MJ：JM

合规性能优化方案详细设计–多线程池方案与阈值逃生方案

仅供内部使用

目录

[目录 2](#_Toc35617499)

[1. 需求 3](#_Toc35617500)

[2. SRS 3](#_Toc35617501)

[3. 功能设计 3](#_Toc35617502)

[3.1. 线程池方案 3](#_Toc35617503)

[3.1.1. 前置条件分析（以下条件在方案分析中视为不变） 3](#_Toc35617504)

[3.1.2. 规模估算 3](#_Toc35617505)

[3.1.3. 合规脚本模型 3](#_Toc35617506)

[3.1.4. 线程池方案设计 3](#_Toc35617507)

[3.2. 心跳逃生方案 5](#_Toc35617508)

[3.2.1. 前置条件分析 5](#_Toc35617509)

[3.2.2. 阈值逃生方案 5](#_Toc35617510)

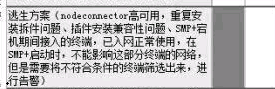
[3.2.3. 心跳逃生方案 6](#_Toc35617511)

[3.2.4. 方案比较 7](#_Toc35617512)

[4. 限制与约束 7](#_Toc35617513)

[5. 附件及其他 7](#_Toc35617514)

# 需求



# SRS

提高合规执行性能的吞吐量、增强合规执行的稳定性。

# 功能设计

## 相关参数计算

1. 变量定义
   1. SMemory：connector服务器具有总内存；
   2. CPercent：合规所占服务器内存百分比；
   3. PCount：单终端平均具有的合规策略数目；
   4. ACount：单条合规策略平均具有的合规动作数目；
   5. ATime：单条合规动作平均执行时间；
   6. SMemory：单个powershell session平均所占用的内存；
   7. TCount：执行线程池线程数；
   8. RTime：终端合规策略响应时间；
   9. Throughput：终端吞吐量；
2. 计算公式：

## 多线程池方案

### 前置条件分析（以下条件在方案分析中视为不变）

* **Connector合规执行模型**：

当前Connector模型中，合规执行以“策略”为维度，“策略”内多个“合规动作”串行执行。因为connector使用线程池并发模式，可认为不同“策略”并行执行。

* **其他组件执行时间：0s**

方案中主要考虑connector的执行效率，其他组件的执行时间可忽略不计（理论分析，其他组件可认为不是性能瓶颈）。这些组件包括前端业务执行、SMP服务器业务执行（如规则引擎、框架代码、数据库操作等）、网络性能（前端-SMP、SMP-connector）、SMP服务器RocketMq性能等。

* **脚本执行时间：4s**

经过测试（单个防火墙开启时间connector端执行时间约为2.5s）并考虑其他因素（如不同脚本的执行时间不同、忽略的其他组件执行时间等），可大致估算平均一个脚本的执行时间为4s。

* **终端合规策略估算：2条策略，每条策略2个动作**
* **合规session内存占用：40M**

经系统观测，一个合规session的内存占用可大致估算为40M。

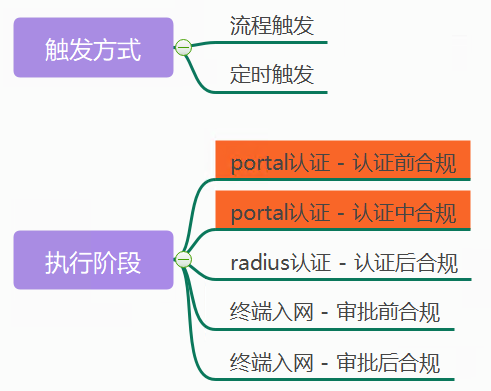
* **Connector服务器分析：满负载合规**

认为connector服务器资源均用作合规处理

### 规模估算

* **企业终端数**：**10000**台（包括服务器、工作机、生产终端、个人携带电脑、哑终端等，不包括手机、平板等接入移动终端，也不包括交换机、路由器等设备）；
  + 可合规终端数：10000 \* **70%** = 7000（去除的部分包括window xp、mac、linux服务器/终端、免合规终端、生产终端、哑终端等）；
    - 进行合规终端数：7000 \* **70%** ≈ 5000（去除部分包括不进行合规控制的终端、区域）；
      * 入网合规终端数：5000 \* **90%** = 4500（去除部分包括类似Windows服务器的终端，一般不会关机离网以及信息更改，因此不会有频繁的入网动作）；
      * Radius认证合规终端数：5000 \* **80%** = 4000（去除部分包括不由认证控制入网的终端，如一些生产终端、公共资源终端等）；
        + Portal认证合规终端数：4000 \* **10%** = 400（考虑通常使用场景下，去除部分包括已注册无感的终端）；
* **Connector服务器性能**：16G \* 75% = **12G**（标准场景）；
* **同时间触发比率（预估最大值）**：**5%**

