智能运维系列（八）| 事件指纹库：构建异常案例的“博物馆”

博物馆为人类留存珍贵的记忆，以史为鉴方可开创未来。在运维领域，是否也需要一个类似“博物馆”的场所，来记录异常案例，从而助力寻找隐藏的根因定位规律呢？

在微众银行的运维实践中，运维人员结合了大量的长期历史异常数据，基于知识图谱，构建了一个异常案例的“博物馆”，实现了根因的辅助定位，并搭建起一个监督学习框架，既能让机器自动学习，又能以恰当的方式融入专家经验，从而助力运维人员快速精准地定位、解决问题。本文就针对构建该“博物馆”的过程及其应用进行详细说明，如异常案例包含哪些数据、如何存入图谱中、如何进行历史根因的推荐等。

**前文回顾**

[专题 | 智能时代下的运维](https://www.infoq.cn/theme/68)

**历史事件对比流程**

当前智能运维中的根因定位大多从某一个具体和方面入手去推断异常，如日志的根因定位，未能做到宏观的比较。然而，现实中的异常需要分析的维度非常广，包括且不限于告警、接口指标曲线、日志、应用版本发布、SQL 操作、推广活动、应用变更以及业务批量操作等。同时，新案例也不会跟历史案例进行对比，白白浪费了历史案例的宝贵经验。



图1 常见的异常分析维度

因此，我们需要一个更综合的方法，对比历史相似案例进行根因定位，建立一套实时存储、抽取特征、对比指纹、推荐根因的解决方案，辅助当前的分析。一个完整历史案例对比过程包括以下三个步骤：

