

1.在Topology的任务提交中，最后调用StormSubmitter的submitTopology方法将任务提交给nimbus；实际上，在StormSubmitter的submitTopology中，是NimbusClient通过RPC（远程过程调用），调用Nimbus的submitTopologyWithOpts方法或者submitTopology方法。

```
1.  if (opts != null) {
2.      // STONE_NOTE 新的提交方式，携带opts参数 提交Topology任务
3.      client.getClient().submitTopologyWithOpts(name, path, serConf, topology, opts)
4.  } else {
5.      // this is for backwards compatibility
6.      // STONE_NOTE 这个是为了兼容之前的版本
7.      client.getClient().submitTopology(name, path, serConf, topology);
8.  }
```

2.submitTopology()方法是Nimbus接口中的一个方法，其实现类有ServiceHandler，在ServiceHandler的submitTopology()方法实现源码如下：

```
1.  @Override
2.  public void submitTopology(String name, String uploadedJarLocation, String jsonConf, StormTopology topology) throws TException, AlreadyAliveException,
3.      InvalidTopologyException, TopologyAssignException {
4.      SubmitOptions options = new SubmitOptions(TopologyInitialStatus.ACTIVE);
5.      submitTopologyWithOpts(name, uploadedJarLocation, jsonConf, topology, options)
6.  ;
7.  }
```

从上面的源码可以看出，调用submitTopology()方法，只是多了一步把SubmitOptions 参数设置为的TopologyInitialStatus.ACTIVE过程，实际上，最终还是调用submitTopologyWithOpts(name, uploadedJarLocation, jsonConf, topology, options)方法。

3.submitTopologyWithOpts()方法的源码如下：

```
1.  @Override
2.  public void submitTopologyWithOpts(String topologyName, String uploadedJarLocation, String jsonConf, StormTopology topology, SubmitOptions options)
3.      throws AlreadyAliveException, InvalidTopologyException, TopologyAssignException, TException {
4.      // STONE_NOTE 再次检查topologyName是否合法
5.      if (!Common.charValidate(topologyName)) {
6.          throw new InvalidTopologyException(topologyName + " is not a valid topology name");
7.      }
8.
9.      try {
10.         // STONE_NOTE 检查topology是否为Active状态
11.         checkTopologyActive(data, topologyName, false);
12.     } catch (AlreadyAliveException e) {}
13. }
```

```

14. String topologyId = null;
15. synchronized (data) {
16.     // avoid to the same topologies were submitted at the same time
17.     // STONE_NOTE 防止在同一时间提交相同的topology，使用同步锁
18.     Set<String> pendingTopologies =
19.         data.getPendingSubmitTopologies().keySet();
20.     for (String cachTopologyId : pendingTopologies) {
21.         if (cachTopologyId.contains(topologyName + "-"))
22.             throw new AlreadyAliveException(
23.                 topologyName + " were submitted");
24.     }
25.     int counter = data.getSubmittedCount().incrementAndGet();
26.     topologyId = Common.topologyNameToId(topologyName, counter);
27.     data.getPendingSubmitTopologies().put(topologyId, null);
28. }
29. try {
30.
31.     // STONE_NOTE 获取topology的序列化配置参数
32.     Map<Object, Object> serializedConf = (Map<Object, Object>) JStormUtils.from
m_json(jsonConf);
33.     if (serializedConf == null) {
34.         LOG.warn("Failed to serialized Configuration");
35.         throw new InvalidTopologyException("Failed to serialize topology confi
guration");
36.     }
37.
38.     serializedConf.put(Config.TOPOLOGY_ID, topologyId);
39.     serializedConf.put(Config.TOPOLOGY_NAME, topologyName);
40.
41.     Map<Object, Object> stormConf;
42.
43.     stormConf = NimbusUtils.normalizeConf(conf, serializedConf, topology);
44.
45.     Map<Object, Object> totalStormConf = new HashMap<Object, Object>(conf);
46.     totalStormConf.putAll(stormConf);
47.
48.     StormTopology normalizedTopology = NimbusUtils.normalizeTopology(stormConf
, topology, true);
49.
50.     // STONE_NOTE 验证topology的结构信息
51.     Common.validate_basic(normalizedTopology, totalStormConf, topologyId);
52.
53.     // STONE_NOTE 获取storm集群状态
54.     StormClusterState stormClusterState = data.getStormClusterState();
55.
56.     double metricsSampleRate = ConfigExtension.getMetricSampleRate(stormConf);
57.     // STONE_NOTE 创建nimbus本地topology任务资源文件的存放目录/local-dir/nimbu
s/topologyId/xxxx files
58.     setupStormCode(conf, topologyId, uploadedJarLocation, stormConf, normalize
dTopology);
59.
60.     // STONE_NOTE 在zookeeper中为每一个bolt或spout生成任务信息/ZK/tasks/topool
ogyId/xxx
61.     setupZkTaskInfo(conf, topologyId, stormClusterState);