### 合规脚本模型



其中，我们主要考虑的合规响应性能为“portal认证-认证前合规”、“portal认证-认证后合规”并且触发方式为“流程触发”（只有这种场景下才与用户有直接交互），我们将其定为为指标RT。

### 线程池方案设计

* **多线程池方案：**
  + **根据触发方式划分：**
    - 划分为“流程”与“定时”两个线程池，关注点为“流程”线程池；
    - “流程线程池”中任务脚本包括“流程-portal认证-认证前合规”、“流程-portal认证-认证中合规”、“流程-radius认证-认证后合规”、“流程-终端入网-审批前合规”、“流程-终端入网-审批后合规”；
    - “定时线程池”中任务脚本包括“定时-portal认证-认证前合规”、“定时-portal认证-认证中合规”、“定时-portal认证-认证后合规”、“定时-终端入网-审批前合规”、“定时-终端入网-审批后合规”、“定时-终端信息检测”、“定时-终端软件信息检测”、“定时-合规连通性检测”、“定时-合规失败告警信息”；
    - “流程线程池”理论性能：
      * 规模：**（5000 + 4500）\* 5% \* 2 = 950** （同一流程同一时刻只能触发一个阶段合规，不同流程可同时触发。其中5000为终端入网数，4500为radius认证数，5%为同时间触发比率峰值，2为单终端的合规策略条数）；
      * 性能估算，以线程池具有100个执行线程估算：**950/100 \* 4 \* 2 = 76 s**（其中4为单条合规动作平均执行时间，2为单条合规策略具有的平均合规动作数）
      * RT的理论最坏执行响应为**76s**，RT的理论平均执行响应为**38s**（RT平均落于队列中部）
      * 内存占用：**100 \* 40M = 4G**；
    - “定时线程池”理论性能：
      * 内存占用：**12 – 4 = 8G**
      * 线程数：**8G/40M = 200**
  + **根据关注目标划分：**
    - 划分为“交互”与“后台”两个线程池，关注点为“交互”线程池；
    - “交互线程池”中任务脚本包括“流程-portal认证-认证前合规”、“流程-portal认证-认证中合规”；
    - “后台线程池”中任务脚本包括其余剩余脚本；
    - “交互线程池”理论性能：
      * 规模：**400 \* 5% \* 2 = 40** （其中400为portal认证终端数，5%为同时间触发比率峰值，2为单终端的合规策略条数）；
      * 性能估算，因为规模只为40，当给与线程数为40时，可认为脚本执行完全并行，RT最坏执行响应为4\*2=**8s**，RT平均/单条理论响应时间为4\*2=**8s**；
      * 内存占用：**40 \* 40M = 1.6 G**；
    - “后台线程池”理论性能：
      * 内存占用：**12 – 1.6 = 10.4G**
      * 线程数：**10.4G/40M = 260**
* **线程池任务优先级方案**
  + 原理：不区分线程池，不同执行脚本区分优先级（仍可以根据“触发方式”或者“关注目标”来划分）。
  + 实现：自定义PriorityRunnable类并继承于Runnable、Comparator<Runnable>，将之传递于线程池中；同时，线程池使用PriorityBlockingQueue代替ArrayBlockingQueue。
* **方案主要优势比较**

