智能时代的新运维

Operations Trends in Intelligent Age

InfoQueue

趋势总览

- 到底该如何理解AIOps？又如何落地AIOps？
- 运维技术大盘点，2017你该关注运维的哪一面？
- 无服务器计算的未来

先锋实践

- 阿里Goldeneye四个环节落地智能监控：预测、检测、报警及定位
- 京东618：升级全链路压测方案，打造军演机器人ForceBot
- 360：集预测、处理、关联和资源优化于一体的智能运维系统

Geekbang.
极客邦科技

关注极客邦科技动态

极客邦科技简介 | INTRODUCTION »

极客邦科技（Geekbang）创立于 2007 年，是集资讯、会议、电商、培训、咨询、图书出版、社交、整合营销、创新孵化等九大产业于一体的 IT 内容综合服务集团。旗下运营 EGO 职业社交、InfoQ 技术媒体、StuQ 斯达克学院职业教育三大品牌。总部设于北京。集团致力于让创新技术推动社会进步，十年来已为超过 100 万技术人、3000 家企业提供服务，业务遍布中国各城市，及美国、法国、德国、荷兰、以色列、日本等国家。

使命 | MISSION »

让创新技术推动社会进步

Help to Build a Better Society with Innovative Technologies

愿景 | VISION »

世界级的 IT 内容服务平台

Become the World-class IT Content Service Platform

三大业务品牌 | BUSINESS »

EGO EXTRA GEEKS' ORGANIZATION NETWORKS

高端技术人员学习型社交平台

面向 CTO、技术 VP 和 CEO，付费会员制，城市分会，学习活动，全球技术领导力峰会

InfoQ

专注中高端技术人员的技术媒体

技术专栏，迷你书，技术微信矩阵，视频品牌，顶级技术大会，垂直技术大会

StuQ 斯达克学院

实践驱动的 IT 教育平台

十大技术领域课程，个人学习服务：公开课、直播课，企业学习服务：企业内训、工作坊、训练营、轻咨询

北 京 | 伦 敦 | 纽 约 | 旧金山 | 圣保罗 | 上 海 | 东 京

QCon

全球软件开发大会2017

[上海站]

主办方 Geekbang. 极客邦科技 InfoQ

2017前瞻热点技术All in QCon

9折购票中，立减680元

截止2017年09月17日 团购享受更多优惠

购票热线：010-6473-8142
会务咨询：qcon@cn.infoq.com
赞助咨询：sponsor@cn.infoq.com
议题提交：speakers@cn.infoq.com
Q Q 咨询：1173834688

上海·宝华万豪酒店 / 2017年10月17–19日

卷首语

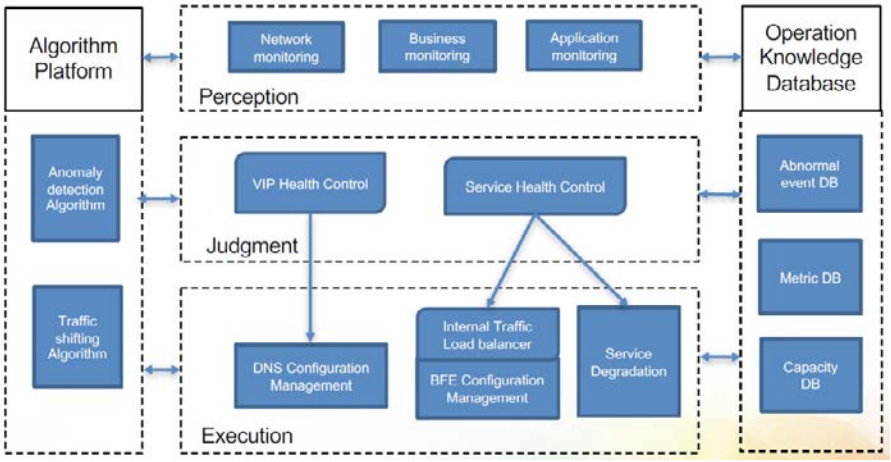
作者：木环

明者远见于未萌，智者避危于无形。

——西汉司马相如《谏猎书》

纵观历史，运维技术经历了业务规模小的原始手工时代，ITIL 管理概念下的脚本时代，和业务量增加原有情况难以维续的改革自动化工具时代。这与机械化、电气化和信息化的三次工业革命竟然有着异曲同工之妙。那么在社会各界纷纷猜测智能化可能引发第四次工业革命的今天，运维人是否应该开始思考智能化会给运维工作带来哪些影响呢？

人工智能的基础是要有可以计算使用的数据，而运维工作中从来不缺少数据，参照 OSI 七层模型，物理层、数据链路层、网络层、传输层、会话层、表示层和应用层都有传输给运维人的数据。AI 对运维的影响并不是说说而已，也不是盲目炒作，业界已经有领先公司研究并取得了一定的成果：IBM 使用了机器学习进行磁盘故障预测，Google 研发 Raft 和 Paxos 算法解决其分布式系统的一致性，阿里基于时序数列预测业务异常，微软则使用智能机器人 SREBot 提升故障修复速度。百度研发的 AIOps 平台为例，其整体机制是搭建一个智能化的算法平台不断学习处理数据，学习积累沉淀出运维知识数据库，并在感知、判断和执行阶段更多地依靠机器工作。

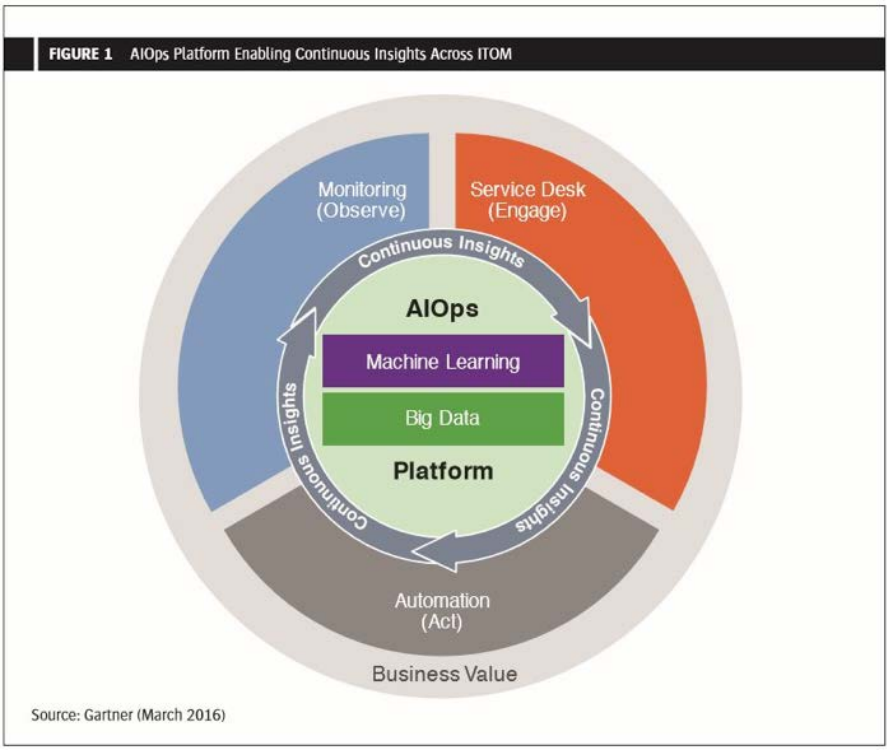


其实挖掘运维数据隐含信息，也并不是新鲜想法，原来业界也早有尝试。ITOA（IT Operations Analytics），将服务和应用数据收集、综合分析、规律发现等流程自动化，在高度分布式、业务多样、技术栈

复杂并且服务迅速变化系统中非常重要，相比人工手动，这样可以快速定位问题并且提高系统表现性能。这里的运维数据不仅仅指机器层面，并且也关注到是应用层面；这是在业界意识到数字化技术浸入到组织每个环节的时候，就已经开始思考和尝试的了。在一次与普元信息的专家交流中，专家提出 AIDevOps 是 AIOps 之后的新阶段。业界有一种看法是，DevOps 是从开发端到运维链路的打通，但是真正可扩展化规模化的工作承受量则要依靠 AIOps。

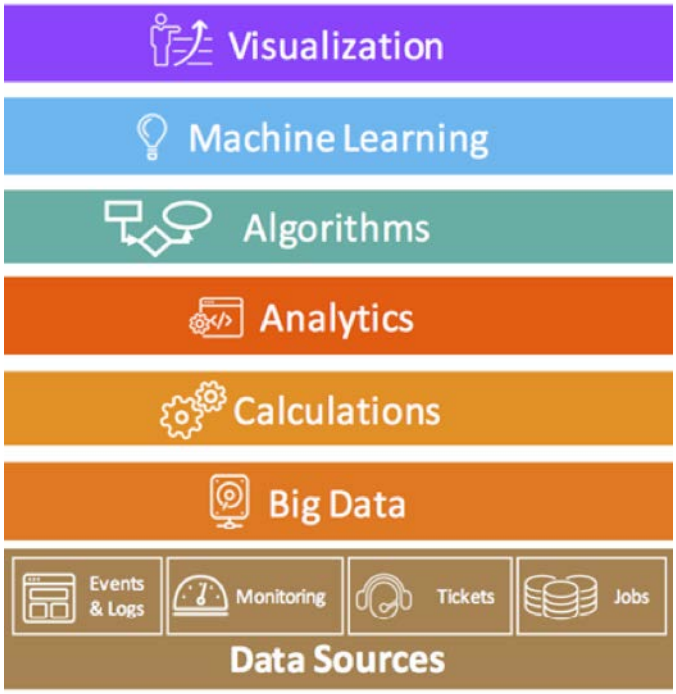
阶段	特征
个人集成开发工具	原始，依靠个人能力，错误难以避免
组织内领域支撑工具	领域内，组织内自动化：统一的需求环境，统一的编译环境
DevOps 1.0	贯穿生命周期
DevOps 2.0	以容器云为基础设施
AI Ops	智能化运维
AIDevOps	智能化开发，智能化运维，智能化生命周期管理

数据量巨大，工作繁琐，人工挖掘是吃力不讨好的工作，交给机器不是正合适吗？ ITOA 提出后，就有很多企业尝试进行 IT 数据的处理，但是那时候似乎没有找到正确的、有效的打开方式；有了机器学习和深度学习，挖掘数据信息如虎添翼，不必像以往单纯依靠传统概率统计论。 也正是因为机器学习和深度学习这样的重大突破之后，学术界已经研究了半个多世纪的人工智能再次回到业界焦点，并且被社会所广泛注视。



上图是 Gartner 给出的 AIOps 平台构成和价值图。除了上文提到的大数据和深度学习（算法）之外，还需要指出的是，AIOps 是服务在商业价值的大范畴之下的，即通过 AIOps 平台，可以更加快速更加精准地进行监控、管理和行动，以此来更好地实现 IT 运维所需要支撑和实现的企业业务价值。新技术层出不穷，没有必要也不应该单纯地追逐技术潮流，必要的是有着客观理性的认识，这是一些冷静的 IT 高手们所常常提到的，“技术终究是要为业务服务的”。

而对于 AIOps 平台化的构成，有如下一张图，从日志、事件、监控系统、故障和工作任务等源头收集数据，通过大数据技术进行计算和分析，同时结合传统算法和机器学习技术将挖掘出的信息和认知可视化地呈现给运维人员。不过，具体到每个公司而言如何真正将 AI 融入运维或者在实现自动化运维之后如何更加智能，打开的方式和实践成果都会有所差别，因为各自的规模、技术力量、业务需求和组织架构不尽相同。



诚实地讲，在刚刚兴起的今天，AIOps 是新概念，业界经验尚且单薄，很难现在就给出 AIOps 的确切且广泛适用的定义，哪怕是著名如 Gartner; 随着业界实践、反思和讨论的不断积累，AIOps 的认知会发生演变。但是，有一点可以明确的是，AIOps 所指代的整体趋势是毋庸置疑的，智能化会逐渐走入 IT 行业乃至社会生活的各个方面。确实没有必要夸大宣传 AIOps 会快速地彻底革新运维，不过见微知著、未雨绸缪和敏而好学终归不是坏事；当机器能越来越智能地工作，我们也应当变得越来越聪明。

到底该如何理解 AIOps？又如何落地 AIOps？

作者：郭蕾

近年来，人工智能技术备受关注，将 AI 引入 IT 运维领域，AIOps 的概念由此应运而生。Gartner 的报告宣称，到 2020 年，将近 50% 的企业将会在他们的业务和 IT 运维方面采用 AIOps，远远高于今天的 10%。尽管 AIOps 还是一个新名词，但并不代表它只是未来的一种趋势而已。在这个数字的年代，任何使用传统技术来管理机器数据的组织要么忽略了信息的价值，要么已经让他们的运维团队不堪重负。

那就当下而言，我们应该如何理解 AIOps？AIOps 应该如何落地？能否通过 AIOps 支持更好的运营？带着这些问题，我们采访了宜信技术研发中心高级架构师张真，请他从宜信近几个月落地 AIOps 的角度聊聊他的想法和洞见。另外，张真也会在 CNUTCon 全球运维技术大会 上分享他们的实践案例。

InfoQ：你是如何理解 AIOps 的？AIOps 的关键点是什么？

张真：我认为 AI 的生态体系与大数据类似，存在两种基本角色：AI 科学家和 AI 领域工程师（FE）。前者推动 AI 科学的发展，创造新的 AI 知识体系；后者是将 AI 知识运用到生产生活的某个领域，创造现实价值。AIOps 正是将 AI 技术应用到 IT 运维领域，帮助变革运维模式，提升效率和创造现实价值的“工程化”过程。

从实施角度，AIOps 这个词本身就体现了两个关键点。其一，Ops 代表运维的场景，这是主旨，识别什么样的场景存在哪些痛点，AI 可以帮助解决；同时也要清楚认识目前的 AI 技术擅长什么，不擅长什么，有哪些限制，切忌凡事尽 AI。其二，AI 作为前缀代表技术，这是手段，AI 技术门类很多，选择合适的，正确的技术去解决真正的问题，是需要切实履行的原则。此外，要从实际出发，考虑投入与产出。

从技术角度，也有两个关键点。首先 AI 的目标是系统类人化，而 AIOps 是将运维系统类人化。它的技术栈应该涵盖三个基本特征：类人交互，主动决策，理解执行。这是与自动化的本质区别。其二，与 DevOps 工具链深度集成是必由之路。我认为 AIOps 不是要替代现有的工具链，而是通过类人化提升“智慧”，实现 SRE 甚至超 SRE 的效果。要达成这个目标，AIOps 就要“学习，了解”这些工具，并且更好的“使用”这些工具，这个过程就是深度集成，它的核心是对这些工具 API 的自主认知和自主使用。

InfoQ：AIOps 和 DevOps 有什么关系吗？可否聊聊你看到的运维理念的演进过程？

张真：正如刚才提到的，AIOps 在技术层面要对 DevOps 工具链进行深度集成。另外，我也想从发展历程角度谈谈它们的关系，其实我也是伴随着这些发展阶段逐步成长起来的。这应该也是个人运维理念的演进过程吧。个人认为 运维经历了 4 个基本发展阶段。

