智能运维系列（四）| 曝光交易路径

智能运维系列专题简介：智能运维（AIOps），根据 Gartner 的最新阐释，意指整合大数据和机器学习能力，通过松耦合、可扩展方式去提取和分析数据量（volume）、种类（variety）和速度（velocity）这三个维度不断增长的 IT 数据，进而为 IT 运维管理产品提供支撑。在此，微众银行智能运维团队根据一线工作的实践经验与心得体会，特别撰写了《智能运维系列》文章，本公众号后期将陆续发布，敬请持续关注。

**点击回顾：[智能运维系列（一） | AIOps 的崛起与实践](https://www.infoq.cn/article/fqUfkjhecOla1zKUKycN" \t "_blank)**

**点击回顾：[智能运维系列（二）| 智能化监控领域探索](https://www.infoq.cn/article/Qta6VCyjvHdoiJg5wKze" \t "_blank)**

**点击回顾：[智能运维系列（三）| 浅析智能异常检测——“慧识图”核心算法](https://www.infoq.cn/article/mryjNLXOlujV7fkQFUaL" \t "_blank)**

微众银行在事件识别越来越准的同时启动了事件根因定位项目，以什么为定位的依据，以什么为推理的基础一直是一个难点。通过对历史案例的分析学习，我们发现交易的行为路径就像灯塔一样指导我们找到问题的根本。全行的交易都通过消息总线传递，那么消息总线中一定有我们需要的交易行为路径，可以绘制出所有交易的交易树。

微众银行基于自研的消息总线 WEMQ 来提供金融级消息总线服务，所有的交易都会经过 WEMQ 进行通信，每笔交易都拥有唯一的交易流水号来识别。一笔交易从开始到结束，都会在 WEMQ 消息总线中留下调用信息，这里称之为“原始消息日志”。日志中记录了调用方信息、接收方信息、日志记录点（log\_point）、发送时间、接收时间、交易流水号、具体的交易消息体等信息。正是基于此消息日志和既定的规则，系统通过交易树生成算法，从而形成了每笔交易的独特交易树。此交易树可以应用于告警分析、根因分析等功能。

**原始消息日志处理流程图**

如图 1 所示原始消息日志会通过一些规则引擎预处理之后，得到消息对，每个消息对包含两条 message（request 和 response），代表一次调用；结合 CMDB(配置管理数据库) 的数据对消息对进行处理，依据调用的响应方，生成多个 tree node(一个响应方，一个节点)；将离散的 tree node 依据 node 的上游以及下游转换成若干条 tree chain；将若干条 chain 依据相同父链路等规则合并成树，再做一些整理，如横向合并、纵向合并就得到最后的交易树，如图 2 所示为某交易的具体交易链路。

