设计文档：面向云原生的下一代微服务集群监测机制，涵盖Kubernetes、Nacos等

参赛队伍：星轨

队伍成员：陈才，蔡建怿，程兴源

概述

伴随着微服务技术的成熟与云原生的演进，微服务已经成为很多应用系统开发和部署的银弹。而服务集群规模的膨胀，服务关系的日益复杂，部署方式的灵活多变则给可观测带来了新的复杂度与挑战：如何适配不同的运行形态，如何有机地将各个维度的数据聚合起来并进行展示，如何合理展示集群中复杂的关联关系……

Dubbo定位为一款微服务框架，在RPC的基础上增加了许多开箱即用的服务治理功能，赢得了很多用户的选择和支持。一直以来，Dubbo Admin充当着Dubbo服务集群的观测组件这个角色。但时至今日，Dubbo Admin的能力越来越受限，这主要体现在三个方面：

1. 用户对于一些开源观测组件（如prometheus，grafana，opentelemetry）的使用越来越多，而在Dubbo Admin现有的框架下无法很好地拓展融合这些开源组件，无法满足用户的需求。
2. 容器和k8s的使用越来越广泛，而这一部分运行时数据无法通过拓展Dubbo Admin来获取到。
3. 用户对于流量管控的需求日益增加，虽然Dubbo 本身具备了流量路由的能力，但用户在Dubbo Admin没有用户友好的方式来下发流量规则，这就导致了用户的学习成本很高

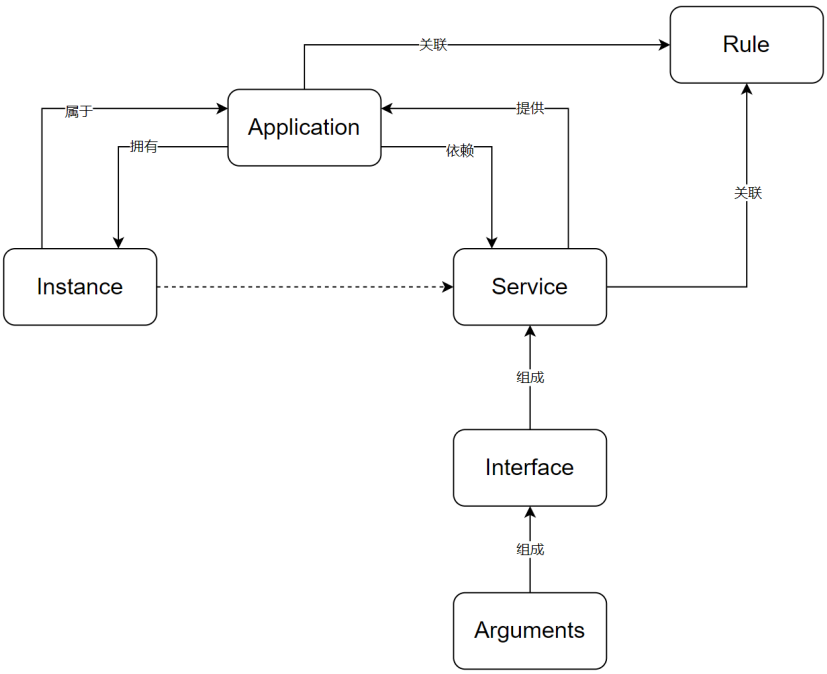
因此从用户的视角触发，对于新一代Dubbo服务集群控制台——Dubbo Console，在Dubbo Admin原有的基础上，应该有如下目标和要求：

1. 拥抱可观测领域的开源标准，仅需少量配置即可接入标准组件，增加更多的观测手段，拓宽观测边界
2. 适配不同的服务运行形态（VM，k8s），提供统一的服务数据观测视角
3. 优化流量规则的查看搜索，编辑下发等流程，以更友好的方式展现给用户

