运维平台

综述

服务可管理的前提是运维数据的准确性，而标准化和流程化是保证数据准确性的前提。只有提供准确的运维数据，才能进一步实现服务的运维自动化。所以，一个能够准确记录和管理服务信息的运维平台，对于运维的发展至关重要。

在运维团队组建初期，运维平台建设一直属于运维团队的工作重点。通过标准和流程的约束，保证信息准确地录入到平台，以便能够准确提供运维所需要的各种维度信息，帮助运维人员开发更上层的系统，获取运行状态、资源占用等信息，与部署系统联动进行服务的动态调度部署和故障容错。

一个真实案例中，早期的运维平台有服务器管理、IDC 管理、监控（Zabbix）、密码管理、故障记录等这几个模块，更多的是信息记录，更像一个网页版的 Excel。没有流程的引入，信息录入完全依赖于人。这个时候的信息仅仅用来对账，滞后不准确的数据无法作为运维工具的基础依据，更谈不上自动化。平台各个功能模块之间没有信息关联，所有信息如一个个孤岛，对于运维的价值非常低。

随着需求场景的进一步明确，平台在不断建设。形成了两个大的运维平台，即：资产管理平台和服务管理平台。

资产管理平台

负责记录基础的物理信息，如：IDC、服务器（资产编号、参数、采购时间、供应商）、配件、网络设备、IP 地址、ACL 等。提供了多个子功能，如：预算管理、自助装机、故障报修、IP 地址管理、ACL 管理、LVS 管理等。资产管理平台作为所有物理资源的唯一出入口，通过流程将预算管理、故障管理这些可能导致资产信息变更的环节打通。新采购的服务器录入到资产管理平台，服务器报废也必须经过它。通过资产管理平台，可以很方便地查询各种物理资源的使用情况。比如，一共有多少服务器、有哪些机房、机房的机柜分布情况、每个机柜摆放的服务器位置等信息。

服务管理平台

记录了业务运维所需的逻辑信息，提供一个基于树状结构（注：后续简称“服务树”）和权限绑定的管理模型。基于服务树和权限管理，实现域名管理、监控系统、部署自动化、环境

初始化等子功能。服务管理平台记录了多个维度的服务信息，比如，产品线内有多少台服务器；谁具备这些服务器的登录权限；产品线对外使用了哪些域名；服务器上部署了什么服务；服务运行的状态、版本、路径；服务都添加了哪些监控等各维度信息。

可以认为资产管理平台和服务管理平台的信息集合就是 ITIL 里的 CMDB（ConfigurationManagement Database）。由于每个运维子团队的分工不同，平台定位和用户场景不同，出于敏捷建设的考虑，我们将它拆分成了两个平台。资产管理平台的主要用户是系统运维工程师，他们关注设备的出入、维修等管理工作，交付资源给上层业务；服务管理平台的主要用户是应用运维工程师、研发工程师和测试工程师，他们关注服务运行的相关数据。虽然是分开的两个平台，但平台之间通过流程和 API 接口，实现了数据的相互关联。

资产管理平台负责底层的物理信息管理，提供 API 供服务管理平台查询和同步。服务管理平台通过 API 获取新交付的服务器列表及其详细信息，将它们归属到服务树产品线节点，分配对应的权限。应用运维工程师在服务树上领取空闲服务器，进行一系列的环境初始化、服务部署、监控添加等工作。应用运维工程师在服务管理平台提交报修申请、服务器归还等操作，通过 API 将信息推送到资产管理平台，由系统运维工程师进行相应处理。

使用场景介绍

两个平台负责所提供信息的准确性，对外提供 API 接口，可以供更上层的业务使用。基于这些信息，我们可以做更多智能化、自动化的工具开发。下面分享几个实际案例中的应用场景。

场景 1 Hadoop 数据存储管理

我们有大量的数据存储在 Hadoop 集群上，出于节省成本的考虑，我们将以前的 3 副本变更为 1.5 副本，降低一倍存储量。为了避免相同数据存储在同一个机柜的服务器内，降低由于单机柜断电或者同机柜服务器多块磁盘故障导致数据丢失的可能性，我们通过平台提供的 API，获取 Hadoop 集群所有服务器的机房、机柜分布和机架位置信息，在存储数据的时候进行合理的动态调配。

场景 2 智能报警合并

当服务器死机、机柜断电或接入交换机故障、机房断电或核心网络故障时，往往会收到大量的报警信息。我们可以通过平台提供的信息，对报警信息进行最大程度的聚合，减少报警发送的条目，而且能更好地帮助运维人员快速定位故障。当一台服务器死机的时候，通过监控项与服务器的关联信息，将这台服务器相关的 SSHD 监控、Nginx 监控等报警信息进行聚合，合并成一条服务器宕机报警；当一个机柜断电后，我们可以将该机柜下接入交换机交换机和每台服务器的报警进行聚合，合并成一条机柜或接入交换机故障报警。

场景 3 磁盘故障自动报修

在互联网业务中大数据应用已经很广泛，Hadoop 服务器数量占比很大，大量的数据计算导致磁盘故障率比较高，每天都有大量的故障磁盘需要更换维修。以前都是通过硬件监控或应用监控发现问题，然后由应用运维工程师登录服务器确认磁盘故障，尝试工具修复。如果修复失败摘掉磁盘，再发起故障报修申请。现在我们研发了磁盘故障自动维修系统，通过平台提供的 API 接口和监控系统联动，当监控系统发现磁盘故障后，通过回调接口启动磁盘工具进行软修复，修复失败后摘掉磁盘，并在服务管理平台进行记录，自动发起故障报修工单。服务器供应商收到维修工单通知后，根据所提供的机房、机柜、磁盘位置，进行集中更换。更换完成后进行通知，再由系统将磁盘分区格式化挂载，开始提供数据存储服务。

在运维平台建设的过程中，我们借鉴 ITIL 的思想，但没有完全照搬。ITIL 能够帮助 IT 部门提高用户的满意度和运行效率，但它的实施难度比较大，不能满足互联网运维的敏捷要求。我们希望贴近 DevOps 的理念，管理和提供准确的运维数据，封装各种灵活的运维工具，让运维工作前置到产品研发阶段，帮助研发、测试人员快速完成产品的发布、测试、上线工作，让运维工具在产品的整个生命周期中联动起来。

平台化不等于自动化，我们的平台更多的是通过流程和标准的保证，提供运维数据的可视化，还算不上真正意义的自动化。我们希望研发和运维人员不再需要关心服务具体部署在哪台服务器、哪个 IDC 中，由调度系统负责服务运行状态的监控，对资源进行合理的调度、伸缩，对一定范围内的故障进行自动处理，实现真正的运维自动化。

资产管理平台

资产管理平台是记录和管理企业 IT 架构中各种设备配置的信息平台。

在系统工作中，它与所有服务支持和服务交付流程都紧密相关，最大价值是支撑流程的运转。同时又依靠流程关联来保证数据的准确性。对上层服务管理平台，能通过 API 提供数据查询能力，如设备的硬件配置、机架位等信息；
同时也提供服务器的重启、报修和系统重装等接口。

系统运维的早期阶段，主要是用 Excel 记录各种信息，用邮件来做各种流程审批。

随着业务的快速发展，设备数量爆发式增长，系统工作中的配置变更越来越频繁，信息遗漏变得非常严重，不断产生脏数据，已经不具备准确性和可用性。
同时对上层系统的查询需求也无法响应，这种靠 Excel 维护信息的方式已经不再适用。

我们开始在资产管理平台研发上投入专门的人力，并和工作流程强关联起来，通过流程保证资产平台数据的准确性。资产管理数据的准确性和功能完备性决定了整个系统运维的自动化程度。因此，要支撑公司不断增长的服务器运营，首先需要建立和维护一套基础架构的资产管理平台。