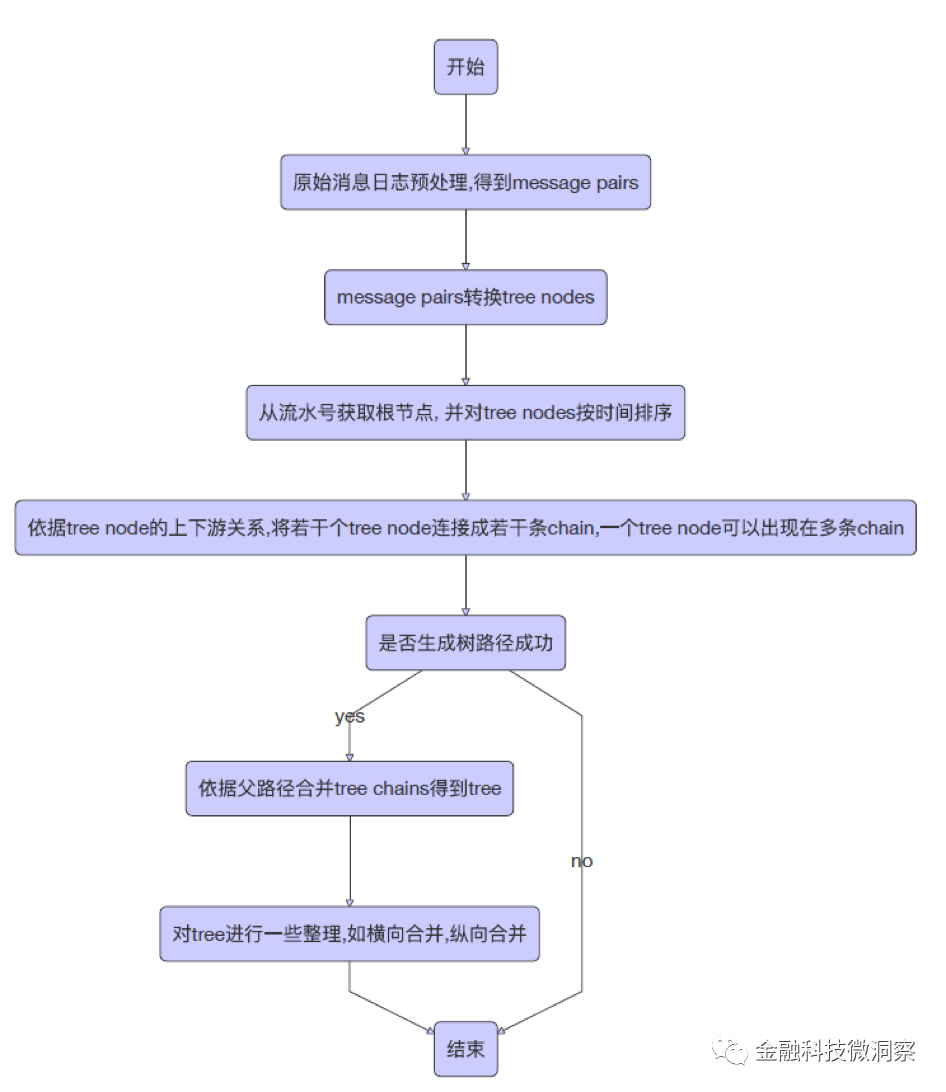


图 1  交易树处理流程图

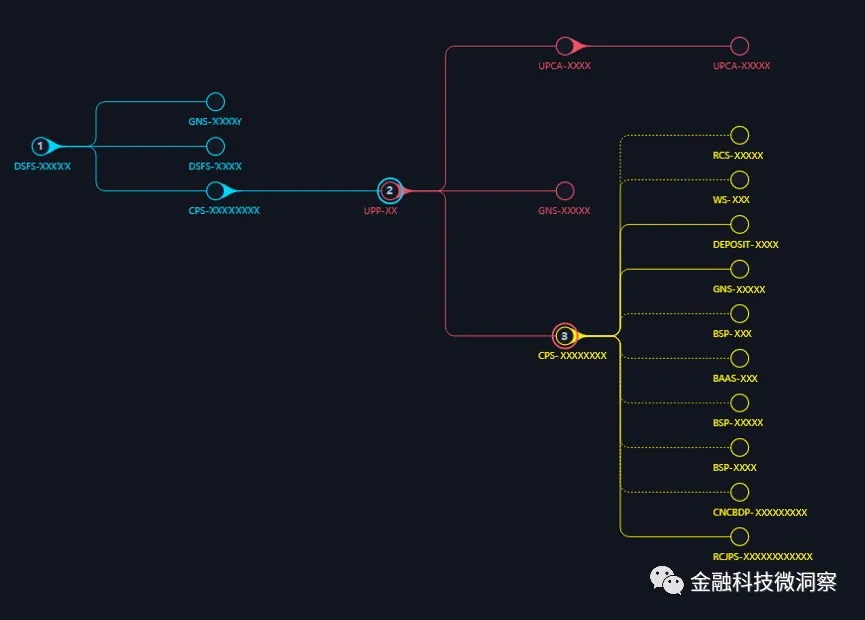


图 2 交易树示例

**获取需要处理的流水号对应的原始消息列表**

首先我们需要获取一笔交易所有的消息日志，基于经验大部分交易会在 3 分钟之内完成，为此我们在对消息日志经过简单的预处理后将其存储在 redis 中，三分钟后将具有相同流水号的数据进行聚合得到完整的交易流水日志。一笔交易以交易流水号来辨识，包含多条原始消息；同一笔交易中存在唯一标识符（unique\_id）来辨识是哪两个子系统之间发生了调用。如 A–>B，A 向 B 发送请求 request，B 会回复请求 response。

