1.在Supervisor的启动过程中，即在mkSupervisor()方法中，源码如下：

```
1.  //仅截取mkSupervisor()方法中的部分源码
2.  SyncSupervisorEvent syncSupervisorEvent =
3.          new SyncSupervisorEvent(supervisorId, conf, syncSupEventManager, stormClus
    terState, localState, syncProcessEvent, hb);
4.
5.  int syncFrequence = JStormUtils.parseInt(conf.get(Config.SUPERVISOR_MONITOR_FREQUE
    NCY_SECS));
6.  EventManagerPusher syncSupervisorPusher = new EventManagerPusher(syncSupEventManag
    er, syncSupervisorEvent, syncFrequence);
7.  AsyncLoopThread syncSupervisorThread = new AsyncLoopThread(syncSupervisorPusher);
8.  threads.add(syncSupervisorThread);
```

SyncSupervisorEvent会定时的循环扫描Zookeeper的任务分配目录，看是否有自己的任务，如果有，那么把对应的信息写到本地（Supervisor）机器的指定目录中，这个工作主要是有SyncSupervisorEvent 线程中的run方法来完成的。

2.SyncSupervisorEvent 的成员信息

```
1.  class SyncSupervisorEvent extends RunnableCallback {
2.      //本地Supervisor的id
3.      private String supervisorId;
4.      private EventManager syncSupEventManager;
5.      //Zookeeper操作客户端实例对象
6.      private StormClusterState stormClusterState;
7.      //Supervisor信息存储的本地kv数据库实例对象
8.      private LocalState localState;
9.      private Map<Object, Object> conf;
10.     private SyncProcessEvent syncProcesses;
11.     private int lastTime;
12.     private Heartbeat heartbeat;
13.
14. }
```

3.任务配置信息Assignment对象所包含的相关信息

```
1.  public class Assignment implements Serializable {
2.      //任务分配的类型
3.      public enum AssignmentType {
4.          //Assign-->新分配的任务
5.          //UpdateTopology-->更新的任务
6.          //ScaleTopology-->这个是什么鬼，暂时还没弄明白
7.          Assign, UpdateTopology, ScaleTopology
8.      }
9.
10.     private static final long serialVersionUID = 6087667851333314069L;
11.     //nimbus上代码所在的目录
12.     private final String masterCodeDir;
```

```
    //任务执行的Supervisor节点信息<supervisorId, hostname>，第一个String是supervisorId，第二个String是Supervisor的主机名或者IP地址
    private final Map<String, String> nodeHost;
    //各个task的启动时间，第一个Integer是TaskId，第二个是时间
    private final Map<Integer, Integer> taskStartTimeSecs;
    //任务执行的worker信息
    private final Set<ResourceWorkerSlot> workers;
    //时间戳
    private long timeStamp;
    //任务类型
    private AssignmentType type;
}
```

