

## EXata 扩展（九）：为应用层协议添加统计量

EXata User guide（包括5.1和7.0）上介绍的统计方法跟代码中的实现方式不同，文档中是在数据结构中直接用变量进行统计，而代码中则是用一个统计类（STAT\_AppStatistics）类型的成员变量指针 \*stats，专门负责进行统计。本章结合新增协议PRODUCER和CONSUMER实现应用层统计量的收集和打印，应能在GUI观察到正确的统计量，并进行对比分析。

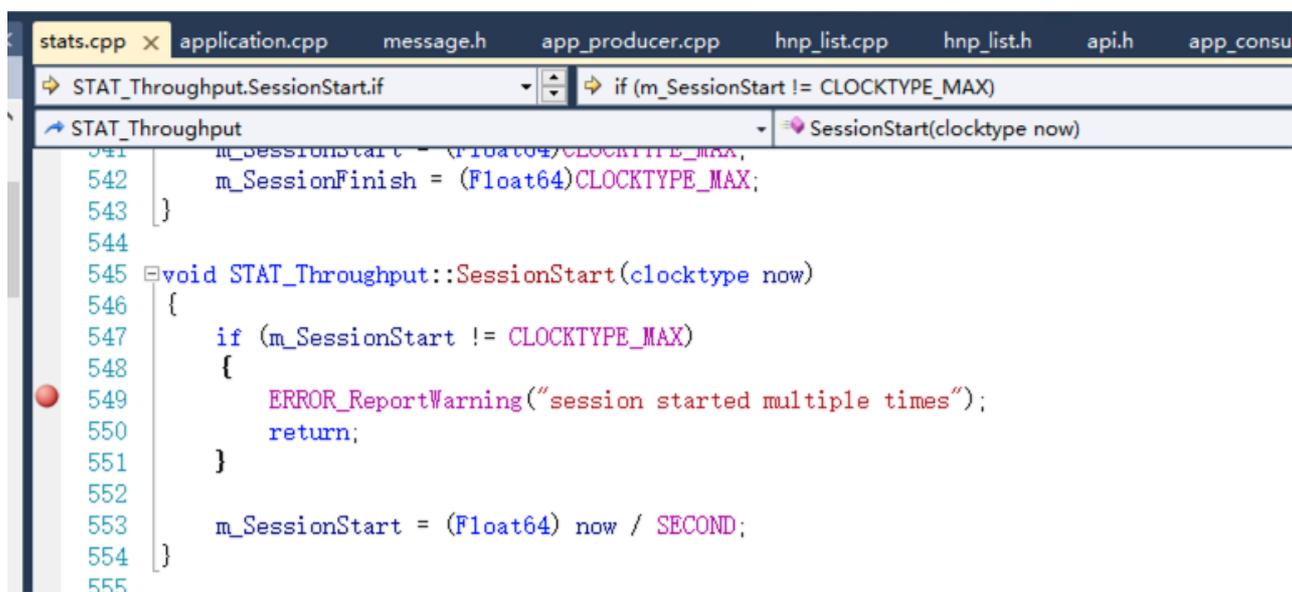
### 1. 为 PRODUCER 添加统计量

#### 1.1 添加统计成员及方法

检查确认Producer数据结构中已添加统计量成员STAT\_AppStatistics\* stats；以及相关方法，如AppProducerPrintStats。

#### 1.2 统计初始化

1. 在Producer创建时（AppProducerNewProducer方法），stats指针暂时置空（NULL）；
2. 在Producer初始化时（AppProducerInit方法），创建统计成员，用new STAT\_AppStatistics方法。位置在Producer创建之后，而且条件是节点激活了应用层统计。名字为“producer”，类型为“STAT\_Multicast”
3. 随即进行统计量的初始化：调用STAT\_AppStatistics::Initialize方法：remoteAddr为组播地址。
4. 问题：运行时出现警告：“Warning in file ..\main\stats.cpp:549; session started multiple times”



```
stats.cpp x application.cpp message.h app_producer.cpp hnp_list.cpp hnp_list.h api.h app_cons...
STAT_Throughput.SessionStart.if if (m_SessionStart != CLOCKTYPE_MAX)
STAT_Throughput SessionStart(clocktype now)
541 m_SessionStart = (Float64)CLOCKTYPE_MAX;
542 m_SessionFinish = (Float64)CLOCKTYPE_MAX;
543 }
544
545 void STAT_Throughput::SessionStart(clocktype now)
546 {
547     if (m_SessionStart != CLOCKTYPE_MAX)
548     {
549         ERROR_ReportWarning("session started multiple times");
550         return;
551     }
552
553     m_SessionStart = (Float64) now / SECOND;
554 }
555
```

5.