方案设计

领域模型

针对上文提到的三个目标，我们先从Dubbo的领域模型以及云原生微服务体系出发，梳理一下现有的概念：



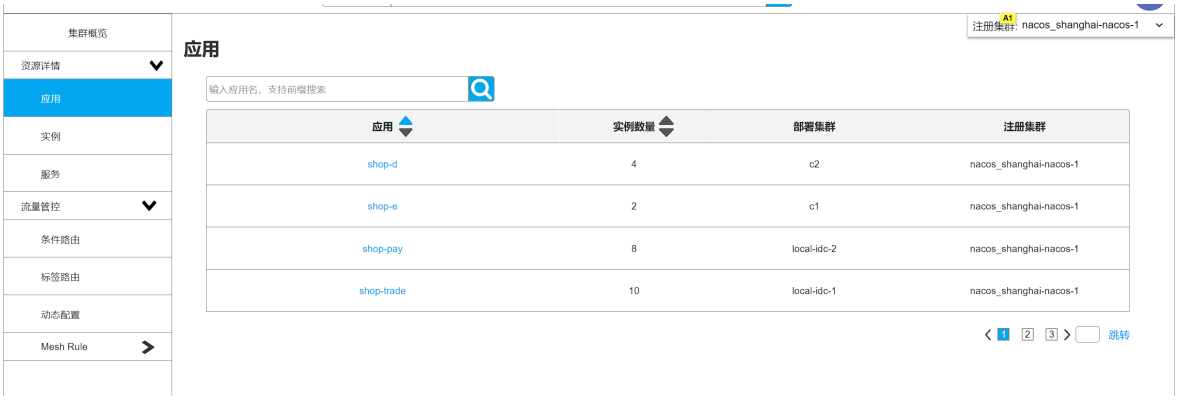
在微服务体系下，应用即一个微服务，可以独立部署和独立运行，通常会以多实例部署的方式来实现高可用。一个应用可以对外以接口的方式提供自己的服务，同时也可以消费其他应用提供的服务。因此应用其实是一个聚合点，实例以及服务都以应用名来作为聚合焦点。在此基础上，流量规则可以基于应用或者服务，来调节集群内的流量走向，以满足灰度发布，金丝雀发布，A/B测试，降级熔断等场景需求。

在云原生场景下，在应用和实例中间多了一层“工作负载”的概念。工作负载一般管理一组实例，同一个的应用的不同工作负载管理的实例通常在版本等方面有差异。因此工作负载可以简单理解为适用于单应用多版本管理的工具。但工作负载的概念可能对于大多数用户会比较难理解，且在虚拟机的部署环境下是没有工作负载这一概念的，因此我们将工作负载简化成了“版本”。