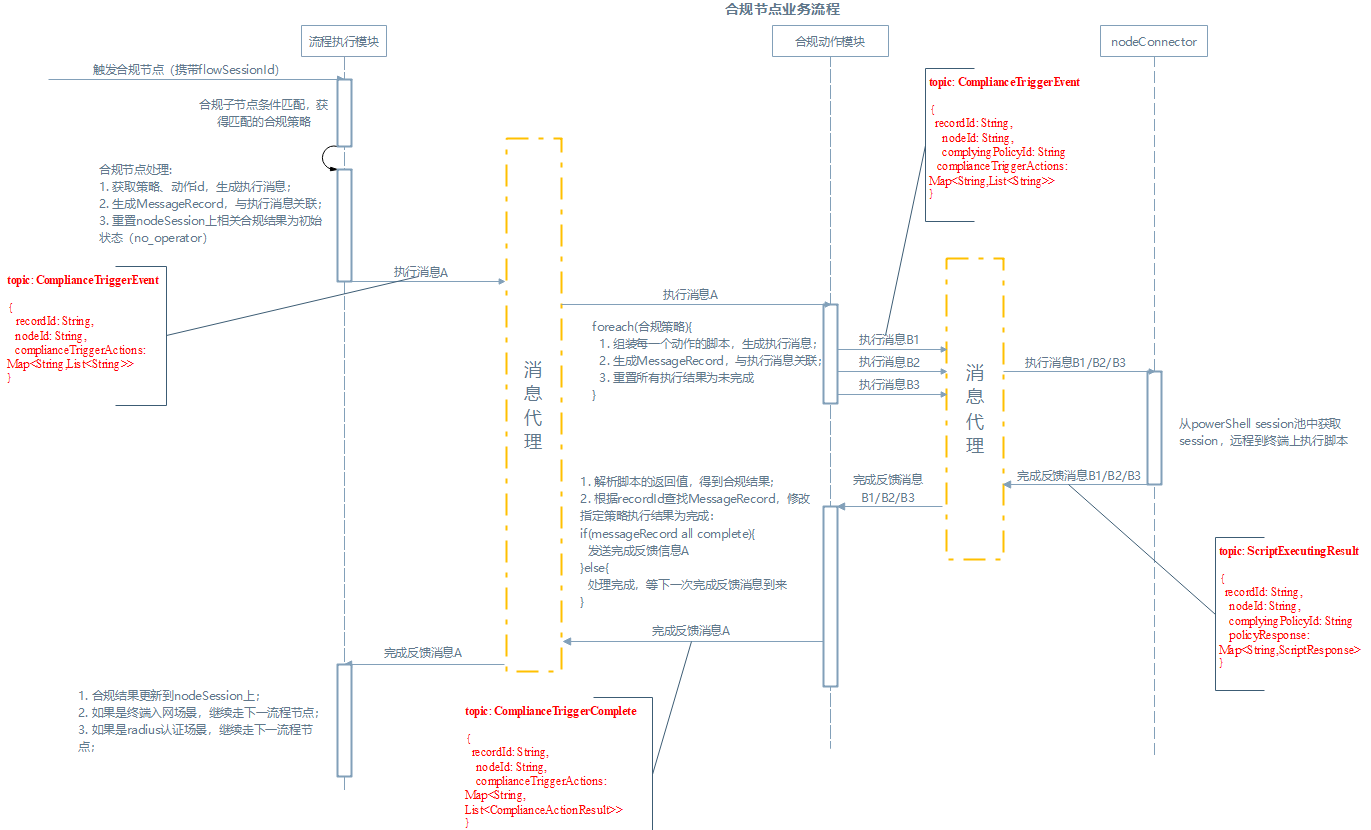
|  |  |  |  |
| --- | --- | --- | --- |
|  | 主要优势 | 主要劣势 | 解决方式 |
| 多线程池 | 实现简单，能做到“不同关注对象”在抽象层面上完全隔离 | 一部分资源会闲置，如“交互”线程池的默认线程 | 设置“交互”线程池的固定线程为较小值，并且keepAlive时间较短 |
| 线程池任务优先级 | 线程池资源可以复用 | 实现稍微复杂些，且不同任务仍然会相互影响，如队列满了后的处理场景；同时，PriorityBlockingQueue插入性能略低于ArrayBlockingQueue | 可指定任务队列满了后的处理方式，如删除队列尾的任务 |

结论：在资源足够的前提下，建议使用“多线程池”方案。

## 心跳逃生方案

### 前置条件分析

* 已经运用了线程池方案；
* 由以上分析可得，线程池方案运用后，若使用得当（无论是单独使用线程池，还是使用优先级），对于界面发起的“交互”合规脚本，不会溢出队列，也不会响应超时，因此该方案下主要针对的是“后台”合规脚本；
* “后台”合规脚本中，不考虑“定时-终端信息检测”、“定时-终端软件信息检测”、“定时-合规连通性检测”、“定时-合规失败告警信息”等任务；
* 方案主要是针对SMP服务器侧的设计；
* 合规执行业务流程：