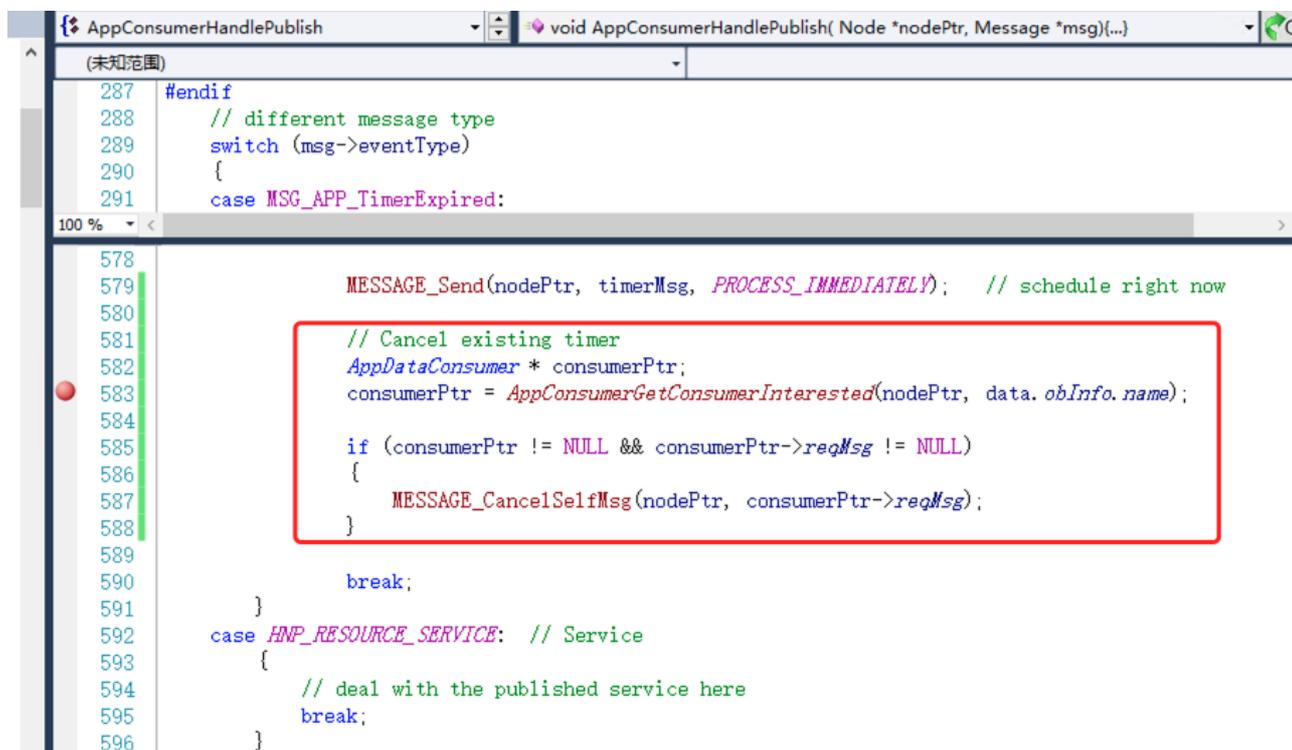
#### 1.3 修改 Consumer 的请求机制

原先的设计是 Consumer 初始化时会启动一个 Send\_PKT\_Timer，到时后将发送一个 GET，对象名字等属性由APP 配置文件设定；此外，如果在发出请求之前 Consumer 收到一个 Publish 消息，且与 Consumer 请求的名字一样，则会立即发出一个 GET 消息。这样就会发生重复请求。因此修改如下：

收到 Publish 时如果1) Timer 没到期则，Cancel 该 Timer，立即发出 GET 消息；2) 如果已经发出，则不再发出请求。

具体流程：Publish 消息处理流程

1. 每个节点维护一个全局的 ObjectInfo 队列，收到 Publish 的也好，包括请求的，都在这里；
2. 检查收到的发布对象信息 Object info 本地链表中是否已存在？如果已存在，则直接返回。
3. (两个列表都没有) 将Object Info 插入本地的发布列表；
4. 检查本地是否有感兴趣的 Consumer；
  - a. 如果有；则检查是否已发送请求的Timer；如果有，则取消原timer。设定立即发送请求的 Timer。
    - i. 在 Consumer 数据结构AppDataConsumer中添加一个 MESSAGE 成员
    - ii. 初始化时，设定发包定时器后，设定该消息。
    - iii. 收到 Publish 后，立即请求时，利用 MESSAGE\_CancelSelfMsg 方法删除原消息



```
void AppConsumerHandlePublish( Node *nodePtr, Message *msg){...}

287 #endif
288 // different message type
289 switch (msg->eventType)
290 {
291 case MSG_APP_TimerExpired:
...
578
579 MESSAGE_Send(nodePtr, timerMsg, PROCESS_IMMEDIATELY); // schedule right now
580
581 // Cancel existing timer
582 AppDataConsumer * consumerPtr;
583 consumerPtr = AppConsumerGetConsumerInterested(nodePtr, data.obInfo.name);
584
585 if (consumerPtr != NULL && consumerPtr->reqMsg != NULL)
586 {
587 MESSAGE_CancelSelfMsg(nodePtr, consumerPtr->reqMsg);
588 }
589
590 break;
591 }
592 case HNP_RESOURCE_SERVICE: // Service
593 {
594 // deal with the published service here
595 break;
596 }
```