- 早期工具时代：这个时期是 IT 运维的软件工具，流程初始化的时期，工具的目标仅仅只是计算机化，流程尚属摸索阶段，还没有形成行业共识。
- Pre-DevOps 阶段：ITIL，DevOps 等理念在这个时期提出，ITIL 强调流程管理质量，而 DevOps 强调打破开发，测试，运维的边界，One Culture as One Team with Closed Cycle，这个时期也开始了围绕如何落地 DevOps 工具链的技术研究，业内就 IT 研发与运维逐步达成了共识。
- DevOps 阶段：DevOps 的工具链已经比较成熟，甚至出现了一些高级形式，比如 SRE，ChatOps 等，其中 ChatOps 通过对大量运维工具的封装，构建了一个代理，它认识人类定义的特定文本指令，并按照指令处理问题。这个时期自动化运维出现了，更加强调从运维流程，运维措施等层面实现完全的自动化，在特定情况下，甚至实现无人干预。
- AIOps 阶段：自动化运维带来了很大进步，但毕竟系统软件是死的，只能 100% 按照人类制定的流程来运行，不能自主适应，甚至不能处理“相似”的“新”问题。于是 AI 被尝试运用到 IT 运维这个领域，这个阶段应该说才刚刚起步，行业对 AIOps 充满期待。

InfoQ：在运维过程中，有哪些技术痛点适合使用人工智能技术来解决？

张真：我认为有两类痛点可以关注。

1. 时效类问题：运维的本质是提供稳定可靠的服务，而达成这个目标的关键是足够好的时效。时效类的场景还是很多的，例如更短的 MTTR（平均故障恢复时间），特别在服务规模很大的情况下，监控数据的获取 / 集中 / 分析，问题的跟踪 / 定位，恢复的执行规划，如果再加上海量数据和状态频繁变化，AIOps 时效都会远高于有经验的人 + 工具。此外，人类有“工作时间”和“工作活力”的限制，自动化依然离不开人的决策，但智能化可以自主决策，当然目前是在经过检验的人类经验范围的扩展学习。“无中生有”的经验创造依然是个难题。

2. 协作类问题：人类的生产离不开协作。尽管有了自动化运维平台或工具链，运维很多场景还是需要许多人工协作。一个经典的例子：业务发现了问题提交工单给 IT 服务台，IT 服务台根据经验初步判断可能与哪些系统相关，再通知相关团队，相关团队判断是否是自己的问题，如果是自己的问题则考虑的修复方案，然后修复，再反馈给 IT 服务台，通知业务。这是典型的 ITIL 流程。

可是实践表明，科学的流程未必带来理想的结果。因为这个过程中人既是参与者，也是驱动者，人可能懈怠，可能 miss 信息，可能误解，可能情绪化。

另一个例子：自动化运维系统能够通过报警通知

系统团队，比业务更快发现问题从而解决问题，甚至直接通过重启等自愈手段自动的解决问题，这是自动化运维带来的价值。但同样也要看到这里的问题判断与恢复规划仍然是人做的，自动化自愈等也只是人把某种情况下的问题识别，判断和处理的经验封装成执行代码而已，如果情况发生改变，系统将“不知所措”；而且系统团队也可能不了解业务影响，还是要找业务团队确认，如果业务团队太多，还是要通过 IT 服务台。那么这里的问题是什么呢？其实是缺少一个“全知”（掌握业务，系统，基础，组织的各种信息）能够客观的，全面的“协调”人，系统，业务的角色。

InfoQ：可否谈谈宜信 AIOps 探索情况？你觉得什么样的团队来搞 AIOps ？

张真： 首先，来谈谈背景和原则。在规划 AIOps 项目之初，我们确立了几点原则。目标是 从实际痛点入手，找到适合场景以及正确的问题来试点，而不是“大而全”的 AIOps 解决方案。技术选型上 充分利用已经比较成熟的开源 AI 技术，可以做必要改进，但尽量不重复造轮子。充分使用我们现有的 DevOps 工具链，而不是全面推倒重来。

AI 技术还不是“平民技术”，尽管已经发展了很长时间，也有人说我们处于第二次人工智能革命，但它的投入产出比可能并不像使用 Spring、Tomcat、RabbitMQ 这些开源技术栈那样的直接。所以先做“点”的事情，再考虑“面”。而且确实并不是所有场景都适合。前面也提到了，要避免凡事尽 AI。

其实问到什么样的团队来搞 AIOps。这个事与技术选型相关，也与团队定位相关。我们团队的定位是 AI FE，是将 AI 技术工程化的团队，这样的团队应该具备几个特征。

1. 对现有 AI 技术充分了解和掌握。
2. 选择较成熟的开源 AI 技术是必由之路。
3. 对运维领域的技术（比如监控、容器技术、CI/CD、问题诊断等）是清楚的，最好是专家。
4. 对运维领域的场景是熟悉的，明白运维的标准，逻辑，原则。

另外，尽管 AIOps 会带来颠覆性的运维思维和效应，但是否也要对现有系统软件来一把推倒重来呢？这里的考虑是我们的 DevOps 工具链已经比较成熟且运行稳定。同时正如前面提到 AIOps 并非是要取代现有系统，而是赋予现有系统智能。所以与 DevOps 工具链深度集成是必由之路。复用现有 IT 优良资产，最大化资产价值也是必要的考量。

再来谈谈进展。我们目前 AIOps 落地的 形态是任务机器人，相关技术也围绕它展开，涉及自然语言处理、搜索技术、知识图谱、监督学习、在线学习、深度学习等。现在处于实验落地阶段，有三个基本场景。

DevOps 的一个典型场景：系统上线。上线的几个痛点是时机选择，上线条件判断，部署验证，功能验证。这些部分有的是需要人工判断的，有的通过工具进行，但都是人工驱动的逻辑判断。这是个时效类场景，如何上线更快，更可靠。

另一个场景是运维的日常工作：巡检。尽管监控系统已经可以掌握全方位的数据，比如应用性能，日志，调用链，基础设施等，还是需要有人值守；而当报警出来的时候，往往又滞后了；此外微服务架构下，人工也跟不上规模的增长和状态的快速变化。而任务机器人是可以正真全天候运行的。这也是时效类的场景，对问题的及时发现，甚至预判。特别值的一提的是，这是主动行为，而系统上线是被动触发，这两个场景正好体现了类人化智能的两面。

我们相信运维的价值在于更好的业务价值转化，Better ITOps for Better Business。这个场景是协作类的，涵盖运维和运营。从业务同事来看他们有两个痛点：一方面他们不懂 IT 术语，玩不转运维系统，但也想时刻掌握系统运行状况；部门以及团队在运营过程的信息不对称，不能随时快速同步，造成运营效率下降。任务机器人作为中间协调者，所有人有问题就找它，它会“不厌其烦”的，“孜孜不倦”的予以解答。

如果说我们通过 AIOps 有什么收益，从前面提到的场景的痛点出发，收益是显而易见。在 此次大会的分享中 会对这三个场景做深度解读。

关于下一步计划，主要会考虑三个方面。

1. 不断提高基本意图理解，系统 API 理解以及个性化交流语义理解的正确率。

2. 加强自主问题诊断分析上的研究和应用，希望从离线方式逐步转变为在线方式。

3. 尝试在更多时效类，协作类场景中应用。

InfoQ：AI 的前提是数据，那你们数据是怎么来的？

张真：关于数据来源，诚如我提到的，深度集成 DevOps 工具链是必由之路（重要的事情说三遍），因为它们就是数据的来源。当然其中监控系统是主要的数据提供者，我们的监控系统代号 UAV（含义：无人机），它提供了几种主要数据：应用画像，服务图谱，应用性能，基础设施性能，日志，调用链，业务指标。

比如通过对应用画像的学习，提取 API 模型，让系统可以使用 API，这是一种新的系统关联方式。又比如通过对服务图谱的学习，让系统掌握应用之间的关联关系，这是自主跨应用问题跟踪和影响分析的基础。还可以通过对应用性能指标的特征提取，找出异常点等。

此外，UAV 会在 9 月份正式开源，与 CNUTCon 2017 大会共襄盛举，它不但能够帮助大家实现三维一体（业务，应用性能，基础）的监控，也能如同我们一样便捷的，集中的获得 AIOps 的机器学习数据来源，欢迎大家的关注。

InfoQ：任务机器人落地过程中，难点是什么？

张真：从我们的实践经验来看，实现任务机器人有三个主要难点。

1. 基本意图理解：就是要理解人想做什么，这是类人交互的体现。目标是能够从自然语言中提取目标信息，并与可识别目标进行匹配，从而理解意图。我们采用了词向量与句型匹配相结合的手段，并对词向量的实现方法做了一些改进，以大幅缩减词向量空间，提高词向量的匹配速率。这个部分会介绍词向量与句型匹配相结合的基本意图理解的原理。

2. 系统 API 理解：除了需要理解人的意图，任务机器人也同样需要像人一样去理解与之交互的系统 API 的含义以及如何使用，而且要自动适应系统 API 的变化，这是自主决策，理解执行的体现。我们采用了“微智能”（自动发现，自我维护，自动适应）与半监督学习类算法相结合的手段，让它能够认识并使用系统 API。这个部分会介绍我们的 API 模型库如何建立以及如何应用。

3. 个性化交流上下文构建及语义理解：FreeStyle 的交流方式是不限制人必须记住某个指令或特定关键词（与 ChatOps 的区别），这是我们的基本目标。

在金融运维 / 运营的垂直领域，尽管比广义领域的词量范围要小，但仍需要解决以下问题。

1. 由于每个人的认知差异（不同专业背景，不同的个体说话习惯），所以会用不同的词汇与句式来描述同一个事物。例如 SRE 会说“贷款系统是否健康”？应用研发会说“贷款平台有没有线上 bug”？业务同事会说“城市信贷业务运行得怎样”？实际上都是指贷款系统的服务运行情况，有没有系统或业务异常。

2. 提取调用系统 API 的参数。NLP 只是按照人的语言习惯来提取语素信息，而任务机器人还需要从自然语言中，识别要调用的系统以及相关 API 的实际参数。例如说“告诉我电签的运行状况”，从基本意图识别来说“运行状况”指的目标服务是监控平台，“电签”是要提取的监控信息的参数。

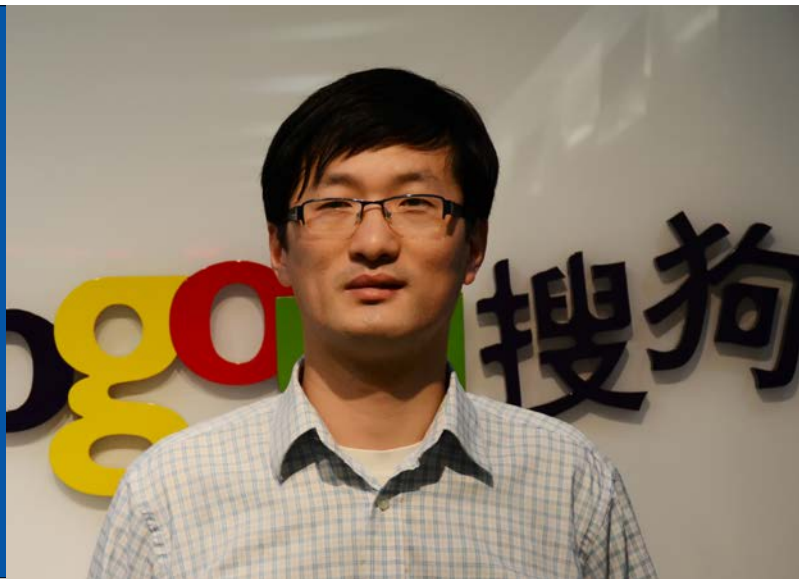
这些问题我都会在 CNUTCon 上和大家分享我们的解决方案，当然也欢迎大家一起交流探讨。

InfoQ：在 CNUTCon 全球运维技术大会上，你会为大家重点分享哪些技术点？

张真：本次分享的议题是《AIOps 的核心技术之一：任务机器人如何在金融运维 / 运营中落地》。我将会介绍任务机器人的构建理念是什么，采用什么样的架构原理，然后从难点问题出发剖析实现原理，也会介绍前面提到的典型应用场景以及如何落地。

运维技术大盘点，2017 你该关注运维的哪一面?

作者：刘建



近些年来，软件领域发生了翻天覆地的变化。从操作系统、数据库等底层基础架构，到分布式系统、大数据、云计算、机器学习等基础领域，从单体应用、MVC、服务化，到微服务化等应用开发模式，从 IaaS、PaaS、CaaS 到 FaaS，运维技术（特别是大规模复杂分布式系统的运维）也变得越来越重要，它已成为 IT 类企业提升生产力的核心。

随着运维受到越来越多的重视，运维体系也逐步丰富，出现了 DevOps 等理念将研发、测试、运维等流程连接起来。而容器技术更是从底层重构了运维，连接了开发、测试、部署、运行和监控全流程，进一步推动了运维体系从工具化逐步往平台化、自动化和智能化方向迁移。本文将对运维技术从底层到顶层做一个彻底的梳理和盘点。

微服务

微服务是近几年提出的概念，它通过将应用解耦成多个服务的方式来改善其模块化程度，使其更容易被理解、开发、测试和部署，更适用于小团队快速迭代式协作开发。同时，每个服务也能够采用不同的技术，便于持续进化。业界前沿互联网公司都构建了微服务框架（例如基于 Spring Boot/Spring Cloud 等开源项目）来应对其业务复杂性以及快速迭代过程中的效率问题。最近，微服务配置管理、容器化部署、自动化测试、微服务治理、微服务监控、安全、故障容忍等领域也受到越来越多的关注。

SRE

SRE（Site Reliability Engineering, 网站可靠性工程）是来自于谷歌的一个最佳实践，它用于服务的容量规划和实施、保障服务的可靠性和性能，更多的在软件基础架构层面构建自动化工具来取代人工操作，从而更好地应对其业务复杂多变的需求。

DevOps & CI/CD

DevOps 逐步成为软件开发的主流，容器也已在过去两年中迅速成长为 DevOps 的核心，在持续集成、持续部署和持续发布等方面也越发受到重视。随着新的 DevOps 自动化工具不断涌现、容器及其相关生态的成熟（特别是容器编排工具及其对有状态服务的支持）、微服务的广泛应用，越来越多的相关工具将会集成在持续集成过程中，同时自动化持续测试也会变得更加流行，从而更有效地控制质量、保障安全、降低成本、控制风险、提升效率，更加高效的支持复杂的大型分布式应用。

容器优化与实践

过去几年间，以 Docker 为核心的容器技术在持续进化，以其构建、分发和部署的简易性成为 IT 基础架构中的关键技术。容器技术通过标准化运行环境的方式来连接了应用的研发、测试和运维。它简单、轻量，具备很强的可移植性，能更高效的利用资源，还能够有效的解决软件依赖问题，提高研发效率，降

低研发成本，因此产业界也持续通过容器来优化其软件发布流程，对已有应用进行容器化。

然而，容器技术本身也面临了不少挑战。未来，在容器标准化、容器安全、容器网络、容器存储特别是对数据库等有状态服务的支持等方面还存在很大的改进空间，容器的可管理性及易用性也需要进一步提升。

