目录

目录

[第壹章 前言 1](#_Toc456169402)

[1.背景 1](#_Toc456169403)

[运维平台的边界 1](#_Toc456169404)

[2.目标 3](#_Toc456169405)

[第贰章 参考 4](#_Toc456169406)

[第叁章 方法 4](#_Toc456169407)

[第肆章 架构 4](#_Toc456169408)

[第伍章 成果 9](#_Toc456169409)

[运维的本质——可视化 9](#_Toc456169410)

[第一部分：可视化的服务交付 9](#_Toc456169411)

[第二部分，可视化服务度量 12](#_Toc456169412)

# 第壹章 前言

## 1.背景

### 运维平台的边界

识别运维平台的边界在哪儿，才能更好地构建平台，从而协助运维的日常工作。

在后续的篇章中又介绍了“互联网运维的价值体系”，里面分解了几个维度：质量、成本、效率、安全等。以上都是为了清楚地梳理运维的内容边界，基于这个边界，我们再考虑如何进行平台支撑。可以说前两篇文章都是为今天这篇文章作为铺垫，用理念先行，然后再考虑平台落地，最后再细化其中每个内容。我更习惯用如下的方式来整体表达运维的工作方法和思路。

首先，价值导向。找到一个价值方向来牵引整个团队很难，但又必须找到，因这个牵引力就决定了团队的气质及后续的工作方法；之前的文章“运维价值体系”有详述，在此不细谈。

其次要有一个分而治之的系统，最后面向业务自底向上的集成，此时便能帮忙实现更好、更快、更省的交付价值。平台的建设需遵循一些的方法（自底向上、先后顺序等），先建设各个运维专业子系统，通过API的方式对上暴露服务，最后不同的业务平台去调用这些服务接口即可。缺少平台的支持，运维的质量、成本、效率都会直接受到影响。

如果要做好服务器精细化成本控制，此时需要一个平台来处理从服务器资源上采集的资源使用状态数据，并生成可视化数据报表，共享到所有团队中，在一致理解下，去驱动成本优化，越海量的业务对这个平台的要求就越高，从采集、处理、模型算法等都有很高的要求。

不要忘了这个平台还包含面向业务技术栈构建的平台。这地方有一个非常好的例子，在2012年左右，我了解到Google有一个非常强大的资源管理平台Borg（后面叫Omega），它的设计目标是“把数据中心看成一个芯片”。Google研发人员将开发的服务交给Borg，后续的服务生命周期（扩容、缩容、调度）都由Borg统一接管，服务被Borg部署到哪个IDC、哪个服务器，研发人员不用关心。后来Twitter根据Borg的思想，也开源实现了一个平台——Mesos，不过Mesos对LongTime的服务调度（如Nginx）支持不是太好，更适合MapReduce的事务调度。这两个资源管理平台背后的思想都值得深究，建议看看。

第三，基于平台，提供透明服务，确保服务提供者和服务交互者之间的交互越少越好。有了整合性的平台，透明提供服务也成为可能。平台整合就是避免服务被碎片化，从而让使用的用户看到的不是一个一个工具或者孤立系统，而是面向业务的整合服务。此时成本便可降低、变更的质量也会变成一个稳定态。不同的人、不同的时间执行相同的事务流程都能取得一致的执行结果。

最后，数据驱动。因所有线上业务服务和线下运维服务都有状态，需数据平台提供服务状态数据的采集、处理、分析处理能力，最后还能让运维人员自定义分析报表。技术运营数据和产品数据的一个很大的区别是，前者在数据挖掘方面的能力要求很少。这个地方有个建议，把线上服务的数据驱动作为重点（80%），把运维内部服务的数据驱动为辅（20%）。因为线上服务的状态会反作用于运维内部事务的优化。比如说从数据中发现现网的服务有一个故障，需要紧急发布版本，此时就会直接检验运维的变更部署流程、平台的完备性。

## 2.目标

自动化运维平台的目标就是自动化和数据化一切，并且最终可视化，从而确保质量、效率和成本几者之间的平衡。

# 第贰章 参考

# 第叁章 方法

但对于这么一个庞大的复杂体系来说，不可能一蹴而就，可以借鉴一下经验。

1. 自底向上。一定要把握这个原则，这就相当于我们造车一样，把各个零件造好了，最后就是组装。
2. 加强跨团队之间的合作与沟通。很多事情一旦研发、测试和运维彼此合作，事半功倍。在合作的过程中，把彼此的需求都统一到平台中，这样有利于后续的推广和使用。
3. 平台建设先后有序，优先级顺序如下：