### 阈值逃生方案

* **原理**：定时检测当前正在执行的定时合规消息的数目N，若N超过一定阈值比例，则进入逃生；若已为逃生模式，且N低于一定阈值比例，则从逃生中恢复；
* **实现**：
  + **阈值比例**：如，进入逃生 - 150%，逃生恢复 – 50%；
  + **合规执行常数**： CT =（5000 \* 2 + 4500 + 400 \* 2）\* 2 ≈ 15000（30000）（根据3.1.2的规模估算，5000为入网合规终端数，4500为radius认证合规终端数，400为portal认证合规终端数，（同一终端的radius、portal、入网会同时出现，且每个节点合规都可能同时执行）；2为单终端具有的合规策略条数，见3.2.1，若逃生判断在“流程执行模块”进行，则不需要该参数；若逃生判断在“合规动作模块”进行，需要该常数参数）；
  + **逃生判断（定时器，如3分钟执行一次）**：
    - If（N > CT \* 150%）{ escape = true }
    - If（N < CT \* 50%）{ escape = false }
  + **进入逃生与恢复逃生**：
    - 若在“流程执行模块”进行，发送“执行消息A”（见3.2.1图）前：

If（escape）{

设置nodeSession指定合规策略为合规，消息提示为逃生合规；

} else {

设置nodeSession指定合规策略为“no\_operate”；

发送执行消息A；

}

* + - 若在“合规动作模块”进行，发送“执行消息B1（B2、B3）”（见3.2.1图）前：

If（escape）{

设定指定策略执行结果为“complete”；

If（messageRecord all complete）{

发送完成反馈消息A；

}

} else {

发送执行消息B1（B2、B3）；

}

* + **逃生效果：逃生时均作为合规处理**。

### 心跳逃生方案

* **原理**：SMP服务器端定时发送“心跳合规”消息，记录发送消息的发送时间Ts，消息完成时，记录消息的结束时间TE，若TE – TS > Ttu（上限阈值），进入逃生；若TE – TS < Ttd（下限阈值），恢复逃生。同时，由于消息可能堆积，可能得不到TE，因此还需要引入定时器，并在定时器中判断当前时间TC – TS是否大于Ttu；
* **性能估算**：（（5000 \* 2 + 4500 + 400 \* 2）\* 2）/260 \* 4 \* 2 ≈ 900s，（5000、4500、400分别为入网合规终端数，radius认证合规终端数，portal认证合规终端数；260为执行线程数，4为单合规动作的平均执行时间，2为单策略具有的平均合规动作数） ；最坏合规响应时间为900s，平均合规响应时间为450s；
* **根据以上计算**：Ttu = 900s \* 150 % = 1250s；Ttd = 900s \* 50% = 450s；
* **实现**：
  + 系统中正在处理的“心跳检测”消息只有一条，发送时标记为“executing”，消息完成时标记为“complete”，或者超时时标记为“complete”；
  + “心跳检测消息”发送与逃生判断，使用定时器（如5分钟）：

If(系统中不存在“executing”的消息){

发送“心跳信息”；

}else{

If(TC – executing消息开始时间TS > Ttu){

escape = true；

置消息为“complete”；

}else {

do nothing；

}

}

* + “心跳检测消息”接收与逃生判断：

message = findById(mqMessage.messageRecordId);

If(message.状态 == “executing”){

If(TC – TS > Ttu){

Escape = true;

}else if(TC – TS < Ttd){

Escape = false;

}

置message为“complete”；

}

* + **进入逃生与恢复逃生**：与阈值逃生方案一致；
  + **逃生效果**：与阈值逃生方案一致；

### 方案比较

|  |  |  |
| --- | --- | --- |
| 方案 | 优点 | 缺点 |
| 阈值逃生方案 | 1. 实现较简单； 2. 响应及时（每次定时器周期都能反映当前执行消息的数目，并判定是否进行逃生或恢复）； 3. 不会引入额外消息（心跳）； | 1. 不能直观反映消息执行的“时间指标”； 2. 从发起方判断，只能反映特定消息的执行效果，如对于“定时软件收集”消息，不会参与到逃生的判断中。 |
| 心跳逃生方案 | 1. 能直接反映消息的“执行时间”； 2. 从结果处判断，可包含同一线程池中的其他消息，如“定时软件收集”； | 1. 实现相对较复杂； 2. 响应有延后（由分析可知，最长检测到的时间可能为Ttu + 定时周期）； 3. 会引入额外消息（心跳）； 4. 若使用该方案，尽量不要使用“优先级”线程池，否则可能因为优先级原因，返回的执行时间无法反映真实场景； |

结论：建议在要求精度不高的场景下，使用“阈值逃生方案”；或者结合使用两种方案，其中“阈值逃生方案”作为主方案，“心跳逃生方案”作为辅助判断。

# 限制与约束

提示：场景、功能、实现 限制

# 附件及其他