容器编排与管理

随着 Docker 等容器技术的广泛应用，容器编排和管理也受到了越来越多的关注，涌现出了诸于 Kubernetes、Apache Mesos、Docker Swarm Mode 等优秀的开源生态和解决方案。它们试图将目前以资源为中心的管理方式过渡到以应用为中心的管理方式，并且试图对应用的基础构成组件（例如配置、服务、负载均衡等）进行标准化，从而获得更好的可管理性。随着 CaaS 的发展，私有或公有的容器云也越来越多，越来越成熟，用户体验越来越好，从而显著降低迁移成本。

然而，在大规模的实践中，在灰度发布、资源调度、隔离性、运维监控、日志等方面仍有待进一步成熟和标准化，在跨数据中心的应用管理，混和云环境支持，跨云服务迁移，安全性等方面仍然面临着困难和挑战。

自动化运维

随着虚拟化和容器化等技术的出现，运维管理的复杂度和难度大大增加，因此必须通过专业化、标准化和流程化的手段来实现运维的自动化。业界出现了很多提升效率的自动化工具，例如 Puppet、Chef、Ansible、Saltstack 等。各大主流互联网公司也逐步从工具自动化往一站式自动化运维管理平台的方向进行演化，从而使得能够对部署、配置、监控、告警等进行一站式处理，实现资源和流程的标准化统一化、应用运行状态可视化管理，提升运维质量，降低运维成本。

智能化运维

随着监控范围的不断扩大，其产生的数据具备多样性、多维性和非结构化等特点，并且可能同业务数据存在相关性，传统的手动分析处理方式效率低且成本高。随着大数据和人工智能的兴起，越来越多的智能分析算法也应用于运维领域，它们通过分析运维系统本身所拥有和产生的海量数据，在问题定位、流量预测、辅助决策、智能报警和自动故障恢复等方面发挥出较大的作用，从而进一步降低运维成本。

运维基础架构

运维基础架构涵盖网络、机器、机房、机架、存储等的管理，涉及基础资源、机架设计和交付、网络架构设计、数据架构规划、操作系统、系统软件、环境交付和机器报废替换等方向。

产业界构建了 CMDB 以支持服务交付流程和相应的管理流程，也都构建了相应的初始化、部署、运行、监控、日志等工具。随着虚拟化、容器化和云计算的发展，运维基础架构也从提供资源往提供能力的方向进行转变，从而提高基础架构对上层应用的透明性，进而提高基础架构的灵活性 。

数据库运维

数据库运维涉及数据库部署架构、容量规划、性能调优、数据备份和恢复、数据迁移、数据库监控审计、数据库运维管理、故障排除等一系列服务。

随着互联网更加广泛的使用，数据库运维也呈现出新的形态。近年来，在异地多活等部署模式、在线表模式变更、海量数据迁移、故障排除时，都会通过一系列的工具，来尽可能的减少数据库整体的不可用时间，从而尽可能的降低对用户的影响。 同时，为了简化数据库的部署和管理，以容器化的方式来对数据库进行管理和调度也逐步成为热点之一。最后，通过对数据库各项指标的分析和挖掘，提供智能化诊断方案，提前预知和管控风险，提升处理效率，提升系统整体稳定性。

大数据运维

随着数据的快速增长，以 Hadoop 为基础的生态系统也扮演了越来越重要的角色，它涵盖离线计算、流式计算、即席查询等多种使用方式，也涌现了 Hadoop、Spark、Kafka、Hbase、Storm、Phoenix 等优秀开源项目。在大数据平台的运维中，由于涉及分布式架构、多源异构海量数据存储、数据的处理框架更为多样化和复杂化等问题，大数据的运维也变得异常复杂。

大数据运维的主要目标是提高资源利用率，降低了大数据系统的运维复杂度，提升用户友好性。其中，计算资源的统一管理和调度能力，以容器为基础的多种类型大数据系统混合部署能力，快速弹性扩缩容能力，跨数据中心容灾能力，大数据应用监控能力和快速灵活的故障定位能力也变得越来越重要。

运维监控

监控是 IT 系统运维中保障核心业务稳定可用的重要环节，它涵盖网络、主机、业务、应用、性能等方面，涉及快速的故障通知，精准的故障定位和性能分析诊断等。当前比较流行并且在业界广泛应用开源的监控软件包括 Nagios、Cacti、Zabbix、Ganglia 等。

随着应用规模的迅速扩大以及 DevOps、微服务、容器等技术的快速发展，监控也出现新的形态。监控方式也已经从类 Nagios 风格演化为流式风格，它基于监控指标对海量数据进行流式处理，同时通过可视化平台来实时展示这些监控指标。另外，随着基础设施变得更加动态，监控不但关心单个节点的运行状态，更关心整个应用的健康状态，全链路追踪等技术也已经出现并得到广泛应用。

运维安全

在互联网化和移动化的背景下，应用逐渐往云中迁移，传统的边界变得越来越模糊，安全也有了新的发展趋势。过去的安全技术是以防御为主，采用传统防火墙、入侵防御系统等。现在，除了对传统的安全措施进行加强之外，还会在开发流程中引入威胁建模，自动安全扫描、安全功能性测试等安全实践，从而降低安全风险，缩短安全问题的反馈周期。同时，安全也从事先预防转向为持续检测和快速响应，通过对攻击行为的持续检测，对安全事件进行快速响应，从而大幅降低损失。

游戏开发与运维

近年来，网络游戏的增长非常迅速，游戏开发采用通用化框架和引擎的趋势越来越明显。在游戏运维方面，除了常规的运维手段，游戏还有其自身的特点。首先，端游、页游和手游由于形式的不同， 其在联网方式、分发渠道、生命周期长短等方面存在差异，因此给网络接入、多渠道分发、容量规划、网络延时、档案数据高可靠存取等方面的运维都带来了挑战。

其次，由于用户增长存在不可预知性，游戏运维必须具备快速的扩缩容能力，多采用混合云或者公有云的技术架构，从而最大程度的提升其水平可扩展性。最后，在受到大规模 DDOS 异常流量攻击时，游戏运维应当具备多级流量清洗保护机制，具备服务降级的能力，从而尽可能的保证可用性。

互联网金融与运维

近几年来，互联网金融出现了井喷式发展，Fintech 也为其注入了技术创新基因。微服务、容器化、大数据和云计算等技术为互联网金融的快速迭代提供了基础。然而，相对于目前的应用运维，互联网金融行业有其自身的特点，其在数据留存、安全合规、防攻击能力、支付清算、金融监管、数据安全、大数据风控和高等级安全防护等方面都有较强需求甚至强制性的金融监管规范，也对互联网金融的运维提出了更高的挑战。

作者简介

刘建，搜狗资深架构师，负责搜狗商业平台的基础研发和平台架构，涉及广告计费、报文等核心业务服务，致力于解决分布式、高并发、大数据量等带来的各种技术难题及挑战，构建和持续优化商业平台基础架构，保证高可靠、高性能、低成本的快速支撑新业务。在多个技术方向有较深刻的理解，有多年大规模复杂系统架构实践经验。

无服务器计算的未来

作者 Mike Roberts　译者 麦克周

现在是 2017 年，距离两年前无服务器计算革新只取得了少许进展（你听见人们在唱歌吗？）。这次变革并不是像 Docker 那样突飞猛进地前进，而是采用了相对平稳的发展节奏。Amazon 新发布了 Web Services 的 Lambda 特性，产品保持了一个有规律的发布节奏，另一个重要的第三方（微软）正在一个接一个地发布生产环境版本，也有一些新的开源项目频繁地加入这场盛宴。

当我们接近早期阶段的末尾时，来做一个有趣的游戏，让我们戴上预测护目镜（开个玩笑），深入思考下一步会走向哪里，如何走到那一步，以及我们的组织需要去做什么来支持这一步的到来。所以，加入我们，让我们看看无服务器计算未来可能发生什么。

让读者了解真实的未来！你可以通过阅读这篇文章开始了解细节。我们离无服务器还有多远？ 2020 年怎么样？请寄一张明信片给我！

无服务器能力展望

计算

过去十年我们目睹了云计算的出现，然后它迅速地异军突起。9 年以前，虚拟公有云服务器还只是手上的玩具而已，但是在相当短的时间里，当我们考虑任何新的部署架构方案时，它变成了大多数行业的首选平台。

无服务器计算，或者 Functions-as-a-Service（Faas），当我们考虑“IT”如何变化时，它会是这次重大变化的最新出现部分。这次变革是一次我们一直以来所期待的自然过程，从我们如何交付应用程序给客户开始，到希望能够删除所有的部件和基础设施。

我们开发的相当一部分应用程序，是由许多小的组件所组成的。每个这样的组件包含了一个小的输入集和上下文信息，需要消耗 10 毫秒或 100 毫秒完成组件的一些内部工作，最终可能反馈一个结果，也可能更新相关内容。这类场景最适合无服务器计算。

我们预测未来由于 FaaS 在部署上的方便、快速、廉价，会有越来越多的团队基于 FaaS 开发，这样便于管理和扩展基础设施。我们对 FaaS 可以有很多种使用方式，包括：

- 由大量的顺序消息处理器所组成的完整的后端数据管道
- 通过 HTTP API 调用的同步服务
- 通过独立的胶水代码提供针对部署、监控的自定义操作逻辑
- 对于计算密集任务，由实体服务器直接调用平台缩放功能组成的混合动力系统

使用 FaaS 的企业相较没有使用的企业具有竞争优势，包括价格、投向市场时间等。

管理应用程序状态

广泛采用 FaaS 的一个先决条件是针对快速而简单的状态管理方法的解决方案，或者一系列解决方案。无服务器计算是一种无状态模式。我们不能假定任何有用的状态存在，即不存在即时执行环境内不同的运行时调用之间的有用状态。一些应用程序在这种限制下依然适用。例如，单纯进行数据转换的消息驱动组件不需要访问外部状态，拥有无限制响应时间需求的 Web Service 组件，其每一次调用时需要连接到一台远程数据库，这种做法也是可以接受的。但是对于其他应用程序来说，这种限制就显得有些不足了。

解决这类不足的一种方案是混合动力系统，即在不同类型的组件里管理状态，而不是执行我们的 FaaS 代码。比较流行的混合动力方案是通过云端基础设施提供其他服务的前端 FaaS 功能。我们已经见到了这种类似于 API 网管的特定上下文逻辑的组件，它们提供了 HTTP 路由、授权，以及我们经常在典型的 Web Service 里看到的节流逻辑，采用定义替换配置方式实现。Amazon 最近也在管理状态的通用方式上露了一手，使用他们的 Step Functions 服务，允许团队基于可配置的状态机器定义应用程序。Step Funcations 服务本身可能没什么过人之处，但是通常情况下这种无码解决方案是很受欢迎的。

当供应商服务不充足时，对于混合动力系统来说，一种改变方式是团队坚持开发追踪状态的长生命周期组件。这些组件可能被部署在 CaaS（容器即组件，Containers-as-a-Service）或者 PaaS（平台即组件，Platform-as-a-Service）环境，和 FaaS 功能协同工作。

这种混合动力系统组合了长期运行的组件逻辑和每一次 FaaS 功能请求逻辑。另一类做法是完全地聚焦于 FaaS 功能，让这些 FaaS 功能超越它们当前执行的环境，极其快速地获取和持久化状态。一种可能的实施方式是确保一个特定的 FaaS 功能，或者一系列 FaaS 功能，确保它们拥有类似于 Redis 这样的外部缓存机制，起到低延时访问作用。通过启动类似于 Amazon 的 same-zone plancement groups（置放组群）这样的特性可以做到这一点。虽然这种解决方案较内存 / 本地磁盘状态方案来说有些延迟，但是很多应用程序会认同这种解决方案。

混合方法的好处是经常被访问的状态可以和逻辑一起保存在环境当中，那样并不复杂，但是可能有点贵，必须要有逻辑网络选址和外部状态。另一方面，一个单纯的 FaaS 方式的有点是更加一致性的编程模型，外加无服务器带来的更为宽广的伸缩使用和可操作性优势。当前的发展势头显示最终混合方式会胜出，但是我们也应该对其他方式开放，比如类似于启用置放群组 Lambdas。

无服务器协作服务

超越业务管理和状态管理，我们可以预见到其他组件的服务化和商业化，即使在云端环境，传统意义上我们希望开发，或者至少自己管理这些服务。例如我们可以停止运行在 EC2 机器上面的 Mysql 数据库服务器，转而使用 Amazon 的 RDS 服务器，我们可以使用 Kinesis 替换我们自管理的 Kafka 消息总线安装程序。其他的基础设施服务包括文件系统和数据仓库，而更多的面向应用示例包括认证和语音分析。

这种趋势还会继续，我们需要进一步地减少创建或者维护产品所带来的工作量。我们可以设想更多的预安装的消息逻辑（把 Apache Camel 想象成服务，构建到 Amazon Kinesis 或者 SQS 里面），并且进一步开发通用机器学习服务。

这里比较有意思的一个想法是 FaaS 功能，由于它们轻量级的应用模式，可以将自己紧紧地绑定一个服务，使得 FaaS 调用服务功能的生态环境时可以调用其他的 FaaS 功能，诸如此类等等。这会导致“有趣的”级联错误问题，对于这种错误我们需要更强大的监控工具，会在本文稍后介绍。

站在数据中心后面

目前来看，绝大多数的无服务器计算是运行在供应商数据中心平台上的。这就给出了一个替代方案，即如何运行你的代码，而不是在哪里运行代码。Amazon 发布了有趣的新特性，即是允许它们的客户在不同的地点运行 Lambda 函数，例如，和 Lamdba@Edge 一起运行在 CDN 内，甚至在无服务器地点，例如，和 Greengrass 一起运行的物联网（IoT）设备。这样做的原因是，Lamdba 是一个极端轻量级的编程模型，本质上的事件驱动的，并且非常容易适配相同的知识理念、新地点的代码风格。Lambda@Edge 是个特别有趣的例子，因为它提供了在一个地点进行程序定制

的可选项，这在以前是没有出现过的情况。

当然，这种做法的缺点是和供应商深度绑定！对于那些不想使用第三方平台，但是又想利用无服务器计算优势的厂商来说，有一种可以接受的解决方案，类似于 Cloud Foundry 已经推出的 PaaS。来自 Kubernetes 的 Galactic Fog、IronFunctions 以及 Fission，是这种方案的早期作品。

我们将来需要的工具和技术

正如我之前描述的，这里有一个明显的减速，使用无服务器方式时存在条件限制、性价比权衡。天下没有免费的午餐。对于已经过了早期适应阶段的无服务器用户来说，我们需要解决或者缓解这些问题。所幸，这方面目前发展势头良好。

部署工具

使用 AWS 的标准工具向 Lambda 部署函数挺复杂的，也比较容易出错。向 API 网关中添加 Lambda 函数，以响应 HTTP 请求，你更多要做的工作是安装和配置。无服务器和 ClaudiaJS 开源项目项目已经推动部署改进措施达一年之久，AWSSAM（AWS 无服务器应用模型）也在 2016 年加入到了这一行动。所有这些项目通过在 AWS 标准工具的顶层增加大量自动化程序，简单化了无服务器应用程序的创建、配置和部署。但是我们还有很多工作要做。未来将会有两个关键动作实现完全自动化：