资产管理平台主要规划了配置管理、预算系统、故障管理和自助系统四大部分，如图 7-1 所示。

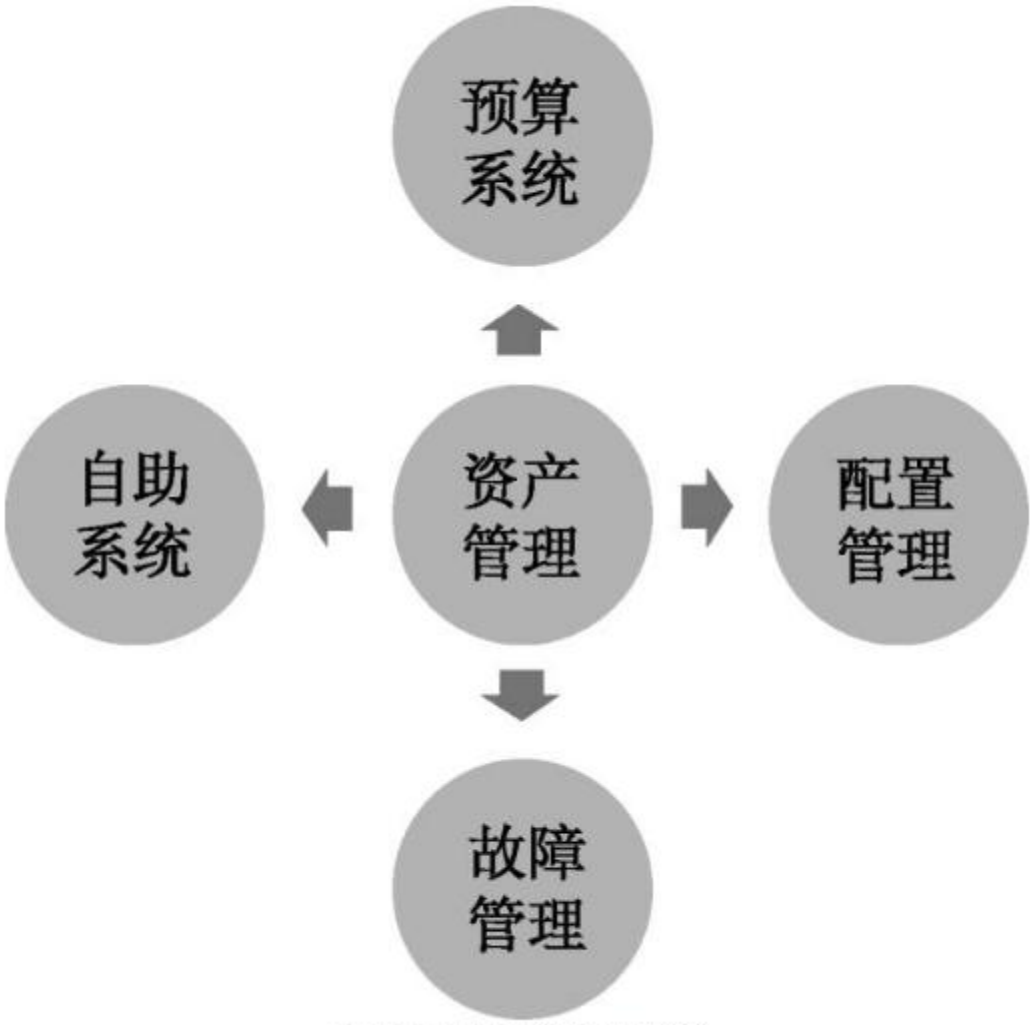

图7-1 资产管理平台规划的四大部分

业务通过预算系统提交服务器申请，系统工程师通过配置管理平台录入服务器信息，再通过 API 的形式将服务器信息同步给服务管理平台。

业务运维工程师可以通过调用自助系统对服务器进行重启、重装等操作。当服务器发生故障时，服务管理平台将服务器标记为故障节点，通过 API 的形式将服务器故障信息同步到故障管理平台进行维修，维修完成后故障管理平台也通过 API 将信息同步给服务管理平台。

当服务器要归还时，服务管理平台清除相关信息，通过 API 的形式将服务器同步给资产管理

平台。

配置管理

配置管理一直被认为是 ITIL 服务管理的核心，因此在系统运维中也是核心部分，它在结构上处于最底层。

为什么说配置管理的作用是核心的？举个例子。
某服务器已经到了报废期，我们需要下架这台服务器，需要有系统记录这台服务器的数据中心、机架等属性，也需要记录这台服务器的硬件信息，依靠这些信息才能找到这台服务器。

但仅有这些信息还不够，仍需要知道这台服务器的使用者（即业务归属），通知其业务方将业务下线，确认下线后，需要再通知数据中心现场人员下架服务器，同时还需要释放服务器相关的其他资源，比如 IP 地址、ACL 配置、LVS 配置等，这一系列信息都需要由配置管理系统来维护和保存。因此可以看到，配置管理是一项非常重要又多信息关联的系统工作。

早期我们同样是用 Excel 记录数据中心、服务器（资产编号、参数、采购时间、供应商）、配件、网络设备、IP 地址、ACL 等信息。

随着设备数量的增加，配置变更频繁，又没有统一上线和变更流程，经常出现记录和实际有差别。比如服务器机架信息核对不上，设备实际用途和记录不符，甚至服务器下架了 IP 地址却没有释放，导致 IP 地址占用和冲突等问题。

通过配置管理系统的研发，强化了这些资产和配置信息的管理，准确度有显著提高，系统工作以及与外部系统的自动化联动开始向前发展。

在配置管理系统中，我们将各类信息管理定义成小的功能模块，通过 API 接口串联起来，实现灵活有序的方式管理各类资源。在机柜管理上开发机柜视图功能，把服务器的机架信息、位置占用等录入系统，在系统中可以非常直观地了解服务器的摆放情况、整体机柜使用率。

设备上线前由系统自动分配摆放位置，设备下线后自动释放资源，从而减少了手动分配的工作量。机柜管理还有一些高级功能，在网络规划中按照安全级别划分出了基础服务、普通业务和安全隔离三个区域，系统会按照设定的分配规则给不同的服务器分配不同区域的机柜。

比如，Hadoop 等分布式业务对机架摆放有特殊要求，太分散不利于数据节点间的数据同步，太集中冗余度又不够，这些特别的需求都有特殊的分配策略。

通过定期扫描机柜使用率，可以及时调整和关闭空闲机柜，从而避免浪费。开发的 iHealth 具有运行功率实时监控功能，可以从服务器监控中采集电流信息，能统计出整个机柜的电量使用，进行机柜的超电预警。

在 IP 管理系统中，把内网、公网、VIP、管理卡都分别设定了不同的 IP 池，根据类型、机柜、业务的不同，通过管理接口分配和回收 IP 地址，从而减少了工程师手动分配和回收 IP 资源的烦琐工作，进一步保证了 IP 资源管理的准确性。IP 管理和 ACL 管理等模块都有关联功能，在 IP 地址释放前都会查询并释放相应的 ACL 记录。

为每台设备设定一个唯一序号，作为主键使用，确保设备的唯一性标识。服务器在记录中的占比最大，网络设备其次。在虚拟化部分中虚拟机也有唯一性标识，并和底层的物理服务器有关联记录，可以快速地查询物理服务器上承载了哪些虚拟机。和虚拟化部分有 API 关联，虚拟机的飘移会实时地更新记录。

为每台设备粘贴了对应的序号二维码标签。在收货、上下架、资产管理、设备报修等环节，通过扫描二维码的方式进行信息的变更和查询，极大地提高了设备管理的效率和准确性。二维码系统设有权限管理，比如现场的维修人员和系统管理员获取的信息量是不同的，主要是出于安全考虑。

经常会遇到 IP 地址随着服务器下线释放，但 ACL 没有及时清理的问题，久而久之，积累的 ACL 记录很容易达到设备的上限，存在一定的安全隐患。

我们遇到过一个真实的安全案例，两个区域隔离的交换机由于设备 BUG 的问题，在 ACL 条目达到上限后并没有任何错误提示，还可以添加 ACL 记录，但原有的 ACL 记录完全失效，导致两个区域的隔离失效，带来了安全问题。
随后我们上线了 ACL 管理功能，业务通过系统发起 ACL 申请，完成审批后由工程师进行变更操作；ACL 和 IP 资源联动，及时清理无效的 ACL 记录；对 ACL 记录匹配次数进行监控，对于长期未使用的记录进行人工检查和清理。同时，我们提供 API 接口，安全人员通过接口获取 ACL 记录信息，扫描验证 ACL 记录的有效性。

