

gmiter 规划蓝图

- 简介
 - 业务价值
 - 目标
 - 可靠性保证
 - 运营成本
- 功能架构
- 功能特性
 - 服务管理
 - 流量管理
 - 访问限流
 - 动态路由
 - 动态路由算法说明
 - Mock管理
 - 节点管理
- 使用指南
 - Dashboard
 - 概览
 - 服务用链路
 - 服务监控
 - 服务管理
 - 流量管理
 - 流量管控
 - 1.单机限流基本配置
 - 2.集群限流-节点均摊基本配置
 - 3.集群限流-redis集群基本配置
 - 4.限流-高级模式
 - 5.限流-参数模式
 - 6.限流-节点限流
 - 动态路由
 - Consumer权重路由
 - Provider权重路由
 - 自定义路由
 - 故障管理
 - 1.服务级别熔断
 - 2.实例熔断
 - 3.接口重试
 - Mock管理
 - 节点管理
- 场景分析
 - 流量管理
 - 全局限流-节点均摊
 - 全局限流-共享Token
 - 参数限流
 - 过载保护
 - 业务防刷
 - 权重路由
 - 故障管理
 - 接口熔断
 - 服务熔断
 - Mock管理
 - 测试自动化
 - DR切换验证

简介

gmiter 是 FastEscrow 团队提供的服务治理平台，旨在解决微服务架构下的服务管理、流量管理、故障容错和可观测性的问题。

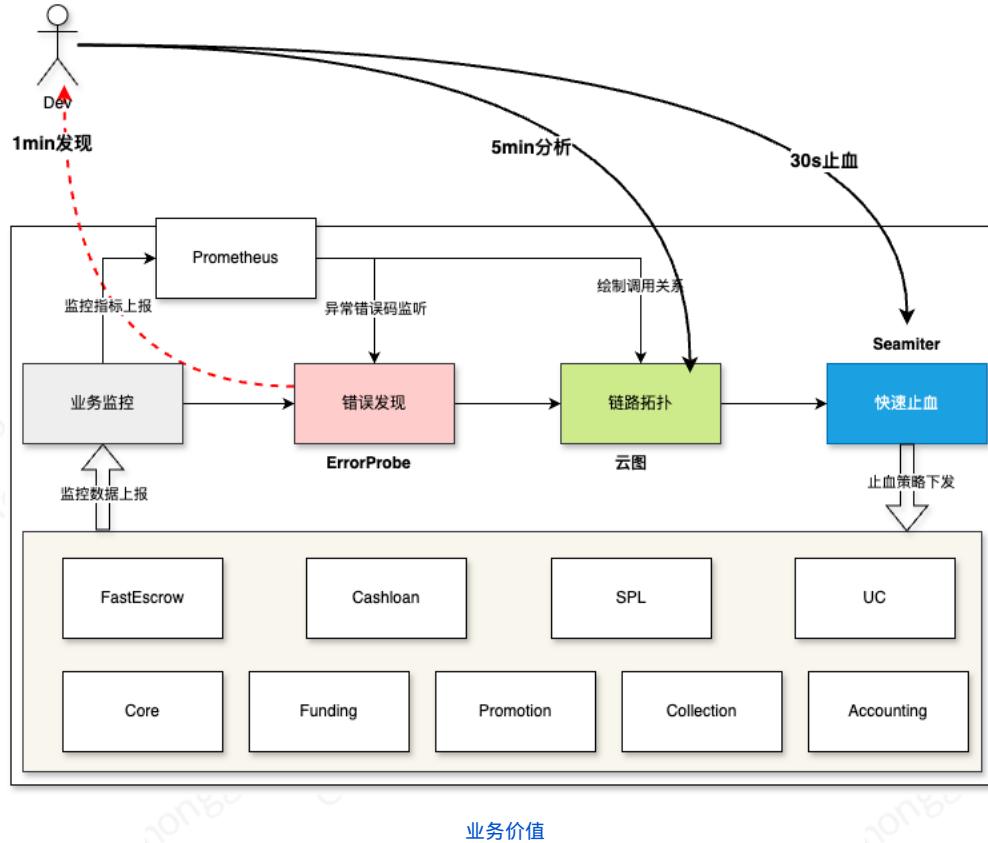
针对不同的组件（go-micro、kafka、调度）提供服务治理的标准方案和最佳实践。

为用户提供自动化的接口收集、流量收集、容器收集等指标，方便用户对所运维系统有全方位的了解，并能够快速为服务设置各种策略（如限流、熔断等），以便提高服务的高可用性。

业务价值

1、提供一个可视化的服务管理平台，如服务接口的自动发现、服务间的调用关系、服务调用的限流熔断快速设置等，使业务方更好的了解自身系统，同时加强系统的可用性，使业务的操作时间由原先的2人日（研发+测试）减少至仅通过UI操作解决credit 限流组件对比。

2、打通错误探测（ErrorProbe）、链路追踪、平台拓扑（云图）、容器管理等功能，做到问题早发现、及时定位、快速止血于一体的服务治理平台，降低Live问题的发生概率。



目标

- 1、打通错误探测（ErrorProbe）、链路追踪、平台拓扑（云图）、容器管理等功能，做到问题早发现(告警)、及时定位、快速止血于一体的服务治理平台
- 2、提供透明化的UI管理平台，操作简单、分类清楚，同时能够通过UI了解项目的实时的资源情况，如QPS、CPU、内存使用率等
- 3、提供微服务治理的通用功能，如接口Mock、接口限流、熔断、降级、热点限流、动态路由等通用功能，同时操作需准实时生效
- 4、高可用，管理平台自身及为业务提供的服务高可用，业务自身需和管理平台解耦，不能因为管理平台的可用性影响到业务，管理平台不能成为业务系统瓶颈
- 5、接入成本低，对业务基本无侵入，接入复杂度较低，需要在单位时间内完成（如0.5h），接入不需要了解新的知识
- 6、高性能，接入后原则上需要对原系统性能基本无影响
- 7、使用简单，功能操作必须简单易懂，同时有丰富的帮助文档以便研发快速接入，学习成本较低
- 8、故障隔离，原则上在gmter引入的故障可以在不修改代码的情况下动态下线相关功能
- 9、维护，接入后提供文档及人员配合回答相关问题，业务能够快速了解问题原因
- 10、Api，需要提供业务自我定制的功能，也可以在没有UI的情况下对业务系统提供Api进行管理

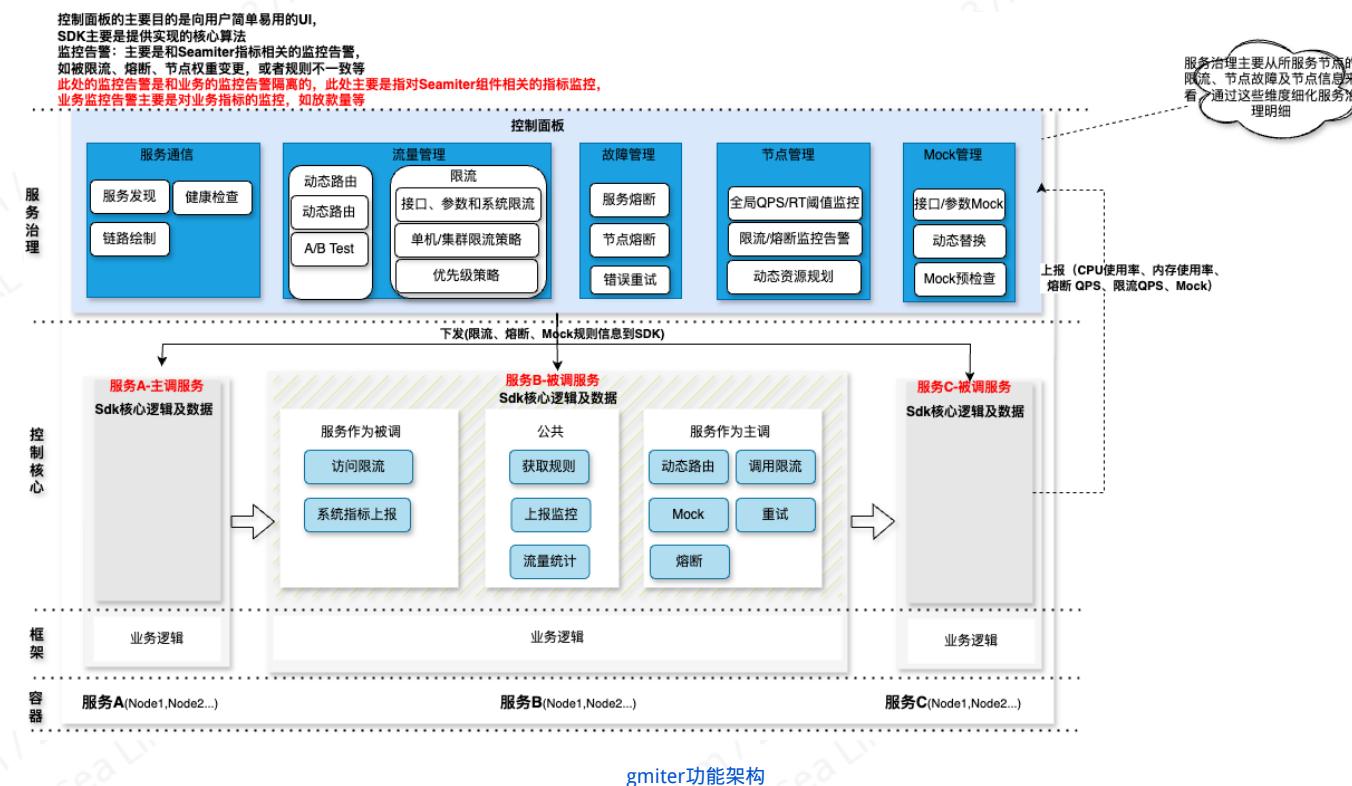
可靠性保证

- 从设计原理上，不存在gmter-server出现故障而影响业务的情况；如果sdk新引进的功能有问题，可以通过dashboard 1s内下线此规则及对应逻辑
- 从算法实现上，所有实现算法都是比较典型的开源算法实现，都在Live环境经过了长时间的验证
- 监控告警，gmter支持了比较丰富的监控告警，可以对接口、系统快速告警，支持seataalk、电话、邮件等告警方式；同时gmter的中间数据可以上报到promethues，可以通过多平台监控系统稳定性
- 从具体实践看，目前credit的网关、cashloan、fast、EA、DR等在Live上已经接入比较久的时间，Invest、Insurance也已经在non-live验证，可以证明系统的稳定性