1.      异常事件的特征（事件指纹）的选取

2.      异常案例在知识图谱中的存储和更新的方案

3.      基于指纹权重的异常案例匹配算法

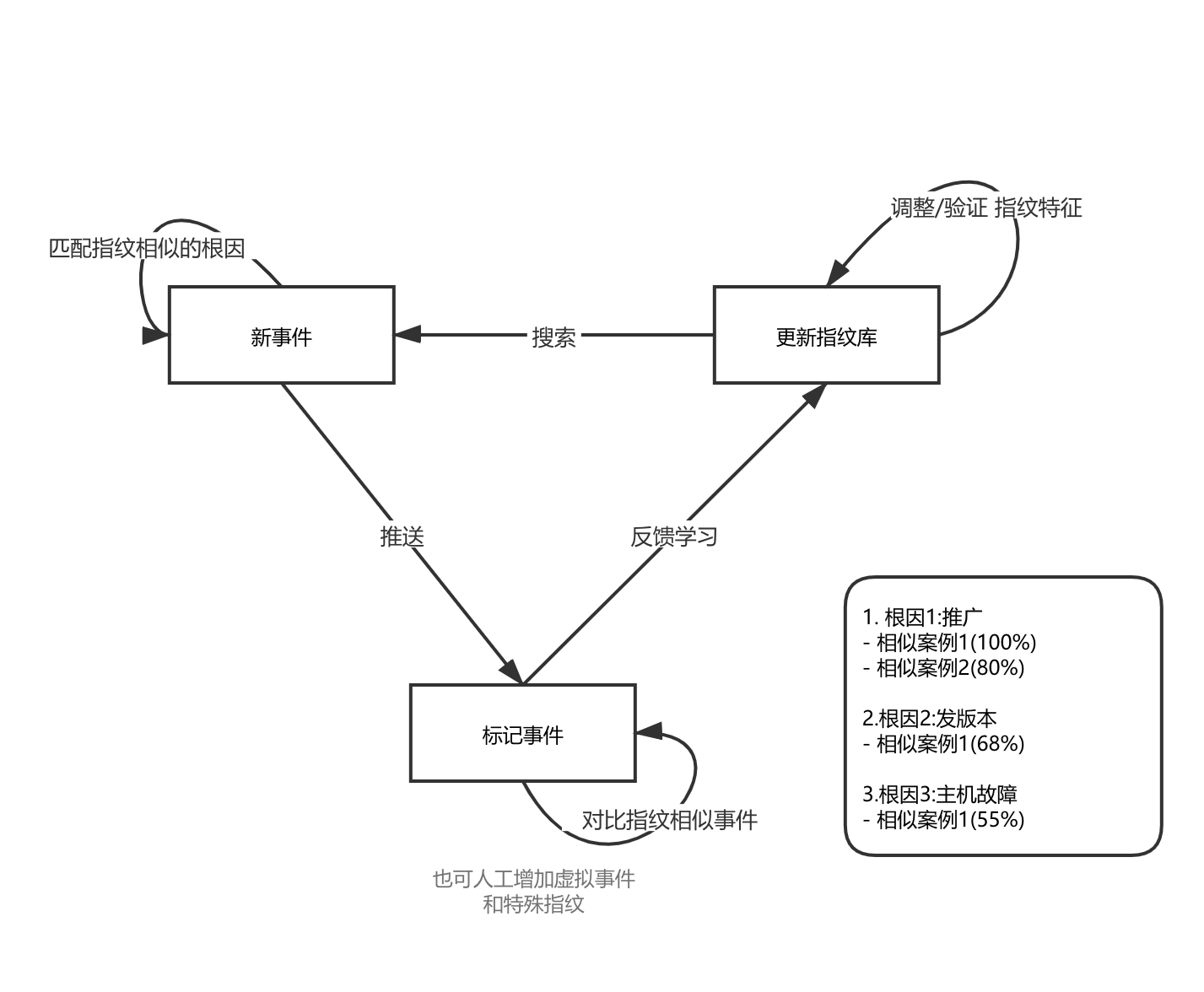


图 2 历史案例比对模型

面对案例分散的信息，我们有必要对异常事件的记录、标记进行标准化。整个流程如上图：事件发生 -> 推送告警群 -> 定位与标记 -> 结构化事件 -> 更新特征指纹库 -> 新事件的相似匹配 -> 推送告警群。

图 2 左上角，发生一个异常事件后会推送到告警群中，经过运维的定位和确认，我们对事件进行标记，如异常类别和责任方是否正确等等。标记的动作会将事件以图 5 方式存入右上角图数据库 Neo4j 的指纹库中。在完成了步骤 2 后，步骤 3 会将存入图谱的事件按表格 1（见下文）进行指纹抽取，从而更新指纹库，步骤 1 则会利用 Tanimoto 系数，根据指纹及权重的不同，计算相似度，进而推出相似历史事件所对应的根因。

对比流程中涉及事件指纹与现象类指纹。事件指纹是异常事件的特征，事件指纹库是所有的事件指纹的集合。如上文中图 1，我们将根因的定位来源分为了七个方面：日志、告警、接口、应用版本发布、SQL 操作、非应用版本变更和推广活动。每个维度，都会产生唯一的事件指纹，用以代表该异常发生时我们所能观察到的 root。

现象类指纹是指产品的关键性黄金指标（如某某产品的某某关键场景下的交易量、平均时延、当前成功率和系统成功率）何时发生了何种异常（如，突增和突降）。异常事件的发生会伴随着多种的现象。例如产品的指标异常，日志关键字，相关告警等等。通过对近半年的上百个历史异常案例分析，我们发现大部分相似的异常事件，都会有着相同的根因。如下图，我行中间件的一些已知故障，会经常造成某产品的查询场景的成功率下降。如果这样的事件可以快速匹配出来, 那么会大大提升根因定位的准确率。

