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。

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

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：

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

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：

```
@Override
public void submitTopologyWithOpts(String topologyName, String uploadedJarLocation, String jsonConf, StormTopology topology, SubmitOptions options)
        throws AlreadyAliveException, InvalidTopologyException, TopologyAssignException, TException {
    // STONE_NOTE 再次检查topologyName是否合法
    if (!Common.charValidate(topologyName)) {
        throw new InvalidTopologyException(topologyName + " is not a valid topology name");
    }

    try {
        // STONE_NOTE 检查topology是否为Active状态
        checkTopologyActive(data, topologyName, false);
    } catch (AlreadyAliveException e) {}

```

```
14.     String topologyId = null;
15.     synchronized (data) {
16.         // avoid to the same topologys wered submmitted 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> pendingTopologys =
19.                 data.getPendingSubmitTopoloygs().keySet();
20.         for (String cachTopologyId : pendingTopologys) {
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.getPendingSubmitTopoloygs().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.fro
    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 pendi
    ngSubmitTopologys
67.         data.getPendingSubmitTopoloygs().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, TopologyInitia
    lStatus 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 LinkedBlocki
    ngQueue<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(even
    t.getTopologyId(), null);
9.          }
10.
```

```
        // STONE_NOTE 任务分配
        assignment = mkAssignment(event);

        // notify jstorm monitor on task assign/reassign/rebalance
        TaskStartEvent taskEvent = new TaskStartEvent();
        taskEvent.oldAssignment = oldAssignment;
        taskEvent.newAssignment = assignment;
        taskEvent.topologyId = event.getTopologyId();
        taskEvent.clusterName = nimbusData.getClusterName();
        taskEvent.timestamp = System.currentTimeMillis();

        Map<Integer, String> task2Component;
        // get from nimbus cache first
        Map<Integer, TaskInfo> taskInfoMap = Cluster.get_all_taskInfo(nimbusData.getStormClusterState(), event.getTopologyId());
        if (taskInfoMap != null) {
            task2Component = Common.getTaskToComponent(taskInfoMap);
        } else {
            task2Component = Common.getTaskToComponent(Cluster.get_all_taskInfo(nimbusData.getStormClusterState(), event.getTopologyId()));
        }
        taskEvent.task2Component = task2Component;
        nimbusData.getMetricRunnable().pushEvent(taskEvent);

        if (!isReassign) {
            setTopologyStatus(event);
        }
    } catch (Throwable e) {
        LOG.error("Failed to assign topology " + event.getTopologyId(), e);
        event.fail(e.getMessage());
        return false;
    }

    if (assignment != null)
        backupAssignment(assignment, event);
    event.done();
    return true;
}
```

（6）在mkAssignment()方法中，进行分配

```
public Assignment mkAssignment(TopologyAssignEvent event) throws Exception {
    String topologyId = event.getTopologyId();

    LOG.info("Determining assignment for " + topologyId);

    // STONE_NOTE 为Topology的任务分配做准备 获取所有空闲的槽（worker）
    TopologyAssignContext context = prepareTopologyAssign(event);

    Set<ResourceWorkerSlot> assignments = null;

    if (!StormConfig.local_mode(nimbusData.getConf())) {

        // STONE_NOTE 集群模式任务分配
```

```
14.         IToplogyScheduler 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("Invalide Assign Type " + assignTy
    pe);
```

```
    }

    DefaultTopologyAssignContext defaultContext = new DefaultTopologyAssignContext(context);
    if (assignType == TopologyAssignContext.ASSIGN_TYPE_REBALANCE) {
        freeUsed(defaultContext);
    }
    LOG.info("Dead tasks:" + defaultContext.getDeadTaskIds());
    LOG.info("Unstopped tasks:" + defaultContext.getUnstoppedTaskIds());

    // 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
    Set<Integer> needAssignTasks = getNeedAssignTasks(defaultContext);

    Set<ResourceWorkerSlot> keepAssigns = getKeepAssign(defaultContext, needAssignTasks);

    // please use tree map to make task sequence
    Set<ResourceWorkerSlot> ret = new HashSet<ResourceWorkerSlot>();
    ret.addAll(keepAssigns);
    ret.addAll(defaultContext.getUnstoppedWorkers());

    // STONE_NOTE 分配worker数量，获取可用的worker数量
    int allocWorkerNum = defaultContext.getTotalWorkerNum() - defaultContext.getUnstoppedWorkerNum() - keepAssigns.size();
    LOG.info("allocWorkerNum=" + allocWorkerNum + ", totalWorkerNum=" + defaultContext.getTotalWorkerNum() + ", keepWorkerNum=" + keepAssigns.size());

    if (allocWorkerNum <= 0) {
        LOG.warn("Don't need assign workers, all workers are fine " + defaultContext.toDetailString());
        throw new FailedAssignTopologyException("Don't need assign worker, all workers are fine ");
    }

    // STONE_NOTE 计算可用的worker
    List<ResourceWorkerSlot> availableWorkers = WorkerScheduler.getInstance().getAvailableWorkers(defaultContext, needAssignTasks, allocWorkerNum);
    // STONE_NOTE 计算每个worker运行多少个task
    TaskScheduler taskScheduler = new TaskScheduler(defaultContext, needAssignTasks, availableWorkers);
    // STONE_NOTE TaskScheduler进行每个worker中的Task的分配
    Set<ResourceWorkerSlot> assignment = new HashSet<ResourceWorkerSlot>(taskScheduler.assign());
    ret.addAll(assignment);
    // STONE_NOTE 返回worker的配置资源信息
    return ret;
}
```

