android-long-task-master-changed

Héctor, debes robustecer todo esto: Que el canal de la app y de AndroidLongTask tenga alguna función del tipo START\_LISTENING\_DART STOP\_LISTENING\_DART o algo así para que no se envíen peticiones si no hay nadie al otro lado de la línea.

Acuérdate de onDetachedFromEngine(@NonNull FlutterPluginBinding binding) {

channel.setMethodCallHandler(null);

}

Canales existentes, el handler en cada terminal (el de Flutter y el de Kotlin) y las funciones implementadas:

CANAL: FSE\_APP\_CHANNEL\_NAME

Comunica app\_client.dart con AndroidLongTask.kt (y viceversa), aunque a través del binder appService con la que accedemos a la única instacia de AppService muchos de los métodos realmente ejecutan cosas a través de appService. En principio, si app\_client.dart está activo también lo estará AndroidLongTask.kt --> no debería haber problema por ese lado. Pero parece que AndroidLongTask.kt no muere pero sí app\_client.dart --> como ya he dicho, hay que hacer dos llamadas al canal: una para decir que app\_client.dart está activo y otra cuando no lo está.

|  |  |  |
| --- | --- | --- |
| Método | Quién lo implementa | Qué hace |
| START\_SERVICE | AndroidLongTask.kt | Llama a startAndBindService que arranca (si no estaba arrancado ya, ver1) el ForegroundService (AppService.kt) y hace el binding entre la Activity y el AppService. Con eso, entre otras cosas, tenemos la variable appService con la que accedemos a la ÚNICA1 instancia creada de AppService. De esta forma, no hacen falta mensajes para enviar info de AndroidLongTask.kt al AppService. |
| SET\_INITIAL\_SERVICE\_DATA | AndroidLongTask.kt | Esto lo único que hace es actualizar la variable initialServiceData de AndroidLongTask. |
| GET\_SERVICE\_DATA | AndroidLongTask.kt | Devuelve appService?.serviceData?.toString() |
| STOP\_SERVICE | AndroidLongTask.kt | appService?.stopDartService()  if (appService != null)  serviceConnection?.let { activity.unbindService(it) }  appService = null  --------------------------  Fíjate que stopDartService de AppService tb se puede llamar desde STOP\_SERVICE desde el canal APP\_SERVICE\_CHANNEL\_NAME  Y stopDartService ejecuta:  stopForeground(removeNotification =true)  stopSelf()  engine?.destroy() |
| RUN\_DART\_FUNCTION | AndroidLongTask.kt | if (appService != null) {  appService!!.runDartFunction()  appService!!.setMethodExecutionResultListener { jObject ->  result.success(jObject.toString())}  }  \*Fíjate que se retrasa el envío de result.sucess hasta que acabe la ejecución de la DartFunction (y eso se envía cuando se ejecute endExecution en AppService.kt)  -----------------  runDartFunction esencialmente inicializa un FlutterEngine sin UI y ejecuta en él serviceMain de lib/main.dart (*en ese serviceMain lo que se hace es definir a qué función hay que llamar cuando se invoce el método "runDartCode" dentro del canal APP\_SERVICE\_CHANNEL\_NAME)*. Tb crea el canal APP\_SERVICE\_CHANNEL\_NAME y define sus MethodCallHandlers (en el lado de kotlin). Finalmente, invoca el método "runDartCode"dentro de ese canal (para que se ejecute desde service\_client.dart. y, de ahí, se ejecute la función que he puesto dentro de ServiceClient.setExecutionCallback en serviceMain())  --------------------  setMethodExecutionResultListener se llama cuando se ejecuta endExecution. En nuestro caso, se devuelve un result.success con el último valor de serviceData. Comprobar que esto siempre es correcto y que no se puede llamar a esa función desde otro sitio en el que no se use esa result.success |
| APP\_PAUSED | AndroidLongTask.kt | Le dice a AndroidLongTask.kt que ya no le envíe NOTIFY\_UPDATE |
| APP\_RESUMED | AndroidLongTask.kt | Le dice a AndroidLongTask.kt que ya sí le puede enviar NOTIFY\_UPDATE |
| APP\_DETACHED | AndroidLongTask.kt | Ejecuta detachApp() que ejecuta:  channel.setMethodCallHandler(null)  if (appService!=null) {appService?.setServiceDataObserver(null) serviceConnection?.let { activity?.unbindService(it) }  }  appService = null |
| NOTIFY\_UPDATE | app\_client.dart | En realidad se llama desde la función setData de AppService.kt a través de la variable observer que apunta a la función serviceDataObserver de AndroidLongTask.kt. Le pasa los datos que ha recibido setData a través del método SET\_SERVICE\_DATA del canal APP\_SERVICE\_CHANNEL\_NAME. Por tanto, es una forma de comunicar el servicio de main.dart con la app. Solo se envía si sabemos que la app está en foreground. |

1 Mira en <https://stackoverflow.com/questions/55101657/context-startforegroundservicestartserviceintent-multiple-time-will-call-oncre>. Si ya estaba funcionando, no se vuelve a llamar a onCreate

CANAL: APP\_SERVICE\_CHANNEL\_NAME

Comunica service\_client.dart con AppService.kt (y viceversa). Ambos deben estar siempre activos y solo debe haber una instacia de cada uno (de hecho, es AppService realmente quien crea el FlutterEngine que ejecuta serviceMain() que, a su vez, crea la instacia de ServiceClient). Si quieres que AppService.kt se comunique con el código que pongas en serviceMain, debes definir tus propios callbacks (igual que defines setExecutionCallback) en lugar de crear un canal nuevo.

|  |  |  |
| --- | --- | --- |
| Método | Quién lo implementa | Qué hace |
| SET\_SERVICE\_DATA | AppService.kt | setData:  serviceData = data.  se invoca al observer con esos datos  se actualiza el título y descripción de la notificación |
| END\_EXECUTION | AppService.kt | endExecution:  updateNotification con los valores finales y se llama a result.success de cuando se invocó RUN\_DART\_FUNCTION desde el otro canal FSE\_APP\_CHANNEL\_NAME. |
| stop\_service | AppService.kt | stopDartService:  stopForeground(removeNotification = true)  stopSelf()  engine?.destroy()engine  -----------------  \* Es mejor que se llame desde FSE\_APP\_CHANNEL\_NAME ya que, en ese caso, aparte de ejecutar stopDartService se hace el unbinding entre AndroidLongTask.kt y AppService.kt. Si no, habrá un puntero appService.kt, entre otras cosas, que apuntará a una clase destruida. |
| runDartCode | service\_client.dart | Ejecuta la función dart que se le ha pasado antes como PRIMER argumento a setExecutionCallback. Los parámetros de entrada son los que se especifican en esta invocación. |
| endDartCode | service\_client.dart | Ejecuta la función dart que se le ha pasado antes como SEGUNDO argumento a setExecutionCallback. La idea es notificar que vamos a parar la ejecución del dart code así que se deben parar las cosas y hacer los disposes necesarios. |