Worker的启动是在Supervisor获取任务的过程中完成的：

1.Supervisor启动后，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。在SyncSupervisorEvent 线程的run方法中，最后给processEventManager添加一个syncProcesses事件，而Worker的创建和启动就是在syncProcesses的run方法中完成的。

2.在syncProcesses的run方法中调用startNewWorkers()方法，启动新的Worker。

```
1.  public void run(Map<Integer, LocalAssignment> localAssignments, Set<String> downlo
    adFailedTopologyIds) {
2.      lastTime = TimeUtils.current_time_secs();
3.      try {
4.          // STONE_NOTE 从LocalState中获取所有分配的Tasks
5.          if (localAssignments == null) {
6.              localAssignments = new HashMap<>();
7.          }
8.          // STONE_NOTE 从local_dir/worker/ids/heartbeat中获得本地所有的worker的状态
    , Map<workerid [WorkerHeartbeat, state]>
9.          Map<String, StateHeartbeat> localWorkerStats;
10.         try {
11.             localWorkerStats = getLocalWorkerStats(conf, localState, localAssignme
    nts);
12.         } catch (Exception e) {}String, Integer> taskCleaupTimeoutMap;
13.         Set<Integer> keepPorts = null;
14.         try {
15.             taskCleaupTimeoutMap = (Map<String, Integer>) localState.get(Common.LS
    _TASK_CLEANUP_TIMEOUT);
16.             // STONE_NOTE 杀死没有用的worker，并从LocalState中移除
17.             keepPorts = killUselessWorkers(localWorkerStats, localAssignments, tas
    kCleaupTimeoutMap);
18.             localState.put(Common.LS_TASK_CLEANUP_TIMEOUT, taskCleaupTimeoutMap);
19.         } catch (IOException e) {}
20.         // STONE_NOTE 检测新的worker
21.         checkNewWorkers(conf);
22.         // STONE_NOTE 检测哪个Topology需要更新
23.         checkNeedUpdateTopologys(localWorkerStats, localAssignments);
24.         // STONE_NOTE 为下载失败的Topology在空闲的端口上，启动新的worker
25.         startNewWorkers(keepPorts, localAssignments, downloadFailedTopologyIds);
26.
27.     } catch (Exception e) {}
28.
29. }
```

3.在startNewWorkers()方法中，生成新的worker id、创建新的worker目录，调用launchWorker()方法启动worker

```
1.  private void startNewWorkers(Set<Integer> keepPorts, Map<Integer, LocalAssignment>
```

```
 localAssignments,
                                Set<String> downloadFailedTopologyIds) throws Exception {

    // STONE_NOTE 获得分配好的任务
    Map<Integer, LocalAssignment> newWorkers = JStormUtils.select_keys_pred(keepPorts, localAssignments);
    // STONE_NOTE 生成一个的worker id
    Map<Integer, String> newWorkerIds = new HashMap<>();

    for (Entry<Integer, LocalAssignment> entry : newWorkers.entrySet()) {
        Integer port = entry.getKey();
        LocalAssignment assignment = entry.getValue();
        if (assignment != null && assignment.getTopologyId() != null &&
                downloadFailedTopologyIds.contains(assignment.getTopologyId())) {
            LOG.info("Can't start this worker: " + port + " about the topology: "
+ assignment.getTopologyId()
                    + ", due to the damaged binary !!");
            continue;
        }
        String workerId = UUID.randomUUID().toString();
        newWorkerIds.put(port, workerId);
        // STONE_NOTE 创建一个新的Worker的id目录   LOCALDIR/workers/newworkid/pids
        try {
            StormConfig.worker_pids_root(conf, workerId);
        } catch (IOException e1) {}

        StringBuilder sb = new StringBuilder();
        sb.append("Launching worker with assiangment ");
        sb.append(assignment).append(" for the supervisor ").append(supervisorId)
                .append(" on port ").append(port).append(" with id ").append(workerId);

        try {
            String clusterMode = StormConfig.cluster_mode(conf);

            // STONE_NOTE 启动worker
            if (clusterMode.equals("distributed")) {
                // STONE_NOTE 调用launchWorker方法启动worker
                launchWorker(conf, sharedContext, assignment.getTopologyId(), supervisorId, port, workerId, assignment);
            } else if (clusterMode.equals("local")) {
                launchWorker(conf, sharedContext, assignment.getTopologyId(), supervisorId, port, workerId, workerThreadPids);
            }
        } catch (Exception e) {}
    }
    markAllNewWorkers(newWorkerIds);
}
```