运营成本

- 权限：当前权限可以控制到用户、组级别，未来会做到数据级别，保证数据安全性
- 规则变更：规则变更会给出变更对比，变更通知，同时策略经过审批才能生效
- 部署：当前live只需要2实例，同时存储数据较少，请求QPS低，不会对live的其他系统造成压力
- 维护：目前gmter的运营成本较低，同时控制到一个Q只会做一次大的变更，并且业务可以按需引入，动态插拔，向后兼容

功能架构



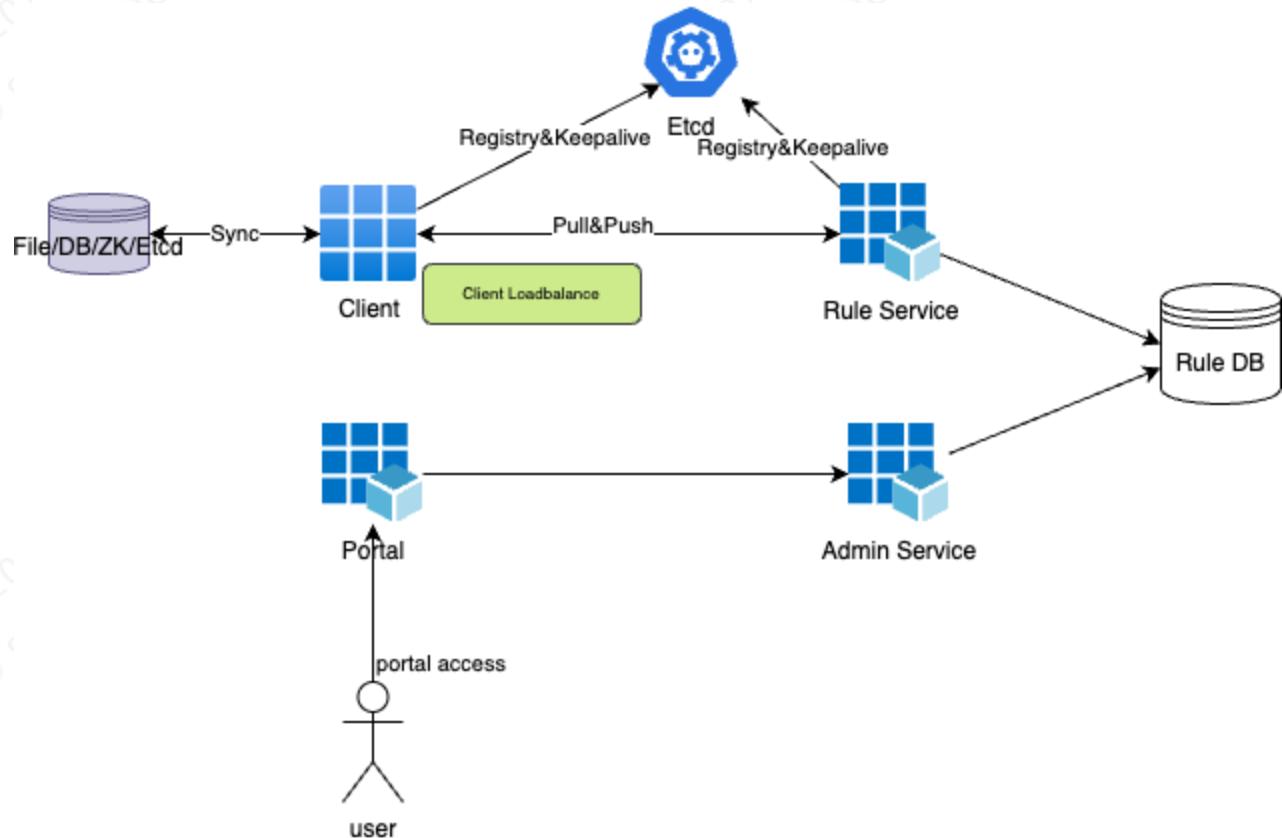
gmter系统包含：流量管理、故障容错、Mock管理、服务及节点管理和可观测性六大功能：

- 流量管理：包含动态路由、负载均衡和访问限流
 - 访问限流：支持本地和分布式限流两种模式，被限流的请求支持排队等待和直接拒绝的限流效果，可以对接口限流、请求参数限流和用户自定义限流场景，同时可以针对特定请求源进行限流。
 - 动态路由：支持自定义路由策略，如支持按照标签、请求参数等将服务的部分请求路由到特定实例，用于灰度发布等场景；支持自动/手动下线部分故障节点，并提供对应的检测及监控；支持路由标签全链路传递。
 - 负载均衡：支持权重路由、随机、轮询、一致性Hash等负载均衡算法。
- 故障管理：包含服务熔断、节点熔断及接口重试
 - 服务熔断：对于服务或者接口进行熔断，如果服务或者接口发生熔断，返回熔断错误或者自定义响应。

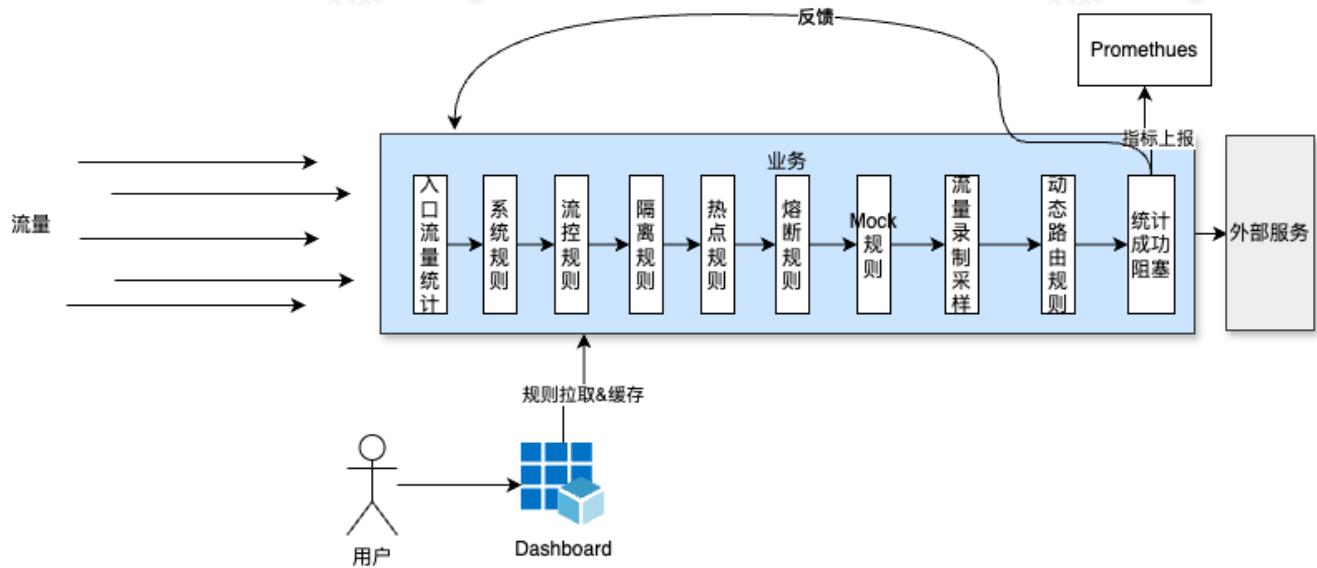
- b. 节点熔断：对于服务实例进行熔断，在发现节点故障后，不会将请求路由到熔断的服务实例，降低请求失败率。
- c. 主动探测：支持主动探测，包含请求时自动统计、Ping接口自动试探及手动熔断接口或者服务。
- d. 接口重试：在请求失败后，可以根据配置主动重试接口，增加成功率。
- Mock管理：包含接口/参数Mock管理、请求资源管理、Mock预检查、动态Mock等功能
 - a. 接口/参数Mock管理：支持对配置的接口或者接口的入参返回Mock数据。
 - b. 请求资源管理：SDK自动收集一次请求数据上报到控制面板，加快Mock数据的构造。
 - c. Mock预检查：对于配置的接口Mock，支持在正式生效前检查被Mock返回的数据Check，避免不正确的Mock导致的Live问题。
 - d. 动态Mock：支持动态生成Mock数据，可以按照请求体或者函数自动控制Mock返回。
- 服务及节点管理：包含服务发现、健康检查、链路绘制、系统指标管理、监控告警及动态资源规划
 - a. 系统指标管理：包含CPU、内存等使用率的实时查看。
 - b. 监控告警：对于系统、接口等触发异常上自动上报并告警，形成监控报告。
 - c. 动态资源规划：根据配置的使用率策略和集群的剩余资源，动态增减节点。
 - d. 服务发现：支持业务通过SDK注册到gmriter Server中。
 - e. 健康检查：支持服务实例上报心跳，通过心跳检查实例是否健康，同时可以对不健康实例及时告警及剔除异常实例。
 - f. 链路绘制：支持按照调用链路绘制整体业务架构图，支持按照接口纬度绘制准实时链路图，支持按照所属服务绘制一级两级拓扑图。
- 可观测性：提供业务流量的实时统计、观测事件的实时统计及告警、用户操作记录、应用实例列表、TOP流量等监控试图。

gmriter的功能分为两个层面：

- 控制面板：提供简单易用的管理界面，支持用户和权限管理，负责创建服务及告警，同时负责流量管理、熔断管理、重试管理、Mock管理的规则数据配置及下发。
- SDK：负责服务发现的客户端，负责规则数据的拉取及生效，采用插件化设置，支持按需加载和使用，同时支持各种组件的适配，如gRPC, gin-web, go-micro等
 - a. 引用服务作为被调方：当一个服务被其他服务调用时，可以使用服务注册、心跳上报、访问限流、访问鉴权、Mock、接口管理等功能
 - b. 引用服务作为主调方：当一个服务调用其他服务时，可以使用动态路由、负载均衡、熔断降级、接口隔离、调用限流等功能
 - c. 公共部分：规则数据拉取及上报监控