1. 初始化一个应用程序或者环境的创建（例如，初始化生产环境，以及初始化临时测试环境）；
2. 持续多部件应用程序的交付 / 部署。

第一条很重要，我们也已经开始认识到，以便更广泛地推广“生产提前期概念”。部署一个全新的无服务器应用程序应该是像创建一个新的 Github 仓库一样容易，填充少量字段，然后按下按钮，通过这种一键部署方式让系统自动创建你所需要的所有东西。

然而，光有简便的初始化部署方式是不够的。我们也需要有比较好的工具，支撑前面提到的混合动力系统的持续交付和持续部署。这意味着我们应该可以部署一系列的计算函数以及 CaaS/PaaS 组件，连同所有应用程序封装服务的变化（例如，在一个 API 网关配置 http 路由，或者一个被单一应用程序使用的 Dynamo 表），一键生效和回滚能力。此外，这些动作都不应该是很费脑力去理解的，也不会需要几天时间去完成安装和维护任务。

这是一个很艰难的抉择，但是我前面提到的工具（类似 Terraform 这样的混合动力工具）正在指引解决这些问题的方式，我完全相信他们在未来的几个月或者几年时间里可以在很大程度上解决问题。

本文不仅仅讨论部署代码和配置服务。其他一些操作上关心的问题也会被讨论。安全问题是一大问题。当前，获取 AWS 凭证、角色，以及设置和维护都可能是一大麻烦事。AWS 拥有一套完善的安全模型，但是我们需要一个工具，这个工具可以让这套安全模型对于开发人员来说更加友好。

总之，我们需要开发人员在开发他们的 Webtask 产品时，做到 UX 和 Auth0 都很好，就像 AWS 一样的宽广而有价值的生态系统。

监控、日志和调试

一旦我们的应用程序被部署完毕，我们就会需要针对监控和日志的良好的解决方案，这类工具目前有几个组织正在尝试积极地发展着。除了评估其中一个组件的功能，我们也需要号的工具追踪请求，这些请求穿越了一个完整的多个无服务器计算功能和配套服务体系的分布式系统。Amazon 正在将 X-Ray 推向该领域，目前说这个还有点为时尚早。

调试也是挺重要的。程序员很少在第一次代码运行通过之前不犯错误，我们也别寄希望于这种情况会有所改变。我们依赖于监控，在 FaaS 功能的开发阶段评估问题，但是这种调试方式是石器时代的工具。

当我们调试传统的应用程序时，我们从 IDE 工具那里可以得到很大的支持，通过设置断点、单步调试代码，等等。使用现代化的基于 Java 的 IDE 工具，你可以绑定一个正在运行的远程进程，并且远程执行调试工作。因为我们更加倾向于使用云端部署的 FaaS 功能完成大量的部署工作，希望未来你的 IDE 工具也可以具有类似的功能，可以连接到一台正在运行的无服务器平台，查询每个功能的执行情况。这需要工具和平台开发商之间的协作，如果想要让无服务器被广泛采用，这些措施都是必要的。这些想法对于云计算来说有一定开发工作量，也有大量的测试工作量。

测试

我到目前为止所讨论的所有关于无服务器工具的话题，我认为最落后的是测试工具。值得关注的是，无服务器方案较传统解决方案来说有着相当大的测试优势，主要是两点，（a）.无服务器计算的各个功能的单元测试很成熟，(b).无服务器服务写的代码更少，并且至少在单元测试层面，只需要做简单的测试。

但是这并没有解决跨组件功能 / 集成 / 验收 / 业务流程等测试问题。无服务器计算时我们的逻辑是分散在几个函数和服务内的，因此，更高级别的测试甚至比使用接近单一方法的组件更重要。当我们如此依赖于在云端基础设施上运行时，我们应该怎么做呢?

对于我们来说，测试可能是最没有看清楚的。我猜测未来基于云端的测试会变得很普遍。这一部分会变得更加容易部署、监控，以及调试我们的无服务器 apps，甚至于比我现在描述的这些原因更加丰富。

换句话说，为了运行更高级别的测试，我们将会部署整个生态系统的一部分到云端，并且对部署在那里的组件执行测试用例，而不是针对部署在我们自己开发机器上的系统运行测试用例。这种做法有一定的优势。

- 执行部署在云端的组件的真实度较本地模拟来说更高。
- 我们较过去，更有可能可以运行高负载 / 高丰富度数据测试。
- 生产环境数据源的测试组件（例如，一个发布订阅模式的消息总线，或者一个数据库）会更加容易，虽然显而易见我们需要关注能力 / 安全问题。

但是这种解决方案也有弱点。首先，执行测试的周期时间很有可能由于部署和网络延迟而相应增加。其次，当网络连接中断以后，我们就不可以继续运行测试用例了（例如，在飞机上）。最后，因为生产环境和测试环境最终部署方案很相近，我们也需要格外小心，当我们打算改变测试用例时，不要发生不小心改变了生产环境的事故。如果我们使用 AWS，我们可能需要通过类似于 IAM 角色这样的工具安全地部署，或者对于不同类型的环境使用完全不同的账号部署。

测试并不仅仅是一个二进制程序运行成功或者失败，我们也想要去弄清楚测试是如何失败的。我们应该可以调试本地运行测试和正在运行的远端组件，包括可以单步调试一个运行在 AWS 上的 Lambda 函数，因为它可以相应测试。所以所有的远端调试，例如，我前面章节提到的工具也需要测试，而不是仅仅交互式开发。

请注意，我并不是基于这些暗示我们的开发工具需要运行在云端，也不是测试本身需要运行在云端，虽然两者将来都会或多或少地走到这一步。我只是表示正在测试的系统仅运行在云端，而不是一个非云端环境。

使用无服务器作为测试驱动环境可以收获有用的结果。一个例子被称为“无服务器火炮”，这是一种负载测试工具，由运行着的许多并行的 AWS Lamdbas 组成，执行即时、廉价、易于扩展性能测试规模的负载测试用例。

值得指出的是，在某种程度上，我们避免了一些失误。由于技术进步，传统的高层及测试实际上正在变得不那么重要，例如（a）生产环境测试 / 使用监控驱动开发，(b) 平均解决时间 (MTTR) 的显著降低，（c）基于持续部署。对于许多的无服务器 apps 应用广泛的单元测试，度量业务水平的生产环境监控 & 预警，以及一个专用于减少 MTTR 和基于持续开发的方法，都将会是有效的代码验证策略。

架构：有很多问题需要回答

系统架构较好的无服务器应用程序是怎样的？是如何演变的?

我们正在逐渐看到一些无服务器被有效地应用的案例，即系统架构的学习案例正在逐渐增多，但是我们还没有看到针对无服务器 Apps 的“模式组”。在 2000 年早些时候，我们看到了一些这方面的书，比 如 Fowler 的《Patterns Of Enterprise Application Architecture》，以及 Hohpe / Woolf 的《Enterprise Integration Patterns》。这些书着眼于很多项目，派生出横贯不同领域的通用系统架构知识。

重要的是，在做出统一意见之前，这些书着眼于基础工具几年的使用经验。无服务器技术存在时间太短，还不足以需要编写一本书进行描述，但是这一时刻正在逼近，一年内我们会看到一些有用的实践案例

出现（当无服务器架构需要出一本高调的书时，大家一般会选用“最佳实践”这样的术语描述）。

系统架构之后（即无服务器应用程序是如何被构建的），我们需要思考部署系统架构（无服务器应用程序如何部署）。我已经谈了一些部署工具，但是我们可以如何使用这些工具呢？例如：

- 环境这样的术语在世界上意味着什么？“生产”看上去较过去有点不明确。
- 什么是一个软件栈的 side-by-side 部署？看上去像是从一组功能 / 服务版本缓慢地移动业务到另一组功能 / 服务版本（滚动部署）？
- 世界上有么有类似于“蓝 - 绿”这样的部署方式？
- 现在的回滚方式是怎样的？
- 我们如何管理数据库的升级 / 回滚，当我们可能有多个不同的代码生产版本，并且这些版本在同一个功能内运行，这类有状态组件应该如何管理？
- 当使用第三方服务时，如果你不能够完全下线服务或者重新完整地部署，那么一台 phoenix-server 看起来更像什么？

最后，当我们从一种系统架构样式迁移到其他架构，什么迁移模式是比较有效的？或者是否包括无服务器组件？我们的架构以怎样的方式进化？

许多这些尚未定义的模式（反模式）都不是很明显的，通过我们幼稚的想法明显表现出来的是，如何最好地管理无服务器系统内的状态。毫无疑问，有一些神奇的模式出现了。

我们的组织将会如何变化

成本效益是无服务器前进的一项驱动，最有意思的优势是“生产提前期概念”的降低。通过提供“超级能力”方式，无服务器为大多数既不是系统管理专家，也不是分布式系统开发专家的美国工程师提供了进入无服务器领域的可行性。这些只有一点点技术的应用程序开发工程师，不再需要编写一行 Shell 脚本，即可完成整套 MVP（即 Minimum Viable Product，最小可行性产品）的部署，扩展平台能力，或者配置一个 nginx 服务器。前文中我提到了配置工具还在开发当中，我们现在还没有这类“简单的 MVP”解决方案，能够解决所有类型的应用程序问题。但是，我们确实看到了相对于简单的 Web Services 服务，甚至为其他类型的应用程序部署一些 Lambda 函数，也比管理操作系统进程或者容器来得更容易。

除了 MVP 以外，我们也看到了重新部署应用程序的周期时间正在缩短，不再需要关心脚本维护、系统补丁级别，等等。

无服务器为我们提供了技术手段去实现这些需求，但是还不足以真正实现对于一个组织的改进。为了实现这些目标，公司需要去克服、适应以下这些变化。

真正的DevOps

DevOps 已经在很多领域都变得很重要了。在开发工作上，额外技术的技术操作越来越常见。我所看见的是系统管理内部的自动化增加和自动化测试，这只是 Patrick Debois 在创造 DevOps 概念时所想到的很小一部分。

真正的 DevOps 是我们思维方式上的变化，以及文化上的变化。让我们假设有这么一个团队，这个团队需要紧密合作、开发和维护一个产品。这就意味着写作，而不是基

于协商的工作序列方式。也意味着开发人员需要提供技术支持。而意味着开发工程师需要参与应用系统架构。换句话说，意味着技能与责任的融合。

如果一个公司分离了开发团队和运维团队，即将“DevOps”团队分离，那么他们不会在无服务器领域有任何收获。如果一个开发人员仅仅只是对应用程序进行编码，而部署工作又交给另一个外部团队负责，那就会没有真正意义上的系统部署情况反馈。如果一个业务工程师不会到应用程序的部署环节，那么他们也不可能适应生产环境的部署模型。

换句话说，未来会从无服务器领域收获实际收益的公司，必然是真正使用 DevOps 的公司。

政策/访问控制的变化

即便一个组一个组地尝试改变文化，也是做得不够的。很多时候，一个大公司里的一个很有工作热情的团队，往往面对的是冷冰冰的公司政策。这可能意

味着在缺乏外部批准的情况下，缺少部署新系统的能力。很有可能是由于对于所有现有应用程序的数据访问限制。也可能是因为超级严格的支出控制。

虽然我不提倡公司把所有与安全和成本相关的问题抛到外部解决，但是为了尽可能做到无服务器化，需要调整他们的政策，允许团队对操作请求作出改变，而不是每一次小的更新操作都需要一个团队外部人员的批准。访问控制政策目前还不是很有必要构建。团队需要被给予一定范围内的预算自由。所有的实验应该被尽可能多地提供免费的沙盒，同事还可以保护公司内部真正敏感的数据或其他需求。

通过我之前提到过的 IAM 规则和多个 AWS 账户的使用，访问控制工具正在逐渐完善。然而，不是那么简单的，针对更好的自动化方式正在成熟。同样，无服务器还存在通过几个账户实现基本预算控制，我们需要更容易控制每个团队执行能力限制，对于不同的环境有不同的执行限制范围。

好消息是通过加强权限控制工具，所有这些问题都有可能解决，我们会看到 y 预算分配模式上的进步，等等，因为无服务器工具在持续改进。事实上，我认为访问自动化和成本控制将会变成新的 shell 脚本，换句话说，当团队思考 suanfa 软件的操作问题时，他们不会想要去开始 / 停止脚本、升级补丁以及磁盘使用率，反而他们会严谨地思考他们需要怎样的数据访问方式，以及需要怎样的预算。因为团队将会经常需要思考这个问题，工程师们会用自动化取代这些问题，仅仅像我们之前做部署那样。

鉴于这种能力和严谨性，未来即便是数据最敏感的企业，也会有富有热情的团队会使用无服务器技术，使用它们去尝试自己的想法，这种做法是之前在白板上从未做过的，最终他们会认识到这种做法真正意义上保护了他们的知识或者避免财务损失。

产品所有权

过去几年时间里我们看到的另一个转变是许多高效的工程团队的聚焦正在从项目专项产品。这一转变的感觉是对于项目规划、迭代和燃尽图等的关注在降低，转而更加关注看板方式的进展、轻量级预估以及持续交付。比这一结构性改变更重要的是虽然角色和心态在转变，转变为更多的职责较差，同样我们看到真正的 DevOps。

举个例子，现在很有可能产品经理和开发人员将会密切地充实新思路，开发人员会做一些原型，产品经理在最终产品设计方案明确之前，会深入进行一些技术上的数据分析。相似地，创新的火花，即新的想法或者概念也会进入某人的大脑，可能属于团队中的任何一个人。这个团队的许多成员，不仅仅是一个，现在正在接触到客户喜欢的想法。

无服务器方法为这些团队提供了一个关键好处，即接受整个团队产品思维。当团队中的任何一个人都可以想出一个点子，并且迅速地针对一种尽可能新的创新模式实现一个原型。现在精益启动式试验变成默认的思维方式，而不是由“黑客时代”保留的那样，因为这样做的成本和时间正在大幅缩减。

另一种看待这一问题的方法是，不接受整个团队产品思维的团队很有可能错误这一关键利益。如果团队不鼓励超越项目结构的思考方式，他们就很难尽可能多地使用无服务器所带来的加速交付可能性。

结论

无服务器在软件架构领域相对来说是一个新的概念，但是它也是一个可能和其他云计算创新一样，具有巨大影响力的技术创新。随着技术的发展、工具提升以及无服务器应用架构方面的心得交流，越来越多的工程团队将会拥有提升开发速度的工具，甚至于可能转变他们产品开发方式。适应无服务器，并且适应支撑该技术的文化，这类公司将会在未来领导我们前进。

作者简介

Mike Roberts 是 Symphonia 公司的合伙人，同时也负责公司的工程团队，该公司提供关于无服务器和云计算技术的咨询。Mike 是敏捷开发和 DevOps 价值的长期支持者，并且认为云计算技术已经让许多高级软件开发团队实现了这两个技术的价值。

阿里 Goldeneye 四个环节落地智能监控：预测、检测、报警及定位

作者 马小鹏

背景介绍

这个分享主要包括智能监控的技术实现，以及大规模日志监测数据的自动化接入两部分。我先介绍一下智能监控部分，下一期分享中我的两位同事将给大家着重介绍日志分析处理的计算存储。智能监控现在其他一些公司也有在做，希望通过这次分享能够给大家带来一些新的启发，也欢迎大家能够提出问题和建议，互相切磋交流经验。

