Linkis Entrance 高可用实现梳理

在消费队列的任务处于等待状态，任务还未真正执行，所以任务很容易换个节点执行。

在执行队列的任务，是任务已经开始执行，任务的一些中间态都保存在内存里，无法被其他节点接管执行。

服务安全下线

触发方式：Entrance 调用/api/rest_j/v1/entrance/operation/label/markoffline接口。

- 清空ConsumeQueue队列中所有任务(Inited)，并设置任务instance为空，等待failover
- 等待runningJobs队列任务执行完成
  - 如果开启*ENTRANCE_FAILOVER_RETRY_JOB_ENABLED*，则会将runningjobs中retry任务移除，并设置任务instance为空，等待failover

服务正常下线

触发方式：执行 sbin/linkis-daemon.sh stop cg-entrance

- ConsumeQueue队列和runningJobs队列任务全部kill，置为取消状态(Cancelled)，并在日志最后提示信息:

  "Entrance exits the automatic cleanup task and can be rerun(服务退出自动清理任务，可以重跑)"

  - *如果开启ENTRANCE_SHUTDOWN_FAILOVER_CONSUME_QUEUE_ENABLED*，则将ConsumeQueue队列任务设置instance为空，等待failover，默认开启

服务异常下线

触发方式：Kill -9 pid

- 所有任务等待failover

FailoverJobServer

- 标记offline的服务不进行failover处理
- 多个服务争夺锁*ENTRANCE_FAILOVER_LOCK*，同一时间只有一个服务能够处理(单点)
- 通过sql获取需要failover的任务，满足以下条件：
  - 任务状态为未完成（Inited,Scheduled,Running,WaitForRetry）
  - 任务创建时间在有效failover范围内，通过设置*ENTRANCE_FAILOVER_DATA_INTERVAL_TIME*来限制有效时间范围，默认7天，设置为0表示不限制时间范围
  - 任务数据限制，通过*ENTRANCE_FAILOVER_DATA_NUM_LIMIT*来限制每次failover任务数量，避免任务failover到某一台，导致任务分配不均匀，默认10条
  - 以及满足以下任一条件
    - 任务instance为空
    - 任务所属instance不在eureka注册列表中
    - 任务所属instance在eureka注册列表中，但是任务创建时间小于服务注册时间的

```sql
SELECT
    a.*
FROM
    linkis_ps_job_history_group_history a
WHERE
    -- 状态未完成
    STATUS IN ( 'Inited', 'Running', 'Scheduled', 'WaitForRetry' )
    -- 有效failover范围
    AND UNIX_TIMESTAMP( a.created_time ) * 1000 >= 1666239054098
    AND (
        -- 任务instance为空
        a.instances = '' OR a.instances IS NULL
        -- 任务所属instance不在eureka注册列表中
        OR a.instances NOT IN ( '192.168.1.123:9104', '192.168.1.124:9104' )
        -- 任务所属instance在eureka注册列表中，但是任务创建时间小于服务注册时间的
        OR EXISTS (
```

```
            SELECT
                1
            FROM
                (
                SELECT '192.168.1.123:9104' AS instances, 1697775054098 AS regis
                UNION ALL
                SELECT '192.168.1.124:9104' AS instances, 1666239054098 AS regis
                ) b
            WHERE
                a.instances = b.instances
                AND UNIX_TIMESTAMP( a.created_time ) * 1000 < b.registryTime
        )
    )
    -- 任务数据限制
    LIMIT 10
```

- 当前Entrance服务接管查询到的所有任务，进行处理
  - 通过metric值获取任务最新引擎信息，主动发起强杀引擎
    - 如果是executeOnce，则直接通过linkismanager强杀引擎
    - 如果不是executeOnce但engineConnTaskId不为空，则通过ecInstance强杀任务
  - 任务正常接管，再次提交，生成execId等信息，并在日志追加提示信息，表明发生failover

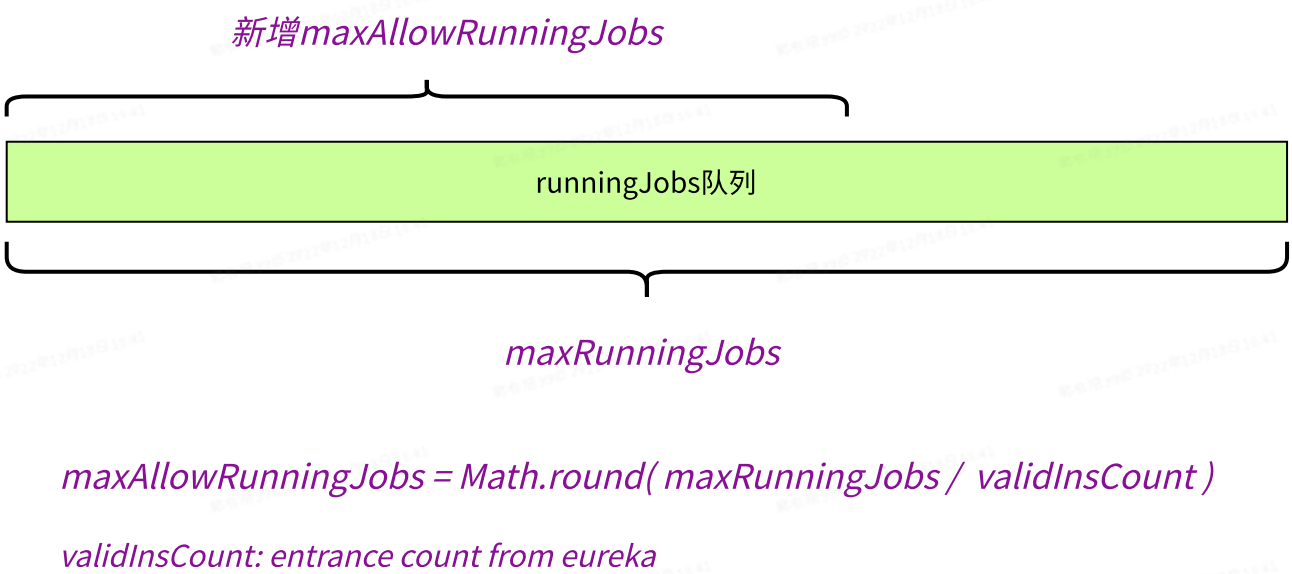
    "**********FAILOVER**********"
    - 如果开启*ENTRANCE_FAILOVER_RUNNING_KILL_ENABLED*，则状态为Running的任务将会被kill处理，而不会提交执行，状态最终变更为Cancelled，并在日志最后提示信息：