```

```

62.
63.         // STONE_NOTE 给topology开始分配任务
64.         makeAssignment(topologyName, topologyId, options.get_initial_status());
65.
66.         // when make assignment for a topology,so remove the topologyid form pendingSubmitTopologys
67.         data.getPendingSubmitTopologys().remove(topologyId);
68.
69.         // push start event after startup
70.         StartTopologyEvent startEvent = new StartTopologyEvent();
71.         startEvent.clusterName = this.data.getClusterName();
72.         startEvent.topologyId = topologyId;
73.         startEvent.timestamp = System.currentTimeMillis();
74.         startEvent.sampleRate = metricsSampleRate;
75.         this.data.getMetricRunnable().pushEvent(startEvent);
76.
77.         notifyTopologyActionListener(topologyName, "submitTopology");
78.
79.     } catch (FailedAssignTopologyException e) {}
80.
81. }

```

4.从submitTopologyWithOpts()方法的源码可以看出，nimbus的任务分配做了以下两步。

(1) 将bolt或spout的任务信息写入到zookeeper中

// STONE\_NOTE 在zookeeper中为每一个bolt或spout生成任务信息/ZK/tasks/topoologyId/xxx  
 setupZkTaskInfo(conf, topologyId, stormClusterState);

(2) 将任务信息放入到队列中，makeAssignment(topologyName, topologyId,  
 options.get\_initial\_status());

```

1. private void makeAssignment(String topologyName, String topologyId, TopologyInitialStatus status) throws FailedAssignTopologyException {
2.     TopologyAssignEvent assignEvent = new TopologyAssignEvent();
3.     assignEvent.setTopologyId(topologyId);
4.     assignEvent.setScratch(false);
5.     assignEvent.setTopologyName(topologyName);
6.     assignEvent.setOldStatus(Thrift.topologyInitialStatusToStormStatus(status));
7.
8.     TopologyAssign.push(assignEvent);
9.
10.    boolean isSuccess = assignEvent.waitFinish();
11.    if (isSuccess == true) {
12.        LOG.info("Finish submit for " + topologyName);
13.    } else {
14.        throw new FailedAssignTopologyException(assignEvent.getErrorMsg());
15.    }
16. }

```

(3) 调用TopologyAssign.push(assignEvent)方法，放入队列中。

```

1. protected static LinkedBlockingQueue<TopologyAssignEvent> queue = new LinkedBlockingQueue<TopologyAssignEvent>();

```

```

2.
3. public static void push(TopologyAssignEvent event) {
4.     queue.offer(event);
5. }