预算系统

在早期运维中，设备采购数量少，申请采购主要依赖邮件审批的模式。系统组负责整理汇总各个需求邮件，再进行统一采购上架，流程非常烦琐，而且容易有遗漏的地方。采购的进度也不透明，业务线无法及时获知进度。

为了简化和规范设备采购的方式，我们上线了预算系统，该系统作为设备资源的唯一入口，包括服务器、网络设备、配件等。

对于服务器资源，支持实体机、基于 OpenStack 的私有云、公有云以及整机租赁等多种资源的申请。私有云和公有云等通过调用 API 创建虚拟机，同步录入管理系统；对于租赁的实体服务器通过 API 下单；自购的服务器先从资产管理系统查询是否有满足需求的备机资源，如果没有再发起采购。

预算系统和监控系统的资源利用率对接，为业务申请采购服务器提供数据说明。当业务运维在采购管理系统发申请之后，审批人根据已有的资源使用情况来审核申请的合理性。

故障管理系统

随着服务器数量的增多，每天发生故障的服务器的数量也呈线性增长。早期系统工程师需要逐台定位服务器故障原因，和业务部门沟通停机时间以及跟进供应商的报修，效率非常低下，业务部门也常常因为故障服务器维修周期长而增加烦恼。为了改善故障持续增长造成的维修效率低下，开发了故障管理系统。

通过监控插件来上报故障信息，并完善插件使输出的故障信息详细、准确，能直接将信息发给数据中心现场或供应商来解决，比如磁盘故障会定位到槽位等。

通过服务管理平台将收集到的监控插件上报的数据信息展示在页面中，相应的负责人可以查询到自己的服务器故障信息，在负责人确认服务停止之后直接点报修。

故障管理系统通过 API 和服务管理平台同步故障信息，系统汇总整理后通过邮件或对接的 API 分别发送给供应商和数据中心现场。由于邮件中有供应商和数据中心维修需要的各类信息，所以基本上普通故障在两个工作日内就能修复，修复好后也无须人工确认，监控上报信息中自然会消除相应的故障信息。

如有需要手工干预的操作，也会通过监控系统体现出来，比如磁盘需要重新挂载和格式化等。故障都消失后，更改设备状态，即可交付给业务。故障管理系统大大提高了报修的效率，也提高了硬件监控的准确性和有效性，为线上服务的稳定运行提供了底层支持。

故障管理系统建设的上下游是应用运维和供应商，需要大家一起来协调工作，为了配合默契，一起约定梳理了一些标准。

① 故障信息的描述
早期报障需要应用运维人员填写描述信息，不统一、不标准，对诊断和定位故障影响较大。后来约定的方案是先将常见故障分类，实现自动化采集并分类。开发了监控和识别的程序，自动发现和告警给应用运维，由他们判断是否需要报障维修。针对常见故障以外的案例，通过人工分析和识别重新定义到故障类型中。

② 维修前是否要和应用运维确认
早期在维修服务器前要和业务沟通好时间，业务停止服务后再修，花在沟通和跟进进度上的时间非常多。维修前需要确认停服的根本原因是维修的周期较长，很多时候故障服务器仅仅是在排队维修，而这类服务器依然可以正常工作，所以业务方希望仅仅是在维修的时候再暂停服务。

后来达成共识，应用运维要停完服务后提交报障，系统组承诺在一定时间内修复完毕，若是无法完成，需要提供备机。当然，由于承诺了对不同故障类型的响应和维修时间，也就无须

反馈维修进度，应用运维也不会再询问，大大减少了沟通时间。

③ 和供应商定义诊断方式和处理机制
早期对于服务器故障定位都是系统组人员和供应商的工程师一起完成的，无法自动化，后来和供应商沟通，让其提供各类故障定位工具，并将相应的日志作为更换配件的标准，也和供应商对不同级别的故障响应和处理时间达成一致，方便批量报修。

自助系统

为了更方便地对服务器进行操作，开发建设了一些自助系统，主要有以下功能。

① 自助查询功能
当交换机发生故障或有计划割接时，可以通过交换机维度查询到同一接入交换机的所有服务器列表，从而达到快速的通告效果。业务线可以通过自助查询页面批量输入主机名查询服务器在数据中心中的分布情况，为业务的冗余性部署做准备。还可以通过源 IP 地址和目的 IP 地址查询 ACL 规则等。

② 自助重装、重启功能
在系统运维平台中开发了自动重装和重启功能，逐步地开放给了应用运维，应用运维在工作平台中可以直接调用 API 完成 OS 重装工作。发生故障时可以自助地重启、停机，提高了应用运维对故障的响应时间，也避免了系统工程师的重复性机械工作。

③ 屏幕截图功能
程序异常时可能导致 kernel panic，在这种情况下一般需要系统运维登录管理卡查看服务器屏幕状态并截图给业务定位故障。后来开发了屏幕截图功能，并通过 API 给服务管理平台使用，业务运维只要在网页中点击按钮就能看到服务器的屏幕截图，如图 7-2 所示。

图7-2 自助服务器屏幕截图功能

通过资产管理系统的建设，资产和配置的准确性得到了有效保障，系统运维的效率大幅提升，系统日常运维走出了依靠人工信息维护的初级阶段。

服务管理平台

在前面的章节中，我们讲到了自动化运维的重要性。运维基础信息的管理是自动化运维的前提，自动化水平完全取决于运维基础信息的准确性和完善程度。本章主要讲述我们在服务管理平台建设方面的一些思考和实践。

服务树

简介

在运维团队成立之初，运维人员只能通过 Excel 表记录所负责产品线的服务、服务器及之间的关联关系。在手工运维阶段，运维人员对服务或服务器进行操作时，需要经常维护和查询 Excel 表。随着业务的增长，服务和服务器越来越多，之间的关联关系越来越复杂，维护成

本也逐渐增加。这时需要一个简单的系统来记录此表格信息，并提供 API 给运维人员使用，可以理解为有 API 功能 Web 版本的 Excel 表格。同时，我们也需要把域名、监控、服务部署等信息通过此服务信息进行关联。

根据运维经验，我们设计了一个树状结构的数据管理模型，用来记录运维各个维度信息的管理。这样的系统我们称之为服务树。

服务树主要有三大功能：维护服务与服务器的关联关系；按业务组织成树状结构；与周边业务系统的联动。本节将围绕这三方面来介绍我们设计开发的服务树。主要有如下几个特点：
◎ 服务树是整合运维业务系统的核心组件
◎ 灵活的服务树结构和展示，满足运维人员的管理需求
◎ 多样化的数据记录和支持各种结构化的查询，联动周边系统

服务树的重要性

服务与服务器的关联关系，是运维业务中的基础核心数据。服务树管理着这样的关联关系，并且给上层业务提供接口和服务，自然成为了运维平台的核心组件。服务树本身可以理解为一个类名字服务的简易系统，侧重于运维人员的管理需求。通过服务树的节点可以较好地整合上、下游系统与工具，自动变更关联关系，提升自动化水平。

服务树的关联关系数据，是由部署系统自动维护的，上层业务系统只作为使用者。通过如图 15-1 所示的运维平台关系图，来协助用户理解“服务树作为运维核心组件”这个概念。

