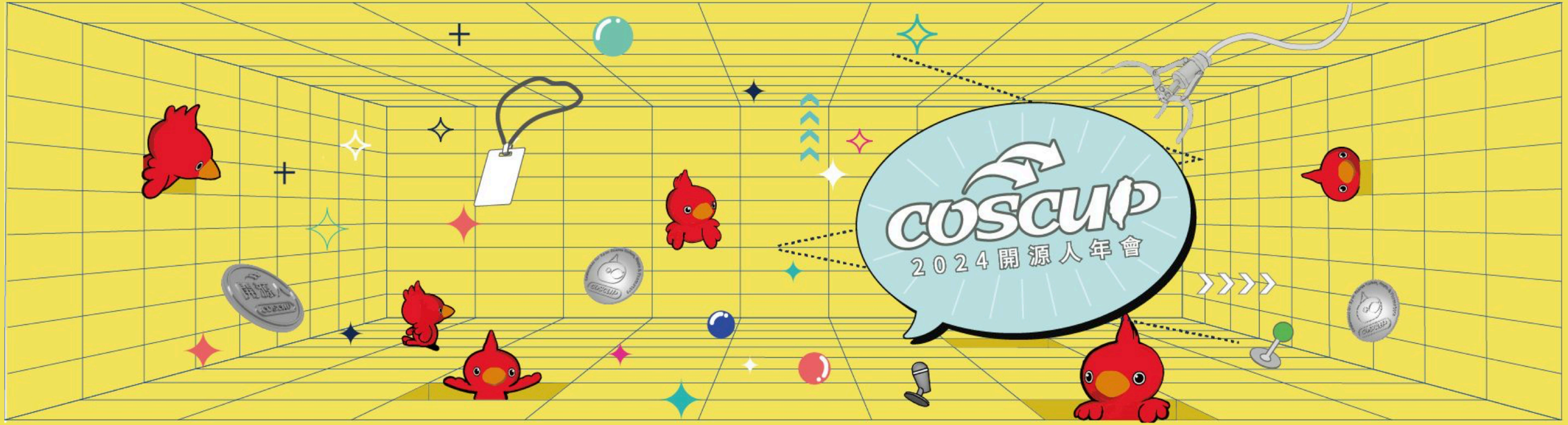
Migrating from Foreground Service to WorkManager

Aayush Gupta



臺灣科技大學NTUST @ Taiwan

August 03 - 04, 2024

https://coscup.org

開源人年會

Conference for Open Source Coders, Users & Promoters

About Me

- Independent Contractor
- Android Developer @ The Calyx Institute, working on CalyxOS
- Senior Staff Member, DevRel @ XDA Developers (Forums)
- FOSS Developer & Contributor

Index

- Running background tasks in Android
- Introduction to WorkManager
- WorkManager in action

Running Background Tasks

Service

Golden Days of Background Tasks

- Simple & easy, no need for notifications or permission
- Deprecated from Android 8.0+, will crash app if used
- System apps can still run background services (not-recommended)

Foreground Service

This ~~is~~ Was The Way

- The de-facto way to run background tasks now, backwards compatible too
- Needs an ongoing notification and bunch of permissions
- Android 12 blocks starting FGS from background unless a permitted use case

Foreground Service

It’s Still Relevant Though

- Since Android 14, Foreground Service types are required as well as a dedicated permission for the specific type
- Additionally, Android 14 recommends migrating to WorkManager for data-sync related jobs
- Android 15 will impose 6 hour time limit on data-sync type services
- Other than data-sync related jobs, Foreground Services are still the preferred way to run background jobs

Introduction to WorkManager

WorkManager

The Cool Kid in the Block

- The primary recommended API for background processing
- Built-upon the Job Scheduler, Foreground Service, and more APIs
- Simple to use and manage
- Compatible with both Java and Kotlin, no dependency upon play services
- Allows to specify multiple constraints to the work as well
- One time and periodic are some of the most used work types

Expedited Work

Right Now, Hopefully

- Expedited work runs immediately on triggering
- Requires specifying setExpedited() method while building work
- Affected by App Standby Quotas and Doze restrictions
- Choice to drop work or run as non-expedited on quota exhaustion
- Periodic work cannot be expedited

Long-Running Work

10 Minutes Not Enough?

- Works are allowed a time-limit of 10 minutes by the OS
- Long-running work should be considered in case more time is required
- Requires calling setForeground() and overriding getForegroundInfo() methods
- Delegated to FGS above Android 12
- Affected by FGS restrictions too (permissions, constraints, etc)

WorkManager in Action

Constraints

Work, But When?

