Executor的创建与启动是在Worker的execute()方法中完成的。
1.在我们启动Worker时，调用Worker的mk_worker()方法，mk_worker方法创建Worker实例，并调用worker的execute()方法。

```java
public WorkerShutdown execute() throws Exception {
    List<AsyncLoopThread> threads = new ArrayList<AsyncLoopThread>();

    // STONE_NOTE 创建接收数据的线程连接
    AsyncLoopThread controlRvthread = startDispatchThread();
    threads.add(controlRvthread);
    // STONE_NOTE 创建用于更新的连接
    RefreshConnections refreshConn = makeRefreshConnections();
    AsyncLoopThread refreshconn = new AsyncLoopThread(refreshConn, false, Thread.MIN_PRIORITY, true);
    threads.add(refreshconn);
    // STONE_NOTE 更新Zookeeper中的活跃状态
    RefreshActive refreshZkActive = new RefreshActive(workerData);
    AsyncLoopThread refreshzk = new AsyncLoopThread(refreshZkActive, false, Thread.MIN_PRIORITY, true);
    threads.add(refreshzk);
    DrainerCtrlRunable drainerCtrlRunable;
    boolean isTaskBatchTuple = ConfigExtension.isTaskBatchTuple(workerData.getStormConf());
    if (isTaskBatchTuple) {
        drainerCtrlRunable = new DrainerBatchCtrlRunable(workerData, MetricDef.BATCH_SEND_THREAD);
    } else {
        drainerCtrlRunable = new DrainerCtrlRunable(workerData, MetricDef.SEND_THREAD);
    }
    AsyncLoopThread controlSendThread = new AsyncLoopThread(drainerCtrlRunable, false, Thread.MAX_PRIORITY, true);
    threads.add(controlSendThread);
    AsyncLoopThread syncContainerHbThread = SyncContainerHb.mkWorkerInstance(workerData.getStormConf());
    if (syncContainerHbThread != null) {
        threads.add(syncContainerHbThread);
    }

    JStormMetricsReporter metricReporter = new JStormMetricsReporter(workerData);
    metricReporter.init();
    workerData.setMetricsReporter(metricReporter);
    // STONE_NOTE 更新心跳信息到本地目录
    RunnableCallback heartbeat_fn = new WorkerHeartbeatRunable(workerData);
    AsyncLoopThread hb = new AsyncLoopThread(heartbeat_fn, false, null, Thread.NORM_PRIORITY, true);
    threads.add(hb);
    // STONE_NOTE 创建task，并注册Task停止的回调监听
    List<TaskShutdownDameon> shutdowntasks = createTasks();
    workerData.setShutdownTasks(shutdowntasks);
    return new WorkerShutdown(workerData, threads);

}
```

2.调用createTasks()方法，用来创建Task

```
// STONE_NOTE 创建Task，无非就是开启线程
private List<TaskShutdownDameon> createTasks() throws Exception {
    List<TaskShutdownDameon> shutdowntasks =
            new ArrayList<TaskShutdownDameon>();

    // STONE_NOTE 获取所有的线程id
    Set<Integer> taskids = workerData.getTaskids();

    Set<Thread> threads = new HashSet<Thread>();
    List<Task> taskArrayList = new ArrayList<Task>();
    for (int taskid : taskids) {
        // STONE_NOTE 创建Task，即new Task的线程
        Task task = new Task(workerData, taskid);
        Thread thread = new Thread(task);
        threads.add(thread);
        taskArrayList.add(task);
        // STONE_NOTE 开启Task线程，即启动Task任务
        thread.start();
    }
    for (Thread thread : threads) {
        thread.join();
    }
    for (Task t : taskArrayList) {
        shutdowntasks.add(t.getTaskShutdownDameon());
    }
    return shutdowntasks;
}
```