图：semiter完成的服务流程



图：完整SDK处理流程

功能特性

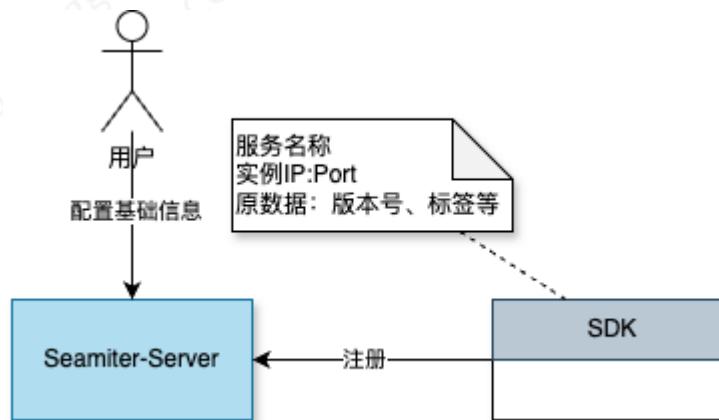
服务管理

主要是指被调服务按照gmriter的服务模型把自己的服务数据注册到gmriter，以供gmriter做服务发现、服务管理及主调方的规则数据拉取。

注册数据主要包含以下部分：

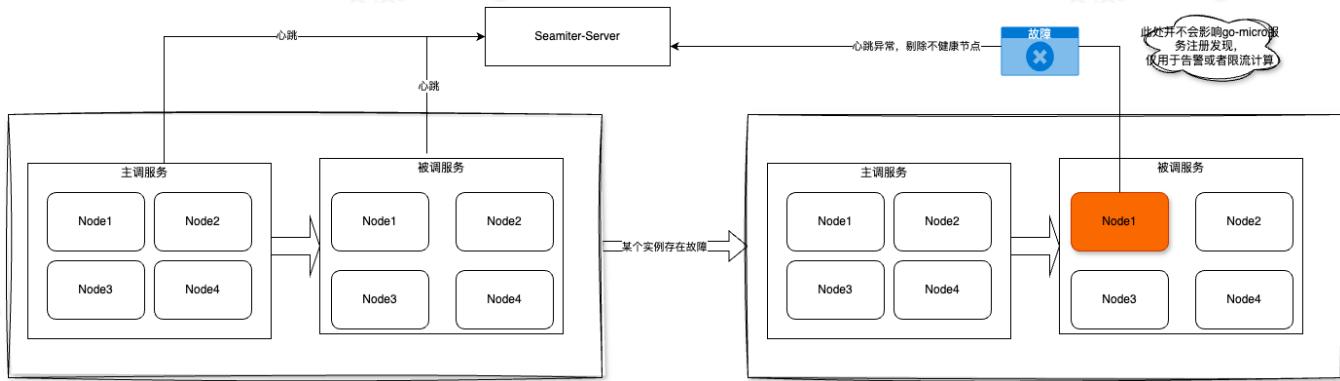
- 服务名称：服务的唯一标识，不区分大小写。（Dashboard配置）
- 服务元数据：包含一些服务的标签信息，以便对服务进行分类，可用于过滤。（Dashboard配置）
- 服务实例：提供服务的节点列表，以IP:PORT的方式提供。
- 服务实例元数据：包含版本号、标签信息等，用于后续的流量治理等操作。

当前支持通过SDK注册的gmriter Server，未来会支持openapi方式注册到gmriter



图：服务注册原数据

节点检查：节点和server之间通过心跳来保持健康检查，不健康的节点会被gmriter-server移除。此处的移除仅作为gmriter自身的需要，如告警及限流计算等。



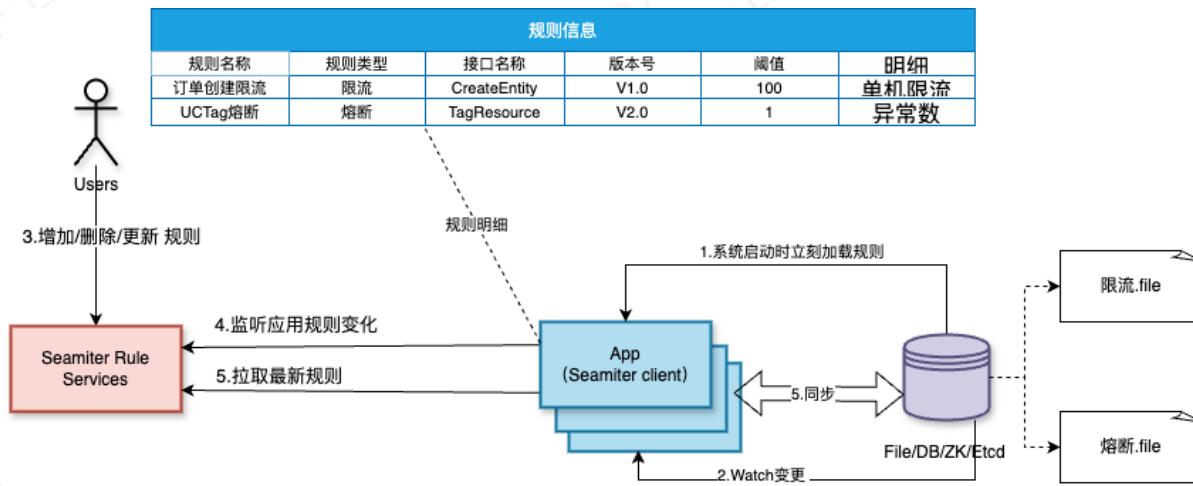
图：节点检查

规则加载：

- 1、在系统启动前，Client会立刻从外部存储中加载规则，避免加载缝隙。
- 2、Client会监听外部存储变化，如果由变化会准时时加载到业务内存。因此如果Dashboard不可用时，业务也可以通过外部存储修改规则，避免了对Rule Services的强依赖。
- 3、Client会监听Rule Service的变更，如果由变更，会实时把规则拉到业务内存。
- 4、变更的规则会更新到外部存储中。