- WorkManager allows specifying several constraints for works
- Build with Constraints.Builder() and apply using setConstraints() method
- Developers can restrict running works based on metered/unmetered data, battery levels, device activity and more
- Fine-grained network control coming in WorkManager 2.10
- Possible to update existing work constraints too

```kotlin
private const val TAG = "UpdateWorker"
private const val UPDATE_WORKER = "UPDATE_WORKER"

fun scheduleAutomatedCheck(context: Context) {
    Log.i(TAG,"Scheduling periodic app updates!")
    WorkManager.getInstance(context)
        .enqueueUniquePeriodicWork(UPDATE_WORKER, KEEP, buildUpdateWork(context))
}

private fun buildUpdateWork(context: Context): PeriodicWorkRequest {
    val updateCheckInterval = Preferences.getInteger(
        context,
        PREFERENCE_UPDATES_CHECK_INTERVAL,
        3
    ).toLong()

    val constraints = Constraints.Builder()
        .setRequiredNetworkType(NetworkType.UNMETERED)
        .setRequiresBatteryNotLow(true)

    if (isMAndAbove()) constraints.setRequiresDeviceIdle(true)

    return PeriodicWorkRequestBuilder<UpdateWorker>(
        repeatInterval = updateCheckInterval,
        repeatIntervalTimeUnit = HOURS,
        flexTimeInterval = 30,
        flexTimeIntervalUnit = MINUTES
    ).setConstraints(constraints.build()).build()
}
```

Working
The Time is Now

- Developers can do their task in the doWork() method
- Automatically ran in background thread
- Returns a Result in the end

```kotlin
override suspend fun doWork(): Result {
    Log.i(TAG, "Cleaning cache")

    PathUtil.getOldDownloadDirectories(appContext).forEach { downloadDir -> // Downloads
        Log.i(TAG, "Deleting old unused download directory: $downloadDir")
        downloadDir.deleteRecursively()
    }

    PathUtil.getDownloadDirectory(appContext).listFiles()?.forEach { download -> // com.example.app
        // Delete if the download directory is empty
        if (download.listFiles().isNullOrEmpty()) {
            Log.i(TAG, "Removing empty download directory for ${download.name}")
            download.deleteRecursively(); return@forEach
        }

        download.listFiles()!!.forEach { versionCode -> // 20240325
            if (versionCode.listFiles().isNullOrEmpty()) {
                // Purge empty non-accessible directory
                Log.i(TAG, "Removing empty directory for ${download.name}, ${versionCode.name}")
                versionCode.deleteRecursively()
            } else {
                versionCode.deleteIfOld()
            }
        }
    }

    return Result.success()
}
```

Sharing Data with/from Workers

This and That

- Possible to share data with Workers using setInputData() method
- Also possible to share data from Workers using setProgress() method
- Shared data can be observed from the UI using LiveData or Kotlin Flows
- Data can be built with Data.Builder()

```kotlin
private fun trigger(download: Download) {
    val inputData = Data.Builder()
        .putString(DOWNLOAD_DATA, gson.toJson(download))
        .build()

    val work = OneTimeWorkRequestBuilder<DownloadWorker>()
        .addTag(DOWNLOAD_WORKER)
        .addTag("$PACKAGE_NAME:${download.packageName}")
        .addTag("$VERSION_CODE:${download.versionCode}")
        .addTag(if (download.isInstalled) DOWNLOAD_UPDATE else DOWNLOAD_APP)
        .setExpedited(OutOfQuotaPolicy.DROP_WORK_REQUEST)
        .setInputData(inputData)
        .build()

    // Ensure all app downloads are unique to preserve individual records
    WorkManager.getInstance(context)
        .enqueueUniqueWork(
            "${DOWNLOAD_WORKER}/${download.packageName}",
            ExistingWorkPolicy.KEEP, work
        )
}
```

Final Thoughts

Work or Not?

- Good but not perfect replacement for FGS with data-sync tasks
- Just another yearly migration

Thank You!