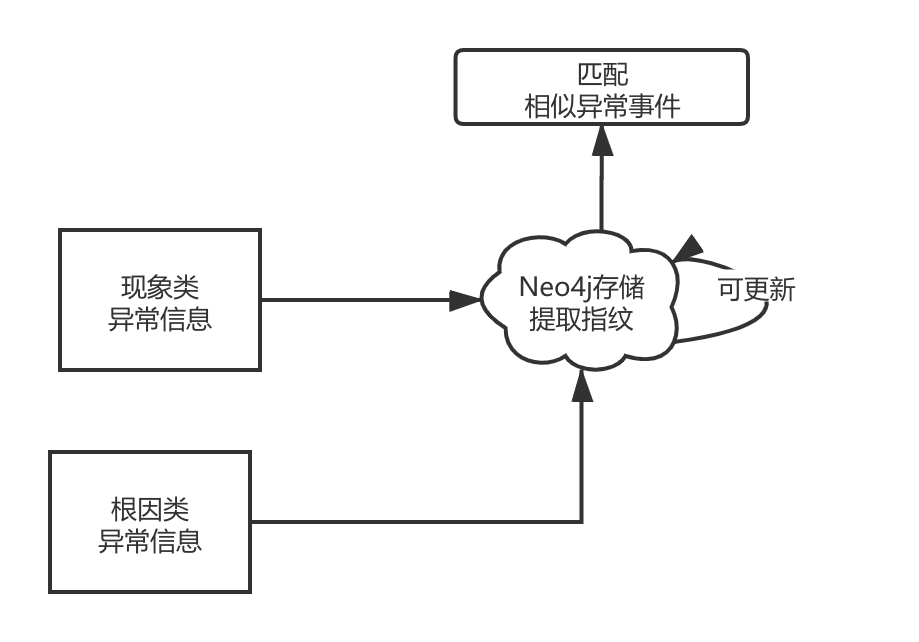


图 3 一些常见的故障现象

值得一提的是，真相往往就在交易量、平均时延、当前成功率和系统成功率这四大指标的组合中。这是一类非常重要的现象，其命中率可高达 80% 以上。



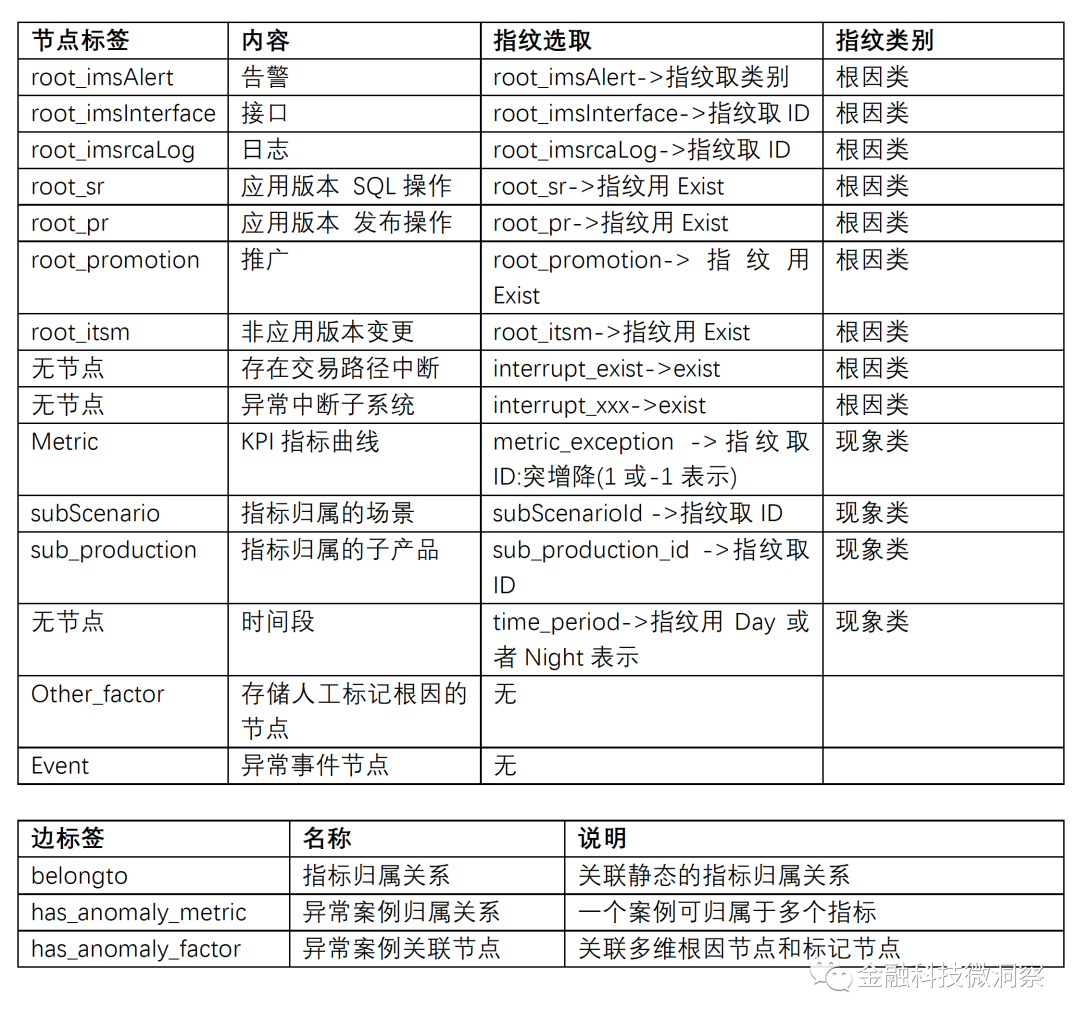
图 4 事件指纹的选取流程

**异常案例在知识图谱中的存储和更新方案**

事件的发生是有脉络的，会从一个故障传至另一个现象，这时就需要将复杂的联系固化成节点和边的形式来展现，即知识图谱。

图谱以事件节点 Event 为中心，关联的每个节点可以存储其相关的信息。如下表，一个指纹可以用 [类别 -> 属性值] 的方式来生成和存储，我们将一个事件的指纹集合存储于 Event 节点，从而纳入指纹库中，便于检索和修改。