```

(4) TopologyAssign中开启一条线程，在其run()方法中，不停的从消息队列中获取消息

```

1. public void run() {
2.     LOG.info("TopologyAssign thread has been started");
3.     runFlag = true;
4.
5.     while (runFlag) {
6.         TopologyAssignEvent event;
7.         try {
8.             // STONE_NOTE 从消息队列中获取任务信息事件
9.             event = queue.take();
10.        } catch (InterruptedException e1) {
11.            continue;
12.        }
13.        if (event == null) {
14.            continue;
15.        }
16.
17.        // STONE_NOTE 开始任务分配
18.        boolean isSuccess = doTopologyAssignment(event);
19.
20.        if (isSuccess == false) {
21.        } else {
22.            try {
23.                cleanupDisappearedTopology();
24.            } catch (Exception e) {
25.                LOG.error("Failed to do cleanup disappear topology ", e);
26.                continue;
27.            }
28.        }
29.    }
30.
31. }

```

(5) 在run()方法中，调用doTopologyAssignment()方法，将获取到的任务事件消息进行任务分配。

```

1. protected boolean doTopologyAssignment(TopologyAssignEvent event) {
2.     Assignment assignment;
3.     try {
4.         Assignment oldAssignment = null;
5.         boolean isReassign = event.isScratch();
6.         // STONE_NOTE 如果任务需要重新分配
7.         if (isReassign) {
8.             oldAssignment = nimbusData.getStormClusterState().assignment_info(event.getTopologyId(), null);
9.         }
10.    }

```

```

11. // STONE_NOTE 任务分配
12. assignment = mkAssignment(event);
13.
14. // notify jstorm monitor on task assign/reassign/rebalance
15. TaskStartEvent taskEvent = new TaskStartEvent();
16. taskEvent.oldAssignment = oldAssignment;
17. taskEvent.newAssignment = assignment;
18. taskEvent.topologyId = event.getTopologyId();
19. taskEvent.clusterName = nimbusData.getClusterName();
20. taskEvent.timestamp = System.currentTimeMillis();
21.
22. Map<Integer, String> task2Component;
23. // get from nimbus cache first
24. Map<Integer, TaskInfo> taskInfoMap = Cluster.get_all_taskInfo(nimbusData.g
etStormClusterState(), event.getTopologyId());
25. if (taskInfoMap != null) {
26.     task2Component = Common.getTaskToComponent(taskInfoMap);
27. } else {
28.     task2Component = Common.getTaskToComponent(Cluster.get_all_taskInfo(ni
mbusData.getStormClusterState(), event.getTopologyId()));
29. }
30. taskEvent.task2Component = task2Component;
31. nimbusData.getMetricRunnable().pushEvent(taskEvent);
32.
33. if (!isReassign) {
34.     setTopologyStatus(event);
35. }
36. } catch (Throwable e) {
37.     LOG.error("Failed to assign topology " + event.getTopologyId(), e);
38.     event.fail(e.getMessage());
39.     return false;
40. }
41.
42. if (assignment != null)
43.     backupAssignment(assignment, event);
44. event.done();
45. return true;
46. }

```

## (6) 在mkAssignment()方法中，进行分配

```

1. public Assignment mkAssignment(TopologyAssignEvent event) throws Exception {
2.     String topologyId = event.getTopologyId();
3.
4.     LOG.info("Determining assignment for " + topologyId);
5.
6.     // STONE_NOTE 为Topology的任务分配做准备 获取所有空闲的槽 (worker)
7.     TopologyAssignContext context = prepareTopologyAssign(event);
8.
9.     Set<ResourceWorkerSlot> assignments = null;
10.
11.     if (!StormConfig.local_mode(nimbusData.getConf())) {
12.
13.         // STONE_NOTE 集群模式任务分配