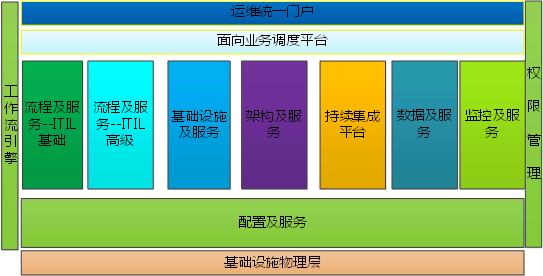
l P1（最高）：CMDB、基础架构及服务、数据及服务、监控及服务、持续集成；

l P2（次高）：面向业务的运维平台；

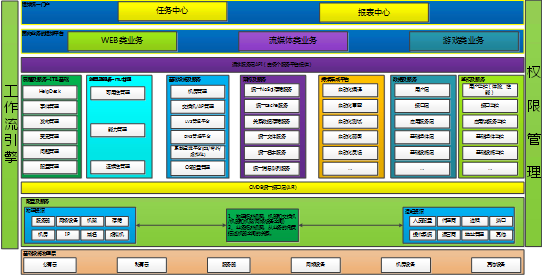
l P3（低）：ITIL相关、运维统一门户。

# 第肆章 架构

在平台体系部分，我采用逐级构建的方法，不断去细化其中的内容，因此会有一级视图和二级视图，在这个地方，我不敢到三级的模块级别，基本上不可看，下图是参照的是eTOM模型构建方法。



继续往下，可以分解出二级视图。



有了整体的平台体系视图，接下来看看每一部分到底是干什么的。

工作流引擎、权限管理。这两者都是基本的功能，因为其中会涉及流程，所以需要统一的流程引擎平台。另外需要部门、角色、用户的权限管理统一管理，不同业务配置不同系统的使用策略即可，这一块可以统一实现在单点登陆系统中。

1. 基础设施物理层。这个视角和传统模式有些不同，主要是公有云的存在。因此在基础设施物理层这块，已经把云端资源当作一个底层基础设施来看待，后续的资源获取完全不同，其他的资源对象依然没有变化，依然是机房、机柜、网络、服务器，等等。
2. 配置及服务，把配置当作服务来看待。在ITIL中叫CMDB，Configuration Management Database， CMDB也可以理解成统一的元数据库，比如说机房信息、服务器信息、人员信息、服务信息、业务信息以及他们之间的物理和业务拓扑关系等，上层的所有系统都应该关联到CMDB，变更后的信息必须实时反馈到CMDB中，确保其他系统能同步这份变化。因此大家都把CMDB系统当作运维的核心系统来对待，便于后续各个系统之间的互通。

在我的经验中，CMDB建设还是有非常多的坑。如果你把iTop或者oneCMDB的产品当着标杆（都是开源，没见过商业的），那你的CMDB建设就完了。之前在一家传统企业，他们把文档都放到CMDB中管理，不建议这么做，文档就是SCM的事情。CMDB建设的核心准则：CMDB管理的数据一定要为了业务管理，业务管理上不需要的东西，就果断舍弃，比如说文档，和业务没有任何关系，就可以不考虑纳入，后续会有专门的文章介绍。