表格 1  Neo4j 节点和边的类型



根据上表，可抽象事件图谱的 Schema 如下图：

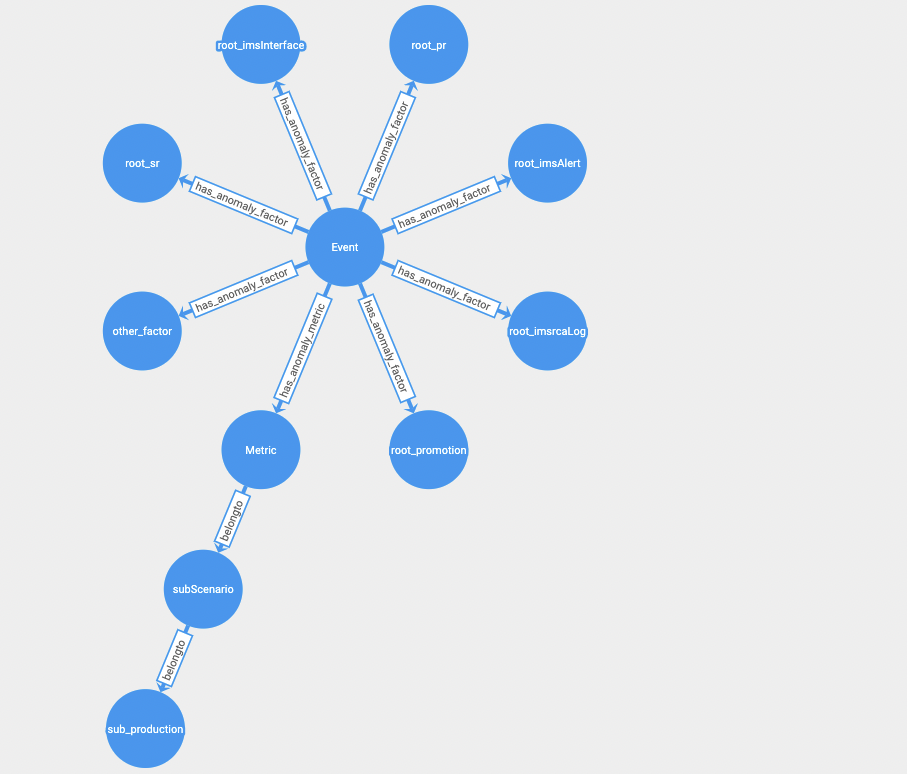


图 5  抽象事件图谱的 Schema

利用 Neo4j 自带的可视化工具，我们来看看一个案例如何存储的。这是一张静态的产品 - 场景 - 指标角度结构图：紫色的是产品，蓝色的是场景，绿色的是指标，包含成功率、时延和交易量。节点通过 belongto 的边进行连接，如下图：

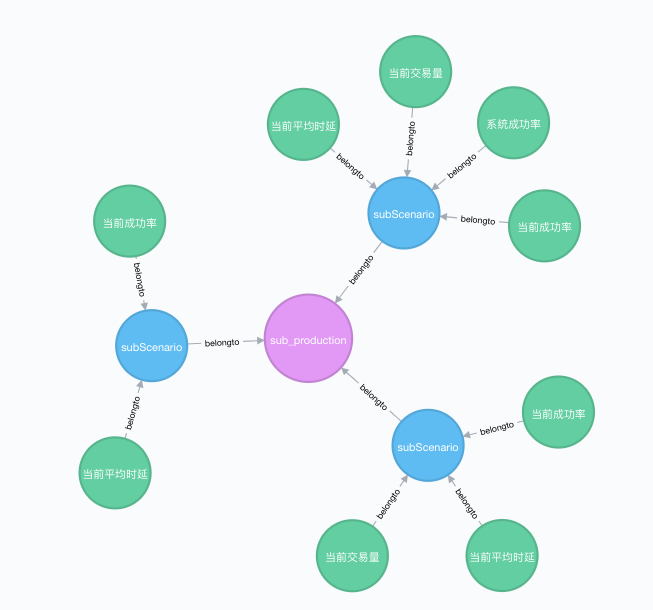


图 6  静态的产品 - 场景 - 指标角度结构图

随着异常案例的增加，绿色指标节点本身不变，但其通过 has\_anomaly\_metric 边关联的红色 Event 节点会相应增加，红色节点会通过 has\_anomaly\_factor 关联其具体的多维根因信息。事件角度视图如下：

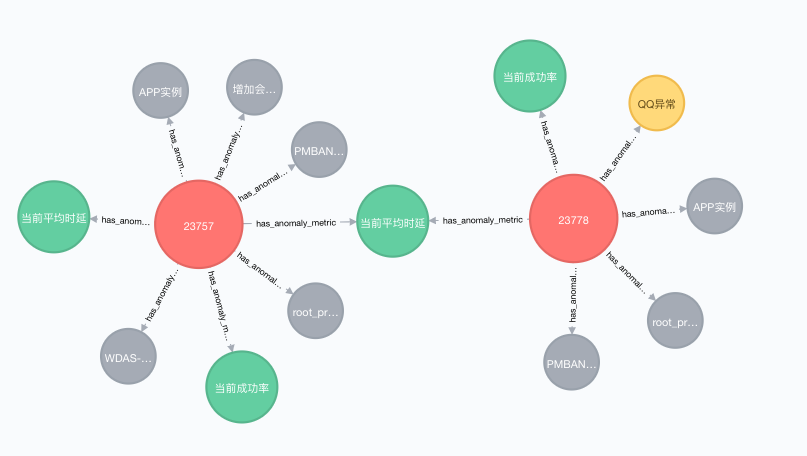
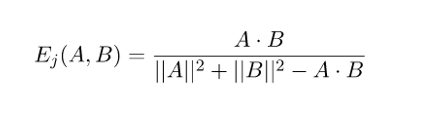