在梳理清楚概念后，我们从前端，后端，监控&链路追踪三个方面对Console进行了拆解和设计。

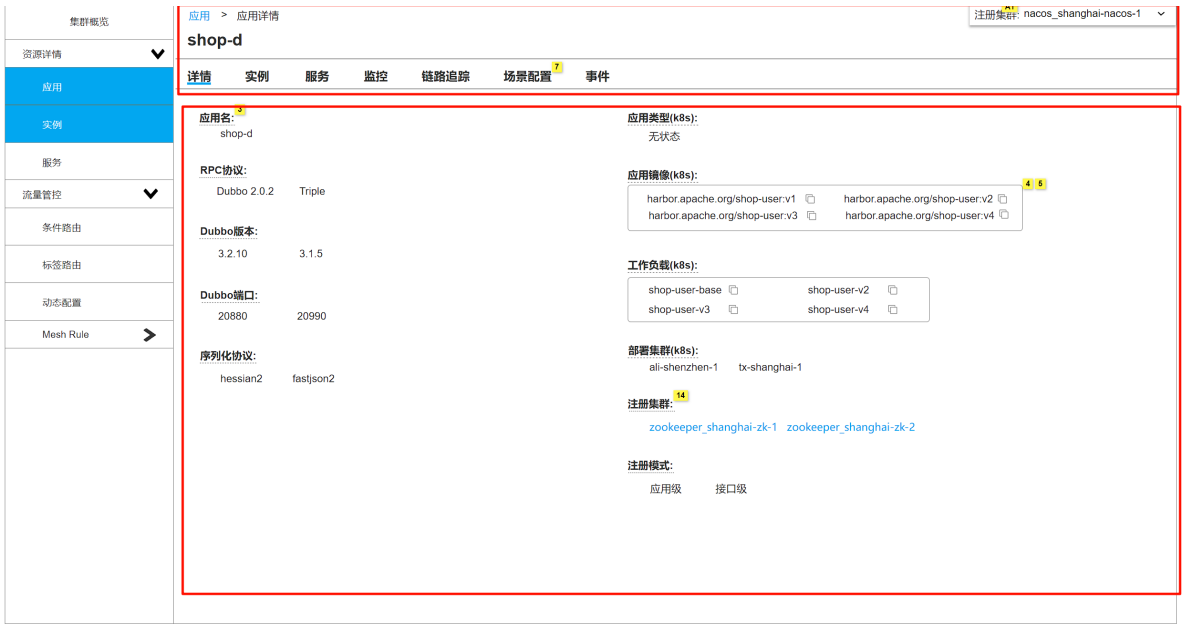
前端设计

前端设计上紧贴领域模型。从大框架上来看，Dubbo的领域模型中主要包括的Application（应用），Instance（实例），Service（服务），Rule（规则）这四个对象，被划分到资源详情这一个一级菜单下，而流量管控这一栏则包含了Dubbo原生支持的一些路由规则：

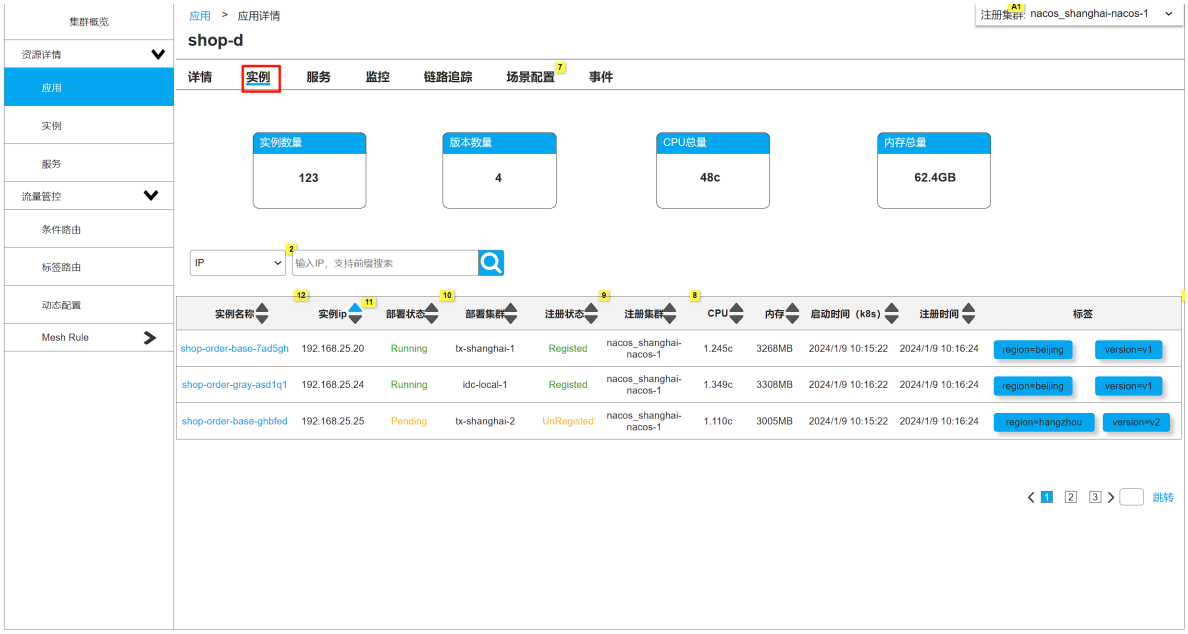


下面以“应用”为例，阐述其设计思路以及用户故事。从用户视角来看，如果需要查找某一个应用，则点击左侧菜单栏进入应用搜索，选中一个应用。选中后会进入到应用页面，进入应用页面后上下文就切换到了当前应用，也就是说这个页面的所有东西都和这个应用关联上，用户在这个页面流量就大致掌握当前应用的运行状态。