1. ITIL服务——基础、ITIL服务——高级。在早期的文章中把DevOps和ITIL做了对比，ITIL是面向流程的，这个可以在运维平台建设中不做重点，不要主动去构建流程，会影响运维的敏捷性。基础部分实现一个事件和HelpDesk即可，事件管理在告警转换成事件之后，可以完整地记录，便于我们事后的原因分析，能挖掘一些问题，比如说是否某个业务、某个人、某类机器经常性故障，那就需要重点关注下。高级服务的部分，大家需关注一下，它是可以带来价值的，比如说可用性管理、能力管理和连续性管理。可用性直接的导向就是业务的质量；能力管理直接的导向就是成本管理；连续性管理也是和质量戚戚相关，如业务的容灾、备份管理等。但这些管理都不要在流程层面上去看，需要在一个平台中进行全面的可视化管理。后续的篇章也会有相应的介绍。
2. 基础设施及服务。把底层运维资源的管理封装成一个一个的服务，供业务自动化平台使用。我把DNS、LVS（或者F5）甚至OS上的配置管理都看着基础设施部分，适当地向上延伸了一下。简单的划分原则是，在业务架构之外的，都可当着基础架构部分了。很多运维团队的建设重点都在这块。
3. 架构及服务。把业务架构中的共性需求都剥离出来，抽象成一个一个的服务，最终让研发只需要关注自己的业务代码即可，比如说统一文件存储、统一Nosql存储、统一RDS存储、统一队列等。这块对运维的质量、效率、能力等影响最大，在之前的文章“如何化解研发和产品之间的矛盾”中重点阐述过服务公共化是唯一的解决之道。现实中如果有研发开发了一个公共组件交给运维，而不提供完整的Webadmin或者API的话，你也就可以认为他是在耍流氓，运维必须有严格的完整性交付要求。
4. 数据及服务。只要有线上服务在运行，服务数据流经过的一切节点产生的数据，你都要采集、存储和分析起来，供不同的运维场景使用。比如说自动化调度，可以根据业务涉及的基础节点资源使用情况，制定对应的自动化调度策略；可以在数据中直接进行故障定位；可以在数据中做安全分析。之前的文章“数据驱动运维”中介绍过我做的一个数据分层体系。
5. 监控及服务，有数据的地方才有监控。脱离这个原则，你做的都是告警，并且告警的成本会越来越大，不成体系。个人观点：所有的监控视图都是来源于我们对数据的采集以及我们到底有多少经验来看待数据。
6. 持续集成。这条线是把一个个的程序包交付到各个环境，在【持续部署】之上的部分可以通过和持续集成工具Jenkins或者Go作对接即可。持续反馈非常重要，一个程序部署到生产环境之后，需要实时的运行报告反馈回来，确认变更的效果。如果持续部署平台化之后，真正的执行部署工作会不断前移，甚至可能直接交付给研发。此时的状态报告，更是有必要，不需要人去登录主机tail日志看是否正常。这个地方和“数据及服务”的能力关联很大，没有前面强大的数据服务能力。
7. 面向业务的运维平台。不同的业务会有不同的调度策略和服务使用策略，需要在更上层完成面向业务的统一调度，这个是全应用的视角，和持续集成是有一些区别的。在没有这个平台之前，一个完整的业务上线，需要做很多操作，比如说DNS变更、LVS变更、OS初始化、自动化测试、持续部署、持续反馈、监控、业务调用关系配置，等等。面向业务的调度平台，就需要有一种调度能力，指挥底层各个平台为它服务，它本身不实现任何服务接口，是一个服务的集成者。
8. 运维统一门户。每个运维系统都有任务或者信息与自己相关，如果运维人员每天要去面对那么多的运维系统，会非常痛苦。在统一门户里面分成两个部分，一部分是任务中心，把底层所有的事务状态都同步到任务中心中，表示我要做什么；信息中心，就是让运维人平时关注的业务状态Dashboard直接推送到信息中心中，表示我要关注什么。

# 第伍章 成果

## 运维的本质——可视化

没有比“可视化”更好的一个词能概括运维的本质，而“可视化”又应该分成两部分：**可视化的服务交付**和**可视化的服务度量**!

## 第一部分：可视化的服务交付

早期的运维是从ITIL开始的，那个时候大家都不知道运维是什么，幸好**找到了一个IT服务最佳实践——ITIL**。开始了互联网运维的摸索之路，从CMDB、服务台、事件管理、变更管理、可用性管理、容量管理等逐步去了解，并同步建设对应的管理平台。但我们很快发现，这一完备的流程框架如果遇到了大规模运维的情况，就无法应对，原因在于过多的聚焦于流程以及规范，我们发现很难提升运维敏捷度和精细性，并且我们还是不知道一个完整的IT服务边界在哪儿？如何实现它？

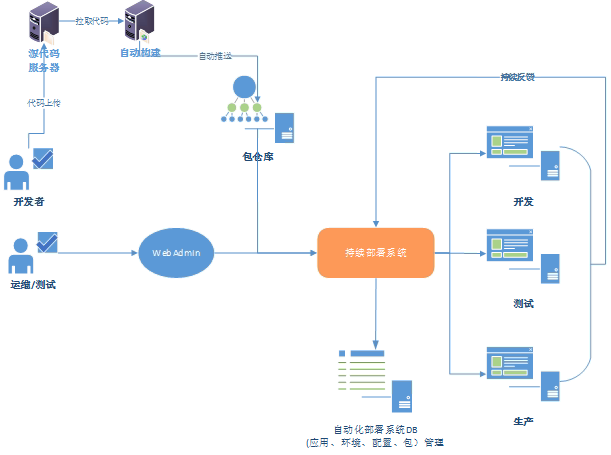
不过在ITIL的实践过程中，其实提出了一个很好的概念——**IT服务**。**对于运维来说，提供一种高效、一致性、透明化、面向用户的服务是运维的价值所在，这样就要求运维屏蔽其提供的服务背后的所有实现细节**。