3.SyncSupervisorEvent 的run()方法

```
@Override
public void run() {
    // STONE_NOTE 记录上次扫描的时间
    lastTime = TimeUtils.current_time_secs();
    //In order to ensure that the status is the same for each execution of syncsupervisor
    // STONE_NOTE 检查Supervisor的心跳状态
    MachineCheckStatus checkStatus = new MachineCheckStatus();
    checkStatus.SetType(heartbeat.getCheckStatus().getType());

    try {
        RunnableCallback syncCallback = new EventManagerZkPusher(this, syncSupEventManager);

        // STONE_NOTE 获取本地LocalStat库中的任务分配的版本信息
        Map<String, Integer> assignmentVersion = (Map<String, Integer>) localState.get(Common.LS_LOCAL_ZK_ASSIGNMENT_VERSION);
        if (assignmentVersion == null) {
            assignmentVersion = new HashMap<String, Integer>();
        }
        // STONE_NOTE 获取本地LocalStat库中的任务分配信息
        Map<String, Assignment> assignments = (Map<String, Assignment>) localState.get(Common.LS_LOCAl_ZK_ASSIGNMENTS);
        if (assignments == null) {
            assignments = new HashMap<String, Assignment>();
        }

        // STONE_NOTE 检查Supervisor的状态信息，如果为panic或者error
        if (checkStatus.getType().equals(MachineCheckStatus.StatusType.panic) || checkStatus.getType().equals(MachineCheckStatus.StatusType.error)){
            // if statuts is pannic or error, it will clear all assignments and kill all workers;
            // STONE_NOTE 清理掉所有的任务分配信息，并杀掉所有的worker
            assignmentVersion.clear();
            assignments.clear();
          } else {
            // STONE_NOTE 获取所有的任务信息，放入本地库中  into
```

```
32.             getAllAssignments(assignmentVersion, assignments, syncCallback);
33.         }
34.
35.         // STONE_NOTE 从STORM-LOCAL-DIR/supervisor/stormdist/目录下获取所有的topol
    ogyIds
36.         List<String> downloadedTopologyIds = StormConfig.get_supervisor_toplogy_li
    st(conf);
37.
38.         // STONE_NOTE 获取当前Supervisor的任务配置信息及对应的端口号
39.         Map<Integer, LocalAssignment> zkAssignment;
40.         zkAssignment = getLocalAssign(stormClusterState, supervisorId, assignments
    );
41.
42.         Map<Integer, LocalAssignment> localAssignment;
43.
44.         // STONE_NOTE 将当前Supervisor任务配置信息写入到LocalState中
45.         try {
46.             LOG.debug("Writing local assignment " + zkAssignment);
47.             localAssignment = (Map<Integer, LocalAssignment>) localState.get(Commo
    n.LS_LOCAL_ASSIGNMENTS);
48.             if (localAssignment == null) {
49.                 localAssignment = new HashMap<Integer, LocalAssignment>();
50.             }
51.             localState.put(Common.LS_LOCAL_ASSIGNMENTS, zkAssignment);
52.
53.         } catch (IOException e) {}
54.
55.         // STONE_NOTE 获取所有更新的Topology任务 根据assignment的时间戳判断任务是
    否更新
56.         Set<String> updateTopologys;
57.         updateTopologys = getUpdateTopologys(localAssignment, zkAssignment, assign
    ments);
58.         // STONE_NOTE 获取所有需要重新下载的Topology任务
59.         Set<String> reDownloadTopologys = getNeedReDownloadTopologys(localAssignme
    nt);
60.         if (reDownloadTopologys != null) {
61.             updateTopologys.addAll(reDownloadTopologys);
62.         }
63.
64.         // STONE_NOTE 从Zookeeper上下载要执行的任务代码
65.         Map<String, String> topologyCodes = getTopologyCodeLocations(assignments,
    supervisorId);
66.         // downloadFailedTopologyIds which can't finished download binary from nim
    bus
67.         // STONE_NOTE 记录所有执行代码下载失败的TopologyId
68.         Set<String> downloadFailedTopologyIds = new HashSet<String>();
69.
70.         // STONE_NOTE 下载执行代码
71.         downloadTopology(topologyCodes, downloadedTopologyIds, updateTopologys, as
    signments, downloadFailedTopologyIds);
72.
73.         // STONE_NOTE 移除没有用的Topology代码，即在downloadedTopologyIds的id中没
    有的代码
74.         removeUselessTopology(topologyCodes, downloadedTopologyIds);
```

```
75.
76.         // STONE_NOTE 运行同步进程事件
77.         syncProcesses.run(zkAssignment, downloadFailedTopologyIds);
78.
79.         // STONE_NOTE 触发心跳更新
80.         heartbeat.updateHbTrigger(true);
81.
82.         try {
83.             // update localState
84.             localState.put(Common.LS_LOCAL_ZK_ASSIGNMENT_VERSION, assignmentVersio
    n);
85.             localState.put(Common.LS_LOCAl_ZK_ASSIGNMENTS, assignments);
86.
87.         } catch (IOException e) {}
88.     } catch (Exception e) {}
89.     if (checkStatus.getType().equals(MachineCheckStatus.StatusType.panic)){
90.         // if statuts is pannic, it will kill supervisor;
91.         JStormUtils.halt_process(0, "Supervisor Machine Check Status : Panic , !!!
    !shutdown!!!!");
92.     }
93.
94. }
```