图 7  产品 - 场景 - 指标角度结构图的变化情况

红色的节点是异常事件，灰色的是关联的根因（root\_imsAlert，root\_imsInterface，root\_imsrcaLog、推广等等），绿色的是异常指标曲线，黄色的是人工标记的根因。图 6、图 7 就是图 5 schema 的具体实现。相似历史根因如何匹配呢？我们细看图 7，左边的事件可以视为新异常，右边是根因为【外部合作伙伴】【QQ 异常】的历史案例，两者通过中间的时延指标产生关联，计算其指纹的相似度，达到阈值便可将 QQ 异常作为左边新事件的历史根因推送出来了。

**基于指纹权重的异常案例匹配算法**

相似度的计算采用 Tanimoto 系数，A 是当前异常事件的向量，B 是对比的事件向量，向量中每个元素是一个指纹，值的大小与该指纹的权重有关，权重反映了指纹对于区分不同类型事件的贡献值：



建议的默认权重设置如下：

features\_weight = {

（告警权重值为 3）’root\_imsAlert’: 3,

（接口权重值为 3）’root\_imsInterface’:2,

（日志权重值为 3）’root\_imsrcaLog’:2,

（应用版本 SQL 操作权重值为 3）’root\_sr’:4,

（应用版本发布操作权重值为 3）’root\_pr’:3,

（推广权重值为 3）’root\_promotion’:7,

（非应用版本变更权重值为 3）’root\_itsm’:3,

（KPI 异常项目数值曲线权重值为 3）’metric\_exception’:5,

（异常项目数值归属的子产品权重值为 3）’sub\_production\_id’:1,

（异常项目数值归属的场景权重值为 3）’subScenarioId’:2,

（时间段权重值为 3）’time\_period’:5}

一个推荐正确的典型案例是某业务的查询成功率指标突降，其经常是由于版本发布造成的，但有一次因为运维版本多次发布失败而未被关联到，导致根因定位到了接口上。然而，历史根因首位推荐的就是应用版本发布。

当前异常事件的当前指纹信息包括：时间段：白天、KPI 异常项目数值曲线 ID：69766:-1、KPI 异常项目数值曲线 ID：17319:-1、接口 ID：CPUPCA\_47758、异常项目数值归属的子产品 ID：401，因此，A 的（指纹）fps=[“subScenarioId->4010101”,

“time\_period->day”,

“metric\_exception->69766:-1”,

“metric\_exception->17319:-1”,

“root\_imsInterface->CPUPCA\_47758”,

“sub\_production\_id->401”]

历史异常事件的历史指纹信息包括：时间段：白天、接口 ID：CPUPCA\_47758、异常项目数值归属的子产品 ID：401、异常项目数值归属的场景 ID：4010101、日志 ID：UPP\_11077、KPI 异常项目数值曲线 ID：17319:-1，因此，B 的（指纹）fps=[“time\_period->day”,

“root\_imsInterface->CPUPCA\_47758”,

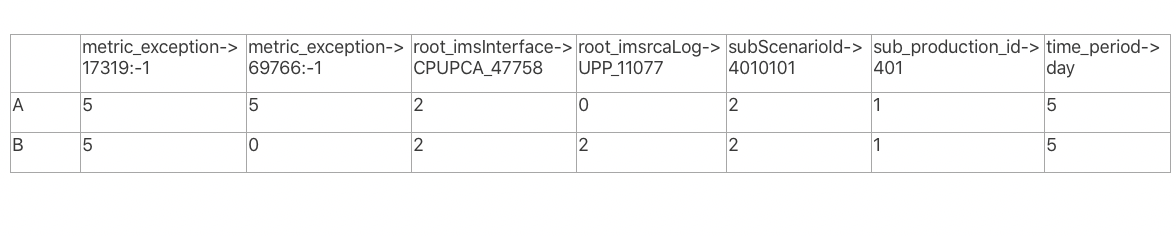
“sub\_production\_id->401”,

“subScenarioId->4010101”,

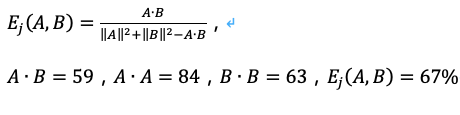
“root\_imsrcaLog->UPP\_11077”,

“metric\_exception->17319:-1”,]