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

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

```
        assignments.addAll(getRestAssignedWorkers());
        return assignments;
    }
    Set<Integer> assignedTasks = assignForDifferNodeTask();
    tasks.removeAll(assignedTasks);
    Map<Integer, String> systemTasks = new HashMap<Integer, String>();
    for (Integer task : tasks) {
        String name = context.getTaskToComponent().get(task);
        if (Common.isSystemComponent(name)) {
            systemTasks.put(task, name);
            continue;
        }
        assignForTask(name, task);
    }
    for (Entry<Integer, String> entry : systemTasks.entrySet()) {
        assignForTask(entry.getValue(), entry.getKey());
    }
    assignments.addAll(getRestAssignedWorkers());
    // STONE_NOTE 最后返回所有的任务配置信息
    return assignments;
}
```

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

```
private Set<ResourceWorkerSlot> getRestAssignedWorkers() {
    Set<ResourceWorkerSlot> ret = new HashSet<ResourceWorkerSlot>();
    // STONE_NOTE getWorkerToTaskNum分配每一个worker中的task数量
    for (ResourceWorkerSlot worker : taskContext.getWorkerToTaskNum().keySet()) {
        if (worker.getTasks() != null && worker.getTasks().size() > 0) {
            ret.add(worker);
        }
    }
    return ret;
}
```

（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

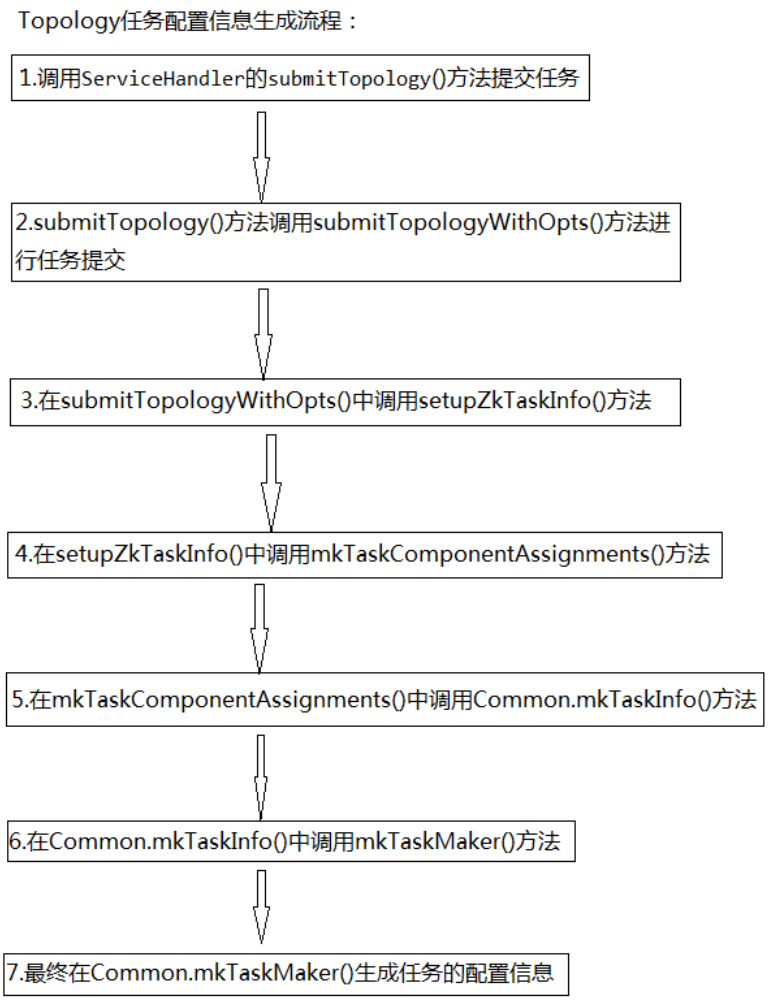
```
public TaskAssignContext(Map<String, List<ResourceWorkerSlot>> supervisorToWorker,
 Map<String, Set<String>> relationship, Map<Integer, String> taskToComponent) {
    this.taskToComponent = taskToComponent;
    this.supervisorToWorker = supervisorToWorker;
    this.relationship = relationship;

    // 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
    for (Entry<String, List<ResourceWorkerSlot>> entry : supervisorToWorker.entryS
et()) {
        for (ResourceWorkerSlot worker : entry.getValue()) {
            workerToTaskNum.put(worker, (worker.getTasks() != null ? worker.getTas
ks().size() : 0));
            HostPortToWorkerMap.put(worker.getHostPort(), worker);

            if (worker.getTasks() != null) {
                Map<String, Integer> componentToNum = new HashMap<String, Integer>
```

```
();
                for (Integer taskId : worker.getTasks()) {
                    String componentId = taskToComponent.get(taskId);
                    int num = componentToNum.get(componentId) == null ? 0 : componentToNum.get(componentId);
                    componentToNum.put(componentId, ++num);
                }
                workerToComponentNum.put(worker, componentToNum);
            }
        }
    }
}
```