规则加载对于整个流程的运转非常重要。规则加载需要做到无缝加载。如果在业务启动后才加载规则，这可能导致预期外的效果。

如限流策略，由于没有规则作用，有可能导致刚启动的节点被大流量瞬时击垮。如果Mock规则在启动后才加载，有可能导致脏数据。



图：规则加载

流量管理

访问限流

限流能力是高并发系统中，对于服务提供方的一种保护手段。通过限流功能，我们可以通过控制QPS的手段，以避免业务被瞬时的流量高峰冲垮，从而保障系统的高可用性。

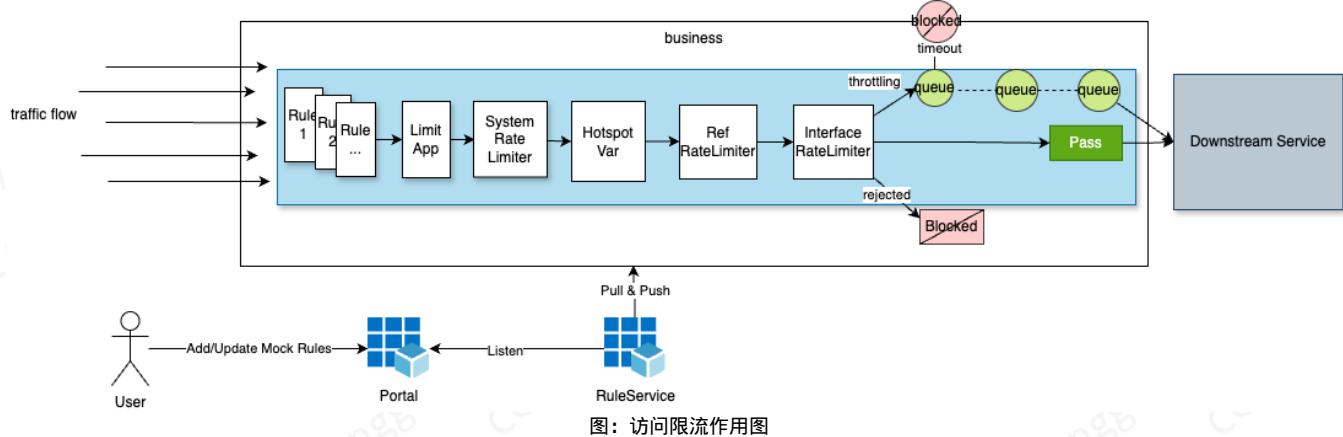
访问限流主要有如下两个应用场景：

过载保护：保护业务不被突发流量打垮

业务防刷：防止恶意用户发送过多流量影响其他正常用户

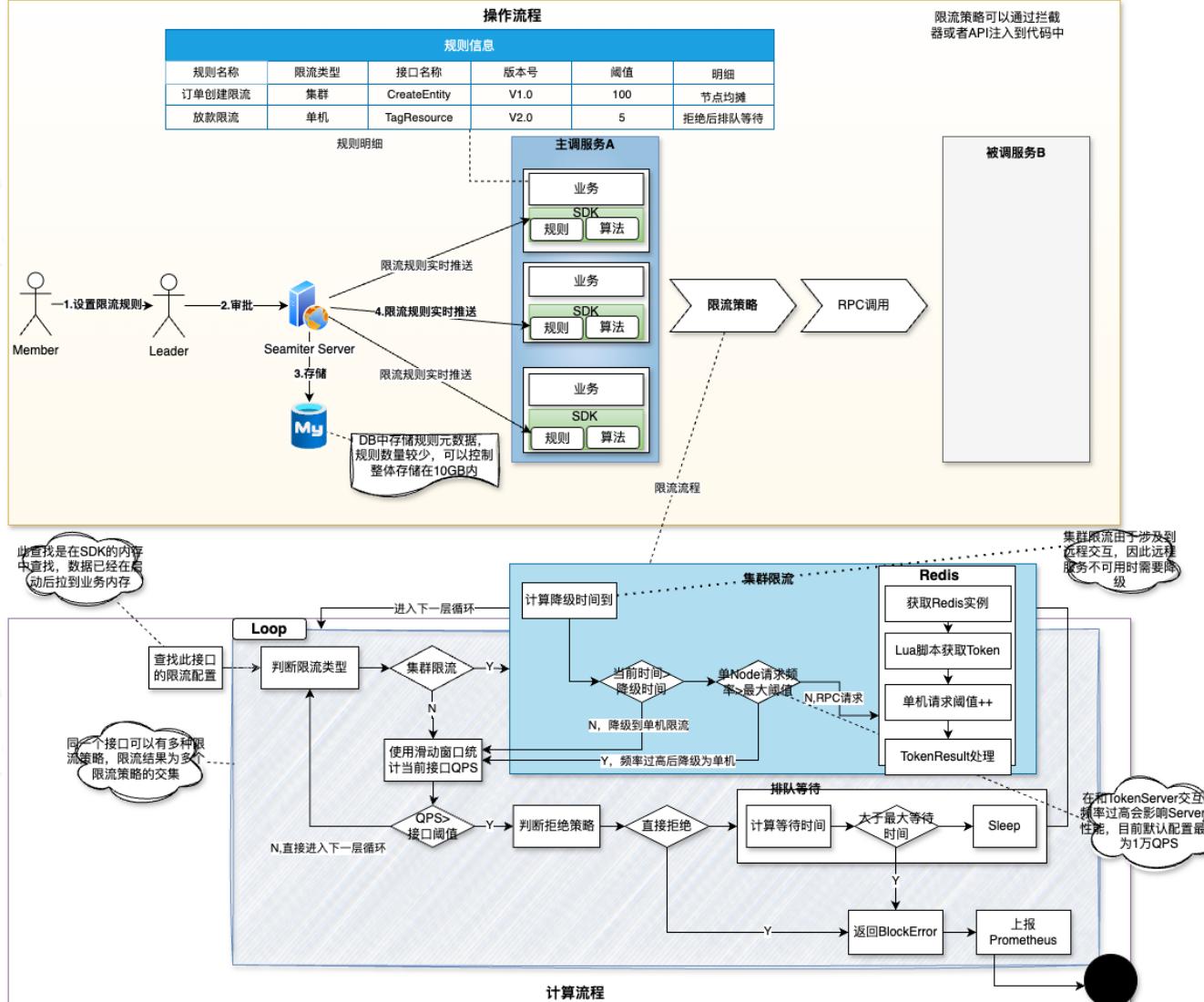
gmiter限流可以用于主调服务和被调服务，主调服务使用限流主要是从调用方保护下游，如网关保护下游服务不被冲垮或者下游自身限流能力有限或者和下游约定（如funding调用银行接口）。

被调服务限流主要是保护系统自身，如限制上游调用自身的频率等。



图：访问限流作用图

gmiter定制了多种限流策略，如果业务设置了一种或者几种限流策略，则流量会按照上面的流程进行逐个遍历，最终的限流结果取决于多种策略的交集，也就是只要有一个被限制，则后续策略不会在运作。

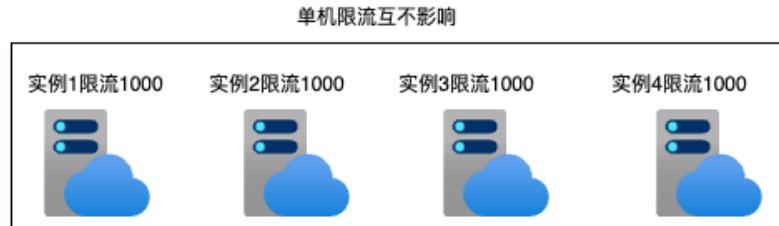


图：限流策略的完整架构图

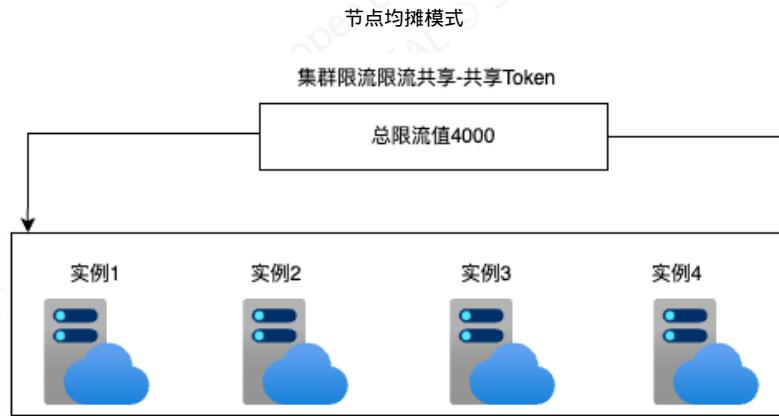
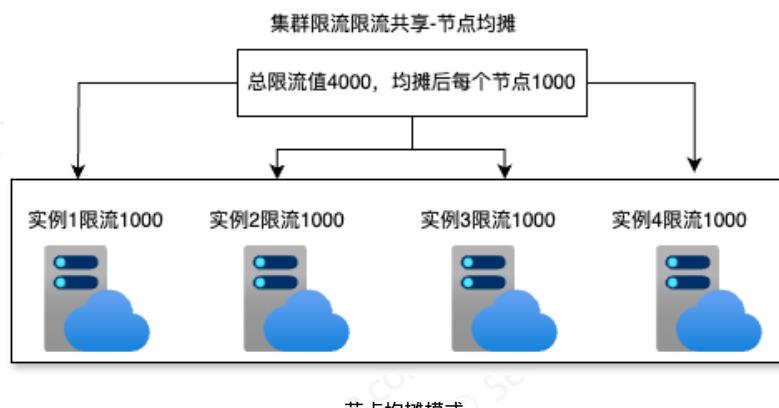
此图主要是为了说明限流从设置到生效的完整流程。gmiter通过dashboard下发规则到sdk。业务可以通过拦截器或者API使用gmiter限流。gmiter提供了多种限流算法，业务流程在进入系统后会根据研发的配置走到相应的限流策略。

gmiter为业务提供了2中类型的访问限流能力：

单机限流: 针对单个被调实例级别的限流，流量限额只针对当前被调实例生效，不共享。



分布式限流: 针对服务下所有实例级别的限流，多个服务实例共享同一个全局流量限额。目前gmiter的限流包含节点均摊和共享Token两种方式，其中节点均摊可以理解为是一种伪全局限流。