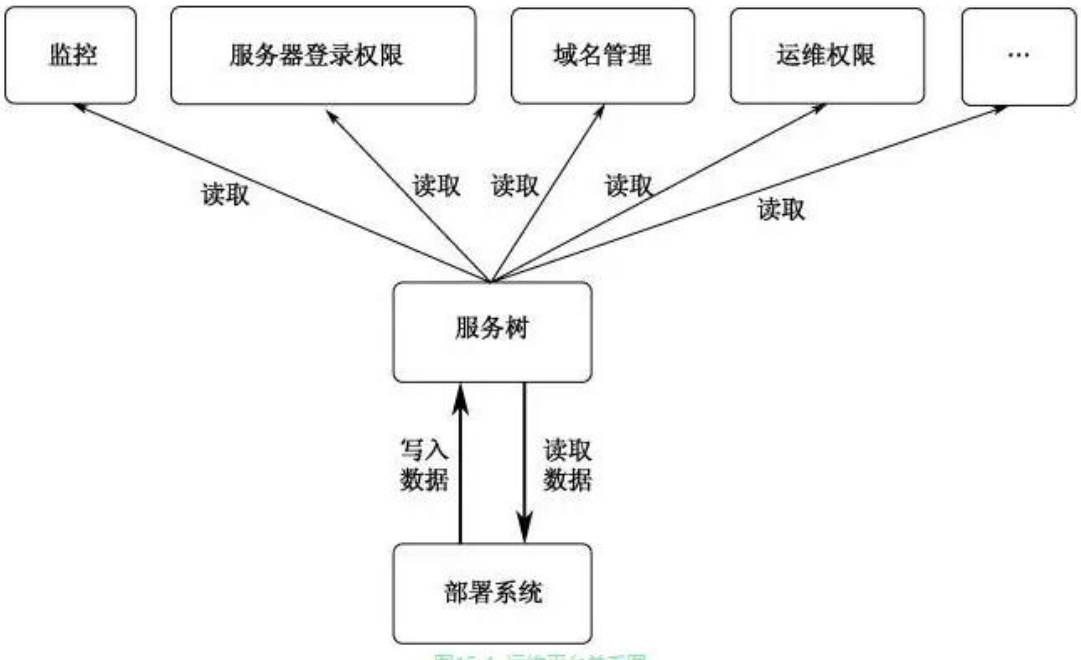

图15-1 运维平台关系图

部署系统是服务发布的具体执行者，是服务树数据来源的核心入口，作为服务信息的生产者。其他系统都作为使用者来获取此服务信息，例如：监控根据服务名获取服务器列表。当服务发布完成时，部署系统通知服务树完成关联关系的变更，监控根据服务名即可实时获取到最新的服务器列表。

设计实现

在介绍上层业务系统如何使用服务树节点之前，先了解一下服务树的整体设计实现。
服务树主要维护服务与服务器的关联关系，因此核心功能是将这些关联关系对外友好地输出。根据公司业务需求及运维积累的经验，为了支持灵活的树视图及查询方式，我们设计了以 Tag 为核心的服务树。其中 Tag 是一个类型和值构成的键值对，并分为两大类：服务相关及服务器属性。
为方便读者更容易理解，对 Tag 进行详细说明，如表 15-1 所示。

	Tag	说　明
服务相关	公司（Corp）	标识所属公司
	部门（Dep）	标识所属部门
	产品线（Pdl）	标识一级产品线，可根据自身业务划分
	集群（Cluster）	根据机房大小划分的逻辑区域，多个小机房可合并为一个集群
	服务分组（Service group）	一组服务的集合
	服务（Service）	配置、代码等基本信息可自包含的程序，并可独立部署
	实例分组（Job group）	一组实例的集合
	实例（Job）	服务的最细小单位，我们一般理解为服务所运行的实例
服务器属性	服务器状态（Status）	服务器在不同生命周期的体现。例如系统已初始化、线上服务中、无服务处于备机状态、已下线、已故障等
	机房（IDC）	服务器所处机房
	地域（LOC）	服务器所处地理位置
	类型（Type）	区别实体机、金山云服务器、OpenStack 虚拟机、Docker 容器、AWS 虚拟机等
	操作系统版本（OS）	操作系统的版本
	其他服务器硬件属性	

表15-2 Tag详细说明

服务相关的 Tag 需要组合在一起才能独立使用。例如 Nginx 服务 Tag，用户不能区分具体是哪个产品线的 Nginx 服务，会产生歧义。因此，为了解决“不产生歧义的服务名”，我们通过这样的格式使其具有唯一性：公司_部门_产品线_服务_实例。这种格式的 Tag 组合我们称为节点串。它是贯穿各个系统的通用描述方法。
由于服务树功能并不复杂，所以可以把数据表设计得比较简单些。我们是通过如下三个数据表来实现服务树的，如表 15-3 所示。

数 据 表	用 途
服务器	用于存储服务器的基本属性，如服务器名、机房、类型等基本信息
节点串	多个节点组合在一起才能独立使用。存储时会按照固定格式进行排序写入数据库。 当上层业务指定返回的顺序时，也会按照顺序重新排序组装后输出。
节点串与服务器的绑定关系	通过服务器表的 ID 与节点串表的 ID 进行关联，方便后续节点的迁移及变更。

表15-3 数据表及用途

服务树视图

服务树的一个重要功能是提供 Web 视图，满足用户的管理需求。视图是和用户交流最直观的一个纽带，同时也是引导和限制用户规范性操作的一个重要入口，进而减少程序的管理成本。服务树视图主要支持：服务层级关系的可视化展示，用户通过点击节点或搜索快速定位。

服务树的展示方式、各层级的先后顺序、节点层级显示与否，都可以由用户自定义。同一棵树，可以根据不同的配置、用户的需求，展示成不同的视图，如图 15-4 所示。

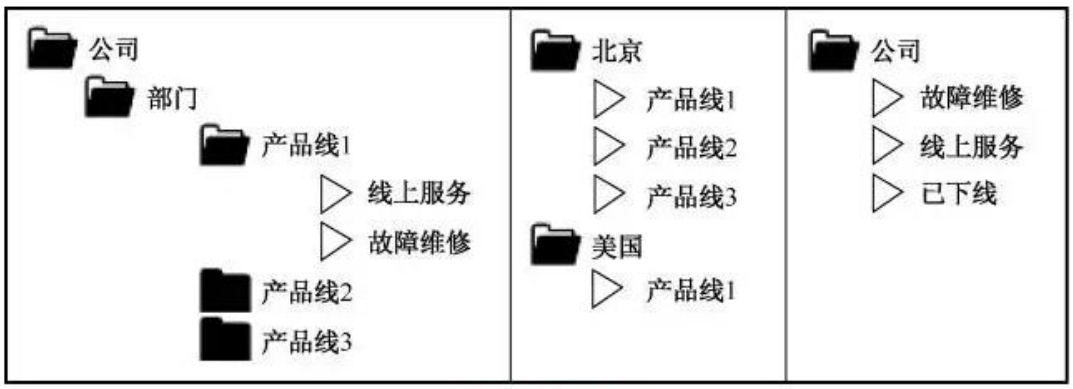

图15-4 服务树视图

不同的运维业务系统需要的视图也可能不一样。运维业务基本都是基于服务的维度进行管理，因此大部分业务（例如服务树、监控、权限系统）的默认视图树结构是公司->部门->产品线->服务组->服务->服务实例组->服务实例。

在部分业务中，为了减少节点变更带来的管理成本，我们需要引导用户将数据关联至产品线，例如域名管理、LVS 管理、部署等业务，视图树结构是公司->部门->产品线。在服务器登录权限集中管理的业务系统中，以默认视图进行展示，同时为了支持一些特殊的需求（例如，给系统组授予公司所有故障机器的 root 登录权限），允许用户在地址栏中修改树结构参数来刷新服务树视图，进而满足对应的特殊需求。

查询功能及命令行工具

服务树的另一个重要功能就是给上层业务和运维人员提供查询功能，主要包括两方面：通过服务信息筛选服务器列表和通过服务器反查询服务信息。

筛选服务器列表是查询某个服务部署在哪些服务器上。服务树提供接口给上层业务查询，同时也将此接口封装为一个脚本工具给运维人员使用。这个工具的输入参数是节点串，比如输入 corp.A_dep.B_pdl.C_service.nginx，可以得到 A 公司 B 部门 C 产品线下服务属于 Nginx 的服务器列表。当某个系统版本出现安全漏洞时，运维人员就可以通过类似 derp.D_pdl.P_os.centos63 来获取本产品线 CentOS 6.3 版本的问题服务器。

反查询是查询某台服务器上有哪些服务。同样也会提供接口给上层业务查询，并且也将此接口封装为一个脚本工具给运维人员使用。这个脚本的输入参数是服务器名，比如 xx-test01.bj，可以查询服务器的所属产品线及状态等基本信息。

目前较多的使用场景是根据节点获取服务器列表，并可以对有登录权限的服务器列表进行相应的操作。同时，公司内部的 Ansible 获取服务器列表也是基于服务树接口改造的，这样运维人员使用 Ansible 时就无须维护服务器列表了。