一、Goldeneye智能监控的背景

Goldeneye 作为阿里妈妈业务监控平台，主要在业务日志、数据的实时统计分析基础上做监控报警以及辅助定位。阿里集团内部也有很多优秀的监控平台，它们在开放性上做的很好，接入成本也不高，但是监控阈值也是开放给用户自己设定。这种情况下，对于业务监控人工维护阈值就比较复杂，需要有丰富的经验来拍定阈值，需要人工持续的维护不同监控项的监控阈值。所以，在业务快速发展的前提下，传统的静态阈值监控很容易出现了误报、漏报的问题，而且人工维护成本高，监控视野局限。Goldeneye 就是在这种基础上，我们试着从大数据应用的角度，去解决业务监控中的问题，由此诞生的。

1. 业务背景

（1）体量大：Goldeneye 现在接入的业务线覆盖了阿里妈妈主体的 90% 业务，每天处理的日志量在 100T 以上，业务监控需要对各业务线的流量分层级实时监控，核心数据以 1 分钟为周期，一般监测数据以 5 分钟或 1 小时为周期，监控目标非常多，按人工维护这些监控的阈值、启停、生效实效等几乎是达不到的。

（2）变化多：业务监控的监测数据大都是业务指标，不同于系统运维指标，比如 RT/QPS/TPS 等一般是比较稳定的，业务指标具有周期性变化的特点，比如工作日和节假日的区别、业务营销策略调整的影响等，在这种情况下人工设定的静态报警阈值准确性就很难保障了。

（3）迭代快：随着阿里妈妈资源整合和业务的快速发展，监控目标也经常发生变化，比如流量监控资源位的调整、效果监控的产品类型划分等，曾经出现过新流量上线后的监控盲点。

2. 技术背景

通常的业务监控系统或平台，都是由采集、数据处理、检测、报警等模块组成的，Goldeneye 也是如此，不过它的技术架构上用了阿里内部的一些技术中间件，比如采集我们使用 TimeTunnel（它有 agent 在各台日志服务器上拉日志到 Topic，并且负责将离线日志放到 ODPS 上），这部分我不再介绍了。

数据处理我们使用的 jstorm 和 ODPS MR job 分别对日志进行实时、离线批处理，主要包括日志解析、

校验、时间周期归一化、聚合、写存储（HBase）等操作，这部分下一期分享中我的同事会详细介绍。今天的分享主要集中在阈值预测、监控检测、报警生成 & 通知、辅助定位这四部分。

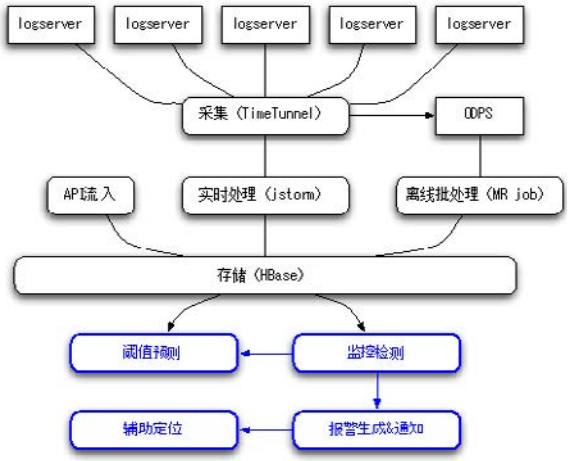


图1 Goldeneye技术背景二、技术思想

二、技术思想

智能监控就是让系统在业务监控的某些环节上代替人工执行和判断的过程。人工维护监控目标和阈值是以经验为参考的，系统如何自动判断哪些目标需要监控、自动设定监控目标的阈值水位、不用人力维护，是基于对历史样本数据统计分析得出判断依据。

通过收集监测数据的样本，并使用智能检测算法模型，让程序自动对监控项指标的基准值、阈值做预测，在检测判断异常报警时使用规则组合和均值漂移算法，能精确地判断需要报警的异常点和变点。

1. 阈值水位自适应变化

以往我们添加监控有两种做法：

- 给指标 M1 设置一个水位线，低于（或高于）水位，触发报警；
- 给指标 M1 设置同比、环比波动幅度，比如同比波动 20%、环比波动 10% 触发报警。

以上两种方式，是平常大家常用的监控方式，但是效果确不理想，这种静态阈值长期来看没有适应变化的能力，需要人工维护，而且报警准确性也依赖于同环比数据的稳定性。

我们能否让系统具备自动适应变化的能力，自动调整阈值水位？就如同手动挡的汽车换成自动挡一样，可以根据速度自己调节档位。

2. 监控项自动发现

当我们的监控系统具备预测动态阈值的能力后，监控项的维护是否也可以交给系统去做？

可能大家也曾遇到过类似的情况，旧的监控项已经没有数据了，新的监控目标却因为各种原因被漏掉，人工维护监控项需要及时同步上下线变更，但是当我们需要监控的目标有一千个、一万个甚至更多的时候，人力是无法一直跟进这些监控项的维护工作的，或者说这种工作比较单调容易被忽视。

我们能否将判断如何筛选监控项的规则交给系统，让它去定期检查哪些监控项已经实效，哪些监控项需要新增，哪些监控项的阈值需要调节。这种发现规则是稳定的，仅仅是依据发现规则得出的监控项内容在不断变化而已。

3. 过滤误报时欲擒故纵

当我们的监控系统具备预测动态阈值、自动发现并维护监控项的能力后，如何达到不漏报和不误报之间的平衡？

对于监控而言，漏报是不可容忍的，但是误报过多也容易使人麻木。

通常的做法是为了不被误报干扰至麻木，会把阈值调节得宽松些，但是这种做法容易产生漏报，尤其是下跌不太明显的情况。

Goldeneye 采取的思路是对误报 case 欲擒故纵，在首先确保不漏报的基础上降低误报率。先监控产生疑似异常点，这一环节我们基于动态阈值去检测时相对严格一些（或者说这一环节不用考虑报警收敛的问题），然后对这些疑似异常点再做验证、过滤，最终生成报警通知，验证和过滤的依据是预先定义的规则，比如指标组合判断、报警收敛表达式等。

三、技术实现细节

下面介绍技术实现的一些细节，分为监控系统的架构、动态阈值、变点检测、智能全景、辅助定位五个点。

1、整体介绍

Goldeneye 监控系统的四个输入：实时监测数据、历史数据、预测策略、报警过滤规则。

其中，历史数据是实时监测数据的积累。

而预测策略主要包括：

（1）阈值参数：设置基于预测基准值的系数决定阈值上下限区间、分时段阈值预测系数、分报警灵敏度阈值预测系数；

（2）预测参数：样本数量、异常样本过滤的高斯函数水位或者过滤比例、基于均值漂移模型的样本分段选取置信度等。

关于报警过滤规则，主要是为了在充分捕捉疑似异常点的前提下，过滤不必要的报警。比如指标 M1 异常，但是组合规则是 M1 和 M2 同时异常才报警，这种就会过滤掉。再比如，按照报警收敛规则，一个监控项的第 1 次，第 2 次，第 10 次，第 50 次连续报警值得关注，可以设置收敛表达式为 1,2,10,50，那么在报警通知生成时对于第 3,4,⋯,9,11,12,⋯,49 次报警可以忽略，因为反复通知的意义不大，这个规则可以按需要达到自动收敛。也可以在同一监控项的多个实例同时发生异常报警的情况下，按规则合并成一条报警，这些规则可以按具体情况去实现，最终的目的是以最简洁的方式暴露最值得关注的报警。这里补充一句，我们最近在考虑新的收敛方式，对第 1 条和最后 1 条报警，并且自动计算出累积 gap，这样异常的起止和影响范围更明显。

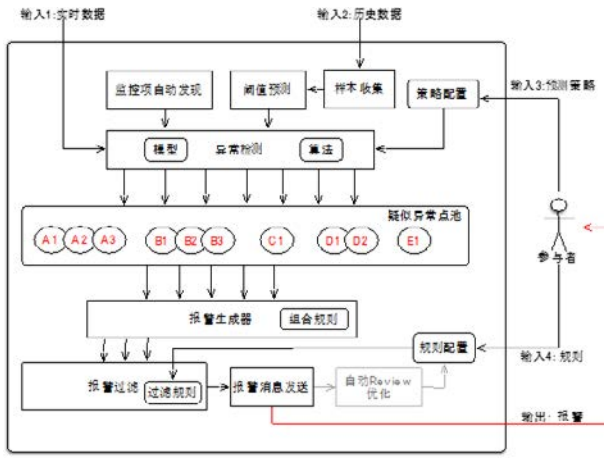


图2 Goldeneye报警系统架构

2、动态阈值

监控使用控制图，对监测指标的时间序列可视化，让人们可以清楚的看到指标的波动。基于控制图的监控，以往很多都是静态阈值方式，比如前面提到的静态水位线、同环比。动态阈值是为控制图的时间序列每个点，预估该点对应时刻这个指标的基准值、阈值上限、阈值下限，从而让程序可以自动判断是否有异常。因为这种预估基于过去几个月甚至更多的历史样本作为参考，所以比同环比两个数据作为参照的准确度要高。动态阈值预测的理论基础是高斯分布和均值漂移模型。

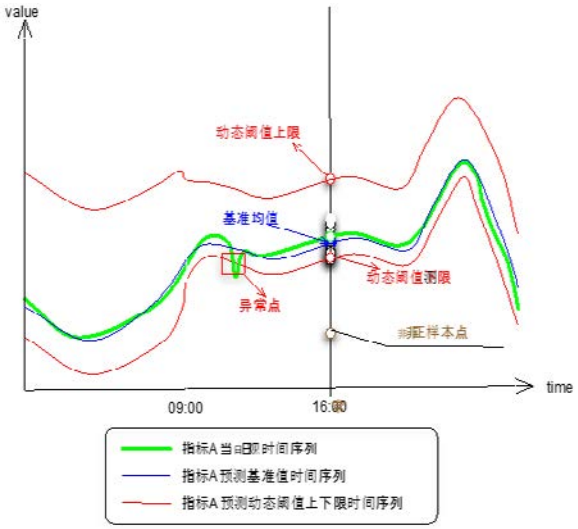


图3 动态阈值原理

动态阈值预测的步骤如下所述。

（1）样本选取：这个根据自己的需要，一般建议选取过去 50 天左右的样本。

（2）异常样本筛除：这个过程主要使用高斯分布函数过滤掉函数值小于 0.01，或者标准方差绝对值大于 1 的样本。

（3）样本截取：因为后来我们优化的版本，在（2）的基础上使用均值漂移模型对历史样本在时间序列上进行分段检验，如果有周期性变化、或者持续单调变化，则会反复迭代均值漂移模型寻找均值漂移点，然后截取离当前日期最近第一段（或者可以理解为最近一段时间最平稳的样本序列）。样本选取还有一个需要注意的问题，节假日和工作日的样本要分开选取，预测工作日的阈值要选择工作日的样本，节假日亦然，

也就是对预测样本从日期、周末、平稳性三个维度拆分选取。

（4）预测基准值：经过（2）和（3）的筛选、截取，剩下的样本基本上是最理想的样本了，在此基础上，保持样本在日期上的顺序，按指数平滑法预测目标日期的基准值，得到基准值以后根据灵敏度或阈值系数，计算阈值上下限。

补充说明：第四步预测基准值，有些人可能之前用过指数平滑法预测，跟第四步我们在样本权重加权时的做法很相近，但是他们预测的效果不理想，因为对样本整体没有充分的过滤选取最稳定的样本集合。

3、变点检测

动态阈值用数据统计分析的办法解决了静态阈值的误报漏报问题，节省了人工维护的成本，一定程度上降低了监控风险。不过在微量波动、持续阴跌的故障面前，动态阈值也有局限性，阈值区间收的太紧误报会增多，区间宽松就会漏报一些不太显著的故障。在 review 漏报 case 时，我们从控制图上发现这些微量波动肉眼可以观察到趋势，但是程序通过阈值区间击穿的判断方式很难控制，所以引入了均值漂移模型来寻找变点。所谓变点，就是持续微量下跌到一定时间，累积变化量到一定程度后，使得变点前后监测指标在一段时间内的均值发生漂移。

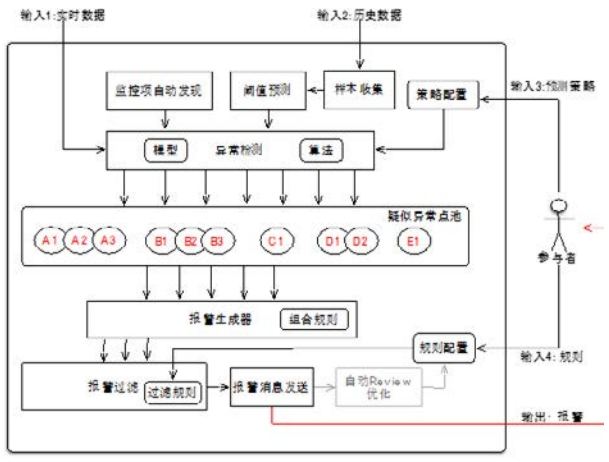


图4 均值漂移原理

从图 4 可以看到，均值漂移模型的算法原理，实际上是把程序不容易识别的阴跌趋势，转换成 CUSUM 时间序列，它的趋势很明显，在变点左侧单调增、右侧单调减，CUSUM 时间序列描述了被监测时间序列每个点偏离均值的累积变化量，它的规律是从 S0=0 开始，到 Sn=0 结束，变点两侧单调变化。

CUSUM = Cumulative Sum。累积和用以在某个相对稳定的数据序列中，检测出开始发生异常的数据点。累积和最典型的应用是在“改变检测”(Change Detection) 中对参量变化的检测问题转化了以后，用程序求 CUSUM 序列上每个点的一阶导数，从持续增变为持续减即可判定为变点，至于持续增、减多少个点，由自己来设定。

关于变点检测使用的 mean-shift 模型，大家可以去网上找找 paper，我这台电脑上找不到了，上面主要说明了发现变点的原理，通俗地讲，就是把问题转化成程序容易解决的状态阴跌线程序不容易量化衡量、判断，那么就用 CUSUM 控制图里的“富士山”形状去寻找，这是我个人的通俗解释。

上面说到我们使用 CUSUM 序列上每个点的一阶导数来判断拐点（变点）是否到来，其实图上这个例子是比较理想的情况，在我应用 mean-shift 模型时，遇到了一些复杂情况，比如这个图上就一个“山头尖”，但是也时候会有多个，这种情况下就要再次转化问题，比如可以把 CUSUM 再差分，或者以我们的做法，记录一阶导数的状态值，从连续 N 个正值变为持续 N 个负值时可以判定。

另外，变点检测的算法实现我这里不方便详细说明，其中变点在反复迭代时自己可以根据实际情况设定迭代次数和置信度，有助于提高变点发现的准确性。

4、智能全景

变点检测弥补了动态阈值对细微波动的检测不足，这两种方式结合起来，基本可以达到不漏报和不误报的平衡，同时也不需要人工长期维护，这是智能全景监控的基础。当监控的人力成本节省了以后，理论上我们可以依赖智能监控无限制的开拓监控视野，并将这些监控报警连结起来分析。