两种限流模式如何选择？

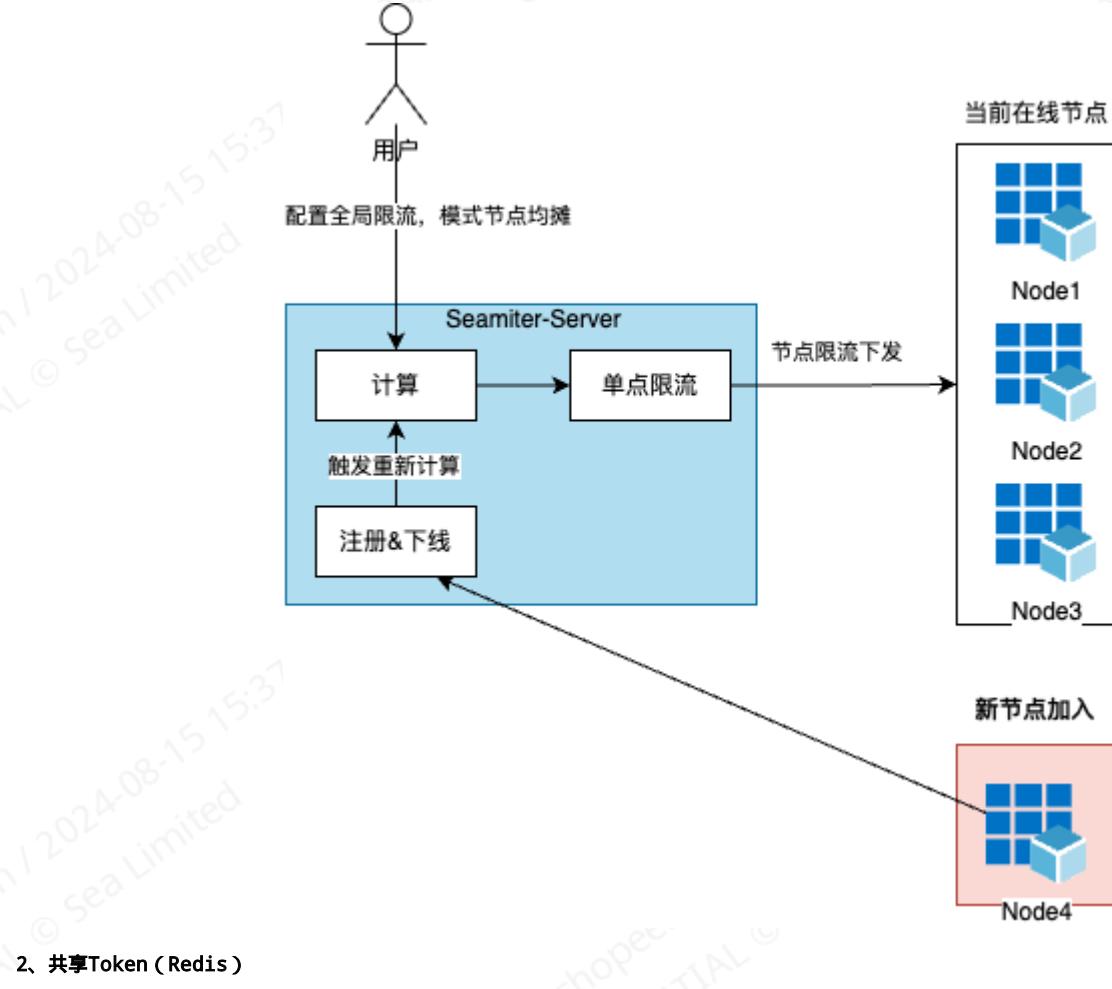
单机限流: 一般适用于保护服务自身不被打垮，按照每个服务集群单机的容量来计算配额。

分布式限流: 一般适用于保护第三方服务或者公共服务（比如保护数据库）；或者是在网关层进行限流，对通过网关接入的后端服务进行保护。

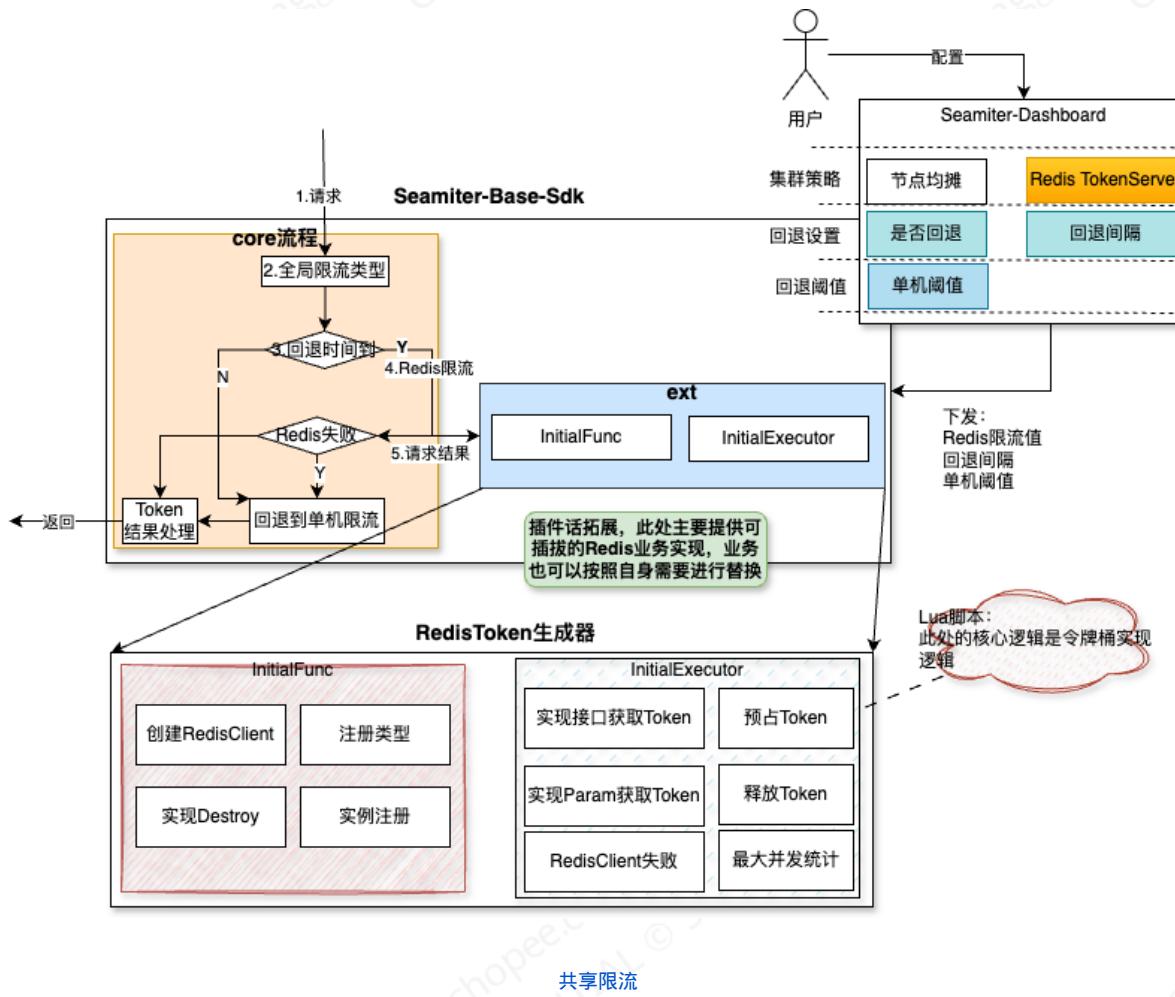
限流算法说明

1、节点均摊

用户配置全局限流值，系统会根据节点在线个数进行均分。当节点个数发生变化时，会触发限流重新计算，并把计算值下发到业务节点中。



2、共享Token (Redis)



规则下发：

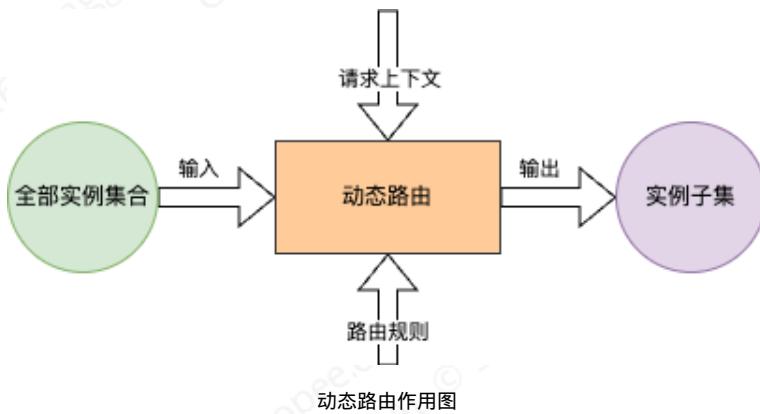
- (1) 业务在Dashboard修改后，gmiter Server会把Redis的限流值、回退间隔和单机阈值下发到SDK
- (2) SDK接收后会更新对应的限流类型和阈值，并构造对应的处理策略

处理流程

- (1) 业务接受请求
- (2) 查看请求对应接口的限流配置，如果是全局限流，则根据类型选择全局限流的实现器
- (3) 如果之前的全局限流器出现问题，则会设置回退时间，如果回退时间到，则转发到Redis申请Token 【为了化简配置，此处的回退时间在SDK中默认配置为2s】
- (4) 如果Redis的回退时间未到，则通过单机限流进行处理，此处走到gmiter的LocalCheck逻辑。
- (5) 统计Client请求RedisServer的频率，如果频率超过阈值，则也会回退到单机限流。【当前Client请求Redis的默认最大频率为10000，业务可以按需修改】
- (5) 对对应的Token结果进行处理，并返回

动态路由

通过动态修改节点的路由属性，将不同节点的不同请求路由到不同实例分组或者集群，我们称之为动态路由。



一般会在以下场景选择动态路由：

- (1) **问题节点下线**: 通过动态路由能力，把有问题的节点流量路由到其他节点，以达到下线此问题节点的能力。
- (2) **灰度**: 新功能上线，需要一定流量或者特定用户的流量进行新功能验证。
- (3) **负载均衡**: 通过动态路由重新修改流量的负载均衡测量，使流量更加均匀或者更合适的到达被调节点。

权重路由: gmter的每个实例都可以设置权重，请求会根据实例的权重，按比例分配权重。权重路由分为静态权重和动态权重两类：

静态权重: 用户可以通过界面配置或者实例注册的方式，调整服务实例的权重，由gmter server推送给所有的gmter sdk生效。

动态权重: gmter sdk在运行过程中，定时上报负载数据给gmter serve。被调方定时（一般2s）上报指标的数值到gmter server，gmter server根据负载信息、规则配置，调整权重数据。主调方定时拉取最新的权重数据，实时更新以前的静态权重字段，以便达到动态修改权重。