    "Job xx failover, status changed from Running to Cancelled (任务故障转移，状态从Running变更为Cancelled)"

Metric字段内容修改

- *ENTRANCEJOB_ENGINECONN_MAP:*
  - key：从*ENGINE_INSTANCE*修改为*TICKET_ID*
  - value：新增*ENGINE_CONN_TASK_ID*，*ENGINE_CONN_SUBMIT_TIME*，*FAILOVER_FLAG*三个值
    - *ENGINE_CONN_TASK_ID:* 提交到ec，ec生成的任务ID
    - *ENGINE_CONN_SUBMIT_TIME:* ec中任务提交的时间
    - *FAILOVER_FLAG:* 表示是否已经发生failover

任务数动态控制



maxAllowRunningJobs = Math.round(maxRunningJobs / validInsCount)

validInsCount: entrance count from eureka

EntranceFIFOUserConsumer

- 新增failover retry任务逻辑，当任务处于waitForRetry时，完全可以进行failover，无需继续提交，这样可以使entrance更早结束

当entrance为offline并且开启*ENTRANCE_FAILOVER_RETRY_JOB_ENABLED*，将retry任务的instances置为空，等待failover

- 新增定时任务，默认60s，刷新所有consume中group的maxAllowRunningJobs值，计算公式

```
maxAllowRunningJobs = Math.round(group.getMaxRunningJobs / validInsCount)
```

FIFOUserConsumer

- 提交任务通用判断逻辑修改maxAllowRunningJobs <= currentRunningJobs

```
if (event.isEmpty) {
  val maxAllowRunningJobs = fifoGroup.getMaxAllowRunningJobs
  val currentRunningJobs = runningJobs.count(e => e != null && !e.isCompleted)
  if (maxAllowRunningJobs <= currentRunningJobs) {
    Utils.tryQuietly(Thread.sleep(1000)) // TODO 还可以优化，通过实现JobListener进
    return
  }
  while (event.isEmpty) {
    val takeEvent = if (getRunningEvents.isEmpty) Option(queue.take()) else queu
    event =
```

```
11        if (
12            takeEvent.exists(e =>
13              Utils.tryCatch(e.turnToScheduled()) { t =>
14                takeEvent.get.asInstanceOf[Job].onFailure("Job状态翻转为Scheduled失
15                false
16              }
17            )
18        ) {
19          takeEvent
20        } else getWaitForRetryEvent
21    }
22  }
```

Rest API 修改

支持taskID查询任务状态，进度，日志等

Gateway

在gateway前置处理请求时，要根据请求参数的taskID查询数据库，获取一些信息:

```
1 status: String // 状态
2 instances: String // 处理的entrance地址，如果在eureka上不存在，则为null
3 jobReqId: String // entranceshortExecID
4 createTimestamp: Long // 任务创建时间
5 instanceRegistryTimestamp: Long // 对应处理的entrance在eureka上的注册时间，如果在eu
  reka上不存在，则为Long.MaxValue
```

获取到以上信息，转为json串，添加到请求Header中

Entrance

判断是taskID的请求，首先解析json串获取信息。

status：`/api/rest_j/v1/entrance/${taskID}/status`

- 如果status是完成状态，则直接返回status
- 如果instanceRegistryTimestamp > createTimestamp，则返回Inited
- 否则jobReqId生成execID，走原有execID逻辑

progress：`/api/rest_j/v1/entrance/${taskID}/progress`

- 如果status是完成状态，则直接返回1.0
- 如果instanceRegistryTimestamp > createTimestamp，则返回0
- 否则jobReqId生成execID，走原有execID逻辑

log：`/api/rest_j/v1/entrance/${taskID}/log?fromLine=${from}&size=${size}`

- 如果status是完成状态，则直接返回

  "The job you just executed has ended. This interface no longer provides a query. It is recommended that you download the log file for viewing.(您刚刚执行的job已经结束，本接口不再提供查询，建议您下载日志文件进行查看)"

- 如果instanceRegistryTimestamp > createTimestamp，则返回

  "The job will failover soon, please try again later.(job很快就会failover，请稍后再试)"

- 否则jobReqId生成execID，走原有execID逻辑

kill：`/api/rest_j/v1/entrance/${taskID}/kill`

- 如果status是完成状态，则直接返回错误信息

  "The job already completed. Do not support kill.(任务已经结束，不支持kill)"

- 如果instanceRegistryTimestamp > createTimestamp，则强制杀死，更新数据库状态
- 否则jobReqId生成execID，走原有execID逻辑