高可用性

服务树属于 Web 型服务，前端使用 AngularJS 和 HTML 开发，使用 Python 开发后台，数据存储在 MySQL 中。在运维平台业务快速发展的过程中，服务树的可用性将影响到上层业务的稳定性。

作为一个重要的基础服务，需要有非常高的稳定性。在稳定性方面我们是如何设计考虑的呢？首先理解服务树的业务特点：大多数时候绝大部分节点数据很少变动，获取实时数据的要求不是很高，允许有一定的延迟；但有些场景需要获取实时数据，比如发起部署时，部署 Agent 首先向服务树注册服务器列表，成功后再向 LVS 注册 Real Server。这时 LVS 就需要向服务树获取实时数据，用于判断服务器是否有权限注册 LVS。

因此，根据这样的业务特点，我们做了两件事情：完善服务端 API 的缓存，以及提供服务树的 SDK。

（1）服务端缓存

服务端缓存主要分三个方面。

◎ 计算数据结果的缓存。例如获取公司、部门级大节点的服务器列表，不需要实时数据，可以直接返回缓存数据。

◎ 数据库配置尽可能多的内存缓存。理论上，可以缓存大部分 select 语句的结果集。好处就是当数据进行变更时，MySQL 自身可以保证缓存数据和实时数据的一致性，进而减少业务代码处理缓存的复杂逻辑。

◎ 数据库多机房多实例。程序通过配置，优先获取本机房数据库，当本机房数据库异常时，程序重试连接耗时最短的数据库，尽量保证可以读取到有效的数据库实例。同时 MySQL 的数据库主从同步功能，目前已非常成熟，进而避免自身业务处理分布式数据的复杂逻辑。

（2）服务树 SDK
多机房多实例部署，是目前通用的高可用部署思路，失败时可以重试不同的实例来提高可用度。这里主要给大家分享 SDK 的思路。SDK 的好处在于可以嵌入对方的代码，部署在应用方的机器上，因此可以做一些缓存数据和重试功能。
◎ 配置多个域名，默认连接本区域的服务实例，允许轮询多个实例；
◎ 默认封装 HTTP 长连接，给出一些最佳实践的用法，引导用户批量、多次请求时使用长连接模式；
◎ 默认缓存响应数据，减少服务端的请求压力，同时当服务端都异常时，还可以使用本地有效的缓存数据，尽量减少因服务端宕机带来的影响。

小结

由于关联关系通过部署系统自动化关联起来，相比人工管理阶段，减少了人工维护成本，同时自动化整合上、下游。
服务树本身就是一个很简单的系统，设计和实现时应该简化设计思路。同时我们觉得此系统的难点在于作为平台底层核心，周边系统一旦依赖，改动代价就比较大了。这就要求设计产品形态时需要尽量考虑周全，同时还得考虑支持后续的业务发展。

运维权限系统

简介

在自动化运维体系中，服务树管理着公司服务器与服务的各维度关系，而运维权限系统维护着公司人员与服务权限的对应关系。通过对人员权限的准确控制，最大限度地减少由于权限管理问题而导致的各类安全风险。
运维权限系统经历了两个版本的迭代，下面分享相关的发展过程和经验。

第一版

在运维团队成立之初，运维平台的绝大部分功能都只面向运维人员，少部分功能提供给其他部门的同事使用。运维平台主要是由运维人员提炼需求，同时依据此阶段“简单、粗暴、有效、满足 80%场景”原则，快速开发完成的。

因此，我们的运维权限系统在此阶段具有两个特点：精简的权限模型和简单有效的人员管理。随着平台业务的发展，运维业务都与服务树节点进行关联，节点成为了贯穿运维相关业务的

“公共语言”。因此，我们的权限模型也是基于服务树节点设计的：服务树节点、功能权限、用户，这么一个简单的权限三元组就可以完成权限校验功能（见图 15-5）。其中功能权限只有读、写两种。

◎ 读权限：只对服务信息有查看权限
◎ 写权限：对节点下的所有数据（服务、服务器、域名等）有查看、执行、更改、删除操作权限

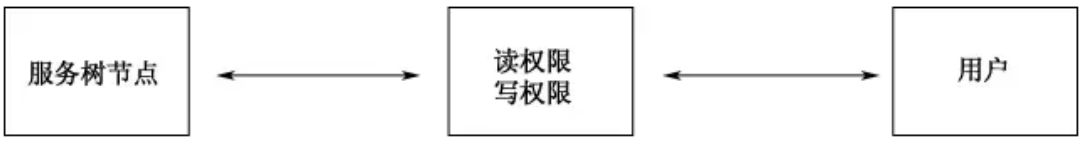

图15-5 权限模型

这时权限系统提供简单的页面进行展示及配置，如图 15-6 所示。

服务树节点	权限	用户
corp公司_dep部门1_产品线2	写	张三
corp公司_dep部门1_产品线2	读	李四
corp公司_status故障	写	小明
corp公司_dep云_产品线.hadoop	写	hadoop服务账号

图15-6 权限列表

权限系统使用这种方式来管理权限，并给周边系统提供用户权限校验服务，周边系统通过 API 对用户和节点的权限关系进行合法性校验。从理论上说，这个校验接口可以满足绝大部分需求，继而支撑运维平台的正常运行。

模型设计完成之后，就可以支持运维人员管理用户的需求了。但如果是针对每个人进行单独权限设定，当运维人员面对几十、上百用户规模时，权限管理将是一件非常痛苦的事情，最终无法管理。因此需要引入用户组的概念，我们把节点与权限的组合称为用户组。

此阶段，我们设计了“人员管理”的简单原则：只要用户在用户组中，这个用户就可以增加、删除此用户组的成员。这样的设计简化了权限模型，使人员管理的成本及实现逻辑非常简单。开通权限的整体过程一般是这样的：管理员首次开通用户组权限，后续就可以由组内用户管理了。

第二版

随着服务管理维度和运维需求的不断增加，第一版的权限模型逐渐不能满足需求了，因此我们设计了第二版权限系统。

1．新需求

随着公司的发展，产品线越来越多，研发、测试人员等公司员工快速增长，权限需求就逐步发生了变化。（1）部分运维工作需要研发或测试人员共同承担。例如，在测试环境下服务器的运维、系统重装、报警等基础运维工作，逐步由对应的研发工程师进行承担；Preview、Staging 环境的程序部署，也可由研发或测试人员负责。
（2）员工的快速增长，业务的快速扩张，需要更细粒度地划分权限，两个简单的读/写权限已不能满足要求。例如，用户可能只需部署系统的发布权限，但基于第一阶段的情况只能授予写权限（可以重启服务器、操作域名、登录服务器权限等），存在一定的安全隐患。运维平台是运维人员提炼需求，并前后参与完成的。研发人员的权限需由运维人员根据需求进行合理授权，避免出现安全事故。

2．设计概述

第二版重新设计了权限模型：服务树节点、角色模板、用户和业务权限点，如图 15-7 所示。

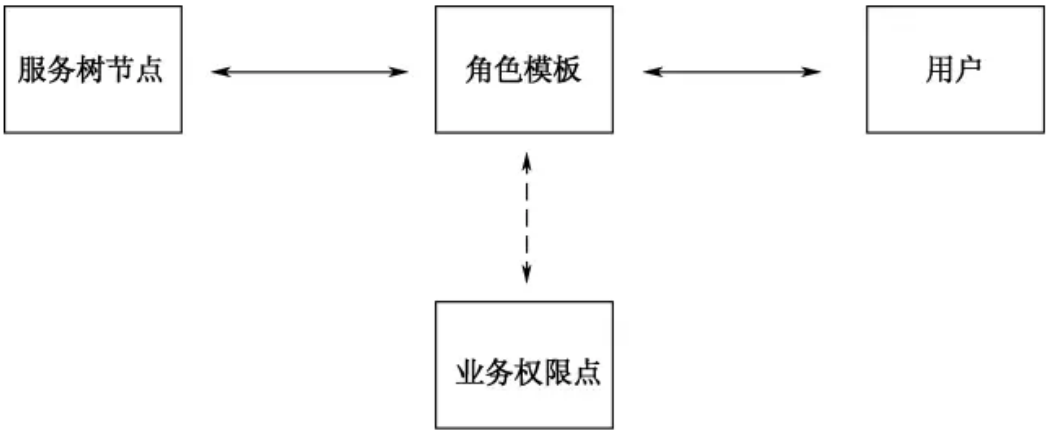

图15-7 权限模型

根据运维业务各个功能的不同，我们允许权限进行细分。不同的运维业务系统，各个操作都可以设置不同的权限点来表示。比如监控系统，可以有增加监控项、增加报警接收人、查看监控图、删除监控等多个业务权限点。为了减少业务系统的管理成本，我们建议业务系统尽量控制权限点的数量，例如服务器重启、服务器改名、服务器关机等相类似操作都用一个权限点来表示。

