1.Nimbus的启动过程

（1）Nimbus 的启动脚本命令：bin/storm nimbus

实际调用过程：

storm脚本中的main方法---->调用 def nimbus---->exec_storm_class

在exec_storm_class中，最终调用java命令，即：

java -server backtype.storm.daemon.nimbus----->执行NimbusServer的main方法。

（2）NimbusServer的main方法源码如下：

```
1.  public static void main(String[] args) throws Exception {
2.
3.      Thread.setDefaultUncaughtExceptionHandler(new DefaultUncaughtExceptionHandler(
    ));
4.
5.      // read configuration files
6.      // STONE_NOTE 获取Storm集群配置
7.      @SuppressWarnings("rawtypes")
8.      Map config = Utils.readStormConfig();
9.
10.     JStormServerUtils.startTaobaoJvmMonitor();
11.
12.     // STONE_NOTE 创建NimbusServer实例
13.     NimbusServer instance = new NimbusServer();
14.
15.     // STONE_NOTE 创建DefaultInimbus实例
16.     INimbus iNimbus = new DefaultInimbus();
17.
18.     // STONE_NOTE 开始启动nimbus
19.     instance.launchServer(config, iNimbus);
20.
21. }
```

（3）在NimbusServer的main方法中，调用launchServer()启动nimbus

```
1.  private void launchServer(final Map conf, INimbus inimbus) {
2.
3.      try {
4.          // STONE_NOTE 检测是不是分布式集群模式
5.          StormConfig.validate_distributed_mode(conf);
6.
7.          createPid(conf);
8.
9.          // STONE_NOTE 初始化停止nimbus，清理相关数据
10.         initShutdownHook();
11.
12.         // STONE_NOTE 屁事都没做，这是要弄啥呢
13.         inimbus.prepare(conf, StormConfig.masterInimbus(conf));
14.
```

```
        // STONE_NOTE 使用conf和inimbus 构建一个实例对象return new NimbusData(conf, inimbus)
        data = createNimbusData(conf, inimbus);

        // STONE_NOTE 初始化follower线程，这个follower线程是干啥的呢
        initFollowerThread(conf);

        // STONE_NOTE deamon--->守护进程
        // STONE_NOTE 获取nimbus守护进程的http端口从配置文件中storm.yaml
        int port = ConfigExtension.getNimbusDeamonHttpserverPort(conf);
        hs = new Httpserver(port, conf);
        // STONE_NOTE 启动一个socket服务进程
        hs.start();

        // STONE_NOTE 初始化一个循环同步的线程容器
        initContainerHBThread(conf);

        while (!data.isLeader())
            Utils.sleep(5000);

        init(conf);
    } catch (Throwable e) {}
}
```

（4）launchServer中的init()方法

```
private void init(Map conf) throws Exception {

    // STONE_NOTE 清除残留的无效的Topology
    // STONE_NOTE 清理本地目录/local-storm-dir/nimbus/stormdist中有，但是zk的storms节点下没有的 或者 zk的storm目录下有而本地目录没有的Topology
    NimbusUtils.cleanupCorruptTopologies(data);

    // STONE_NOTE 初始化Topology分配
    initTopologyAssign();

    // STONE_NOTE 初始化Topology的状态
    initTopologyStatus();

    // STONE_NOTE 初始化一个清理器，用来清理/nimbus/inbox中的jar
    initCleaner(conf);

    serviceHandler = new ServiceHandler(data);

    if (!data.isLocalMode()) {

        //data.startMetricThreads();
        // STONE_NOTE 初始化集群的监控
        initMonitor(conf);

        initThrift(conf);

```

```
    }
}
```

（5）initTopologyStatus()方法源码如下：

```
private void initTopologyStatus() throws Exception {
    // STONE_NOTE 获取Zookeeper中有效的Topology，将所有的有效的Topology id放入到一个list中
    List<String> active_ids = data.getStormClusterState().active_storms();

    if (active_ids != null) {

        for (String topologyid : active_ids) {
            // set the topology status as startup
            // in fact, startup won't change anything
            // STONE_NOTE 将Topology的状态设置为startup，事实上，startup状态并没有任何的改变
            NimbusUtils.transition(data, topologyid, false, StatusType.startup);
            // STONE_NOTE 更新Topology任务超时时间
            NimbusUtils.updateTopologyTaskTimeout(data, topologyid);
            // STONE_NOTE 更新Topology任务的心跳状态  Hb--->heartbeat
            NimbusUtils.updateTopologyTaskHb(data, topologyid);
        }

    }
}
```