3.创建并启动Task线程，Task执行的任务在其run()方法中执行。

```
// STONE_NOTE Task任务的执行，在其run()方法中执行的
public void run(){
    try {
        // STONE_NOTE 在Task的run()方法中，调用Task的execute()方法，执行任务
        taskShutdownDameon=this.execute();
    }catch (Throwable e){
        LOG.error("init task take error", e);
        if (reportErrorDie != null){
            reportErrorDie.report(e);
        }else {
            throw new RuntimeException(e);
        }

    }
}
```

4.在Task的run()方法中调用自己的execute()方法，执行Task任务。

```java
public TaskShutdownDameon execute() throws Exception {

    taskSendTargets = echoToSystemBolt();

    // create thread to get tuple from zeroMQ,
    // and pass the tuple to bolt/spout
    // STONE_NOTE 开启线程获取数据（tuple），并转换成spout或者bolt
    taskTransfer = mkTaskSending(workerData);
    // STONE_NOTE 准备Executor，创建并获得对应的Executor
    RunnableCallback baseExecutor = prepareExecutor();
    AsyncLoopThread executor_threads = new AsyncLoopThread(baseExecutor, false, Thread.MAX_PRIORITY, true);
    // STONE_NOTE 创建一个Task的接收器
    taskReceiver = mkTaskReceiver();

    List<AsyncLoopThread> allThreads = new ArrayList<AsyncLoopThread>();
    allThreads.add(executor_threads);

    LOG.info("Finished loading task " + componentId + ":" + taskId);

    taskShutdownDameon =  getShutdown(allThreads, taskReceiver.getDeserializeQueue(),
            baseExecutor);
    // STONE_NOTE 最后返回一个Task守护进程的停止对象实例
    return taskShutdownDameon;
}
```

5.在Task的execute()方法中，首先调用mkTaskSending()方法，获得一个Tuple的发送对象。

```java
private TaskTransfer mkTaskSending(WorkerData workerData) {
    // sending tuple's serializer
    // STONE_NOTE 创建一个用于发送Tuple的序列化器
    KryoTupleSerializer serializer = new KryoTupleSerializer(workerData.getStormConf(), topologyContext);

    // STONE_NOTE 通过组件的id和taskid获取task的名称 【componentId + ":" + taskId】
    String taskName = JStormServerUtils.getName(componentId, taskId);
    // Task sending all tuples through this Object
    // STONE_NOTE 获得TaskTransfer的对象，通过TaskTransfer对象发送所有的Tuples
    TaskTransfer taskTransfer;
    if (isTaskBatchTuple)
        taskTransfer = new TaskBatchTransfer(this, taskName, serializer, taskStatus, workerData);
    else
        taskTransfer = new TaskTransfer(this, taskName, serializer, taskStatus, workerData);
    return taskTransfer;
}
```

6.创建对应类型的Executor，并在Executor中实现数据的发送与业务逻辑的处理。

```
// STONE_NOTE 创建Executor接收Tuples 并运行spout或bolt的execute方法
private RunnableCallback prepareExecutor() {
    // create report error callback,
    // in fact it is storm_cluster.report-task-error
    ITaskReportErr reportError = new TaskReportError(zkCluster, topologyId, taskId);

    // report error and halt worker
    reportErrorDie = new TaskReportErrorAndDie(reportError, workHalt);

    // STONE_NOTE 创建Executor
    final BaseExecutors baseExecutor = mkExecutor();

    return baseExecutor;
}
```

调用mkExecutor()方法，创建Executor。

```
public BaseExecutors mkExecutor() {
    BaseExecutors baseExecutor = null;

    // STONE_NOTE this.taskObj = Common.get_task_object(topologyContext.getRawTopology(), componentId, WorkerClassLoader.getInstance());
    // STONE_NOTE 根据taskObj的类型，创建对应类型的Executor
    if (taskObj instanceof IBolt) {
        baseExecutor = new BoltExecutors(this);
    } else if (taskObj instanceof ISpout) {
        if (isSingleThread(stormConf) == true) {
            baseExecutor = new SingleThreadSpoutExecutors(this);
        } else {
            baseExecutor = new MultipleThreadSpoutExecutors(this);
        }
    }

    return baseExecutor;
}
```

至此，Executor已经创建并启动。