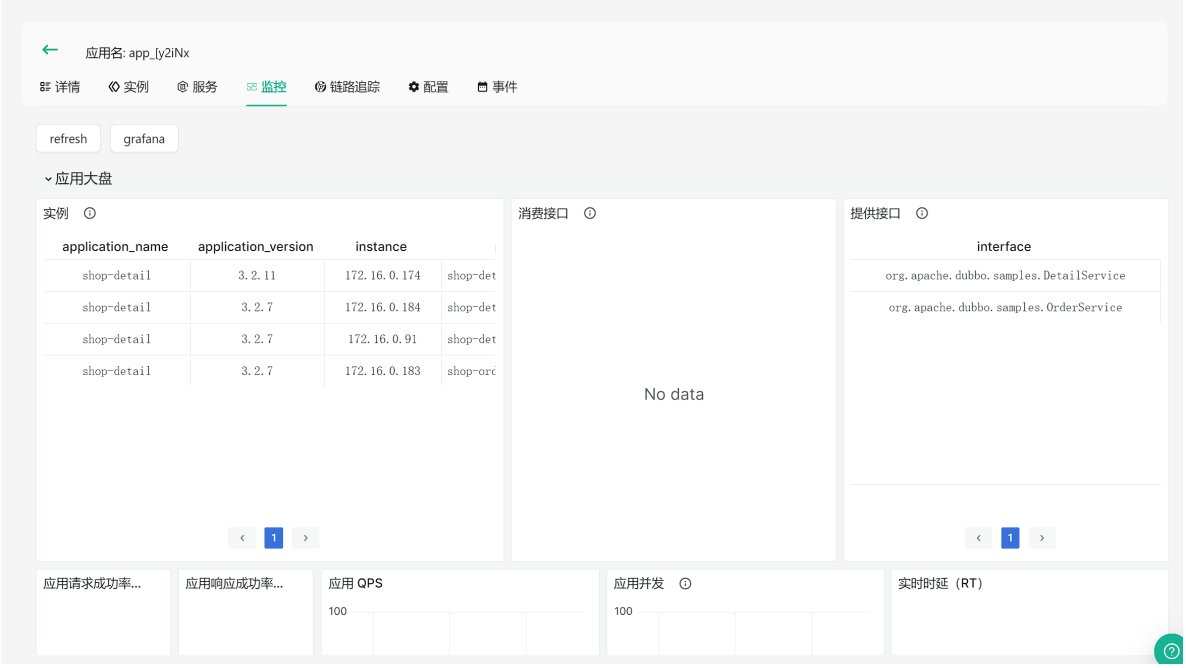
页面从上至下可以分为两块，上块展示的是应用名以及Tab栏，下块的主体是Tab页，展示的是每一个Tab对应的信息。