监控项的自动发现规则，比如对维度 D 的指标 M 做实时监控，维度 D 下可能由 1000 种维度值，而且是不断变化的 1000 种，如何让程序自动维护监控项？你可以制定一个规则，比如指标 M>X 则认为需要监控（毕竟不是所有的都需要监控报警，至少在目前故

障定位处理没有完全自动化的状况下，报警处理也是需要一定人力的）。在满足 M>X 的条件下，为了提高报警准确性，我们还需要根据重要性区分报警灵敏度，也就是对于宏观、核心的维度值我们希望能够非常灵敏的监控波动，而对于非重要的维度值我们预测阈值可以宽松一些，这些可以通过上面说的阈值参数来设定。

以上条件都满足了之后，智能全景监控基本可以运行，不过我们也曾遇到一些其他的问题，比如业务方需要接入监控，但是不一定是必须要我们解析日志，他们有自己的数据，可能是数据库、接口返回、消息中间件里的消息等等。所以，我们在数据接入上采用分层接入，可以从日志标准输出格式、存储的时间序列 schema 约定、阈值预测的接口三个层次接入使用，这个内容将在下一次分享时由我的同事单独介绍。这里之所以提到，是因为全景监控接入的数据比较多，所以接入途径要有层次、灵活性。

5、辅助定位

报警的最终目的是减少损失，所以定位问题原因尤为重要。Goldeneye 尝试着用程序去执行人工定位原因时的套路，当然这些套路目前是通过配置生成的，还没有达到机器学习得出来的地步，不过当业务监控指标接入的越来越多，指标体系逐渐完善以后，通过统计学的相关性分析，这些套路的生成也有可能让程序去完成。这里我介绍一下，程序可以执行的人工总结处的几个套路。

（1）全链路分析

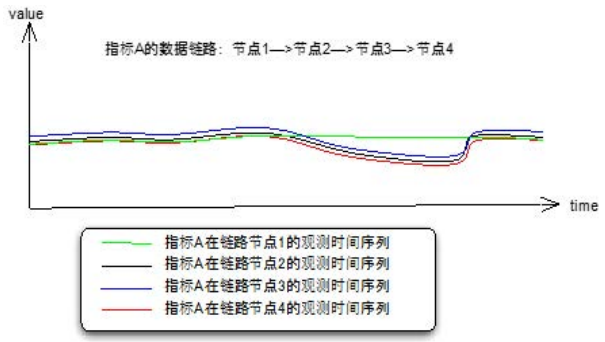


图5 全链路tracing

从技术架构、业务流程的角度，我们的监测指标是否正常，从外部因素分析，一般会受到它的上游影响。按照这个思路，逐一分析上游是否正常，就形成了一条链路。这种例子很多，比如系统架构的模块 A,B,C,D,E 的 QPS。

（2）报警时间点上发生了什么？

这是收到监控报警后大多数人的反应，我们把运维事件、运营调整事件尽可能地收集起来，将这些事件地散点图和监测报警的控制图结合起来，就能看出问题。如果程序自动完成，就是将事件发生的时间点也按相同的方式归一化到固定周期的时间点，检查与报警时间点是否吻合。

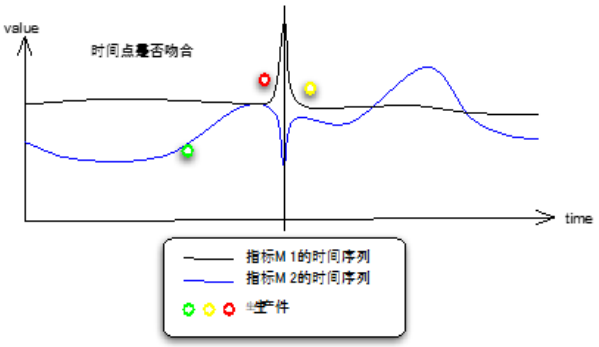


图6 生产事件与时间序列

（3）A/B test 或 TopN

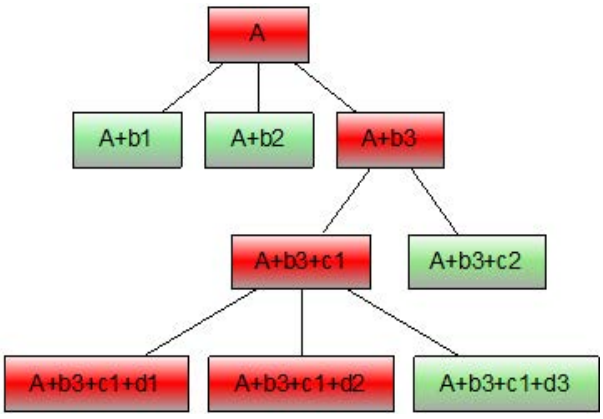


图7 A/B test or TopN

有些人定位问题，使用排除法缩小出问题的范围。比如在维度 D 上指标 M 有异常波动，可以将 D 拆分成 D1，D2，D3 来对比，常见的具体情况比如机房对照、分组对照、版本对照、终端类型对照等等，如果在监测数据层级清晰的基础上，我们可以一层一层的钻取数据做 A/B test，直到定位到具体原因。还有一种方式，

不是通过枚举切分做 A/B test，而是直接以指标 M 为目标，列出维度 D 的子维度 D1，D2，D3,……中指标 M 的 TopN，找出最突出的几项重点排查。

topn 也是类似的。大家可以也能看出来，智能监控和辅助定位是需要一个清晰的数据层级和元数据管理系统来支撑的，这一点很基础。

（4）关联指标

不同的指标在监控中都是持续的时间序列，有些指标之间是函数关系，比如 ctr=click/pv，click 和 pv 的变化必然带来 ctr 的变化，这种联系是函数直接描述的。还有一些指标的关联，无法用函数公式描述，它们之间的相关性用统计学指标来衡量，比如皮尔逊系数。Goldeneye 的指标关联依据，目前还没有自动分析，暂时是人工根据经验设置的，只是视图让程序去完成追踪定位的过程，比如指标 M1 出现异常报警后能够触发相关指标 RMG1/RMG2/RMG3 的检测（因为这些指标可能平时不需要 7*24 小时监控报警，仅在需要的时候 check），以此类推逐级检测定位。

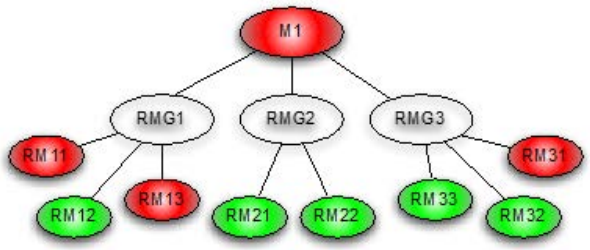


图8 关联指标

这些方式或许大家平时也尝试着去做过一些程序化的处理，我个人认为关联指标的方式，基础在于构建指标体系，这个构建过程可以是人工经验和程序统计分析的结合，指标体系至少能够描述指标的分类、数据出处、具体含义、影响相关指标的权重等等，有了这些基础才能应用统计学的分析方法完成。

四、难点

1、时间序列平稳化

平稳化的时间序列，对预测准确性有非常重要的意义，可是我们的业务监测时间序列恰好大多数都不是平稳化的，以 5 分钟的监测周期为力，除了大盘及核心监测序列，其他的时间序列都是在一定范围内正常波动但总体趋势却是稳定的。我们目前采用的方法是：

（1）滑动平均，比如波动锯齿明显，容易造成误报干扰的化，则加大监控监测周期，将 5 分钟提高到 30 分钟，相当于拟合 6 个时间窗口的数据来平滑时间序列。

（2）持续报警判断，如果觉得 30 分钟发现问题会比较晚，可以按 5 分钟检测，锯齿波动容易发生报警，但可以连续 3 次报警再发通知，这样就避免了锯齿波动的误报。

（3）对于需要**均值漂移来检测细微波动**的情况，24 小时的时间序列本身有流量高峰和低谷，这种情况一般采用差分法做平滑处理，使用几阶差分自己掌握。Goldeneye 没有直接使用差分法，因为我们已经预测了基准值，所以我们使用实际监测值与基准值的 gap 序列作为变点监测的输入样本。

2、埋点代价

业务监控的监测数据来源主要是日志、业务系统模块吐出到中间件、采集接口被 push，从系统各模块吐出数据到中间件似乎比直接写入磁盘的 IO 开销小很多，不过对于请求压力比较大的系统，开旁路写出数据即使是内存级也是有一定开销的。

解决这个问题的办法是数据采样，对于在时间上分布均匀的监测数据，直接按百分比采样。

3、数据标准化

虽然数据接入是分层开放的，但是我们还是制定了标准的数据格式，比如时间序列数据存储 schema，可扩展的日志消息 proto 格式，在这些结构化数据的定义中，可以区分出业务线、产品、流量类型、机房、版本等一些标准的监控维度信息，这样做的目的是以后可以将这些监测数据和故障定位的指标相关性分析衔接起来。

但是，这些标准化的推进需要很多参与者的认可和支持，甚至需要他们在系统架构上的重构，看似是比较困难的。

目前可以想到的办法，就是在旁路吐出监测

数据时，以标准化的消息格式封装，然后保证在 Goldeneye 的存储层有标准的 schema 和接口访问。

五、今后的优化方向

时间序列预测模型，目前的模型只考虑了日期、节假日 / 周末、时间段的因素，没有年同比趋势、大促活动影响、运营调整影响的因素，需要抽象出来。

指标相关性由统计分析程序来确定。

六、Q&A

Q1: 什么样的监控才能称之为智能？

马小鹏 ：这是个好问题！我也一直在考问自己，自从我开始做 Goldeneye 智能监控的那天起。截至目前，我的理解是如果一个监控系统能够具备不费人力、具备自动异常判断、有一定的辅助定位分析的特点，应该能称之为智能，这需要更多的数据去分析支持。不知道我这么解答是否可以。

Q2: 10 秒级别监控海量数据如何存储？

马小鹏 ：这个问题我想留给下一期我的两位同事，牧仁、一鸥，他们会分别从数据存储、大规模实时统计两个方面讲解。

Q3: 报警收敛中，监控项的取用报警次数怎样设定？依靠经验吗？为什么 1、2、10、50 这样的选择？

马小鹏 ：这个只是我举个例子，目前 Goldeneye 的报警收敛表达式是业务方自己配置的，可以根据检测周期的长短、监控项重要性，决定连续报警收敛跳跃的间隔。

Q4: 所使用的的数学检测算法是否有调用比较成熟的整套的第三方的库函数？

马小鹏 ：没有用库函数，是我们自己实现的。我不了解 Python 里是否实现了 mean-shift 模型，不过统计相关性的皮尔逊系数、高斯分布等基本的库，应该是有现成的。我建议你自己实现，因为可以根据实际场景，开放更多的参数去调优。

Q5: 所以说智能监控的基础是大数据处理能力？而对技术人的要求是怎样的，您当时的学习路径是怎样的？

马小鹏 ：你这是两个问题，第一智能监控的基础是大数据处理，但不完全是，因为智能的核心在于统计分析、模型抽象，对于阿里这样的公司我们不必担心计算环境以及资源，因为有人去解决，对于中小公司我觉得按照数据规模一般的计算资源还是有的，我分享的主要是智能监控的思想和技术方案。第二，大数据是当前比较热的词，它也分很多分之，比如数据处理有实时的流式计算、离线的批处理，这些都有很多开源的计算框架，还有就是数据的统计分析、ETL、机器学习等等，你主要先想清楚自己最感兴趣的是哪一方面。我个人最早接触大数据是 2012 年，离线 MR job，到后来的 storm，到现在的统计分析预测，其实没有什么固定的路线，只要找到自己的兴趣就好。希望这样讲对你有所帮助。

Q6: 阿里妈妈告警平台从问题发生到告警到问题定位，大概耗费多少时间？

马小鹏 ：你好，这个问题没有精确答案，我只能告诉你从我们使用了 A/B test 和 topn 自动对比，以及使用事件点的方式以后，比之前定位问题快了很多。但不是所有的问题都能立竿见影的提高定位效率，因为有些系统内部参数指标打埋点很难，无法全部自动化。

作者简介

马小鹏，阿里妈妈全景业务监控平台技术负责人。2013 起在阿里从事大规模系统日志分析及应用的研发，曾经主导了直通车广告主报表平台和实时报表存储选型。在加入阿里之前，曾负责网易电商 App 数据统计平台的研发。

京东 618：升级全链路压测方案，打造军演机器人 ForceBot

作者 张克房

准备电商大促就要准备好应对高流量，全链路压测无疑是必不可少的一个环节，但是同时也涉及到很繁重繁琐的工作。京东研发设计了军演机器人，这为今年备战 618 减负不少。最初，军演机器人 ForceBot 正式立项并组建了一个虚拟的研发团队，彼时计划是基于开源的 nGrinder 项目进行二次开发，随后实现部署即可；随后在深入研发之后，又根据京东的业务场景对 nGrinder 进行了优化，以满足功能需求。

传统的压测方案

传统的系统压测基本都是部署在内网环境，和被压测的系统部署在一个局域网内，比较常用的工具有：loadrunner、jmeter、nGrinder、gatling、iperf 等 等，通过这些工具，模拟生产环境中的真实业务操作，对系统进行压力负载测试，同时监控被压测的系统负载、性能指标等不同压力情况下的表现，并找出潜在的性能优化点和瓶颈，目前流行的压测工具，工作原理基本都是一致的，在压力机端通过多线程或者多进程模拟虚拟用户数并发请求，对服务端进行施压。

以往在京东，备战 618 大促要提前 3 个月准备，需要建立独立系统进行线上的压力评测，这为各个性能压测团队带来了很大的工作量。

除了工作量大的问题之外，传统的压测数据与线上对比，并不准确。以前对某个系统性能的压测需要在内网线下进行多次，压测出的结果各项指标与线上相比差异太大；这是因为线下服务器配置和上下游服务质量不可能与线上一模一样，只能作为一个参考，不能视作线上系统的真实表现。压测后需要思考如何进行各系统容量规划，每次大促前备战会进行基础服务资源的分配，如果不能精确预估容量需求，会发生扩容行为的浪费。

为了更贴近线上真实结果，有些系统也会通过直接内网压测线上系统，这样做需要上下游同步协调，费事费力，这样下来至少一周时间才能搞定。使用了 ForceBot 全链路军演压测系统后，目前只需要 2 天左右就可以完成所有黄金链路系统的性能评测。

变革，制造一个军演机器人

最开始的时候京东并没有直接投入研发力量，京东 618 年中全民购物节技术执行总指挥刘海锋首先提出了 ForceBot 这个想法，京东内部则面向各研发和性能团队进行大量的调研，了解他们现在的痛点和使用的工具情况，同时谈及了 ForceBot 的想法，这个想法获得了大家的赞同并且收集了很多宝贵意见和建议。于是 ForceBot 正式立项并组建了一个虚拟的研发团队，最开始计划是基于开源的 nGrinder 项目进行二次开发，随后实现部署即可。

nGrinder 基于开源的 Java 负载测试框架 grinder 实现，并对其测试引擎做了功能提升，支持 python 脚本和 groovy 脚本；同时提供了易用的控制台功能，包