从运维具体事务或者活动的角度来说，如何对其进行一次或者多次的组合封装，把它们变成一个完整的IT运维服务，是此时的运维自动化重点方向。毕竟繁杂的运维事务不进一步封装，对个人或者团队来说，都意味着很高的学习成本和事务执行成本。在传统的IT运维组织中，我们能看到彼此事务之间的割裂非常明显，比如说网络、机房、服务器、应用部署等，都是在不同的团队完成，彼此工作独立进行。在敏捷和精益运维驱动之下，必须要求有一个集成平台来把这些事务流调度起来，否则无法提高事务执行的效率和质量，真正地把运维**交付功能**变成了**交付服务**的模式

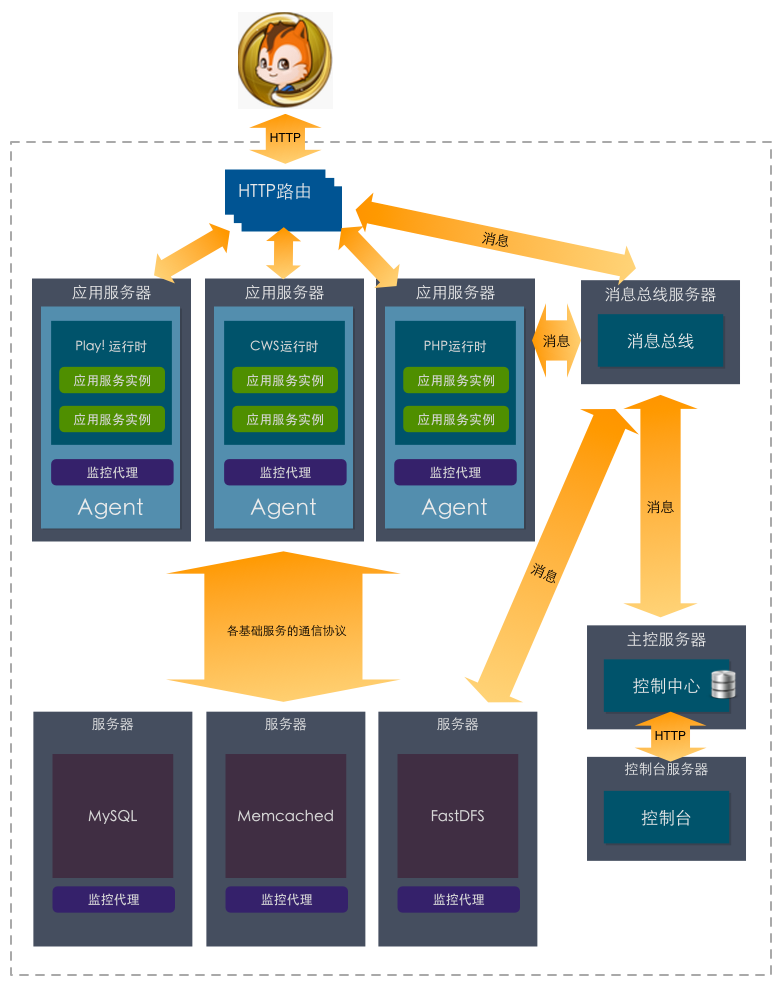
对于如何封装这些事务或者活动，从DevOps提倡的“自动化一切” （Auto everything)可以找到些答案，其核心的自动化主线就是面向用户的敏捷持续交付。我把持续交付又分成两类场景：**一种是持续交付基础设施，一个是持续应用交付 （持续构建、持续测试、持续部署、持续反馈)**，他们有点近似IAAS和PAAS的关系。

持续交付基础设施在公有云IAAS平台中得到很好的解决，利用**软件定义计算、存储、网络**等技术来实现对上层应用所需资源的快速交付。在私有IT环境中，当前有大量客户采用虚拟机方案或者私有云方案来解决交付难和慢的问题。最新的轻量级虚拟化技术Docker更是热点，根本的原因是把应用的交付在镜像级别完成，从而让应用交付更加快速。