**将原始消息转化为消息对**

两个子系统之间的一次调用包含两条消息（一次请求，一次回复），在获取到一笔交易的完整日志后，首先需要对日志按照调用进行聚合，对于同一个 unique\_id 的原始消息会生成一个消息对，即两个子系统之间的调用消息对。为此我们制定了统一格式的消息对（req\_message 和 rsp\_message），需要从日志中提取信息进行填充。

从 WEMQ 获取到的是调用消息的日志，不同版本的 WEMQ API 会上报格式不统一的日志。对于新版 WEMQ API 一次调用存在 2 条日志，对于旧版 WEMQ API 一次调用可能存在 4 条日志。由于不同版本的日志格式不同，会存在不同的 log\_point 即记录日志点不同。根据 WEMQ 使用规范我们总结了原始消息的字段格式。

为了得到消息对，首先将相同 unique\_id 的消息按照 log\_point 分组，将相同 log\_point 的消息数据合并为一条消息。然后初始化消息对 (req\_message, rsp\_message)，根据历史经验总结出了不同 log\_point 的日志置信度，按照一定的规则引擎将多个 log\_point 日志中的信息填充到 req\_message 和 rsp\_message 中，得到最终的消息对。

**消息对转化为树节点**

原始消息转化为消息对后，首先需要将消息对中的属性进行预处理，如某些属性缺失可以通过 CMDB 中的信息进行补充。另外在 CMDB 中还保存有完整的子系统与其提供的服务响应的对应关系，而在消息对的 req\_message 中详细记录了调用哪个子系统的哪个服务，基于此对应关系，我们可以得到所有可能的下游节点。根据消息对中的 rsp\_message 及一定的规则引擎，我们将消息对转化为了树节点。依据调用的响应方，可能会生成多个 tree node（一个响应方，一个节点）。

如果消息对中调用的服务在 CMDB 中找不到提供相应服务的子系统，则无法生成树节点，被称为孤立消息对。

**树节点转化为链路**

至此，我们已经将一笔交易中的原始消息转化为了多个树节点，现在将其按照一定的规则转化为链路。首先要选取一个根节点，然后寻找下游节点依次拼接。在我们的交易流水号中记录了根节点的子系统 id，然后遍历所有的节点，如果某节点的发送方属性（如子系统、ip 等）和根节点的接收方属性相同，则将该节点拼接到根节点后面，形成一条链路。接着重复上述步骤遍历剩余节点，将符合上述条件的节点匹配到每条链路的叶子节点中。

我们就将离散的树节点转换为了若干条链路，一个树节点可能同时出现在多条链路中。若从流水号中没有获取到根节点子系统 id，尝试从已经按照时间排序的树节点中获取前三个节点的子系统 id 作为根节点尝试生成链路。如果能够将所有节点都拼接到链路中则选择该方案。如果有节点拼接不到链路中则选取剩余孤立节点最少的那个树节点作为根节点。

**合并链路生成交易树**

我们已经得到了若干条链路，距离交易树生成还有最后一步。首先对每条链路进行预处理。如果同一条链路中上游节点的接收方属性和发送方属性相同，则表示自己调用自己，做合并处理即纵向合并（调用自循环）。接着对于不同的链路进行处理，如果不同的链路中存在上下游属性完全一致的情况，意味着具有相同的父链路，做合并处理即横向合并。

按照上述规则合并完成得到原始的交易树，对其进行简单整理，将同一层级的子节点按照时间升序排序得到最终的交易树。

**总结**

上面我们详细介绍了如何基于 WEMQ 中的原始消息日志及 CMDB 中的信息生成交易树，我们将原始的消息日志和已经生成的交易树结构进行存储，展示在界面上便于行内运维及开发人员对于具体交易进行搜索，对于交易的整体流程有清晰的感受。另外基于交易树，我们还获取了包括子系统信息，告警信息，日志信息等各维度内容展示在前端，方便查看交易的详细状态。

由于我行的业务量很大，每天有很多交易，这些交易树中包含了很多有价值的信息。我们对于每笔交易利用具体的交易树信息生成交易树的唯一标识 treekey，可以对全量交易进行归类统计分析，得到全行各系统的运行状况。另外我们还通过算法识别出交易归属的产品场景，可以得到整个产品场景的交易森林，对整个产品场景的框架有清晰的感知。交易森林的建立即是交易大数据，在此基础上我们可以做很多的数据挖掘。异常根因定位就是其中的一个应用场景，下一篇将介绍如何使用交易树进行异常根因定位的。期待下次与你分享。