```

```

14.         ITopologyScheduler scheduler = schedulers.get(DEFAULT_SCHEDULER_NAME);
15.
16.         assignments = scheduler.assignTasks(context);
17.
18.     } else {
19.         // STONE_NOTE 本地模式任务分配
20.         assignments = mkLocalAssignment(context);
21.     }
22.
23.     Assignment assignment = null;
24.     if (assignments != null && assignments.size() > 0) {
25.         Map<String, String> nodeHost = getTopologyNodeHost(context.getCluster(), c
ontext.getOldAssignment(), assignments);
26.
27.         Map<Integer, Integer> startTimes = getTaskStartTimes(context, nimbusData,
topologyId, context.getOldAssignment(), assignments);
28.
29.         String codeDir = StormConfig.masterStormdistRoot(nimbusData.getConf(), top
ologyId);
30.
31.         assignment = new Assignment(codeDir, assignments, nodeHost, startTimes);
32.
33.         // the topology binary changed.
34.         if (event.isScaleTopology()){
35.             assignment.setAssignmentType(Assignment.AssignmentType.ScaleTopology);
36.         }
37.         StormClusterState stormClusterState = nimbusData.getStormClusterState();
38.
39.         stormClusterState.set_assignment(topologyId, assignment);
40.
41.         // update task heartbeat's start time
42.         NimbusUtils.updateTaskHbStartTime(nimbusData, assignment, topologyId);
43.
44.         NimbusUtils.updateTopologyTaskTimeout(nimbusData, topologyId);
45.
46.         LOG.info("Successfully make assignment for topology id " + topologyId + ":
" + assignment);
47.     }
48.     // STONE_NOTE 最后返回整个Topology的任务配置信息对象Assignment
49.     return assignment;
50. }

```

( 7 ) Scheduler调用scheduler.assignTasks(context)的方法，进行集群模式的任务分配

```

1.     // STONE_NOTE 分配任务
2.     @Override
3.     public Set<ResourceWorkerSlot> assignTasks(TopologyAssignContext context) throws F
ailedAssignTopologyException {
4.
5.         // STONE_NOTE 获取任务类型
6.         int assignType = context.getAssignType();
7.         if (TopologyAssignContext.isAssignTypeValid(assignType) == false) {
8.             throw new FailedAssignTopologyException("Invalid Assign Type " + assignTy
pe);

```

```

9.     }
10.
11.     DefaultTopologyAssignContext defaultContext = new DefaultTopologyAssignContext
(context);
12.     if (assignType == TopologyAssignContext.ASSIGN_TYPE_REBALANCE) {
13.         freeUsed(defaultContext);
14.     }
15.     LOG.info("Dead tasks:" + defaultContext.getDeadTaskIds());
16.     LOG.info("Unstopped tasks:" + defaultContext.getUnstoppedTaskIds());
17.
18.     // STONE_NOTE 任务计算，计算有多少个Task 即有多少个并行度，累计求和
19.     Set<Integer> needAssignTasks = getNeedAssignTasks(defaultContext);
20.
21.     Set<ResourceWorkerSlot> keepAssigns = getKeepAssign(defaultContext, needAssign
Tasks);
22.
23.     // please use tree map to make task sequence
24.     Set<ResourceWorkerSlot> ret = new HashSet<ResourceWorkerSlot>();
25.     ret.addAll(keepAssigns);
26.     ret.addAll(defaultContext.getUnstoppedWorkers());
27.
28.     // STONE_NOTE 分配worker数量，获取可用的worker数量
29.     int allocWorkerNum = defaultContext.getTotalWorkerNum() - defaultContext.getUn
stoppedWorkerNum() - keepAssigns.size();
30.     LOG.info("allocWorkerNum=" + allocWorkerNum + ", totalWorkerNum=" + defaultCon
text.getTotalWorkerNum() + ", keepWorkerNum=" + keepAssigns.size());
31.
32.     if (allocWorkerNum <= 0) {
33.         LOG.warn("Don't need assign workers, all workers are fine " + defaultConte
xt.toDetailString());
34.         throw new FailedAssignTopologyException("Don't need assign worker, all wor
kers are fine ");
35.     }
36.
37.     // STONE_NOTE 计算可用的worker
38.     List<ResourceWorkerSlot> availableWorkers = WorkerScheduler.getInstance().getA
vailableWorkers(defaultContext, needAssignTasks, allocWorkerNum);
39.     // STONE_NOTE 计算每个worker运行多少个task
40.     TaskScheduler taskScheduler = new TaskScheduler(defaultContext, needAssignTask
s, availableWorkers);
41.     // STONE_NOTE TaskScheduler进行每个worker中的Task的分配
42.     Set<ResourceWorkerSlot> assignment = new HashSet<ResourceWorkerSlot>(taskSched
uler.assign());
43.     ret.addAll(assignment);
44.     // STONE_NOTE 返回worker的配置资源信息
45.     return ret;
46. }

