[SCA] Startup接入文档

1. 项目背景

随着公司项目需求迭代，项目依赖库越来越多， 在项目**中onCreate()等部分**承载的初始化逻辑变得越来越复杂。比如拿**SPX-PDA项目中Application**启动类举例:

项目中的代码逻辑可分为同步逻辑、异步逻辑、以及互相有依赖逻辑关系的代码；如果对这块逻辑不做优化的话，统一都将默认为同步去处理这些逻辑，而且如果有依赖关系的，需要等被依赖同步模块执行完才能继续执行下去；

2. 优化思路

根据上述痛点，可以看出需要对代码逻辑进行聚合分类，并根据不同的任务集群进行分布初始化，举个简单的对比例子：

- 不对任务进行分类初始化，那么到整个应用首帧被用户看到需要经历Application启动类整个onCreate生命流程：
- 若对任务先进行分类：
- 聚合后分类初始化，应用application启动类onCreate生命流程将变成下面这样:

明显可见，极大缩短了Application初始化的时间，所以进行task聚合分类初始化非常有必要；

3.接入方法

3.1 添加仓库和依赖

```
//build.gradle
allprojects {
    repositories {
        maven { url "https://maven.garenanow.com/nexus/content/groups/public/" }
        ...
    }
}

//sdkmodulebuild.gradlesdk xxx
implementation ("com.shopee.sc:startup:xxx")
```

~~3.2 简单场景接入~~

这种接入暂时废弃，存在的问题：未添加anchor的情况下，代码内部是将同步任务通过post的方法添加到主线程。若在Application的onCreate()中通过这种方式创建同步任务，会发现创建的同步任务并未阻塞主线程。

~~当业务场景相对简单，只需要将多个任务简单分类成同步任务和异步任务，两种任务分别依次执行在两个不同的线程：同步任务执行在当前线程，异步任务执行在新的子线程时适合使用此方法接入。~~

~~直接调用StartUpTaskManager.start(AsyncStartUpTask asyncStartUpTask ,SyncStartUpTask syncStartUpTask, boolean debuggable)传入相应的参数即可，该方法调用后内内部会自动立即开始执行任务。~~

~~参数说明：~~

~~AsyncStartUpTask：用于执行异步任务，具体任务代码放在run()中，将会新起一个线程依次执行run()中所有任务。~~

~~SyncStartUpTask：用于执行同步任务，具体的任务代码放在run()中，将直接在当前线程依次执行run()中所有任务。~~

~~debugable：是否是调试模式~~

~~举例说明：假设现在一共有7个任务A,B,C,D,E,F,G需要执行，现需要将A,B,C,D放在当前线程执行，E,F,G在新的子线程执行，接入代码如下：~~

```
StartUpTaskManager.start(new AsyncStartUpTask(this) {
    @Override
    protected void run(@NonNull String taskId) {
        Log.i(TAG, "AsyncStartUpTask run: " + Thread.currentThread());
                //A.execute()
                //B.execute()
                //C.execute()
                //D.execute()
    }
}, new SyncStartUpTask(this) {
    @Override
    protected void run(@NonNull String taskId) {
        Log.i(TAG, "SyncStartUpTask run: " + Thread.currentThread());
                //E.execute()
                //F.execute()
                //G.execute()
    }
    }
}, BuildConfig.DEBUG);
```

3.3 复杂场景接入

当存在多个任务，这些任务之间存在依赖关系，且依赖关系不止一条链时，可参考以下接入方式。

异步任务：

链中的异步任务将被放在线程池中被执行，异步任务受任务依赖关系的限制，但是不受锚点的限制；

同步任务：

同步任务受任务依赖关系的限制，也受锚点的限制；

链中若没有添加anchor锚点，链中的同步任务将通过handler.post()的方式发送至UI线程轮流执行；

链中若添加了anchor锚点，在链中的anchor任务执行完毕前，所有同步任务直接在当前线程串行执行并保证执行顺序，anchor任务执行完毕后其他任务通过handler.post()的方式发送至UI线程轮流执行；

3.3.1 创建子任务

先分别创建各个子任务

```
public Task10 extends Task {

        public Task10(String taskId,boolean isAsyncTask){
                super(taskId,isAsyncTask);
        }

    @Override
    protected void run(@NonNull String taskId) {
                //
    }
}



public Task11 extends Task {

        public Task11(String taskId,boolean isAsyncTask){
                super(taskId,isAsyncTask);
        }

    @Override
    protected void run(@NonNull String taskId) {
                //
    }
}
```

Task的构造方法包含两个参数：

taskId：task id，需唯一；

isAsyncTask：是否是异步任务 true:是　false:不是；

3.3.2 创建TaskCreator的实现类

该类的createTask()通过taskName返回相应的task。这里返回的task可以是同步任务，也可以是异步任务，两种任务可通过Task的构造方法中的第二个参数区分。