- b. 如果没有感兴趣的Consumer，则直接返回。
  - c. 如果有感兴趣的 Consumer，而且也已经发过请求消息，处于则不再发起请求。
- 关于 Consumer 中局部目标列表 pObjectInfoList 的作用
    - 保存收到的发布的 Object；来自 Publish 消息；
    - 请求过的消息：Consumer 已经请求过的 Object info，以避免重复发送。
    - 这就意味着；
      - 如果收到 Publish 消息中包含该队列已有的Object，说明该对象之前已收到发布信息，或者已经请求过，则可以直接忽略。如果队列中没有发现发布的对象，说明没有收到过之前的发布，或者尚未请求

过，则如果有感兴趣的 Consumer 则可以立即发出请求，不需要等到定时器。【如果请求过，但没收到回复，是否立即发送请求？】

- Consumer 正常请求过某个对象后，须将该对象信息保存到列表中。

- 新的问题：Producer 尚未发布Object 之前，收到 Get 消息如何处理？

- Producer 维持两个列表：
    - PublishedObjects：保存该 Producer 发布过的 Object 的相关信息
    - PendingSatisfiedObjects：保存该 Producer 尚未满足的 Object 的相关信息，即在这些 Objects 发布之前收到了Requests。
  - 缓存该请求，当内容准备好后即发送；。问题：是否保存请求者信息？如何定义请求者信息（name + Id）？这是个系统问题，当下处理：暂时不保存请求者信息，暂时像 Publish 那样，如果之前有请求未被满足，则直接广播出去。
  - 广播后，将该 Object 从 Pending list 中清除。
  - 【Consumer】按感兴趣的资源名字匹配，仍然接收。【Done】
- 问题：队列溢出的问题：Producer 响应，如果 chunknum 超过95，可能会溢出。即 Producer 层连续下发 100 个 chunks，但 Consumer 只收到 96 个。

## 1.4 关于调试信息的输出

- 1) 在 Makefile 中添加开关变量 MYDEBUG 到YDEBUG 宏中，即添加 -DMYDEBUG 到 /Zi 后面
- 2) 代码中需要选择编译时，加入ifdef MYDEBUG 宏即可，比如
- 3) 这样可以在 Debug 模式下，很方便的选择打印调试信息。

## 1.5 添加 Trace 功能

以 Consumer 为例，参考 CBR Trace 功能的实现

- 添加 ConsumerInitTrace 和 ConsumerPrintTraceXML 方法
- 在 APP\_TraceInitialize方法中添加 CONSUMER case，在 application.cpp 文件中。
- 注意：在节点设计中要激活应用层 Trace，在场景全局设置中完成。
- 注意：在 PrintTraceXML 方法中，需要区分message 是来自本地，或是远端（Producer），这是比 CBR 复杂的地方。【如何判断消息来源？或者说不管本地定时器消息，因为 Trace 是针对网络内的 packets】
  - 如果是本地的Message，则打印的是 ConsumerData 信息；
  - 如果消息来自远端，则需要打印的是 ProducerData 的信息。
- EXata 5.1 Trace 应用层协议的一个 Bug
  - 问题描述：在 GUI 场景属性-Statistics and Tracing – Packet tracing 中勾选"Trace All Application Layer Protocols"，如果不单独勾选 CBR，则 CBR 不会被 Trace！

- 发现：检查代码发现：
  - 1) CbrInitTrace 时没有问题，即TRACE-ALL YES 或 TRACE-APPLICATION-LAYER YES时，均将激活，调用 TRACE\_EnableTraeXML，赋值打印函数。
  - 2) 问题应该是出在 CbrInittrace 时，如果选择 Trace All Application Layer Protocol，但没有勾选 TRACE\_CBR，则 CBR 仍不会被 Trace。
  - 3) 对于属性框中没有的协议，不存在。
  - TRACE\_CONSUMER := 196; InitTrace后，node->traceData->traceList[protocol] = TRUE，这样才会激活该协议的 TRACE 功能。
  - 注意：Consumer 收到协议是 TRACE\_PRODUCER= 197! 注意之间的差异!
  - 解决方案：将 Consumer 和 Producer 变成一个协议的两个对等端，像 CBR client 和 CBR server；HTTP Client 和 HTTP server! 【重大变动】
- 统一的协议命名：RDP: Resource Dissemination Protocol, 资源扩散协议。