**持续交付软件**从代码产生的那一刻就开始进行管理，到编译、到测试、到灰度环境验收再到正式环境部署，并且希望这条主线完全自动化。**面向程序包的持续集成非常简单**，现在有很多的开源解决方案来实现，如Jenkins、Go等，但有一种情况需要特别注意，就是**程序包的配置管理问题**，这个也往往是影响部署的重要因素。所以我们很多时候使用开源平台只是为了构建程序包，后续包及其其中的配置管理以及实例化部署，特别是大规模集群部署，都是由单独的**持续部署**平台来解决，而非之前的持续集成工具（虽然它们也支持发布），但持续部署平台需要有和持续集成平台无缝对接的能力。



基于软件包的交付解决之后，我们希望交付的粒度更大，**如何实现全应用（从应用的前端接入到后端存储)的交付**，此时便有了PAAS平台和基于应用架构的可视化部署服务两种方案。这两种实现思路有很大的不同，我们知道完整的PAAS平台提供了对底层公共服务的向上API统一抽象，比如说数据库服务、存储服务、Cache服务。PAAS平台最经典的实现应该是Cloud Foudry了，国内很多PAAS平台基本上都是参考CF来实现的。阿里UC也有一个类似的PAAS平台，示意图如下。



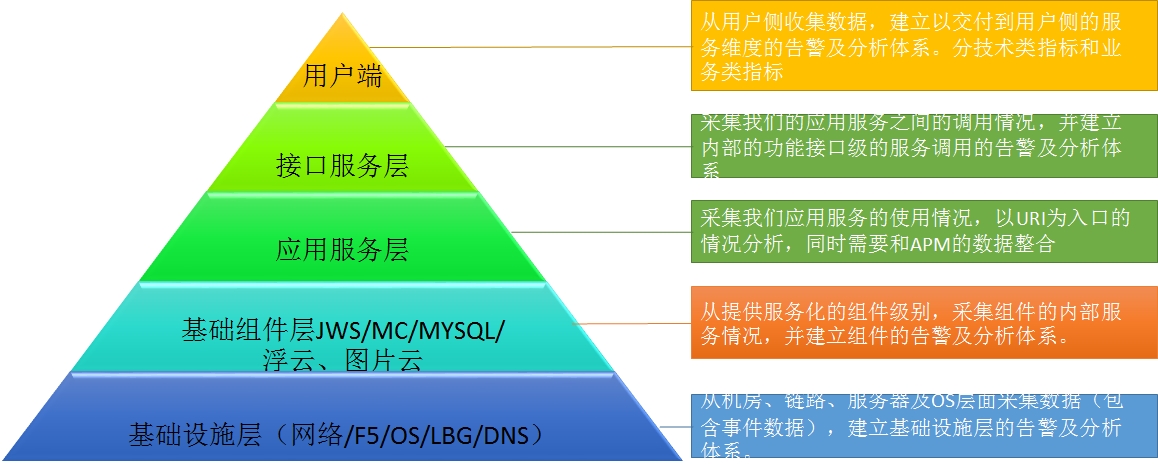
而在现实的情况中，很少公司有能力把MySQL、MC、Fastdfs封装公共服务供上层应用直接调用，意味着对研发程序有着一定的要求，是否还有一种更轻量的无约束自动化方式呢？我们可以把运维的全应用部署转变下思路，此时把应用架构中的各个部分拆解成对象组件（包含属性和状态)，比如说机房、OS、应用包等，全应用部署就是这些对象的编排，类似可视化IDE编程环境。