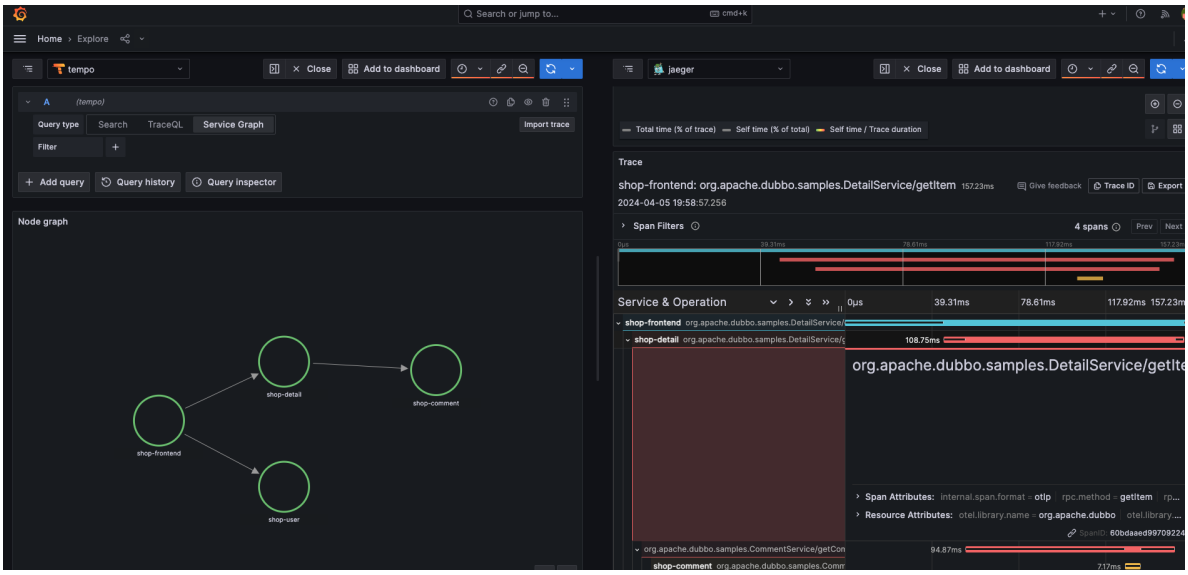


就应用而言，一个应用“拥有”实例，同时它提供一些服务或依赖一些服务，因此它和实例与服务都是有关联的，在原型图中集中体现在”应用>实例“以及“应用>服务”这两个tab页。其中，实例tab页展示的是这个应用拥有的实例信息，用户能够在这里看到实例的一些重要信息，如部署状态，注册状态，注册时间等等，并可以从这里跳转到实例页面以进一步查看实例更多的信息。服务tab页展示的是这个应用提供以及依赖的服务，用户能够在这里看到每个服务的监控数据如QPS,RT,RequstTotal等，方便查找出热点接口。



对于监控以及链路追踪这块，tab页以嵌入grafana的形式展现应用维度的详细metric和trace，用户只需在启动时配置好grafana，metric后端地址，trace后端地址，然后在前端就可以直接展示对应维度的metric以及trace。





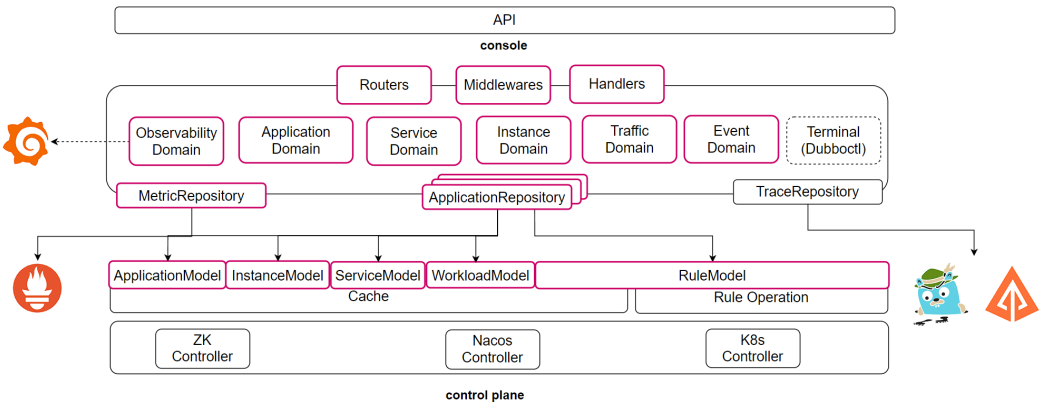
对于场景配置（如下图）这个页面，展示的是一些简化后的规则表单。用户能够在这个tab页就能够操作常见场景下的的一些流量规则，进一步降低用户的学习成本。