可以通过对当前异常事件和历史异常事件中的上述指纹（维度变量）分别进行向量化，采用 one-hot 编码，乘以权重，分别得到当前向量 A、历史向量 B：

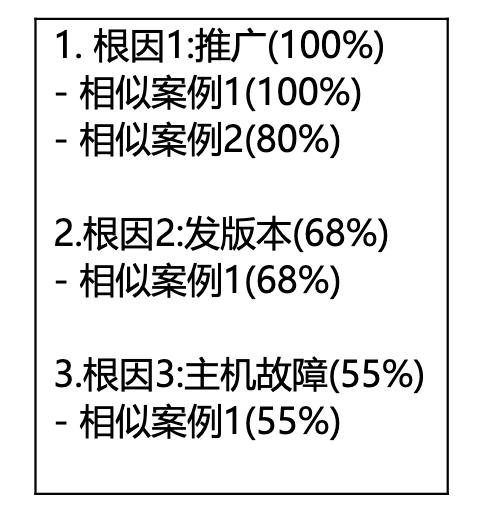


将 A 和 B 代入公式 ：



那么该当前异常事件与该历史异常事件的相似度为 67%。设定阈值为 50%，那么我们就认为 AB，重合度较高，两个事件相似。

接下来，通过聚合排序推荐最有可能的相似根因。（聚合：根据历史案例的标记根因 other\_factor 的值进行聚合；排序：取每个集合内最大的相似度作为该集体根因的相似度，进行排序，最后取前三位进行推荐），从而得到最终的结果：



需要注意的是，标记的根因记录在 other\_factor 中，标记内容需要有一定的概括性，良好规范有助于以后的日报统计和监督学习，具体格式如 [根因大类][细分的错误根因]（如：[中间件][HBase 故障] ，[外部合作伙伴异常][财付通] 等）。根因标记的大类可以分为：‘外部合作伙伴’, ‘中间件 GNS’, ‘TDSQL’, ‘TGW’, ‘HBase’, ‘Redis’, ‘WEMQ’, ‘DNS’, ‘应用主机’, ‘邮路’, ‘应用版本发布’, ‘内部程序异常’, ‘正常业务失败’, ‘网络’, ‘SQL 操作’, ‘灰卡’。

另外，还有一些需要注意的事项：

* 根据我们选取和生成的指纹，可以采用多种算法进行相似性的对比，包括单不仅限于余弦相似度，jaccard 相似系数等。
* 指纹的向量化：除了 one-hot 还可以采用 DeepWalk、TransE 和 NodeVec 等算法进行向量化。
* 特征不多，历史案例稀少，结论的给出会显得非常武断。因而，历史根因往往会变成一种辅助手段，可能很多次的推荐根因都不对，但能有一次拯救运维于水火之中，也是一个不错的提醒信息。

**总结**

规范化、标准化异常事件各维度信息的存储，是保障根因定位快速迭代、快速对比的必要条件。面对不同结构的接口、多种的关联关系，采用 Neo4j 这样的 Graph Database 是个不错的选择。知识图谱是由一些相互连接的实体和他们的属性构成的，借用这样的概念，我们可以去梳理、抽取一条条案例，将经验留存下来，构建异常案例的“博物馆”。

本文提出的推选历史根因，实质上是一个基于异常事件知识图谱的推荐系统。类似物物推荐，当你收藏了一本书，系统会推给你同作者的其他书籍。同理，我们根据事件本身的属性进行向量化，在候选集中计算相似度，推荐该事件最可能的根因。随着数据的积累和标记的完善，可以让系统进行监督学习，直接预测异常分类，指导根因的判断方向。但想建立的一座丰富的“历史博物馆”，还是差得很远。数据像石料一样堆起了建筑实体，但还缺少了人工智慧的点睛。如何在知识库中纳入运维经验、结合规则引擎去动态调整根因？如何自动挖掘异常案例中隐含的规律？这些问题还值得去深入探索！