```

(8) 在这个taskScheduler.assign()方法中，进行每个worker中运行的Task的数量分配

```

1. public List<ResourceWorkerSlot> assign() {
2.     if (tasks.size() == 0) {
3.         // STONE_NOTE 在getRestAssignedWorkers中实现每个worker中的task平均分配的原
则

```



```

4.         assignments.addAll(getRestAssignedWorkers());
5.         return assignments;
6.     }
7.     Set<Integer> assignedTasks = assignForDifferNodeTask();
8.     tasks.removeAll(assignedTasks);
9.     Map<Integer, String> systemTasks = new HashMap<Integer, String>();
10.    for (Integer task : tasks) {
11.        String name = context.getTaskToComponent().get(task);
12.        if (Common.isSystemComponent(name)) {
13.            systemTasks.put(task, name);
14.            continue;
15.        }
16.        assignForTask(name, task);
17.    }
18.    for (Entry<Integer, String> entry : systemTasks.entrySet()) {
19.        assignForTask(entry.getValue(), entry.getKey());
20.    }
21.    assignments.addAll(getRestAssignedWorkers());
22.    // STONE_NOTE 最后返回所有的任务配置信息
23.    return assignments;
24. }

```

( 9 ) 在getRestAssignedWorkers中实现每个worker中的task平均分配的原则

```

1.    private Set<ResourceWorkerSlot> getRestAssignedWorkers() {
2.        Set<ResourceWorkerSlot> ret = new HashSet<ResourceWorkerSlot>();
3.        // STONE_NOTE getWorkerToTaskNum分配每一个worker中的task数量
4.        for (ResourceWorkerSlot worker : taskContext.getWorkerToTaskNum().keySet()) {
5.            if (worker.getTasks() != null && worker.getTasks().size() > 0) {
6.                ret.add(worker);
7.            }
8.        }
9.        return ret;
10.    }

```

( 10 ) 最终，在TaskAssignContext的构造函数中，实现对Worker的Task数量进行平均分配的原则

```

1.    public TaskAssignContext(Map<String, List<ResourceWorkerSlot>> supervisorToWorker,
2.        Map<String, Set<String>> relationship, Map<Integer, String> taskToComponent) {
3.        this.taskToComponent = taskToComponent;
4.        this.supervisorToWorker = supervisorToWorker;
5.        this.relationship = relationship;
6.
7.        // STONE_NOTE 在这个构造函数中进行worker的Task数量分配，实现一个平均分配的原则
8.        for (Entry<String, List<ResourceWorkerSlot>> entry : supervisorToWorker.entrySet()) {
9.            for (ResourceWorkerSlot worker : entry.getValue()) {
10.                workerToTaskNum.put(worker, (worker.getTasks() != null ? worker.getTasks().size() : 0));
11.                HostPortToWorkerMap.put(worker.getHostPort(), worker);
12.
13.                if (worker.getTasks() != null) {
14.                    Map<String, Integer> componentToNum = new HashMap<String, Integer>