括脚本管理、测试计划和压测结果的历史记录、定时执行、递增加压等功能。

根据京东的业务场景对 nGrinder 进行了优化，以满足我们的功能需求。比如：提升 Agent 压力，优化 Controller 集群模式，持久化层的改造，管理页面交互提升等。

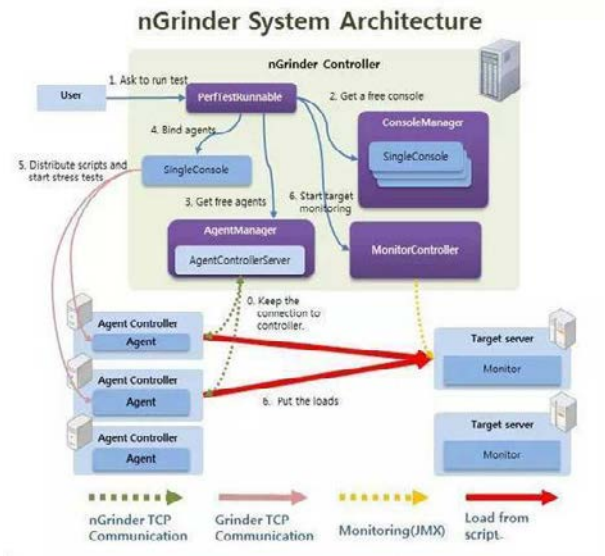


图1 nGrinder 系统架构

NGrinder 能胜任单业务压测，但很难胜任全链路军演压测。分析其原因是 Controller 功能耦合过重，能管理的 Agent 数目有限。原因如下：

- Controller 与 Agent 通讯是 BIO 模式，数据传输速度不会很快；
- Controller 是单点，任务下发和压测结果上报都经过 Controller，当 Agent 数量很大时，Controller 就成为瓶颈了。

也就是说问题出现在：Controller 干的活又多又慢，整体压力提升不上去。

尽管我们优化了 Controller 集群模式，可以同时完成多种测试场景。但是，集群之间没有协作，每个 Controller 只能单独完成一个测试场景。即 nGrinder 整体构架无法满足设想的军演规模和场景，也算是走了一些小弯路，最终在其基础上开始新的架构设计和规划，规避已知瓶颈点，着手研发全链路军演压测系统（ForceBot），为未来做好长远打算。

ForceBot

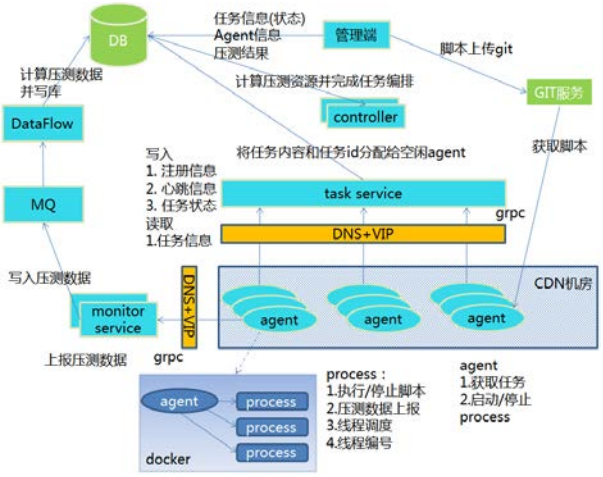


图2 ForceBot 平台

ForceBot 平台在原有功能的基础上，进行了功能模块的解耦，铲除系统瓶颈，便于支持横向扩展。

对 Controller 功能进行了拆解，职责变为单一的任务分配；

- 由 Task Service 负责任务下发，支持横向扩展；
- 由 Agent 注册心跳、拉取任务、执行任务；
- 由 Monitor Service 接受并转发压测数据给 Kafka；
- 由 Dataflow 对压测数据做流式计算，将计算结果持久化至 DB 中；
- 由 Git 来保存压测脚本和类库。GIT 支持分布式，增量更新和压缩。

这样极大的减轻了 Controller 的负载压力，并且提升了压测数据的计算能力，还可以获取更多维度的性能指标。

在此基础上，还融合进来了集合点测试场景、参数下发、TPS 性能指标等新特性。

持续改进

ForceBot 平台在上线提供服务后，在受到好评的同时也发现了一些问题。所以我们对架构和功能实现进行了调整。对问题进行了合理化解决，重新设计了架构中的部分功能实现，并对依托 nGrinder 和 Grinder 的功能，针对京东的使用习惯和场景，进行

了自研，至此，ForceBot 平台由基于 nGrinder 基础上的深度改造升级为完全自研的性能测试平台。

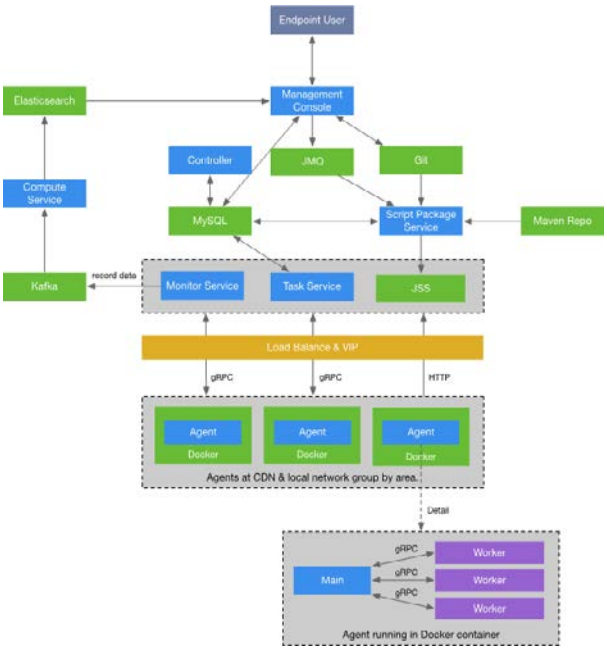


图3 持续改进

Git 作为先进的版本控制系统，用来构建测试脚本工程的资源库再合适不过，但是直接使用它作为资源分发服务就不太合适了。Git 的每次操作如 Clone、Pull 等都是一组交互，传输效率不高。同时一代平台使用 GitLab 构建的 Git 服务集群依赖于一个集中的 NFS 网络共享存储，这就带来性能瓶颈和单点故障的可能。

架构调整中针对 ForceBot 平台的资源分发方式进行了重新设计，借助京东基础平台自研的京东分布式文件系统（JFS）的云存储服务（JSS）进行资源的分发。

新的架构调整中，增加了 Script Package Service 为性能测试脚本提供统一的构建打包支持，并通过对 Maven 的支持，与公司内网 Maven 私有服务交互，为脚本提供更灵活可靠的依赖管理。Script Package Service 将脚本及其依赖类库、配置、数据进行打包，处理成一个统一的 Gzip 压缩文件，并上传至 JSS 以向 Agent 进行分发和归档。

平台在使用过程中，由于调试环境和实际运行环境有所差别，用户有查看 Agent 执行日志的需求。考虑到日志落地带来的磁盘 IO 对性能性能的影响，以及日志内容给平台管理带来的开销，新的架构中提升了日志的收集和处理功能。Agent 的日志不再落地本地磁盘，改为写入到内存一块固定大小的区域，最后经处理切割成一条一条的日志实体并结构化的存入 Elasticsearch 中供用户查询。

技术细节

1. 核心功能

平台核心功能在于需要向性能测试人员提供一个高效的可操作环境用于准确的描述其测试逻辑，并兼容大部门公司业务服务调用方式和场景。京东内部业务系统大部分使用 Java 语言开发，使用基础平台中间件技术部开发的 JSF、JMQ 等中间件进行服务调用，为此提供一个与 Java 语言高度兼容的脚本语言执行环境作为性能测试逻辑编写基础尤为关键。

系统选用了兼容 JSR223 规范的 Groovy 作为主要脚本语言，并效仿 Java 下著名的单元测试框架 Junit 的设计哲学设计了一套高效并友好的测试逻辑开发和执行环境。

平台核心脚本引擎为性能测试脚本设计了多种生命周期控制，以适用不同的场景，并使性能最优化。在脚本编写过程中，用户仅需要在 Groovy 脚本中使用内置的几种注解便可对脚本的执行和数据采集进行精确灵活的控制，如测试类生命周期、事物、执行权重等，大大提升脚本开发效率。

2. 容器部署

为了快速的创建测试集群，Agent 采用 Docker 容器通过镜像方式进行自动化部署。这样做好处如下：

- 利用镜像方式，弹性伸缩快捷；
- 利用 Docker 资源隔离，不影响 CDN 服务；
- 运行环境集成，不需要额外配置运行所需类库；
- 每个 Agent 的资源标准化，能启动的虚拟用户数固定，应用不需要再做资源调度。

3. 服务通信

Task Service 采用了 gRPC 与 Agent 进行通信，通过接口描述语言生成接口服务。gRPC 是基于 http2 协议，序列化使用的是 protobuf3，并在除标准单向 RPC

请求调用方式外，提供了双向流式调用，允许在其基础上进一步构建带状态的长链接调用，并允许被调用服务在会话周期内主动向调用者推送数据， 其 java 语言版采用 netty4 作为网络 IO 通讯。使用 gRPC 作为服务框架，主要原因有两点。

服务调用有可能会跨网络，可以提供 http2 协议；

服务端可以认证加密，在外网环境下，可以保证数据安全。

4. Agent 实现

Agent 采用多进行多线程的结构设计。主进程负责任务的接收、预处理和 Worker 进程的调度。将任务的控制和执行进行进程级别的分离，这样可以为测试的执行提供相对独立且高度灵活的类库环境，使不通的任务之间的类库不会产生冲突，并有益于提升程序运行效率。

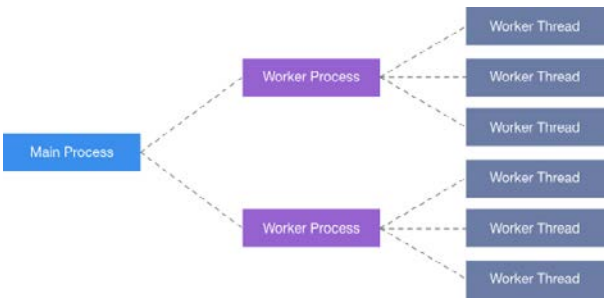


图4 Agent多线程结构

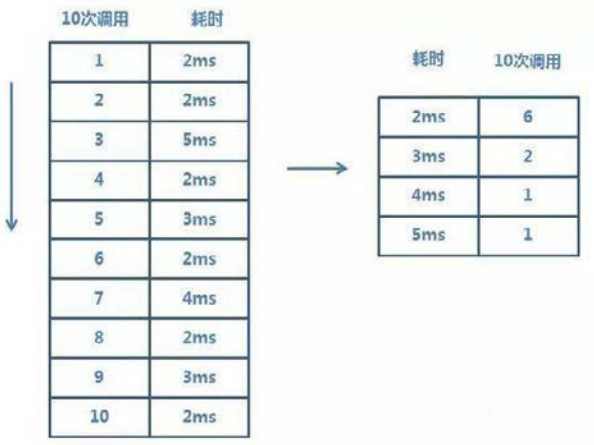
Agent 与 Task Service 保持通信，向系统注册自身并获取指令。根据任务需要启动Worker 进程执行任务，主进程

负责管理 Worker 进程的生命周期。Worker 进程启动后会通过 gRPC 与主进程保持通信，获取新的变更指令，如线程数变化通知，及时进行调整。

5. 数据收集和计算

实现秒级监控。数据的收集工作由 Monitor Service 完成，也是采用 GRPC 作为服务框架。Agent 每秒上报一次数据，包括性能，JVM 等值。

Monitor Service 将数据经 Kafka 发送给 Compute Service，进行数据的流式计算后，产生 TPS，包括 TP999，TP99，TP90，TP50，MAX，MIN 等指标存储到 ES 中进行查询展示。



10次调用	耗时
1	2ms
2	2ms
3	5ms
4	2ms
5	3ms
6	2ms
7	4ms
8	2ms
9	3ms
10	2ms

耗时	10次调用
2ms	6
3ms	2
4ms	1
5ms	1

图5 数据收集和计算

为了计算整体的 TPS，需要每个 Agent 把每次调用的性能数据上报，会产生大量的数据，Agent 对每秒的性能数据进行了必要的合并，组提交到监控服务以进行更有效有的传输。

新式全链路压测的打开方式

ForceBot 的工作基本原理就是站在真实用户角度出发，从公网发起流量请求，模拟数百万级用户 0 点并发抢购和浏览，由于压力机分布在全国各地，不同的运营商，所以最接近真实用户的网络环境，对于读服务通过回放线上日志形式模拟用户请求，数据更分散、更真实，避免热点数据存在，对于写服务则模拟用户加入购物车、结算、更改收货地址、使用优惠券、结算、支付等核心链路环节，每次军演都会发现新的问题和瓶颈，对后期的资源扩容、性能调优都会有针对性的备战方向，最终让研发兄弟对自己系统放心，让消费者购物无忧。

图6 全链路压测

全链路压测方案最重要的部分是被压系统如何去

适应这种压测，如何精准的识别测试流量，如何协调整个研发系统统一打标透传压测流量，并做相应的处理，尤其对脏数据的隔离和清理尤其关键，被压测的系统要和线上环境保持一致，又要隔离这些数据，需要确保万无一失，尤其订单部分环节，出现差错是不可恢复的。

压测数据读的服务都是来自于线上服务器的日志真实数据，数据最终会由各个系统通过标记进入回收站，最终被清理。军演平台本身不会侵入到系统去清理垃圾数据，每次军演压测首先会有一个目标值，比如按照历史峰值的一个倍数去动态加压，如到了这个目标值，各系统性能表现都非常好，那么继续加压，直至有系统出现瓶颈为止。

军演机器人的过去与未来

全链路压测可以理解为网络链路 + 系统链路，网络链路是用户到机房的各个网络路由延迟环境，系统链路是各个系统之间的内部调用关系和强依赖性。其实去年双 11，京东就采用了 ForcBot，只不过仅仅针对重要链路；因为如果没有核心系统首先参与进来，小系统是搞不起来的，因为小系统强依赖核心系统。

今年备战 618，ForceBot 技术架构没有太大调整。我们工作一部分是平台用户体验的一个优化和性能的优化，让有限的压力机产生更大的压力请求；另外一个是整合被压测的系统监控，实时展示。

京东未来希望 ForceBot 可以实现“人工智能预言”。现在还在逐步的做，我们希望未来的全链路压测引入 AI 技术，通过人工智能预言各个系统的流量值和资源分配建议，根据线上的系统军演数据预言未来的大促各系统场景，举个简单的例子，在 ForceBot 平台上录入每秒一千万并发订单场景，以现在的系统去承载，各系统是一个什么样的性能指标和瓶颈点在哪？ 这就是我们思考要做的。