为了减少权限点授权管理的成本，我们引入了角色模板的概念。角色模板是一批权限点的集

合，根据日常的角色进行划分。为了方便运维人员分配、管理权限点，每个节点都可以绑定多个角色模板。我们的角色模板具有如下两个特点。

（1）继承角色模板：在上级节点绑定的角色模板，在下级节点也可以继承使用。同时允许下级节点调整此角色模板的权限点，其最大权限点由上级节点模板决定。
（2）自定义角色模板：权限系统给运维人员提供的公司角色模板，不一定能完全满足所有产品线的权限需求，因此运维人员可根据实际情况，自定义产品线的角色模板。

3．产品形态

第二版解决了只有读/写权限的问题，提供了更加丰富、更加细化的权限点，可以根据实际情况定制化角色模板。下面将详细描述这个产品的细节，给读者更加直观的印象。
如图 15-8 所示就是配置完角色模板与权限点后的整体概况。通过此图，方便运维人员给用户分配合适的角色模板，同时也方便管理员调整权限点。

系统	权限点	运维管理员 (corp.公司_dep.部门1)	云部门的运维人员 (corp.公司_dep.云)	Dev (corp.公司)	Dev管理员 (corp.公司)	运维人员 (corp.公司)	运维管理员 (corp.公司)
服务器登录	1. ⑦ rd	1. ✖	1. ✖	1. ✖	1. ✖	1. ✖	1. ✖
	2. ⑦ work	2. ✖	2. ✖	2. ✖	2. ✔	2. ✖	2. ✖
	3. ⑦ root	3. ✔	3. ✔	3. ✖	3. ✖	3. ✔	3. ✔
权限系统	1. ⑦ 读	1. ✔	1. ✔	1. ✔	1. ✖	1. ✖	1. ✔
	2. ⑦ 管理"Dev"	2. ✔	2. ✔	2. ✖	2. ✔	2. ✔	2. ✔
	3. ⑦ 管理"Dev管理员"	3. ✔	3. ✔	3. ✖	3. ✖	3. ✔	3. ✔
	4. ⑦ 管理"运维人员"	4. ✔	4. ✖	4. ✖	4. ✖	4. ✖	4. ✔
	5. ⑦ 管理"运维管理员"	5. ✔	5. ✖	5. ✖	5. ✖	5. ✖	5. ✔
机器管理	1. ⑦ 读	1. ✔	1. ✔	1. ✖	1. ✖	1. ✖	1. ✔
	2. ⑦ 节点操作	2. ✔	2. ✔	2. ✔	2. ✔	2. ✔	2. ✔
	3. ⑦ 操作机器	3. ✔	3. ✔	3. ✖	3. ✔	3. ✔	3. ✔

图15-8 角色模板详情

根据工程师的实际使用情况，目前我们公司整体分为两类角色：运维人员和非运维人员（用 Dev 表示）。权限系统的所有操作都是通过权限点进行控制的，管理员可随时调整每个角色模板的权限。

角色模板配置完成后，就需要给用户配置权限。如图 15-9 所示，为了方便用户快速查询权限信息，我们以表格的形式进行展示。授权时支持运维人员单个或批量操作，同时也支持用户以正则方式快速搜索权限信息。

权限模板是一批权限点的集合，运维人员可根据用户对服务的掌握程度进行调整。例如小王

是运维部新入职的同事，运维管理员可先授予其 Dev 角色的权限。随着小王对服务的逐渐掌握，运维管理员就可以将部分运维事物交由小王负责，可以通过授予小王运维人员或运维管理员的角色完成。

图15-9 权限列表

如所图 15-10 所示，运维管理员角色模板关联了几乎所有的权限。节点表示此角色模板所生效的范围。如果填写的是公司节点，此角色模板将适用于公司所有节点；如果填写的是部门，则只能适用于部门内部的节点。

图15-10 角色模板

我们允许用户根据自身产品线的实际情况，对上级节点角色模板的权限点进行缩小或调整，如图 15-11 所示。

图15-11 调整权限点

也允许用户新增产品线的自定义角色模板，此角色模板的权限点可以不受限制并由运维人员自行选择，如图 15-12 所示。

图15-12 自定义角色模板

与外部系统交互

权限系统作为平台的基础组件，管理着公司人员与服务的关联关系。为了保证上层业务的正常运转，运维权限需要对外部提供灵活和高可用的服务。如下是我们对外提供的 5 个接口：

```
# 检查权限
check(节点，权限点，用户名)
# 获取节点对应的成员列表
members(节点，角色模板)
# 通过节点发送邮件
mail(节点，邮件标题，邮件内容，邮件发送者，角色模板)
# 通过节点发送短信
sms(节点，短信内容，角色模板)
```

# 将操作日志发送至服务端统一存储、展示、分析及审计
event(事件名, 请求 IP, 用户名, 服务器名, 节点串, 详情备注)
我们把用户的运维操作行为通过 event 函数统一收集起来，如图 15-13 所示。目前已对外提供简单的文本查询，后续将会收集更详细的数据，并提供友好的界面查询及展示。

图15-13 运维操作审计

同时我们还允许上层业务系统，通过接口和服务账号来自动变更有权限的相关数据。例如，DBA 在数据库平台部署数据库实例时，通过调用权限系统的接口就可以对数据库及服务器的登录进行实时授权及变更。也允许公司内部 PaaS 平台通过接口实时授权相关人员登录 Docker 容器。

小结

在我们公司，依赖运维相关资源（服务器、域名、IP 地址等）的系统，基本都会和服务树节点进行关联。服务树和权限系统所构造的运维基础体系，更好地管理和对接各个业务系统。

服务器生命周期管理

一台服务器从业务提出预算需求，到最终可以提供线上服务，要经过很多部门的共同协作配合才能完成。随着业务的发展，对服务器的需求也在不断增大。因此采购变得越发频繁，各业务购买的服务器数量也很多。如果整个过程还靠人来协调各部门协作，靠人来完成一些重复过程的操作（如安装系统、系统参数调整、业务依赖基础软件安装等），将会导致效率的极度下降。最终会因服务器交付延期而导致服务出现问题，甚至影响新功能上线，影响业务发展。

随着时间的推移，服务器逐渐老化，故障概率增大，故障频度也会增加。如果服务器规模较大，那么处理服务器故障更是家常便饭。在处理故障服务器时，运维工程师需要预先做很多工作，比如协调备件或新服务器，将故障服务器从集群中摘除，调整监控等。如果服务器只

修不换的话，还需要进行持续的状态跟进、记录，以免后续忘记。整个过程非常耗费人力。

正因为服务器的采买、安装、运行、故障、归属调整等时刻都在发生，但过程又是如此的繁杂，所以对服务器平台化管理的需求显得非常急迫。而纵观运维自动化系统，服务器是一种非常明确的实体资源。自动化服务管理的关键能力之一就是通过调度实现服务器资源的调配，当服务器资源不足时，可以自动调配服务器，进行服务部署，并将其注册到服务集群中。而当服务器资源过剩时（如深夜），又能自动减少服务器，将过盛的资源进行归还或调度到其他服务。

服务器在各个阶段会有不同的状态，相关系统会根据状态进行相应的处理。这些状态的变化有些是人通过工单系统触发的，有些是系统执行操作后自动改变的（初始化系统、部署系统、监控系统）。我们将运维场景中产生的各种状态进行梳理，明确每个状态变更点的触发条件、保障流程等，最终形成了对服务器全生命周期的闭环管理，也就是所有状态均在平台中通过特定动作改变，不会出现人为干预而导致的服务器状态错误。这对自动化管理系统非常重要。试想：如果一台服务器正处于“维修”状态，而这时却因为使用系统操作而变为了“可使用”状态，那么它将会被调度系统分配给线上业务进行使用，后果可想而知。