5.Topology任务配置信息生成流程：



（1）`setupZkTaskInfo`的源码如下：

```
public void setupZkTaskInfo(Map<Object, Object> conf, String topologyId, StormClusterState stormClusterState) throws Exception {
    Map<Integer, TaskInfo> taskToTaskInfo = mkTaskComponentAssignments(conf, topologyId);

    // mkdir /ZK/taskbeats/topoologyId
```

```
    // STONE_NOTE 在zookeeper上创建任务信息节点/ZK/taskbeats/topoologyId
    int masterId = NimbusUtils.getTopologyMasterId(taskToTaskInfo);
    // STONE_NOTE 使用
    TopologyTaskHbInfo topoTaskHbinfo = new TopologyTaskHbInfo(topologyId, masterId);
    data.getTasksHeartbeat().put(topologyId, topoTaskHbinfo);
    stormClusterState.topology_heartbeat(topologyId, topoTaskHbinfo);

    if (taskToTaskInfo == null || taskToTaskInfo.size() == 0) {
        throw new InvalidTopologyException("Failed to generate TaskIDs map");
    }
    // key is taskid, value is taskinfo
    stormClusterState.set_task(topologyId, taskToTaskInfo);
}
```

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：

```
public Map<Integer, TaskInfo> mkTaskComponentAssignments(Map<Object, Object> conf, String topologyid) throws IOException, InvalidTopologyException {

    Map<Object, Object> stormConf = StormConfig.read_nimbus_topology_conf(conf, topologyid);
    StormTopology stopology = StormConfig.read_nimbus_topology_code(conf, topologyid);
    StormTopology topology = Common.system_topology(stormConf, stopology);

    return Common.mkTaskInfo(stormConf, topology, topologyid);
}
```

（3）mkTaskInfo的源码如下：

```
public static Map<Integer, TaskInfo> mkTaskInfo(Map<Object, Object> stormConf, StormTopology sysTopology, String topologyid) {

    // use TreeMap to make task as sequence
    Map<Integer, TaskInfo> rtn = new TreeMap<Integer, TaskInfo>();

    Integer count = 0;
    count = mkTaskMaker(stormConf, sysTopology.get_bolts(), rtn, count);
    count = mkTaskMaker(stormConf, sysTopology.get_spouts(), rtn, count);
    count = mkTaskMaker(stormConf, sysTopology.get_state_spouts(), rtn, count);

    return rtn;
}
```

（4）mkTaskMaker源码如下：

```
public static Integer mkTaskMaker(Map<Object, Object> stormConf, Map<String, ?> cidSpec, Map<Integer, TaskInfo> rtn, Integer cnt) {
    if (cidSpec == null) {
        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.TOPO
    LOGY_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(), componentTyp
    e);
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！