[SCA] 离线任务组件介绍

SyncManager 目标是作为一个轻量级的同步任务（SPX 中称为离线任务）协调组件，在 App 运行时按照配置协调轻量级数据的同步，比如SPX 中的地址校正、离线签收、通知已读等数据。对于数据量较大的同步任务，推荐使用 androidx WorkManager。

一、项目接入

配置公司 maven 库，在模块 build.gradle 中配置（最新版本可查看 CHANGELOG）：

```
implementation 'com.shopee.android:syncmanager:{lastReleaseVersion}'
```

二、常用配置及用法

1、新建 SyncClient：

```
mSyncClient = new SyncClient.Builder()
        //  Error
        .setLogLevel(Log.DEBUG)
        //  Dispatcher Synchronizer
        //  Dispatcher  Synchronizer
        .setDispatcher(new SyncDispatcher(64, 10))
        //  Schedule Executor
        .setScheduleExecutor(Executor scheduleExecutor)
        //  Executor
        .setIoExecutor(ScheduledExecutorService ioExecutor)
        .build();
```

2、为每一种后台同步任务注册一个 Synchronizer，提供默认实现 GeneralSynchronizer：

```
mSyncClient.registerSynchronizer(ModelA.class, new GeneralSynchronizer
        //  SyncRepository<T>
        .Builder<>(mSyncClient, new GeneralRepo<ModelA>())
        //  Synchronizer  5 Dispatcher  maxWorkersPerSynchronizer
        .setMaxRunningWorkers(10)
        //
        .setRetryDelayPolicy(RetryDelayPolicy retryDelayPolicy)
        //  ms
        .setMaxRetryInterval(long maxRetryInterval)
        //  ms
        .setMinRetryInterval(long minRetryInterval)
        .build());
mSyncClient.registerSynchronizer(ModelB.class, new GeneralSynchronizer
        .Builder<>(mSyncClient, new GeneralRepo<ModelB>())
        .build());
```

3、随后就可以进行启动/入队/停止等操作：

```
//  Synchronizer
mSyncClient.triggerAllSynchronizer(true);


//
mSyncClient.enqueueNewOffline(new ModelA());
mSyncClient.getSynchronizer(ModelA.class).enqueueNewSyncData(new ModelA());

//  Synchronizer
mSyncClient.triggerAllSynchronizer(false);
```

当同步任务无需保存至 DB 且逻辑大部分相同时，提供一个通用的 GeneralRepo 供参考，该类未加入组件库，因为一般的同步任务都需要先保存至 DB，实用性不大：

```
public class GeneralRepo<T extends SyncData> extends SyncRepository<T> {
    private final List<T> mList = new ArrayList<>();


    @Nullable
    @Override
    public T getData(@NonNull String id) {
        for (T data : mList) {
            if (Objects.equals(data.getUniqueId(), id)) {
                return data;
            }
        }
        return null;
    }

    @Nullable
    @Override
    public List<T> getPendingList(int count) {
        if (count < mList.size()) {
            return mList.subList(0, count);
        }
        return new ArrayList<>(mList);
    }

    @Override
    public void saveToLocal(@NonNull T data) {
        mList.add(data);
    }

    @Override
    public void handleSyncResult(@NonNull T data, SyncResult syncResult) {
        for (int i = 0; i < mList.size(); i++) {
            T t = mList.get(i);
            if (Objects.equals(data.getUniqueId(), t.getUniqueId())) {
                mList.remove(i);
                return;
            }
        }
    }

    @Override
    public ListenableWorker<T> newWorker(@NonNull T data, @NonNull SyncClient syncClient) {
        return new Worker<T>(data, syncClient) {
            @NonNull
            @Override
            public SyncResult doWork() {
                // do your real background work
            }
        };
    }
}
```