在本节中，我们会介绍如何通过系统管理服务器状态，以及需要哪些前置条件等。对服务器状态的自动维护、状态的触发是工单系统、监控系统自动发现等。前置条件需要备机池，需要产品线临时机器池。

服务器状态

服务器从业务提出需求到为线上提供服务，再到故障维修、下线归还，主要经过几大环节，包括：采购、装机、服务准备、服务中、故障、下线。每个环节会有多种子状态，涉及不同的自动化系统。

1．采购

预算、采购主要是通过预算系统来完成的。工程师通过该系统提交服务器需求，各团队在此系统中完成审核、审批、生成采购单等工作。采购包括审批中、采购中、已到货三种子状态，这是由人来更新的。运维工程师可以通过此系统跟进服务器的购买进度。

2．装机

服务器到货后，会进入装机环节，系统运维工程师会进行网络资源分配、系统安装等工作，对应的子状态为“网络资源分配完成”、“系统安装完成”。这些动作都是通过相关系统完成的，涉及网络资源管理系统、自动装机系统。

3．服务准备

当系统工程师将系统装好后，服务器会进入到服务准备环节，此时服务器开始按业务的需求进行服务器命名修改、系统环境私有参数修改、服务程序部署。两种子状态分别为“私有环境初始化完成”、“服务部署完成”。此时，虽然服务器已经搭建好，但并未加入到在线集群中。一般构建好一个服务环境后，还需要进行相应的测试。

4．在线服务

线下测试通过后，准备阶段完成，该服务器会被加入到线上集群中，提供线上服务。此时进入到在线服务环节，子状态变为“服务中”。

5．离线

如果服务器资源出现过剩，则需要释放资源，我们会将服务器归还公司。服务器进入离线环节，子状态为“离线中”。在调度系统中，这种状态的机器被定义为健康的、可随时使用的资源，是被优先筛选的。

6．故障

在服务器出现故障时，我们会将其子状态设置为“故障中”，此时服务器进入故障环节。对于故障的处理方式有两种：停机维修和更换服务器。SRE 会根据情况进行选择。如果故障服务器的内存等易换件坏了，我们会选择停机维修，主要是更换机器就涉及应用、数据的迁移，考虑到运维成本，这是没必要的，但这种维修一般需要的时间可能会偏长，需要业务确认是否可以容忍。如果服务器主板等维修起来很麻烦的部件坏了，一般会选择更换服务器。

对于硬件故障的判别，我们主要靠带外监控来主动发现。还有一种情况是从业务上判断服务器整体性能出现下降，不如其他同型号服务器，我们可能不知道是什么原因导致的，这时就会选择更换服务器。具体的故障点系统工程师线下再去追查。当服务器修好后，会自动进入装机环节，重新安装系统等待调度（注：有一种情况不会进入到装机环节，而是进入到服务准备环节，具体场景在业务临时机器池中说明）。

7．报废

这种状态一般 SRE 不太关注，系统工程师会根据服务器的年限、故障情况设置其状态。
具体状态如表 15-14 所示。

环　节	子 状 态
采购	审批中
	采购中
	已到货
装机	网络资源分配完成
	系统安装完成
服务准备	私有环境初始化完成
	服务部署完成
在线服务	服务中
离线	离线中
故障	故障中
报废	已报废

表15-14 具体状态

在机器管理中，需要两个服务器池来放置不同状态的服务器，即公共备机池、业务临时机器池。

前置条件

在服务器管理系统建设中，需要两个服务器池来存放不同状态、不同业务归属的服务器。

1．公共备机池

这里的服务器不属于任何业务，是公共资源。服务器都处于系统安装完成状态。当某个业务资源不够时，调度系统会从中调配服务器。也可以通过系统来进行筛选，并通过工单系统发起备机申请流程，进行服务器借用。最终归还时，也会划分到公共备机池中，继续供其他业务调配使用。进入到备机池的服务器都会被强行重装系统，避免之前业务修改的系统参数影响别的服务。

2．业务临时机器池

这里的服务器属于特定的业务，是私有资源。主要是为两种场景设计的：一是特定业务定向采购的服务器，当安装完操作系统后，会自动进入业务临时机器池中，对应的服务器状态为“系统安装完成”，工程师可以随时使用；二是服务器故障，需要停机维修，因为选择了保存业务环境，所以修复后其依然归特定业务所有，此时服务器状态为“故障中”。在服务器维修好后，不会进入到装机环节，而是进入到服务准备环节的“服务部署完成”状态。在测试通过后，会由人触发，将其加入集群提供服务。如果这个步骤接驳自动化测试，则可以自动完成，但目前主要还是人工干预。

服务器管理系统

在系统设计阶段，我们对服务器管理系统提出了三大原则，以保证该系统可以作为底层服务可靠地运行。

（1）灵活扩充环节与状态。主要因为公司的组织结构有可能发生变化。这时有可能影响整个服务器生命周期涉及的团队及流程。举例来说，公司出于安全考虑，新出规定要求所有报废的服务器必须由安全团队进行审查（审查可能是人工完成，也可能是系统完成）才能最终得到“报废”状态。这时我们就需要增加安全审查环节，制定相应子状态并与其系统对接。

（2）流程的闭环。服务器状态的正确性是非常重要的。一旦状态出现混乱。我们将不知道哪些服务器可用，哪些不可用。调度系统也会将有问题的服务器提供给线上使用。因此，所有状态的变更必须通过明确的平台流程进行控制。主要是为了确保状态信息的准确，不会出现“遗漏”或“配错”。例如大批服务器到货，系统运维工程师安装完操作系统后，会交付给各业务的运维工程师，如果靠人来通知各业务服务器列表，再由各业务手工改变服务器状态，从历史来看一定会出现遗漏。如果自动装机系统能够自动触发状态变化，运维工程师只要查看自己业务的临时机器池中是否有新机器即可。不需要人在中间进行执行，那将不会出现实施层面的遗漏。

（3）提供统一接口获取资源。无论是实体服务器资源还是虚拟机资源，也不管是服务于公司的私有云还是对外的公有云，都需要通过统一接口进行资源申请，获取到资源后再自行安排使用。因为如果是多套系统或多种申请方式，将导致工程师要进行频繁的工作方式切换，降低工作效率。自动化系统也要考虑各个服务器管理系统的融合（这本不该是自动化系统考虑的）。更重要的是，服务器管理涉及业务使用资产成本的问题，不统一管理，将出现资产的混乱。

