链滴

Android WorkManager 工作约束，延迟与查询工作

作者：RustFisher
原文链接：https://ld246.com/article/1629643717828
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

WorkManager工作约束，延迟与查询工作

本文可能会混用“工作”与“任务”这两个词。

本文例子使用Kotlin

准备一个工作类（任务）**UploadWorker2**

```
class UploadWorker2(context: Context, params: WorkerParameters) : Worker(context, params)
{
    override fun doWork(): Result {
        Log.d(TAG, "模拟执行任务2 ${Thread.currentThread()}")
        return Result.success()
    }
}
```

工作约束

约束可让工作**延迟**到满足最佳条件时运行。下面的约束适用于 **WorkManager**。

名称	说明
NetworkType	约束运行工作所需的网络类型。如 Wi-Fi (UNMETERED)。
BatteryNotLow	若为 true，那么当设备处于“量不足模式”时，工作不会运行。
RequiresCharging	若为 true，那么工作只能在备充电时运行。
DeviceIdle	若为 true，则设备必须处于空闲状态能运行工作。如果考虑到其他应用的性能，建议用这个约束。
StorageNotLow	若为 true，那么当设备上的存空间不足时，工作不会运行。

可用 Constraints.Builder()来创建工作约束。然后将**Constraints**实例分配给 WorkRequest.Builder()。

也就是说，工作约束是针对工作**WorkRequest**的。

下面这个工作约束，只有1个要求，在设备使用计费流量是执行任务。

```
val constraints = Constraints.Builder()
        .setRequiredNetworkType(NetworkType.METERED)
        .build()
val d1 = OneTimeWorkRequest.Builder(UploadWorker2::class.java)
        .setConstraints(constraints)
        .addTag("约束").build()
mIdList.add(d1.id) // 登记任务id，可去掉
WorkManager.getInstance(applicationContext).enqueue(d1)
```

真机测试中，如果一开始手机用着wifi，那么任务不会执行。而关掉wifi，打开流量开关，就会执行任。

这就是满足最佳条件时运行。

NetworkType说明

androidx.work.NetworkType是一个枚举类。目前有

枚举	说明
NOT_REQUIRED	这个工作不需要网络
CONNECTED	这个工作需要有网络连接
UNMETERED	需要Wi-Fi
NOT_ROAMING	非漫游网络
METERED	需要按流量计费的网络

延迟工作

如果工作没有约束，或者当工作加入队列时所有约束都得到了满足，那么系统可能会选择立即运行该作。

如果不希望工作立即运行，可以给工作设定一个延迟时间。

下面这个工作是要延迟3秒钟。

```
val d1 = OneTimeWorkRequest.Builder(UploadWorker2::class.java)
        .addTag("延迟1")
        .setInitialDelay(3, TimeUnit.SECONDS)
        .build()
WorkManager.getInstance(applicationContext).enqueue(d1)
```

enqueue任务后，查一下任务状态。工作从 ENQUEUED状态，执行完毕后变成了 SUCCEEDED。

```
WorkInfo{mId='2794b4dd-7c02-4fd0-9851-8798a0da3bb2', mState=ENQUEUED, mOutputDa
a=Data {}, mTags=[com.rustfisher.tutorial2020.workmanaer.WorkManagerAct$UploadWorker
, 延迟1], mProgress=Data {}}

WorkInfo{mId='2794b4dd-7c02-4fd0-9851-8798a0da3bb2', mState=SUCCEEDED, mOutputD
ta=Data {}, mTags=[com.rustfisher.tutorial2020.workmanaer.WorkManagerAct$UploadWorke
2, 延迟1], mProgress=Data {}}
```

查询工作

使用UUID查询

新建工作的时候，系统会分配一个UUID。我们可以记录UUID到 mIdList里。通过UUID可以查询到作当前的状态。

```
val mgr = WorkManager.getInstance(applicationContext)
for (id in mIdList) {
    val cur = mgr.getWorkInfoById(id)
    Log.d(TAG, "查询任务 ${cur.get()}")
}
```

一个工作状态的例子，状态是 ENQUEUED

```
WorkInfo{mId='16cc292d-f140-488f-9ab5-a0d22b95d128', mState=ENQUEUED, mOutputDat
=Data {}, mTags=[com.rustfisher.tutorial2020.workmanaer.WorkManagerAct$UploadWorker,
1], mProgress=Data {}}
```

使用tag查询

前面我们给工作打上tag。可以用**WorkManager**的 getWorkInfosByTag()方法把指定tag的工作查来。

```
val mgr = WorkManager.getInstance(applicationContext)

for (w in mgr.getWorkInfosByTag("约束1").get()) {
    Log.d(TAG, "$w")
}
```

工作状态的例子，下面这个工作 UploadWorker状态是 SUCCEEDED，tag里有 约束1

```
WorkInfo{mId='fed1974c-77e5-46ba-8b38-f6c01d68fe4c', mState=SUCCEEDED, mOutputDat
=Data {}, mTags=[com.rustfisher.tutorial2020.workmanaer.WorkManagerAct$UploadWorker,
约束1], mProgress=Data {}}
```

需要注意的是，工作的状态是存储起来的。即使重启app或者手机，都可以查到这个工作的状态。

假设查询tag是 约束1，我们能得到之前创建的含有这个 约束1标签的工作。

参考

- WorkManager入门示例 https://an.rustfisher.com/android/jetpack/workManager/use1/
- 定义工作请求 https://developer.android.com/topic/libraries/architecture/workmanager/how-o/define-work