TaskCreator调用createTask()方法创建任务时传递的是taskName。

```
public class StartUpTaskCreator implements TaskCreator{
    @Override
    public Task createTask(String taskName) {
        switch(taskName) {
            case TaskConstants.TASK10 :
                    Task task10 = new Task10("task_id_10", false);
                    return task10;
            case TaskConstants.TASK_11 :
                    Task task11 = new Task11("task_id_11", true);
                    return task11;
            defaule:
                    return null;
        }
    }
}
```

3.3.3 构造任务链

将上面创建的任务构造成内部有依赖关系的任务依赖链：每条依赖链以Project开头，中间可添加若干task，这些task可通过dependOn形成依赖关系；

通过Project.Builder调用add()添加任务、dependOn()创建任务依赖关系时传递的是taskName。

通过Project.Buider调用add()添加任务后，如果没有调用dependOn()添加任务之间的依赖关系，后add()的任务与先add()的任务是并列的，无先后顺序。

```
//1.TaskCreatorTaskFactoryTaskFactorytask nametask
 TaskFactory testTaskFactory. = new TaskFactory(new StartUpTaskCreator);

//2.project
Project.Builder builder1 = new Project.Builder(Datas.PROJECT_1, testTaskFactory);
builder1.add(Datas.TASK_10);
builder1.add(Datas.TASK_11).dependOn(Datas.TASK_10);
builder1.add(Datas.TASK_12).dependOn(Datas.TASK_11);
builder1.add(Datas.TASK_13).dependOn(Datas.TASK_12);
Project project1 = builder1.build();

Project.Builder builder2 = new Project.Builder(Datas.PROJECT_2, testTaskFactory);
builder2.add(Datas.TASK_20);
builder2.add(Datas.TASK_21).dependOn(Datas.TASK_20);
builder2.add(Datas.TASK_22).dependOn(Datas.TASK_21);
builder2.add(Datas.TASK_23).dependOn(Datas.TASK_22);
Project project2 = builder2.build();
```

通过以上方式就创建了两条任务链。

task链间无法保证顺序，但是链内的task是保证顺序的，异步任务也遵循task链执行顺序；若异步任务非常耗时，则尝试先执行其他task链中的任务;

3.3.4 给同步任务链添加anchor锚点(可选)

如果同步任务链中的其他任务需要在某个前置任务执行结束后才执行，那么就可以考虑将这个前置任务设置为锚点。

通过调用addAnchor()添加任务锚点时传递的是taskId。

如果将某个同步任务设置为了锚点任务，则SDK内部会将此锚点任务的优先级设置为Interger.MAX_VALUE，便于优先执行该任务。

```
Task startTask = new TASK_10("task_id_10", false);
project1.dependOn(startTask);
AnchorsManager.getInstance().debuggable(BuildConfig.DEBUG)
        .addAnchor("task_id_10")
        .start(project1);
```

3.3.5 开始执行任务链

```
AnchorsManager.getInstance().debuggable(BuildConfig.DEBUG).start(project1);
AnchorsManager.getInstance().debuggable(BuildConfig.DEBUG).start(project2);
```

4 项目实例

以Ops PDA BaseApplication的onCreate()为例进行说明。

优化前的原始代码如下：

```
public abstract class BaseApplication extends CommonApplication {

    @Override
    public void onCreate() {
        super.onCreate();
        // init common lib first
        new CommonLibInitTask(this).run();
        BuildConfigs.init(this, getCountryVersion());

        AppLifeCycleListener.listenerForeground(this);
        AppLifeCycleListener.registerScreenOffReceiver(this);

        // load raw file
        SoundUtils.getInstance().load(getApplicationContext());
        // init arouter
        RouterHelper.init(this, BuildConfigs.isDebug());
        SimpleToast.init(this);
        UIMatchingHelper.setup(this);
        UIMatchingHelper.register(this, 360, UIMatchingHelper.MATCH_BASE_WIDTH);
        ScanHelper.getInstance().registerApplication(this);
        NyearPrintUtil.init(this);

        // set a default error handle for RxJava
        RxJavaPlugins.setErrorHandler(throwable ->
                CommonPlugins.getLogger().w("an unhandled throwable when using RxJava, \n throwable = "
                        + Log.getStackTraceString(throwable)));

        setScanConfig();
        initLanguage(this);
        updateContext(this);
        initTransify();
        TrackUtil.initTracker(getApplicationContext());

        CodeCoverageInternalUtil.init(this);
    }
......
```

修改代码前首先对onCreate()中的众多任务进行分类：

有些任务是必须要阻塞主线程先执行的，例如Arouter的初始化；而有些任务则对及时性要求不高，无需阻塞主线程，可以放在异步线程中做处理，例如打印功能的初始化、扫码功能初始化等。