```



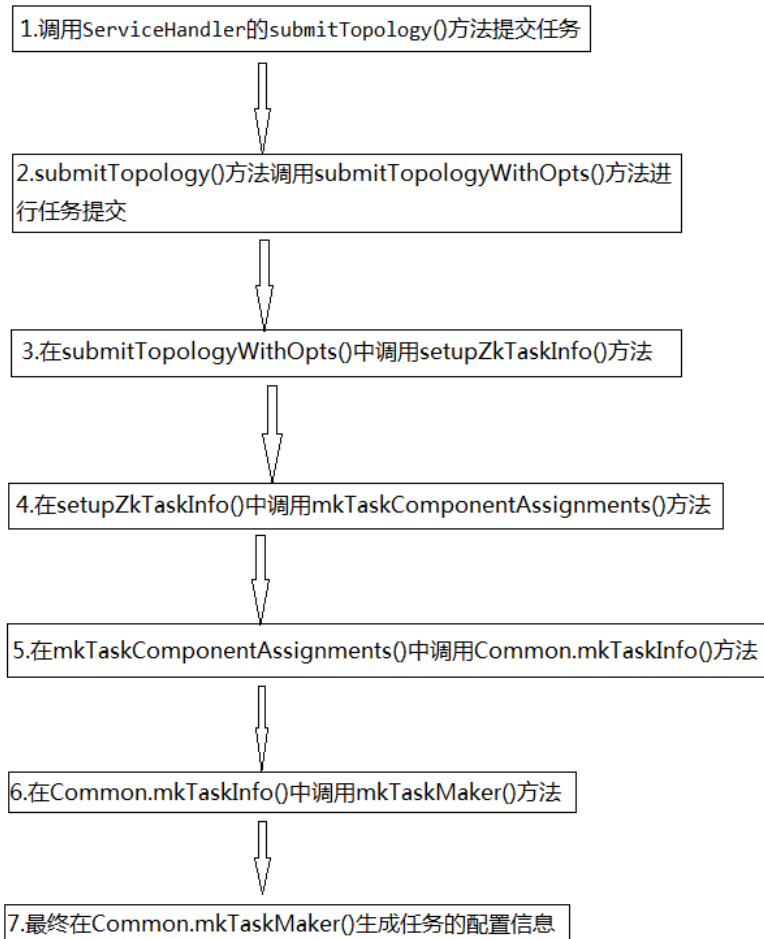
```

14.         ();
15.         for (Integer taskId : worker.getTasks()) {
16.             String componentId = taskToComponent.get(taskId);
17.             int num = componentToNum.get(componentId) == null ? 0 : componentToNum.get(componentId);
18.             componentToNum.put(componentId, ++num);
19.         }
20.         workerToComponentNum.put(worker, componentToNum);
21.     }
22. }
23. }

```

## 5.Topology任务配置信息生成流程：

Topology任务配置信息生成流程：



### ( 1 ) setupZkTaskInfo的源码如下：

```

1. public void setupZkTaskInfo(Map<Object, Object> conf, String topologyId, StormClusterState stormClusterState) throws Exception {
2.     Map<Integer, TaskInfo> taskToTaskInfo = mkTaskComponentAssignments(conf, topologyId);
3.
4.     // mkdir /ZK/taskbeats/topoologyId

```

```

5. // STONE_NOTE 在zookeeper上创建任务信息节点/ZK/taskbeats/topoologyId
6. int masterId = NimbusUtils.getTopologyMasterId(taskToTaskInfo);
7. // STONE_NOTE 使用
8. TopologyTaskHbInfo topoTaskHbinfo = new TopologyTaskHbInfo(topologyId, masterI
d);
9. data.getTasksHeartbeat().put(topologyId, topoTaskHbinfo);
10. stormClusterState.topology_heartbeat(topologyId, topoTaskHbinfo);
11.
12. if (taskToTaskInfo == null || taskToTaskInfo.size() == 0) {
13.     throw new InvalidTopologyException("Failed to generate TaskIDs map");
14. }
15. // key is taskid, value is taskinfo
16. stormClusterState.set_task(topologyId, taskToTaskInfo);
17. }