自定义路由: 自定义路由主要用于灰度验证场景，业务可以按照Metadata、Header、Param等定义匹配规则，以便动态调整接口或者服务的路由策略。

动态路由场景

场景1：灰度发布

FastEscrow引入一个新功能，通过灰度版本的控制，实现符合灰度策略的对象，优先进入灰度服务进行体验，此功能观察和验证后才可以推广到全部用户。

场景2：故障节点下线

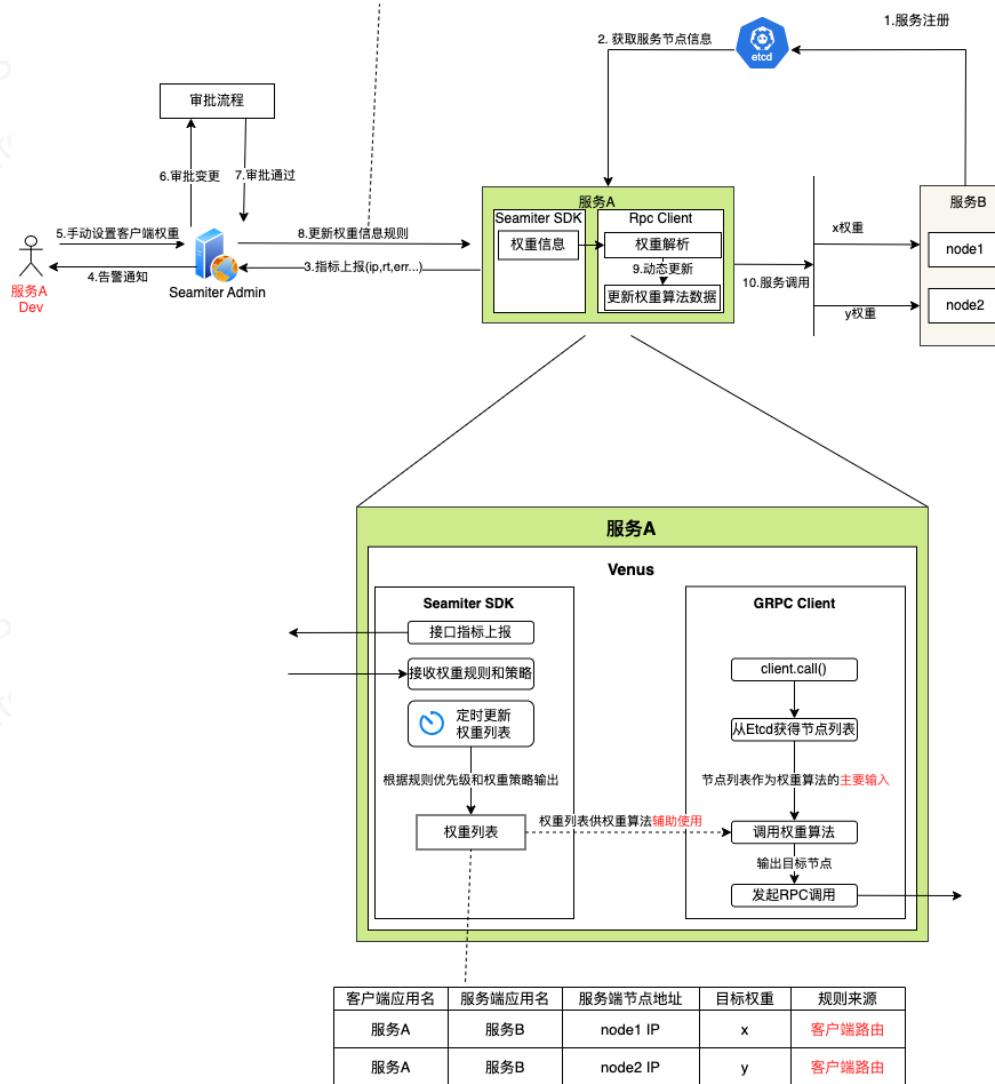
由于机器原因，某些节点持续存在问题，通过权重路由快速下线此节点。

场景3: 特定机器导流，便于线上复杂问题排查

动态路由算法说明

1、客户端动态路由

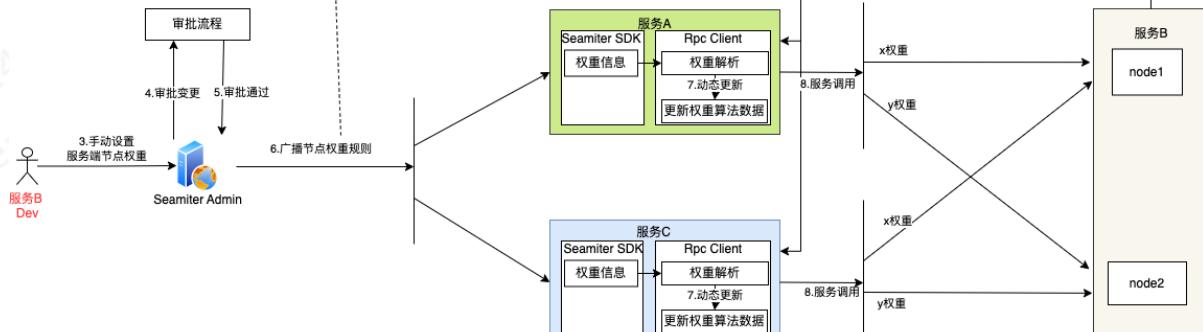
权重规则名称	客户端应用名	服务端应用名	服务端节点地址	目标权重	规则来源
weight-rule-01	服务A	服务B	node1 IP	x	客户端路由
weight-rule-02	服务A	服务B	node2 IP	y	客户端路由



手动设置权重-客户端路由流程图

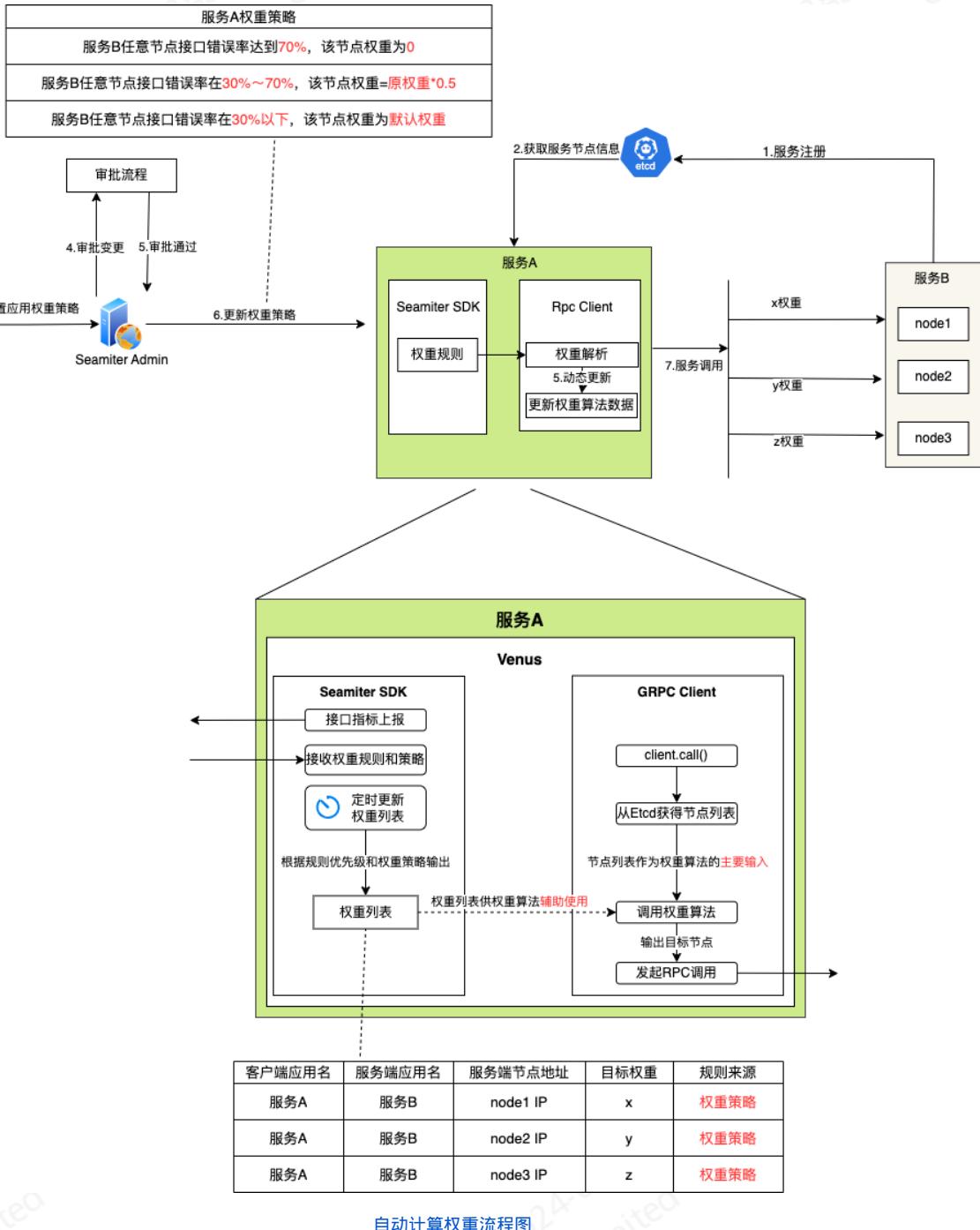
2、服务端动态路由

权重规则名称	客户端应用名	服务端应用名	服务端节点地址	目标权重	规则来源
weight-rule-01	*	服务B	node1 IP	x	服务端路由
weight-rule-02	*	服务B	node2 IP	y	服务端路由