至此，nimbus的启动过程完成！

2.Supervisor的启动过程

（1）Supervisor 的启动脚本命令：bin/storm supervisor

实际调用过程：

storm脚本中的main方法---->调用 def supervisor ---->exec_storm_class

在exec_storm_class中，最终调用java命令，即：

java -server backtype.storm.daemon.supervisor----->执行Supervisor的main方法。

```
/**
 * supervisor daemon 的入口
 */
public static void main(String[] args) {

    Thread.setDefaultUncaughtExceptionHandler(new DefaultUncaughtExceptionHandler());

    JStormServerUtils.startTaobaoJvmMonitor();

    // STONE_NOTE Supervisor启动时，创建实例
    Supervisor instance = new Supervisor();
```

```
12.
13.     // STONE_NOTE 运行其run()方法
14.     instance.run();
15.
16. }
```

（2）Supervisor的main方法中，创建Supervisor的实例，并调用其run()方法。

```
1.  /**
2.   * 在run()方法中启动Supervisor
3.   */
4.  public void run() {
5.      SupervisorManger supervisorManager;
6.      try {
7.          // STONE_NOTE 读取Storm集群的配置
8.          Map<Object, Object> conf = Utils.readStormConfig();
9.
10.         // STONE_NOTE 验证分布式模式
11.         StormConfig.validate_distributed_mode(conf);
12.
13.         createPid(conf);
14.
15.         // STONE_NOTE 开始启动Supervisor
16.         supervisorManager = mkSupervisor(conf, null);
17.
18.         JStormUtils.redirectOutput("/dev/null");
19.
20.         initShutdownHook(supervisorManager);
21.
22.         while (!supervisorManager.isFinishShutdown()) {
23.             try {
24.                 Thread.sleep(1000);
25.             } catch (InterruptedException ignored) {
26.             }
27.         }
28.
29.     } catch (Throwable e) {}
30.
31. }
```

（3）在run()方法中调用`mkSupervisor(conf, null)`方法启动Supervisor。

- 清理Supervisor本地临时目录
- 获取Zookeeper的客户端操作实例对象
- 创建LocalStat的key-value本地数据库
- 使用UUID生成一个Supervisor的id，并放入到LocalStat的key-value本地数据库中
- 移除LocalStat中的"lcoal-zk-assignment-version"和"local-zk-assignments"信息
- 创建Supervisor的心跳对象，设置Supervisor的心跳时间supervisor.heartbeat.frequency.secs，并将Supervisor的信息写入Zookeeper中
- 将心跳同步到Nimbus和Apsara容器

- 创建和启动Supervisor的同步线程，每隔supervisor.monitor.frequency.secs秒，运行一次Supervisor的同步线程
- 检查Supervisor是否运行正常
- 最后，返回一个SupervisorManger，SupervisorManger能够停止所有的Supervisor和Worker