4.getAllAssignments()获取所有的任务信息，放入本地库中

```
1.  // STONE_NOTE 获取所有的任务信息，放入本地库中
2.  private void getAllAssignments(Map<String, Integer> assignmentVersion, Map<String,
     Assignment> localZkAssignments,
3.          RunnableCallback callback) throws Exception {
4.      Map<String, Assignment> ret = new HashMap<String, Assignment>();
5.      Map<String, Integer> updateAssignmentVersion = new HashMap<String, Integer>();
6.
7.      // get /assignments {topology_id}
8.      // STONE_NOTE 获取Zookeeper的 /storm/assignments/下的所有{topology_id}，存放在
    list中
9.      List<String> assignments = stormClusterState.assignments(callback);
10.     if (assignments == null) {
11.         assignmentVersion.clear();
12.         localZkAssignments.clear();
13.         LOG.debug("No assignment of ZK");
14.         return;
15.     }
16.
17.     for (String topology_id : assignments) {
18.
19.         // STONE_NOTE 获取zk上的版本号
20.         Integer zkVersion = stormClusterState.assignment_version(topology_id, call
    back);
21.         LOG.debug(topology_id + "'s assigment version of zk is :" + zkVersion);
22.         // STONE_NOTE 获取本地库中版本号
23.         Integer recordedVersion = assignmentVersion.get(topology_id);
24.         LOG.debug(topology_id + "'s assigment version of local is :" + recordedVer
```

```
sion);

        Assignment assignment = null;
        // STONE_NOTE 如果本地版本号和zk上的版本号一致，则说明是一个已经存在的任务分配
        if (recordedVersion !=null && zkVersion !=null && recordedVersion.equals(zkVersion)) {
            // STONE_NOTE 从本地库中获取任务配置信息
            assignment = localZkAssignments.get(topology_id);
        }
        //because the first version is 0
        // STONE_NOTE 如果本地库中不存在，则从zk上获取
        if (assignment == null) {
            // STONE_NOTE 从Zookeeper上获取任务配置信息，反序列化获得Assignment对象
            assignment = stormClusterState.assignment_info(topology_id, callback);
        }
        // STONE_NOTE 如果zk上没有获取到，则获取失败
        if (assignment == null) {
            LOG.error("Failed to get Assignment of " + topology_id + " from ZK");
            continue;
        }
        // STONE_NOTE 将zk上任务的版本信息更新到本地库中
        updateAssignmentVersion.put(topology_id, zkVersion);
        // STONE_NOTE 将topology_id和任务信息对象放入到map中
        ret.put(topology_id, assignment);
    }
    assignmentVersion.clear();
    assignmentVersion.putAll(updateAssignmentVersion);
    localZkAssignments.clear();
    localZkAssignments.putAll(ret);
}
```

5.syncProcesses.run(zkAssignment, downloadFailedTopologyIds)

```
public void run(Map<Integer, LocalAssignment> localAssignments, Set<String> downloadFailedTopologyIds) {
    LOG.debug("Syncing processes, interval seconds:" + TimeUtils.time_delta(lastTime));
    lastTime = TimeUtils.current_time_secs();
    try {
        // STONE_NOTE 从LocalState中获取所有分配的Tasks
        if (localAssignments == null) {
            localAssignments = new HashMap<>();
        }
        // STONE_NOTE 从local_dir/worker/ids/heartbeat中获得本地所有的worker的状态, Map<workerid [WorkerHeartbeat, state]>
        Map<String, StateHeartbeat> localWorkerStats;
        try {
            localWorkerStats = getLocalWorkerStats(conf, localState, localAssignments);
        } catch (Exception e) {
```

```
14.             LOG.error("Failed to get Local worker stats");
15.             throw e;
16.         }
17.         Map<String, Integer> taskCleaupTimeoutMap;
18.         Set<Integer> keepPorts = null;
19.         try {
20.             taskCleaupTimeoutMap = (Map<String, Integer>) localState.get(Common.LS
    _TASK_CLEANUP_TIMEOUT);
21.             // STONE_NOTE 杀死没有用的worker，并从LocalState中移除
22.             keepPorts = killUselessWorkers(localWorkerStats, localAssignments, tas
    kCleaupTimeoutMap);
23.             localState.put(Common.LS_TASK_CLEANUP_TIMEOUT, taskCleaupTimeoutMap);
24.         } catch (IOException e) {}
25.         // STONE_NOTE 检测新的worker
26.         checkNewWorkers(conf);
27.         // STONE_NOTE 检测哪个Topology需要更新
28.         checkNeedUpdateTopologys(localWorkerStats, localAssignments);
29.         // STONE_NOTE 为下载失败的Topology在空闲的端口上，启动新的worker
30.         startNewWorkers(keepPorts, localAssignments, downloadFailedTopologyIds);
31.
32.     } catch (Exception e) {}
33. }
```