```

调用mkTaskComponentAssignments方法，获取Task之间任务信息，放入到Map中。

(2) mkTaskComponentAssignments的源码如下：

```

1. public Map<Integer, TaskInfo> mkTaskComponentAssignments(Map<Object, Object> conf,
String topologyid) throws IOException, InvalidTopologyException {
2.
3.     Map<Object, Object> stormConf = StormConfig.read_nimbus_topology_conf(conf, to
pologyid);
4.     StormTopology stopology = StormConfig.read_nimbus_topology_code(conf, topology
id);
5.     StormTopology topology = Common.system_topology(stormConf, stopology);
6.
7.     return Common.mkTaskInfo(stormConf, topology, topologyid);
8. }

```

(3) mkTaskInfo的源码如下：

```

1. public static Map<Integer, TaskInfo> mkTaskInfo(Map<Object, Object> stormConf, Sto
rmTopology sysTopology, String topologyid) {
2.
3.     // use TreeMap to make task as sequence
4.     Map<Integer, TaskInfo> rtn = new TreeMap<Integer, TaskInfo>();
5.
6.     Integer count = 0;
7.     count = mkTaskMaker(stormConf, sysTopology.get_bolts(), rtn, count);
8.     count = mkTaskMaker(stormConf, sysTopology.get_spouts(), rtn, count);
9.     count = mkTaskMaker(stormConf, sysTopology.get_state_spouts(), rtn, count);
10.
11.     return rtn;
12. }

```

(4) mkTaskMaker源码如下：

```

1. public static Integer mkTaskMaker(Map<Object, Object> stormConf, Map<String, ?> ci
dSpec, Map<Integer, TaskInfo> rtn, Integer cnt) {
2.     if (cidSpec == null) {
3.         LOG.warn("Component map is empty");

```

```

4.         return cnt;
5.     }
6.
7.     Set<?> entrySet = cidSpec.entrySet();
8.     for (Iterator<?> it = entrySet.iterator(); it.hasNext();) {
9.         Entry entry = (Entry) it.next();
10.        Object obj = entry.getValue();
11.
12.        ComponentCommon common = null;
13.        String componentType = "bolt";
14.        if (obj instanceof Bolt) {
15.            common = ((Bolt) obj).get_common();
16.            componentType = "bolt";
17.        } else if (obj instanceof SpoutSpec) {
18.            common = ((SpoutSpec) obj).get_common();
19.            componentType = "spout";
20.        } else if (obj instanceof StateSpoutSpec) {
21.            common = ((StateSpoutSpec) obj).get_common();
22.            componentType = "spout";
23.        }
24.
25.        if (common == null) {
26.            throw new RuntimeException("No ComponentCommon of " + entry.getKey());
27.        }
28.
29.        int declared = Thrift.parallelismHint(common);
30.        Integer parallelism = declared;
31.        // Map tmp = (Map) Utils_clj.from_json(common.get_json_conf());
32.
33.        Map newStormConf = new HashMap(stormConf);
34.        // newStormConf.putAll(tmp);
35.        Integer maxParallelism = JStormUtils.parseInt(newStormConf.get(Config.TOPOLOGY_MAX_TASK_PARALLELISM));
36.        if (maxParallelism != null) {
37.            parallelism = Math.min(maxParallelism, declared);
38.        }
39.
40.        for (int i = 0; i < parallelism; i++) {
41.            cnt++;
42.            TaskInfo taskInfo = new TaskInfo((String) entry.getKey(), componentType);
43.            rtn.put(cnt, taskInfo);
44.        }
45.    }
46.    return cnt;
47. }

```

最终将任务配置信息放入stormClusterState.set\_task(topologyId, taskToTaskInfo)中。

至此，nimbus的任务分配过程完成！