源码如下：

```
1.  public SupervisorManger mkSupervisor(Map conf, IContext sharedContext) throws Exce
    ption {
2.
3.      // STONE_NOTE 获取Supervisor本地临时目录路径
4.      String path = StormConfig.supervisorTmpDir(conf);
5.      // STONE_NOTE 清理Supervisor本地临时目录的文件
6.      FileUtils.cleanDirectory(new File(path));
7.
8.      // STONE_NOTE 创建Zookeeper的客户端操作实例对象StromClusterState
9.      StormClusterState stormClusterState = Cluster.mk_storm_cluster_state(conf);
10.
11.     // STONE_NOTE 获取主机名HostName
12.     String hostName = JStormServerUtils.getHostName(conf);
13.     WorkerReportError workerReportError =
14.             new WorkerReportError(stormClusterState, hostName);
15.
16.     // STONE_NOTE 创建LocalStat，LocalStat是一个键值对的数据库
17.     LocalState localState = StormConfig.supervisorState(conf);
18.
19.     // STONE_NOTE 获取supervisorId
20.     String supervisorId = (String) localState.get(Common.LS_ID);
21.     // STONE_NOTE 如果supervisorId不存在
22.     if (supervisorId == null) {
23.         // STONE_NOTE 通过UUID.randomUUID()生成supervisorId，并放入LocalState数据
    库中 LS_ID = "supervisor-id"
24.         supervisorId = UUID.randomUUID().toString();
25.         localState.put(Common.LS_ID, supervisorId);
26.     }
27.     //clean LocalStat's zk-assgiment&versions
28.     // STONE_NOTE 移除LocalStat中的"lcoal-zk-assignment-version"和"local-zk-assign
    ments"信息
29.     localState.remove(Common.LS_LOCAl_ZK_ASSIGNMENTS);
30.     localState.remove(Common.LS_LOCAL_ZK_ASSIGNMENT_VERSION);
31.
32.     Vector<AsyncLoopThread> threads = new Vector<>();
33.
34.     // STONE_NOTE 创建Supervisor的心跳对象，设置Supervisor的心跳时间supervisor.hea
    rtbeat.frequency.secs，并将Supervisor的信息写入Zookeeper中
35.     Heartbeat hb = new Heartbeat(conf, stormClusterState, supervisorId, localState
    , checkStatus);
36.     hb.update();
37.     // STONE_NOTE 将心跳同步到Nimbus
38.     AsyncLoopThread heartbeat = new AsyncLoopThread(hb, false, null, Thread.MIN_PR
    IORITY, true);
39.     threads.add(heartbeat);
```

```
40.
41.     // STONE_NOTE 同步心跳到Apsara容器
42.     AsyncLoopThread syncContainerHbThread = SyncContainerHb.mkSupervisorInstance(c
    onf);
43.     if (syncContainerHbThread != null) {
44.         threads.add(syncContainerHbThread);
45.     }
46.
47.     // STONE_NOTE 创建和启动Supervisor的同步线程，每隔supervisor.monitor.frequency
    .secs秒，运行一次Supervisor的同步线程
48.     ConcurrentHashMap<String, String> workerThreadPids = new ConcurrentHashMap<>()
    ;
49.     SyncProcessEvent syncProcessEvent = new SyncProcessEvent(supervisorId, conf, l
    ocalState, workerThreadPids, sharedContext, workerReportError);
50.
51.     EventManagerImp syncSupEventManager = new EventManagerImp();
52.     AsyncLoopThread syncSupEventThread = new AsyncLoopThread(syncSupEventManager);
53.     threads.add(syncSupEventThread);
54.
55.     SyncSupervisorEvent syncSupervisorEvent =
56.             new SyncSupervisorEvent(supervisorId, conf, syncSupEventManager, storm
    ClusterState, localState, syncProcessEvent, hb);
57.
58.     int syncFrequence = JStormUtils.parseInt(conf.get(Config.SUPERVISOR_MONITOR_FR
    EQUENCY_SECS));
59.     EventManagerPusher syncSupervisorPusher = new EventManagerPusher(syncSupEventM
    anager, syncSupervisorEvent, syncFrequence);
60.     AsyncLoopThread syncSupervisorThread = new AsyncLoopThread(syncSupervisorPushe
    r);
61.     threads.add(syncSupervisorThread);
62.
63.     Httpserver httpserver = null;
64.     if (!StormConfig.local_mode(conf)) {
65.         // STONE_NOTE  启动httpserver，这是要弄啥呢？
66.         int port = ConfigExtension.getSupervisorDeamonHttpserverPort(conf);
67.         httpserver = new Httpserver(port, conf);
68.         httpserver.start();
69.     }
70.
71.     // STONE_NOTE 检查Supervisor是否运行正常
72.     if (!StormConfig.local_mode(conf) && ConfigExtension.isEnableCheckSupervisor(c
    onf)) {
73.         SupervisorHealth supervisorHealth = new SupervisorHealth(conf, checkStatus
    ,supervisorId);
74.         AsyncLoopThread healthThread = new AsyncLoopThread(supervisorHealth, false
    , null, Thread.MIN_PRIORITY, true);
75.         threads.add(healthThread);
76.     }
77.
78.     // STONE_NOTE 返回一个SupervisorManger，SupervisorManger能够停止所有的Supervis
    or和Worker
79.     return new SupervisorManger(conf, supervisorId, threads, syncSupEventManager,
    httpserver, stormClusterState, workerThreadPids);
80. }
```

至此，Supervisor的启动过程完成！