综上所述，**运维的自动化最终要实现可视化，复杂的运维工作流必须通过可视化来表达，可视化后的自动化才能让所有人理解一致、执行一致、结果一致**。

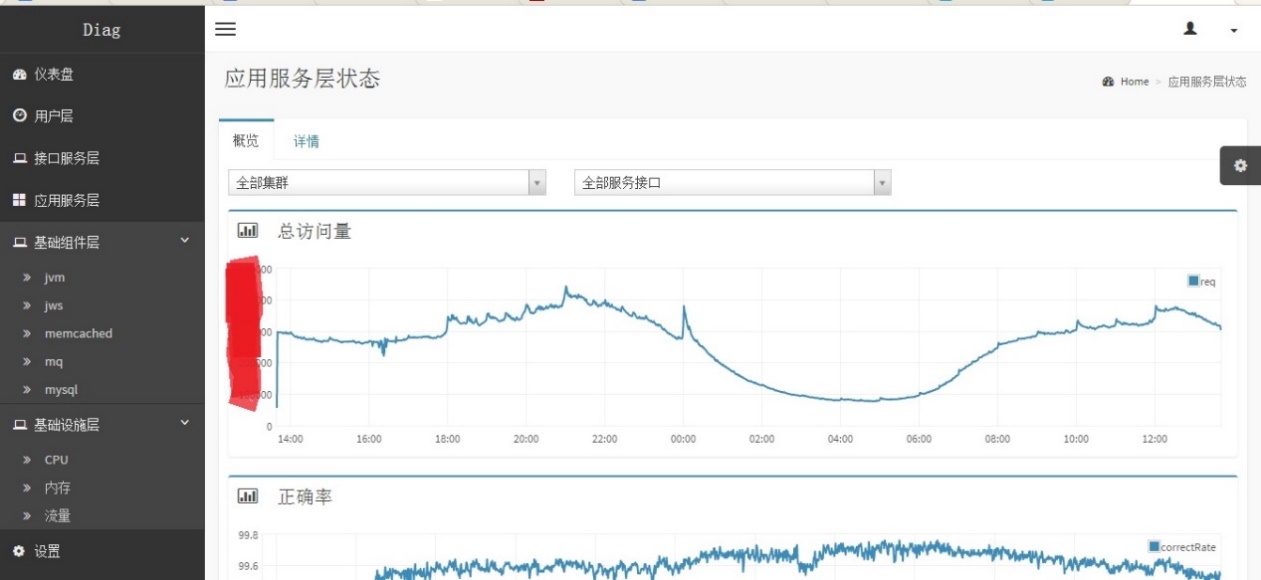
## 第二部分，可视化服务度量

**“除了上帝，一切人都必须用数据说话”**，这是运维人员必须恪守的信条。我写过一篇完整的数据驱动运维的文章“[关于数据驱动运维的几点认识](http://mp.weixin.qq.com/s?__biz=MzA4NjAzMjEyOA==&mid=200143127&idx=1&sn=b8d3f71659958a9ff60291b3892cfb66#rd)”，里面系统地介绍了数据化运维的目的、数据的来源以及如何构建数据体系，等等。

最近也在进行一个数据实践，就是建立面向应用的端到端数据分析体系，该体系对数据有个标准化的分层归类，从**基础设施、上层组件、到应用服务、到接口、再到用户侧**，**基于应用的拓扑架构，收集各类指标，统一到一个分析平台中展现**，如下图所示。



基于这套分层化的数据体系标准，我们也有对应的系统实现，如下图所示。



当形成标准的数据采集、分析和展现体系之后，可以向其他应用不断去复制这套方案，大家只需要遵循一套数据标准即可，最后数据的采集、分析、展现和告警都是标准化完成。这套数据体系建设完成之后，可以在运维的故障定位、服务优化、架构改进、运维规划等各方面找到应用场景。

此时有人会有疑问如何面向应用把这些数据整合关联起来？我们当前是**基于配置文件的静态视图和基于接口调用而生成的动态视图来集成**。动态调用视图生成会复杂一点，可以让线上的接口调用统一由名字服务中心来接管调度，抽样对接口调用进行染色，从而生成动态的访问关系。

以上视图能快速发现和定位规模故障，但对于单个用户的故障指标上则应对乏力。此时分布式Trace服务的作用就显现出来了，可以借鉴Twitter的Zippkin和Google的Dapper的实现思路。当前我们就结合自身的业务架构特点，实现了一个统一的服务调度框架和名字服务中心，在业务代码无侵入的情况下，可以把业务调度链的染色数据上报和关联，实现对于单个问题的快速定位。

数据的可视化能力非常重要，需要在面向整体和面向某个业务流上都有实现。**它首先体现出你对运维的理解是什么样的**，从可视化Dashboard上可以看到最直接的运维经验；**其次基于可视化之上的数据共享，让大家对数据的理解达成一致；**最后**利用一致化的可视化数据发挥运维的驱动能力，驱动DevOps，数据的核心价值就在于此。**

因此**可视化的能力就代表了运维的能力**，可视化的程度越高，运维的能力越高。那么你现在到底可视化了哪些运维服务，并能进行度量呢？