4.集群模式下，在launchWorker()方法中，拼接java命令，启动Worker

```
public void launchWorker(Map conf, IContext sharedContext, String topologyId, String supervisorId,
                         Integer port, String workerId, LocalAssignment assignment) throws IOException {

    Map stormConf = StormConfig.read_supervisor_topology_conf(conf, topologyId);
    String stormHome = System.getProperty("jstorm.home");
    if (StringUtils.isBlank(stormHome)) {
        stormHome = "./";
    }
    Map totalConf = new HashMap();
    totalConf.putAll(conf);
    totalConf.putAll(stormConf);
    Map<String, String> environment = new HashMap<String, String>();
    if (ConfigExtension.getWorkerRedirectOutput(totalConf)) {
        environment.put("REDIRECT", "true");
    } else {
        environment.put("REDIRECT", "false");
    }
    environment.put("LD_LIBRARY_PATH", (String) totalConf.get(Config.JAVA_LIBRARY_PATH));
    environment.put("jstorm.home", stormHome);
    environment.put("jstorm.workerId", workerId);
    String launcherCmd = getLauncherParameter(assignment, totalConf, stormHome, topologyId, port);
    String workerCmd = getWorkerParameter( assignment,
             totalConf,
             stormHome,
             topologyId,
             supervisorId,
             workerId,
             port);
    String cmd = launcherCmd + " " + workerCmd;
    cmd = cmd.replace("%JSTORM_HOME%", stormHome);
    JStormUtils.launchProcess(cmd, environment, true);
}
```

（1）launchWorker()方法调用java命令，启动worker的过程，首先是调用JStormUtils.launchProcess()

```
public static String launchProcess(final String command, final Map<String, String>
 environment, boolean backend) throws IOException {
        String[] cmds = command.split(" ");

        ArrayList<String> cmdList = new ArrayList<String>();
        for (String tok : cmds) {
            if (!StringUtils.isBlank(tok)) {
                cmdList.add(tok);
            }
        }

        return launchProcess(command, cmdList, environment, backend);
    }
```

（2）在launchProcess()方法中，拆分并组装命令，然后调用launchProcess(command, cmdList, environment, backend)方法

```
1.  public static String launchProcess(final String command, final List<String> cmdlis
    t,
2.                                     final Map<String, String> environment, boolean
    backend) throws IOException {
3.      if (backend) {
4.          new Thread(new Runnable() {
5.              @Override
6.              public void run() {
7.                  List<String> cmdWrapper = new ArrayList<String>();
8.
9.                  cmdWrapper.add("nohup");
10.                 cmdWrapper.addAll(cmdlist);
11.                 cmdWrapper.add("&");
12.
13.                 try {
14.                     launchProcess(cmdWrapper, environment);
15.                 } catch (IOException e) {
16.                     LOG.error("Failed to run nohup " + command + " &," + e.getCaus
    e(), e);
17.                 }
18.             }
19.         }).start();
20.         return null;
21.     } else {
22.         try {
23.             Process process = launchProcess(cmdlist, environment);
24.
25.             StringBuilder sb = new StringBuilder();
26.             String output = JStormUtils.getOutput(process.getInputStream());
27.             String errorOutput = JStormUtils.getOutput(process.getErrorStream());
28.             sb.append(output);
29.             sb.append("\n");
30.             sb.append(errorOutput);
31.
32.             int ret = process.waitFor();
33.             if (ret != 0) {
34.                 LOG.warn(command + " is terminated abnormally. ret={}, str={}", re
    t, sb.toString());
35.             }
36.             return sb.toString();
37.         } catch (Throwable e) {
38.             LOG.error("Failed to run " + command + ", " + e.getCause(), e);
39.         }
40.
41.         return "";
42.     }
43. }
```

（3）最后调用jdk的launchProcess(cmdWrapper, environment)方法，执行命令，启动worker

```
protected static java.lang.Process launchProcess(final List<String> cmdlist,
                                                  final Map<String, String> environ
ment) throws IOException {
    ProcessBuilder builder = new ProcessBuilder(cmdlist);
    builder.redirectErrorStream(true);
    Map<String, String> process_evn = builder.environment();
    for (Entry<String, String> entry : environment.entrySet()) {
        process_evn.put(entry.getKey(), entry.getValue());
    }

    return builder.start();
}
```

（5）通过ProcessBuilder执行一条命令 java -server backtype.storm.daemon.worker 调用Worker的main方法，在main方法中调用mk_worker()方法。

以下的流程与本地模式的启动方式一致。

5.在本地模式下，调用Worker类的mk_worker()方法，启动worker

```
public void launchWorker(Map conf, IContext sharedcontext, String topologyId,
                         String supervisorId, Integer port, String workerId,
                         ConcurrentHashMap<String, String> workerThreadPidsAtom) t
hrows Exception {
    String pid = UUID.randomUUID().toString();

    WorkerShutdown worker = Worker.mk_worker(conf, sharedcontext, topologyId, supe
rvisorId, port, workerId, null);

    ProcessSimulator.registerProcess(pid, worker);

    workerThreadPidsAtom.put(workerId, pid);
}
```

6.在Worker类的mk_worker()方法中，拼装worker的相关信息，创建worker实例对象，并调用其execute()方法

```
public static WorkerShutdown mk_worker(Map conf, IContext context, String topology
_id, String supervisor_id, int port, String worker_id, String jar_path)
        throws Exception {

    StringBuilder sb = new StringBuilder();
    sb.append("topologyId:" + topology_id + ", ");
    sb.append("port:" + port + ", ");
    sb.append("workerId:" + worker_id + ", ");
    sb.append("jarPath:" + jar_path + "\n");
    Worker w = new Worker(conf, context, topology_id, supervisor_id, port, worker_
id, jar_path);
    return w.execute();
```

```
11.  }
```