如图 15-15 所示是服务器状态图，我们会针对一些关键流程进行说明。

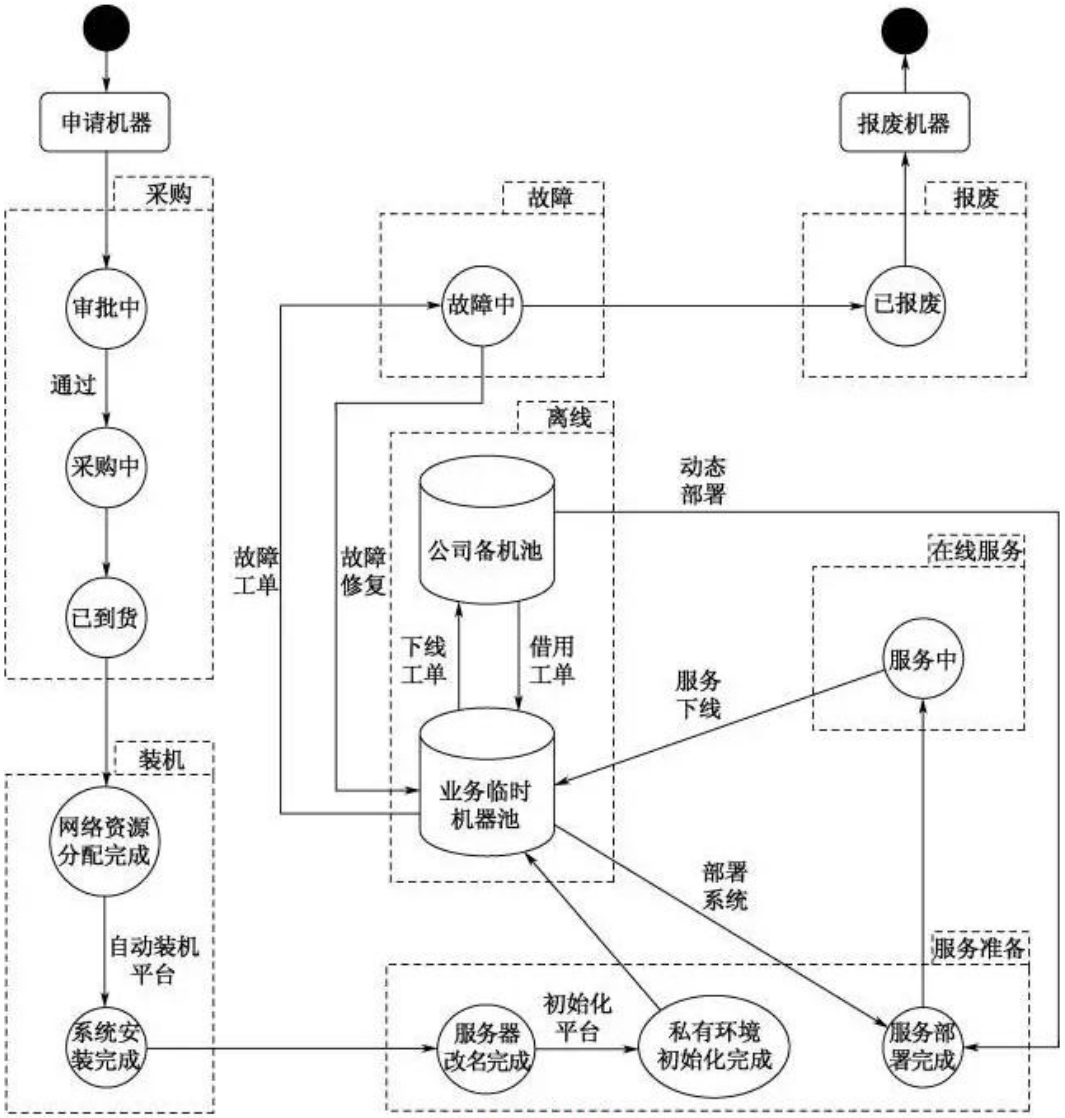

图15-15 服务器状态图

1．私有环境初始化

服务器在系统安装完成后，只有一个临时名称，在业务临时机器池中一般命名为music-tmp100。music 代表业务线，tmp100 代表业务临时机器池内的服务器及数量编号。在公共备机池中的会被命名为 sys-buffer199。不管服务器来源于哪里，业务都需要对其进行私有环境的初始化。我们会将其按业务需求进行改名，如 music-fe30，规则同上，这个名称确定了其在 music 业务中用于什么。这里可能有人会问：为什么不在预算环节就定义好服务器名称，这样就可以省了这一步？这是因为在采购的这段时间，线上服务器也在发生着变化，可能会导致与预算服务器定义名称冲突，从而带来未知问题。

服务器改名完成后，开始进行业务私有环境部署，如 JDK 升级、Python 升级等，我们称之为 runtime，也就是业务运行时所需要的依赖。我们是通过 Ansible 自动完成这个动作的。此时服务器状态被设置为“私有环境初始化完成”。
基础环境准备好后，就可以通过部署系统部署服务程序了。只要有一个程序部署成功，系统就会将服务器置为服务部署完成状态。这里可能大家会有疑问，一台服务器上可能部署多个程序，为什么只要一个部署完成，即会改变状态？这样设计的原因主要是为后续调度系统考虑，它会根据线上情况自动决定服务器上部署什么，所以哪个程序部署成功，即代表其提供什么服务。

2．服务中

测试通过后，就可以提供线上服务了。这种状态目前由人来触发，后续接入自动化测试后，可自动完成。

3．服务器下线

下线分为两种场景。
（1）服务器归还公共备机池。这类服务器一般是业务较长时间内不再需要这些资源，所以释放给公司，作为公共资源。资源统计系统也不再将其计为该业务成本。
（2）服务器放到业务临时机器池中。这种情况是业务短时间内还需要使用这些资源。如果放到公共备机池中，有可能被分配走，所以临时放置在这里。但资源统计系统会定期筛查，超过一段时间没有使用，会自动将其划分给公共备机池。

4．服务器借用

当业务自己的临时备机池筛选不出服务器时，会考虑借用，来源统一为公共备机池。即便是两个不同业务线之间借用服务器，也要遵守这个规则。被借方需要将服务器归还公共备机池，需求方再从中借用。业务线之间私自调换服务器的行为一定要控制，否则在资产管理上将会出现混乱。

5．服务器故障

故障的处理分为两类。
（1）更换服务器。主要适用一些不好修或不确定问题的场景。这个过程比较容易，分配一台新的服务器即可。但需要初始化、部署服务等环节。而旧的服务器下线修好后，会被放置到公共备机池中。
（2）故障维修。如果只是硬盘、内存等坏了，一般会选择这种方式。迁移成本比较小。这

种处理方式，在系统修好后，还属于原来的业务，启动服务即可。

环境初始化

环境初始化的需求是这样产生的：
（1）由于基础设施的区域化自治设计，部分配置在不同区域有差异。比如内网 DNS 区域化自治，不同区域的服务器，在 resolve.conf 中配置的服务器 IP 地址不同，需要进行环境初始化。
（2）随着运维基础设施的建设，每台服务器上越来越多的 Agent 需要安装升级，升级频度远比装机包的更新高，因此也必须额外进行环境初始化。
（3）业务应用程序，对系统环境有特定的需求（库、语言环境、内核参数等），但这些特定的环境需求又无法统一，这是环境初始化最重要的需求。

传统模式的环境初始化，通常使用业内流行的配置管理工具完成，如 Puppet/Chef/Ansible 等。常见的批量管理工具大致分为 Agent 模式和非 Agent 模式两类，在性能、可靠性、易用性、可维护性等方面各有千秋。基于以下几方面考量，我们选择非 Agent 模式的开源工具 Ansible 作为统一的批量管理工具。

◎ Agent 维护成本。Ansible 使用 SSHD 作为 Agent，不产生额外维护成本；省去了 Agent 端的安装、升级、进程守护等额外的管理成本。
◎ 权限和认证管理。Ansible 的设计模式使其天生就具有认证功能，而中心控制模式容易与现有权限管理系统进行对接，实现基于用户身份的认证和访问控制。
◎ 功能可扩展性。模块化实现使扩展功能更容易，并且中心控制模式使扩展功能的升级更加便捷，利用扩展模块与其他自动化系统接驳和联动，使 Ansible 更加易用。
◎ 易用性。Ansible 的使用方式更接近于传统 SSH，更适合支持传统的运维管理方式。
◎ PlayBook。基于 PlayBook 功能，容易将基础环境配置、常用操作等固化，并结合初始化平台进行主机初始化或统一变更。

ZYXW、参考

《运维之下》
第三章、运维平台
http://mp.weixin.qq.com/s?__biz=MzIxMzIyNDkyMw==&mid=2654107988&idx=1&sn=cad957b7fde78888e2d7696b40c5a7b3&scene=21#wechat_redirect
第七章、资产管理
http://mp.weixin.qq.com/s?__biz=MzIxMzIyNDkyMw==&mid=2654108121&idx=1&sn=f4334e1237954a605af92092536bd2aa&chksm=8c7cc56dbb0b4c7b5db1cb4dd396e5b3411321406323545e02ec5aa5293d07174953683c1542&scene=21#wechat_redirect

第十五章、服务管理平台
http://mp.weixin.qq.com/s?__biz=MzIxMzIyNDkyMw==&mid=2654108400&idx=1&sn=c0488b907b176f4ef22b8b633913db87&chksm=8c7cc644bb0b4f521ee2e851ccb250f1ea718449758d2c356c90d1a3190282386ff3fb10e597&scene=21#wechat_redirect