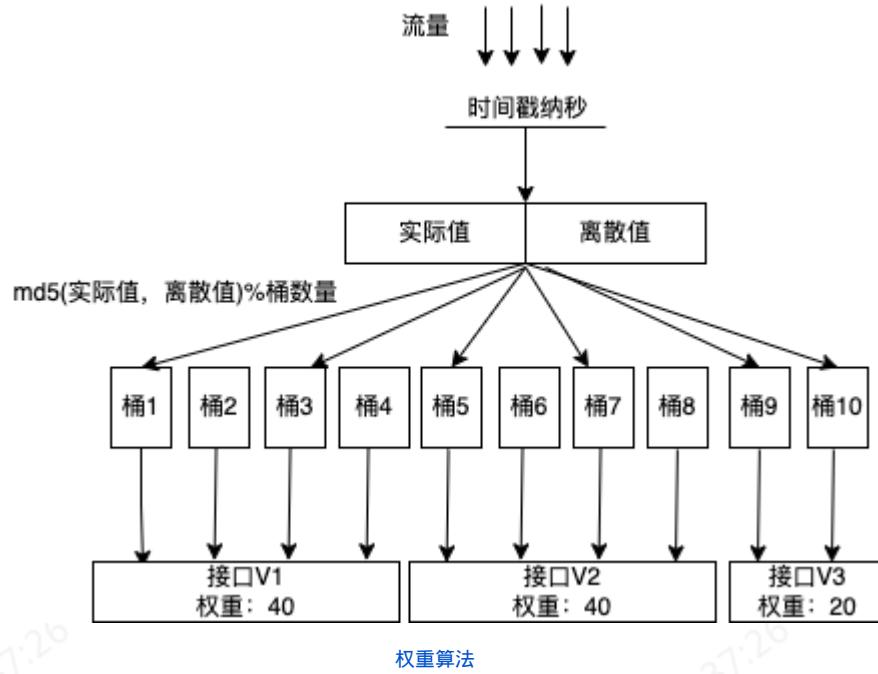
手动设置权重--服务端路由流程图

3、自动计算动态路由



自动计算权重流程图

权重分配算法



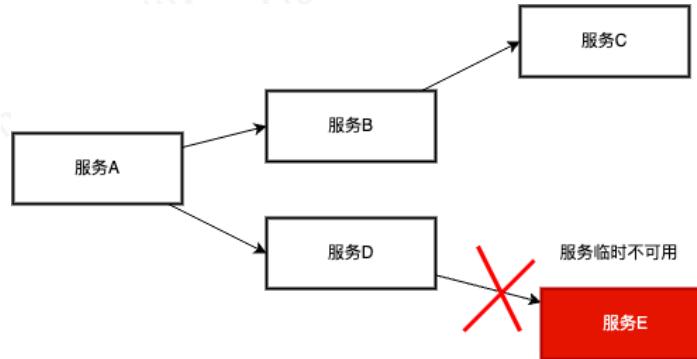
权重的大致思路是桶算法，

把请求均匀划分到桶中，每个节点占用固定的桶数量。针对随机的请求，使用一种比较随机算法 mod 桶数量，就得到了当前桶路由到具体哪个节点的接口中。

故障管理

在微服务架构下，业务功能往往通过多个系统共同完成，因此调度链路也会较长。在依赖服务出现故障后，需要快速发现、快速止血。

业务故障熔断，主要指当下游因过载或者BUG等原因，出现请求错误后，为了防止故障级联扩散导致整个链路出现异常，从而对请求进行拒绝或者重试的一种机制。



图：依赖服务故障

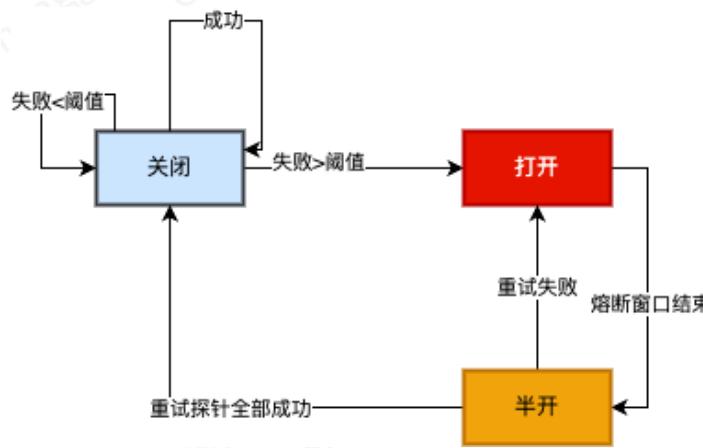
熔断模型

熔断模型的设计遵循业界标准的熔断器模型设计。熔断器有3类状态：

关闭：所有请求皆可访问下游资源，无任何限制。

打开：限制访问下游资源的请求，不允许任何请求的访问。

半开：限制访问下游资源的请求，只允许部分请求达到下游。



图：熔断状态机

熔断场景

熔断一般会发生在以下场景下：

硬件环境出现故障

服务在运营过程中，因为一些不可抗力的因素，可能会出现机器故障、机器重启、机房断电、网络中断等问题。通过熔断机制，对服务实例或者机房分组的快速熔断，可以避免业务请求持续失败。

版本上线引入BUG

版本新特性开发上线后，因为漏测等原因，某些分支触发了BUG，导致部分的业务逻辑出现故障。常见的是部分方法在遇到某些入参的时候，会出现进程报错或者高负载的问题，影响其他方法的请求处理。通过熔断机制，将故障方法进行屏蔽，可以避免其他业务请求受到影响。

服务出现过载

因为路由不均或者峰值流量的到来，导致被调服务出现了高负载，导致请求的时延增大，成功率降低。通过熔断机制，合理的拒绝一部分请求，可以降低服务负载，恢复正常运行状态。

熔断级别

接口级熔断

应用与服务之间的调用都是针对接口进行调用，为避免调用故障接口导致业务整体时延较大，加剧后端的压力。用户可以设置熔断规则，按照整个服务或者服务下某个接口的粒度设置熔断阈值，并统计在调用过程中的错误率时延等数据，达到阈值后会进行熔断（熔断器打开）。熔断后，访问该服务或特定接口的请求都会返回失败或者走降级逻辑。

接口级熔断生效在接口调用前，主调服务访问接口前需要判断接口的熔断状态。

实例级熔断

一般用于远程服务调用（RPC）的场景，针对某个节点或者分组（具备相同标签的节点集）设置熔断阈值，实例级熔断往往按照具体的服务实例进行熔断统计，并统计在调用过程中的错误率时延等数据，达到阈值后会进行熔断。熔断后，该实例会被屏蔽，不会有请求路由进来，直到恢复。