作者简介

张克房，京东商城资深架构师，2010 年 3 月份加入京东商城，2010 年 -2016 年负责京东核心数据中心 IDC 网络、负载均衡、DNS、自建 CDN、DevOps 等多项运维架构和运维管理负责人，是京东早期运维体系架构变革和发展的直接参与者，奠定了京东运维现行的多项技术架构和运维模式，伴随京东从小到大，从无到有、从有到优的飞速变化。

360：集预测、处理、关联和资源优化于一体的智能运维系统

作者 籍鑫璞

DoctorStarange背景介绍

为了保证 360 公司内部的私有云平台的稳定性和可靠性，我们部门开发了监控系统 wonder 来监控系统的状态。但是随着业务量的增加，通过设置单纯的阈值来监控报警是远远不够的。而且这种被动式的触发报警很多时候需要人工去处理。

我们提出来 DoctorStarange，它是一个智能的预测和处理系统，能够提前预测出一些监控项的报警，并提前处理预测的报警，最大程度减少报警次数；它是一个关联不同报警项的系统，能够帮助运维人员去更快地排查报警；它还是一个对机器各个维度进行检测的系统，能够优化机器资源。

下面分三个部分对该系统进行介绍：首先介绍智能预测与处理系统，然后介绍报警关联分析，最后引入机器健康度的概念，来及时了解机器的运行状况，最大化机器资源的利用率。

智能预测与处理系统

我们对监控项的历史数据进行分析后就会发现，有些监控项的历史趋势的增长（或下降）趋势趋于稳定，如磁盘空间使用率，而有些监控项的趋势就波动的比较厉害，如 CPU、网卡流量等。对于这两种趋势，我们需要使用不同的预测模型对未来一段时间的趋势进行预测。预测只是第一步，如果预测后能让机器自动处理一些危险的监控项，就可以最大程度上减少人工干预。

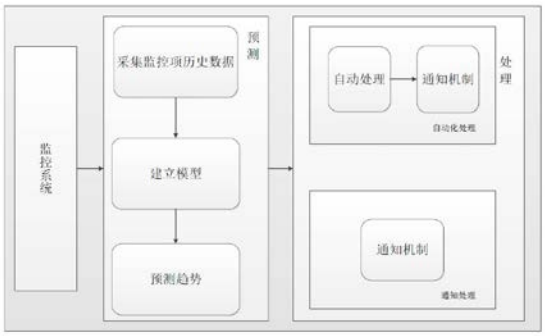


图1 智能预测与处理系统的框架图

趋势稳定的监控项的预测与自动化处理

应用系统出现故障通常不是突然瘫痪造成的，而是一个渐变的过程。本节我们将结合磁盘空间使用率来说明我们的系统如何做到智能预测与处理。很明显，磁盘预测工作应该分为两部分，一部分是预测，另一部分是处理，具体的流程如下所述。

首先说一下预测。系统长时间运行，数据会持续写入存储，存储空间逐渐变少，最终磁盘被写满而导致系统故障。由此可见，在不考虑人为因素的影响时，存储空间随时间变化存在很强的关联性，且历史数据对未来的发展存在一定的影响。所以可以采用时间序列分析法对磁盘容量进行预测分析。

我们使用 ARIMA 模型来处理时间序列，ARIMA 涉及到三个参数：p（自相关系数），q（偏自相关系数）以及 d（差分次数）。差分次数可以使用 ACF 函数来检测差分后该序列是否为平稳序列，而 p 和 q 可以使用 AIC 或者 BIC 准则来得到使 AIC 或者 BIC 信息量达到最小的模型阶数。

有人可能会问这个系统可能依赖于预测模型的准确率。为了验证模型的准确率，我们提出来两个概念来说明效果。

- 预测准确率：预测准确率 = 预测报警且具有报警趋势的机器数 / 预测报警的机器数。
- 报警减少率：预测覆盖率 = 预测报警且真正报警的机器数 / 报警的机器数。

我们对线上 20000+ 台机器未来 24 小时的走势进行预测，跟踪了将近一个月的预测结果。可以发现我们的模型预测准确率能够达到 100%，报警减少率能够达到 70% 左右，这说明我们的模型能够获得比较好的效果。

接下来将重点介绍在预测磁盘使用率将要达到阈值后，我们如何自动处理的过程。用户可以根据自己的意愿选择处理类型：一种是自动处理，一种是通知邮件。

如果是自动处理类型，我们会清理 100% 可以确定删除的日志文件，比如 allweb 文件以及一些归档的日志文件。虽然我们有一定的目录规范和定期的日志轮转，但是因为有些程序编写的不规范，业务访问量的增长，还是会有磁盘被写满的情况，所以单纯依靠自动清理会有一定的风险，所以我们还有一种是通知邮件，我们会定期将扫描出来的占用空间比较大的文件信息发送给用户，由用户自己去处理。

波动剧烈的监控项的预测与故障定位

和磁盘这种趋势稳定的监控项不同的是，波动剧烈的监控项由于历史数据变化情况比较复杂，采用传统的 ARIMA 或者指数平滑的方法很难达到比较好的效果，因为他们很难捕捉到以前从未出现过的情况。相反，神经网络模型由于输入的是非线性方程，可以处理更复杂的时间序列。对于 cpu 预测，我们也分了两个过程：预测以及故障定位。

本节采用神经网络模型去预测 CPU 的走势，提前预测出将要发生报警的机器，尽可能将具有隐患的故障扼杀在摇篮中，提高系统的可靠性和可用性。

建立神经网络模型，首先要确定输入、隐藏和输出层的神经元个数。由于我们的线上业务具有天的周期性，所以我们将输入层的个数设为 24（当然这个值也可以通过求自回归系数来求得），隐藏层的个数一般为输入层个数乘以 2 再加 1，所以输出层为 49，输出层的个数为 1，我们根据此规则建立模型。

和趋势稳定的监控项的预测准确率和报警减少率一样，我们也预测了波动剧烈的监控项的未来一个小时的趋势，同样的机器数量，我们的预测准确率能够达到 80% 左右，报警减少率能够维持在 50%。由于波动剧烈的项相比于趋势稳定的项来说，不太容易预测，能够一定程度减少报警也可以达到比较好的效果。

接下来说明预测报警后，如何进行故障定位。我们会根据 top 值来找到占比比较高的进程，由于直接处理占比比较高的进程会有风险，所以我们只会邮件通知负责人以及运维人员。

以上我们以磁盘使用率和 cpu 为例子介绍两种不同趋势的监控项的预测以及处理工作，其他具有此特征的监控项的思路类似，在此就不一一展开。

报警关联分析

随着监控系统监控项的数目越来越多，报警的次数也会随之暴增。不同报警项他们之间并非孤立的，而是存在着千丝万缕的联系。我们分析出某一时间段内协同发生变化的监控项，能够帮助我们排查出问题，找到问题的根源。报警关联分析系统主要做了两件事情：报警系统：加入报警合并算法，能够减少报警次数；实时报警分析：找到存在因果关系的监控项，帮助排查问题；图 2 是报警关联分析系统的框架图。

怎样去找到关联项呢？我们根据自相关系数来获得该段时间内发送波动的监控项。一起发生波动的项可能非常多，通过求曲线斜率的方法来获得各个波动项的波动剧烈程度，并按照波动剧烈程度排序。利用均值波动法获得波动持续的时间。在综合考虑波动剧烈程度和持续时间后，我们可以给每个波动项赋予一定的权值，并由此得出正关联和负关联的项。

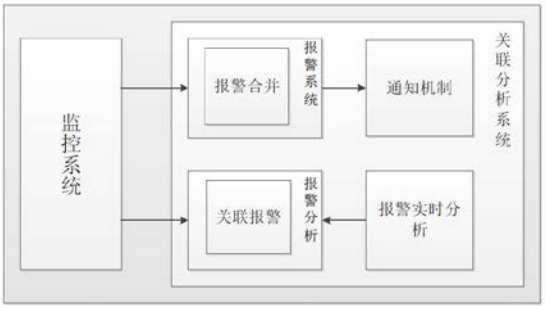


图2 报警关联分析系统的框架图

现实场景中，单纯依靠当前的数据来得到相关联的项可能不太准确，这时候我们就需要一个自学习的关联分析系统，通过老 case 和人工经验的加入，该模型得到不断地修正和补充，因而会获得比较好的效果。

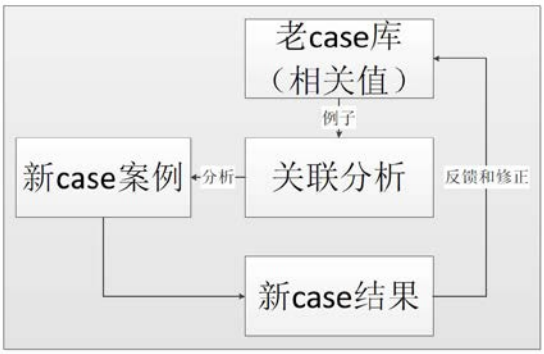


图3 关联分析系统

对于报警系统，我们主要找到正关联系数比较高的监控项，来合并一些报警之后，以最少的报警次数通知给用户。而对于实时报警分析，我们就可以通过输入一个波动项（原因），找到由此原因导致的其他波动的项（结果）。

机器资源优化方案

面对不同业务的机器使用程度不同的问题，如何在不影响业务的同时，最大化机器资源利用率越来越成为业界比较关注的话题。我们不能使机器使用率过高，同样也不能使机器利用率过低，因此我们提出来一个“机器健康度”的概念，该值会反映出该机器过去一段时间内重要指标的使用情况。

首先，如何选择机器指标？在我们的方案中，我们选择 cpu 空闲率、内存使用率、网卡流入流量、网卡流出流量和状态连接数作为考量因素。之所以选择这几个指标是因为这几个监控项能够总体反映出机器的负载情况。

我们分别通过阈值的方法为这六个监控项设置上限和下限后，针对于每个监控项我们得到了四个值：

- 历史数据的均值上限；
- 历史数据的均值下限；
- 预测数据的均值上限；
- 预测数据的均值下限。

经过一个公式计算后，我们将得到一个 -1 到 1 区间内的一个值，即为健康度。如果该值越接近于 -1，则说明机器比较空闲，如果该值越接近于 1，表明机器使用率比较高。

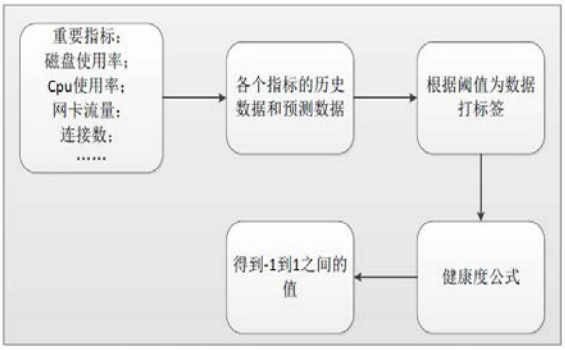


图4方案实施的具体流程

健康度的概念毕竟是一个学术的东西，我们如何将此概念应用到实际的场景中呢？下面将介绍我们具体的应用场景。

场景一 动态扩容和缩容

设置 cron 任务，比如每隔一周全量跑一次线上的机器，将有问题的机器录入到数据库中。 该数据库可以用于以下方面：

- 资源回收的依据（关注健康度值为 -1 的机器）；
- 业务扩容的依据（关注健康度值为 1 的机器）。

场景二 机房物理拓扑（及时了解机房机器的健康状况）

场景二是我们针对于运维人员经常不能了解线上机房机器的运行状况做得一个拓扑图。在该拓扑图里面，运维人员可以更方便知道机器的总体运行状况。

而我们可以以上面提到的“机器健康度”来表示每个机器现在的运行状态。如下图是我们一个简单机房的物理拓扑，在图中，白色的机器表示该机器运行良好，红色的机器表示该机器使用率过高，而灰色的机器则表示该机器使用率过低。

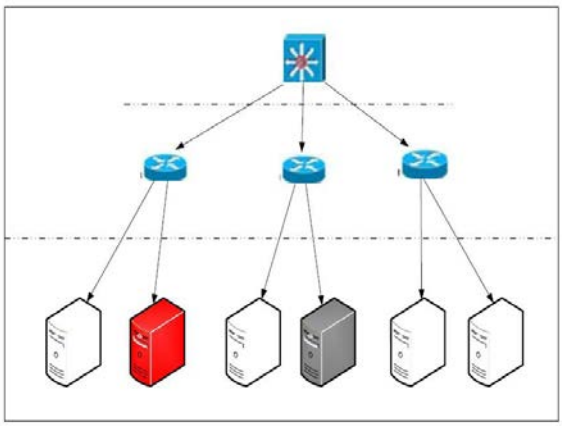


图5 机房物理拓扑

今后方向

- 业务拓扑：业务四层和七层的拓扑图。
- 报警自发现：报警后结合关联分析和分析算法找到问题的根源。

Q&A

Q1：第一部分预测后自动处理能举一个例子吗？

籍鑫璞：比如磁盘空间使用率，在有比较好的预测模型后，预测出将要发生故障的机器后，会有两种处理策略：一种是自动清理，我们会处理一些日志文件（绝大部分的磁盘空间是因此导致的），当然如果觉得不稳妥，我们可以将占比比较高的文件信息发送给用户，由用户来自己处理。

Q2：请问能否说一下你们使用的报警合并算法？你们会发送并处理所有的报警吗？

籍鑫璞：第一个问题：我们的报警合并算法是考虑该报警发生的一段时间内，综合每个波动剧烈程度和持续时间后，给每个波动项赋予一定的权值，并由此得出正关联，我们会选择关联系数比较高的项进行综合。第二个问题：我们现在只做了磁盘使用率和CPU发生报警后如何去处理，我们接下来会做报警自发现的模块，这是我们后期的一个目标。

Q3：自学习的关联分析系统，通过老案例和人工经验的加入，该模型得到不断地修正和补充，能不能详细的说明下，最好能举一个案例。

籍鑫璞：先说老案例吧，将以前的有关联关系的结果保存在案例库里面；人工经验比如某个机器的网络流量突增有可能导致CPU的上升，有可能单纯依靠算法无法达到好的效果，案例库需要不断的丰富。

Q4：是通过什么手段识别误告警的？如何保证关键告警不被其他告警覆盖掉？

籍鑫璞：这个只是辅助手段，最大限度减少报警，报警关键还是人来识别。

Q5：报警自发现和异常检测，我们也很感兴趣，请问能简单介绍下思路吗？

籍鑫璞：先说异常检测吧，最近看到一个开源技术估计对你们有帮助: Skyline timeseries异常判定算法，有兴趣可以看一下。报警自发现：这个就是找到问题的根源，比如CPU报警，可以通过查看top来找到是什么进程导致CPU报警。

Q6: 神经网络模型的输入，隐藏层，输出层具体使用哪些参数数据？

籍鑫璞：输入层是跟踪自相关系数得到的，比如通过求1,2,…,7得到7天的自相关系数最高，那么输入层就可以知道了，而隐藏层一般是输入层的2倍加1，输出层是你想要预测的时间窗口。推荐你看一篇论文：PRACTISE: Robust Prediction of Data Center Time Series

作者简介

籍鑫璞，毕业于南开大学，360公司运维开发工程师，参与Web平台智能运维系统、业务拓扑和APM等项目的开发工作。

扫码关注InfoQ公众号

Geekbang. 极客邦科技 | InfoQ