最后，事件这个页面以瀑布流的形式展示了应用相关的事件，便于用户排查相关时间点的问题。

对于实例以及服务，也都是作为一个独立的对象，展示与其相关的数据以及操作。同时应用，实例，服务之间可以相互跳转，共同形成一个有机整体。

后端设计

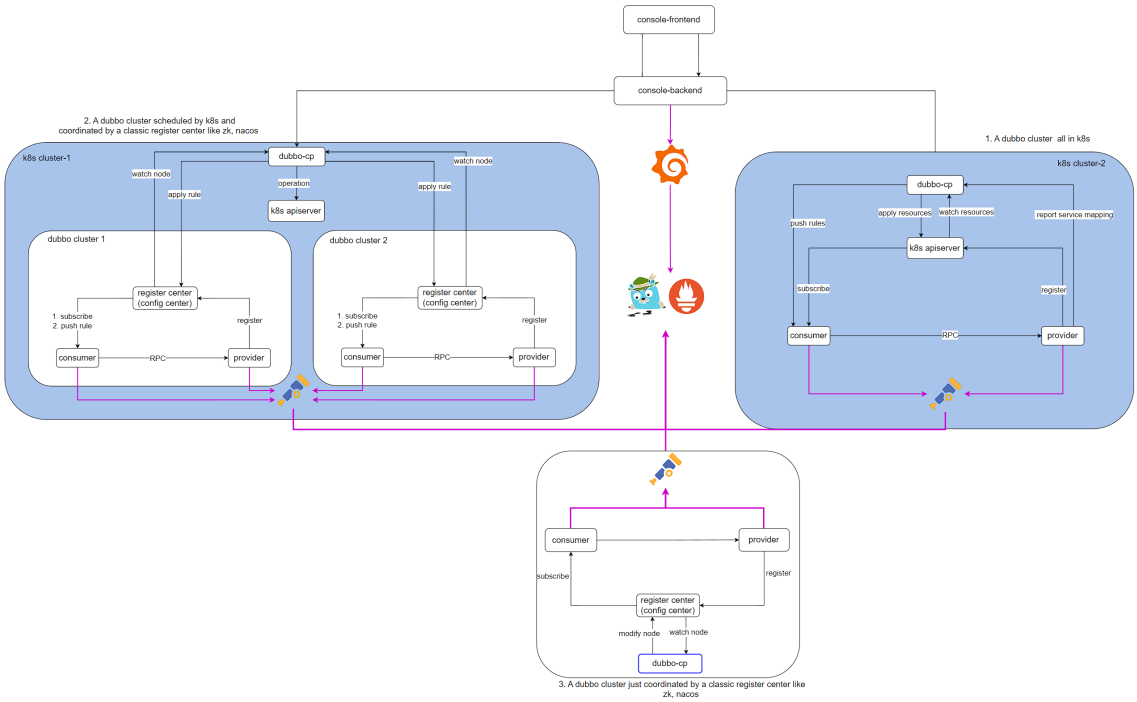
后端设计上，得益于社区其他同学的开发，我们只需要在control plane提供的接口的基础上将各路数据筛选，校验再聚合成领域模型的数据结构即可。因此架构上非常清晰：



整体上分为三层，repository负责与control plane交互，关注的是单个数据的原子操作，而数据的筛选，校验以及聚合则在domain层进行，而handler，router，middleware则负责提供完备的openapi，供前端交互。

监控&链路追踪设计

监控和链路追踪在开源社区已有很多完备的技术方案，用户也大多采用的是开源的组件来构建他们的可观测体系。因此我们会更多地依靠开源组件来构建console，并兼容主流方案。在调研了很多方案后，我们依托主流且完备的可观测技术构建了整个可观测链路：



图中存在监控以及链路追踪这两条数据链路，我们分开来讲。

首先是监控，Dubbo3在RPC请求链路，配置中心下发规则链路，元数据中心上报下发链路都进行了指标埋点，并以标准的prometheus格式暴露在qos端口，因此只需要prometheus依靠服务发现就能把指标采集上来并做后续的持久化存储，然后对接grafana，用户就能在console中嵌入的grafana中看到详细的监控数据，上文中也提到，我们针对不同的维度，提供了不同视角的监控面板，这一部分都会作为grafana的面板配置，用户只需要导入到自托管的grafana,然后将grafana地址在启动时配置进去即可。

对于trace，Dubbo3在RPC请求链路也进行了trace埋点，并可以自由选择不同的上报协议来上报到不同的trace后端，如jaeger，zipkin，opentelemetry（中转）等。在上报到trace后端后，我们可以使用grafana对接到不同的数据源，用户就能在console中嵌入的grafana中看到详细的trace数据。

从上面的图以及链路可以看出，其实metric和trace的数据链路是很多种多样的。对于console而言，只需要去适配主流的方案，并提供统一的面板（grafana）即可，对于用户在链路中使用到的各种开源组件，其实是没有太多依赖和关注的。