对任务进行分类后就可以明确知道哪些任务是同步任务，哪些是异步任务，最后利用启动框架来启动这些任务。

优化后的代码如下：

1.在BaseApplication的onCreate()中创建任务链，并开启任务的执行。

由于我们是想同步任务阻塞住UI线程，因此将同步任务设置为锚点任务(如果不把同步任务设置为锚点，sdk内部是通过Handler.post()的方式将同步任务post到主线程执行的，这样不会阻塞住UI线程。)

```
public abstract class BaseApplication extends CommonApplication {

    @Override
    public void onCreate() {
        super.onCreate();
        AppUtils.init(this, BuildConfigs.isDebug());

        Project.TaskFactory taskFactory = new Project.TaskFactory(new ApplicationInitTaskCreator());
        Project.Builder builder = new Project.Builder(ApplicationInitTaskCreator.PROJECT_APPLICATION_INIT,
taskFactory);
        builder.add(ApplicationInitTaskCreator.TASK_ID_SYNC_APPLICATION_INIT);
        //dependOn()
//        builder.add(ApplicationInitTaskCreator.TASK_ID_ASYNC_APPLICATION_INIT).dependOn
(ApplicationInitTaskCreator.TASK_ID_SYNC_APPLICATION_INIT);
        builder.add(ApplicationInitTaskCreator.TASK_ID_ASYNC_APPLICATION_INIT);

        AnchorsManager.getInstance().addAnchor(ApplicationInitTaskCreator.TASK_ID_SYNC_APPLICATION_INIT)//UI
                .debuggable(BuildConfigs.isDebug())
                .start(builder.build());
    }
```

2.ApplicationInitTaskCreator类负责创建出具体的同步任务和异步任务。

```
public class ApplicationInitTaskCreator implements TaskCreator {
    private static final String TAG = "ApplicationInitTaskCreator";
    public final static String PROJECT_APPLICATION_INIT = "application_init";
    public final static String TASK_ID_SYNC_APPLICATION_INIT = "sync_application_init";
    public final static String TASK_ID_ASYNC_APPLICATION_INIT = "async_application_init";

    @NonNull
    @Override
    public Task createTask(@NonNull String taskName) {
        switch (taskName) {
            case TASK_ID_SYNC_APPLICATION_INIT:
                return new SyncApplicationInitTask(TASK_ID_SYNC_APPLICATION_INIT);
            case TASK_ID_ASYNC_APPLICATION_INIT:
                return new AsyncApplicationInitTask(TASK_ID_ASYNC_APPLICATION_INIT);
            default:
                return null;
        }
    }

    private class SyncApplicationInitTask extends Task {

        private SyncApplicationInitTask(@NonNull String id) {
            super(id, false);
        }

        @Override
        protected void run(@NonNull String taskId) {
            //UIUITask
            Logger.i(TAG, "SyncApplicationInitTask run() start.... thread:" + Thread.currentThread().getName());
            Context context = AppUtils.getContext().getApplicationContext();
            if (!(context instanceof BaseApplication)) {
                return;
            }
            BaseApplication application = (BaseApplication) context;
            // init common lib first
            CommonLibInitTask runnable = new CommonLibInitTask();
            runnable.run();
            initLanguage(context);
            SimpleToast.init(context);

            BuildConfigs.init(context, application.getCountryVersion());
            // init arouter
            RouterHelper.init(application, BuildConfigs.isDebug());

            UIMatchingHelper.setup(application);
```

```
            UIMatchingHelper.register(application, SpxConstants.MATCH_UI_DESIGN_WIDTH, UIMatchingHelper.
MATCH_BASE_WIDTH);

            application.updateContext(context);
            Logger.i(TAG, "SyncApplicationInitTask run() end.... thread:" + Thread.currentThread().getName());

        }
    }

    private class AsyncApplicationInitTask extends Task {

        private AsyncApplicationInitTask(@NonNull String id) {
            super(id, true);
        }

        @Override
        protected void run(@NonNull String taskId) {
            //
            Logger.i(TAG, "AsyncApplicationInitTask run() start.... thread:" + Thread.currentThread().
getName());
            Context context = AppUtils.getContext().getApplicationContext();
            if (!(context instanceof BaseApplication)) {
                return;
            }
            BaseApplication application = (BaseApplication) context;
            // load raw file
            SoundUtils.getInstance().load(context);

            initScan(application);
            NyearPrintUtil.init(application);
            initTransify(application);

            // set a default error handle for RxJava
            RxJavaPlugins.setErrorHandler(throwable ->
                    Logger.w("an unhandled throwable when using RxJava, \n throwable = "
                            + Log.getStackTraceString(throwable)));

            TrackUtil.initTracker(context);
            CodeCoverageInternalUtil.init(context);
            Logger.i(TAG, "AsyncApplicationInitTask run() end.... thread:" + Thread.currentThread().getName());
        }
    }
}
```

参考文档：https://juejin.cn/post/6854573214380032007

https://yummylau.com/2019/03/15/%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90_alpha%E7%9A%84%E7%A0%94%E7%A9%B6%E4%B8%8E%E6%94%B9%E8%BF%9B/