Savepoint 恢复

savepoint的触发和过程与checkpoint大同小异，不做描述。 具体差异在CheckpointCoordinator:[733]

概念

> A job may be resumed from a checkpoint just as from a savepoint by using the checkpoint's meta data file instead (see the savepoint restore guide). Note that if the meta data file is not self-contained, the jobmanager needs to have access to the data files it refers to (see Directory Structure above).
> 一个job可以从检查点路径或保存点路径获取元数据文件并恢复。

恢复流程

有一点需要注意，如果前后两次状态后端类型不同，可能会导致metadata文件无法解析。

程序中需要添加如下配置：

```
    // execution.savepoint.path
    // 从目前来看，savapoint路径需要提前创建，且目录下需要存在检查点生成的metadata文件
    configuration.setString(SavepointConfigOptions.SAVEPOINT_PATH,
"file:///D://savepoint");
    // 其实还有个配置execution.savepoint.ignore-unclaimed-state
    // 指的是当算子变更(新增/删除算子)后，如果false而且metadata用的是之前的文件会起不来

configuration.setBoolean(SavepointConfigOptions.SAVEPOINT_IGNORE_UNCLAIMED_STATE,
true);
```

生成StreamGraph的时候(构造方法，StreamGraphGenerator:[241])，会读取上文的两个配置并初始化为一个SavepointRestoreSettings，如下：

```
public class SavepointRestoreSettings {
    public static SavepointRestoreSettings fromConfiguration(final ReadableConfig
configuration) {
        final String savepointPath =
configuration.get(SavepointConfigOptions.SAVEPOINT_PATH);
        final boolean allowNonRestored =

configuration.get(SavepointConfigOptions.SAVEPOINT_IGNORE_UNCLAIMED_STATE);
        return savepointPath == null
                ? SavepointRestoreSettings.none() // null, false
                : SavepointRestoreSettings.forPath(savepointPath,
allowNonRestored);
    }
}
```

在生成ExecutionGraph的时候，经由DefaultExecutionGraphFactory#createAndRestoreExecutionGraph调用检查点协调器的restoreSavepoint方法：

```
public class CheckpointCoordinator {
    // 从给定的保存点位置恢复
    public boolean restoreSavepoint(
            String savepointPointer,
            boolean allowNonRestored,
            Map<JobVertexID, ExecutionJobVertex> tasks,
            ClassLoader userClassLoader)
            throws Exception {

        Preconditions.checkNotNull(savepointPointer, "The savepoint path cannot be
null.");

        LOG.info(
                "Starting job {} from savepoint {} ({})",
                job,
                savepointPointer,
                (allowNonRestored ? "allowing non restored state" : ""));

        // 获取metadata文件
        final CompletedCheckpointStorageLocation checkpointLocation =
                checkpointStorageView.resolveCheckpoint(savepointPointer);

        // Load the savepoint as a checkpoint into the system
        // 通过MetadataSerializer#deserialize反序列化
        // 对metadata文件做校验，例如并行度
        CompletedCheckpoint savepoint =
                Checkpoints.loadAndValidateCheckpoint(
                        job, tasks, checkpointLocation, userClassLoader,
allowNonRestored);

        completedCheckpointStore.addCheckpoint(
                savepoint, checkpointsCleaner, this::scheduleTriggerRequest);

        // Reset the checkpoint ID counter
        long nextCheckpointId = savepoint.getCheckpointID() + 1;
        checkpointIdCounter.setCount(nextCheckpointId);

        LOG.info("Reset the checkpoint ID of job {} to {}.", job,
nextCheckpointId);

        // 这一步后，流程和普通的失败重启检查点恢复相同
        final OptionalLong restoredCheckpointId =
                restoreLatestCheckpointedStateInternal(
                        new HashSet<>(tasks.values()),

OperatorCoordinatorRestoreBehavior.RESTORE_IF_CHECKPOINT_PRESENT,
                        true,
                        allowNonRestored,
                        true);
```

```
            return restoredCheckpointId.isPresent();
        }
    }
```