接口级熔断生效在接口调用中，在负载均衡过程中完成对熔断状态实例的剔除。

触发熔断条件

连续错误数熔断

请求调用时，统计周期内，出现连续错误数目超过阈值之后，资源进入熔断状态。

错误率熔断

熔断器按照滑窗对请求总数及成功数进行统计，并汇总时间段内的总错误率，一旦超过阈值，资源进入熔断状态。

错误判断条件

系统需要通过错误请求的统计来判断是否需要触发熔断，请求的错误一般会表现出以下2个方面的特性：

返回的状态码

对于标准协议的请求，比如HTTP Response，常见的5XX等状态码，代表着后端出现异常（比如数据库异常）导致业务请求失败。

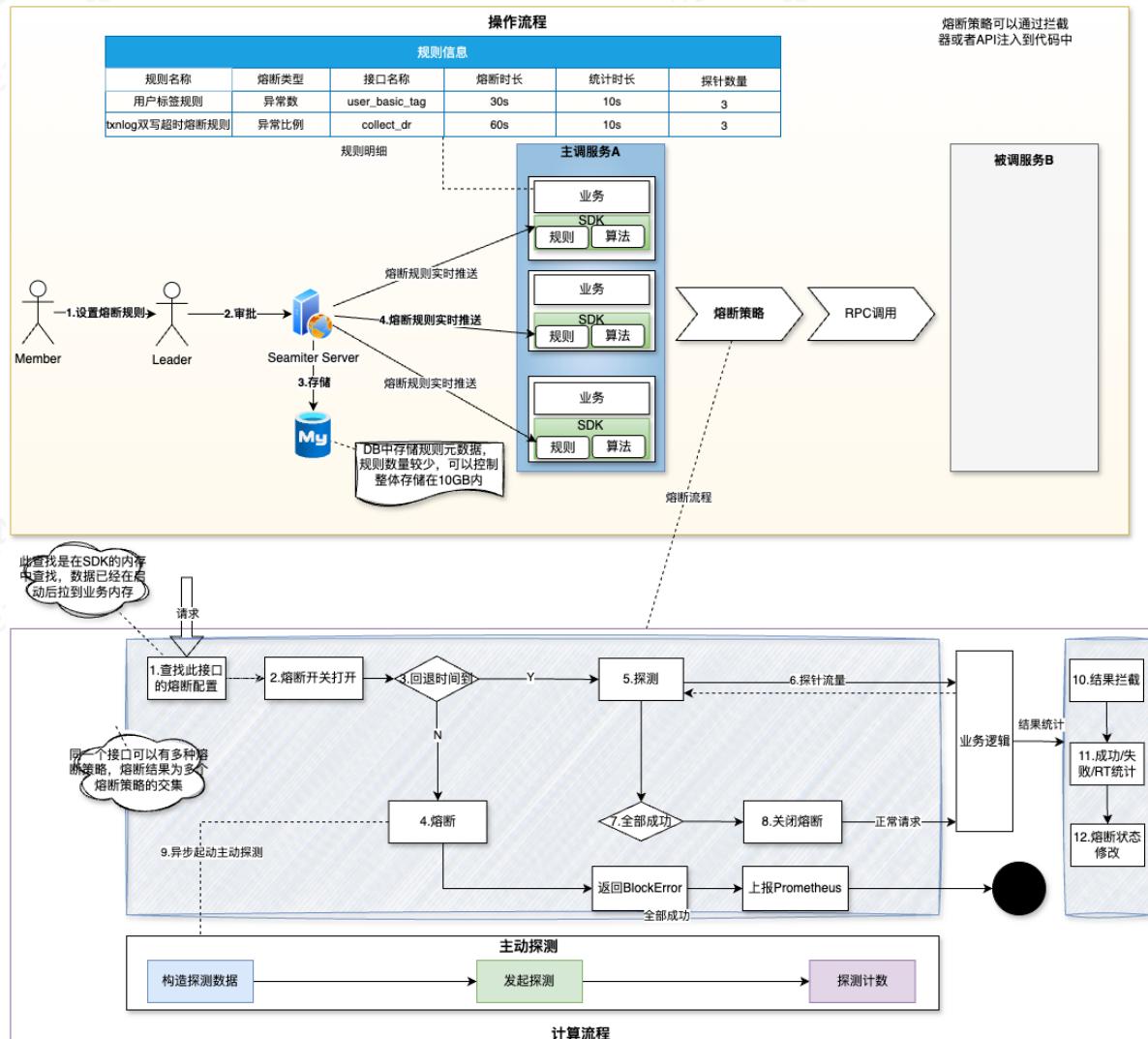
时延

对于交易系统等对时延比较敏感的系统，当出现后端数据库等负载过高的情况，导致部分请求可以正常处理，但是时延普遍过高，此时仍可认为这部分请求是失败请求，触发熔断处理。

熔断恢复

当资源的错误请求统计达到一定阈值后，资源会进入熔断状态，在接下来的一段时间内，该资源将会被屏蔽（不会有请求路由到该资源），渡过屏蔽期后，资源会进入半开状态，此时系统会放少部分业务请求给该资源，并记录请求的处理结果。假如请求全部处理成功，则资源恢复成功（熔断器关闭），取消屏蔽并正常处理业务请求。

但是，假如业务请求扔存在处理失败，则该资源会重新进入熔断状态，继续保持隔离。

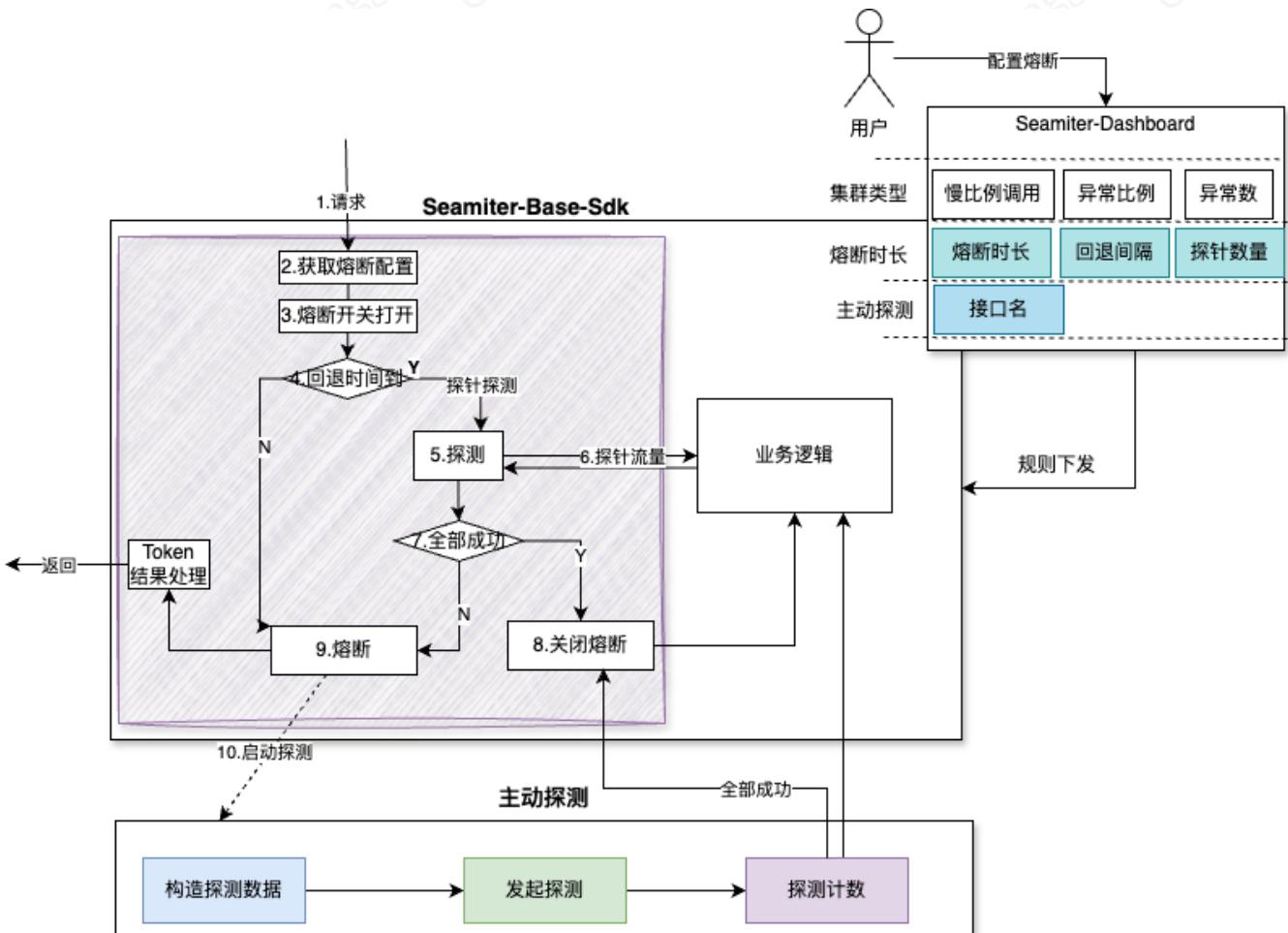


图：熔断策略完整架构图

当用户在Dashboard操作后，熔断策略会快速推送到SDK端。

当请求到达SDK时，请求会根据熔断状态执行对应的熔断策略。同时无论业务请求处理成功还是失败都会进入到熔断统计中，根据统计计算错误率、失败率等。

熔断算法说明

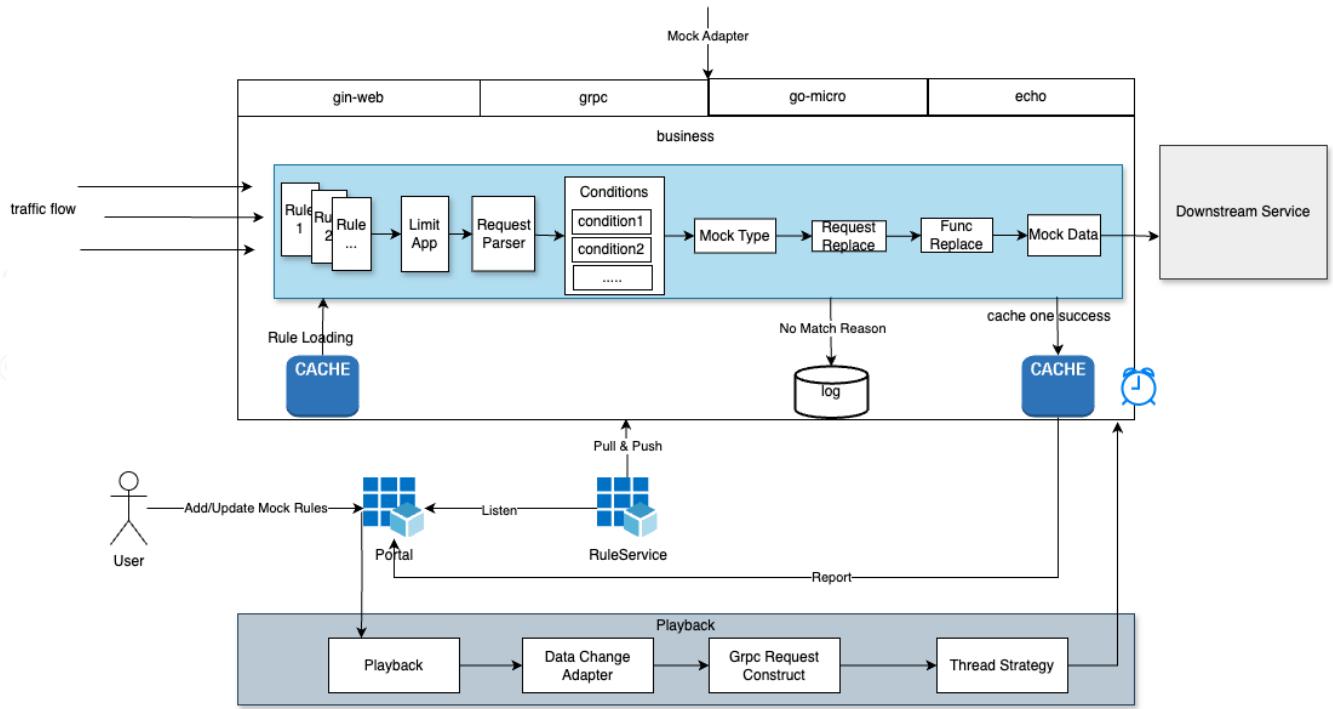


图：熔断算法流程

- (1) 用户需要在Dashboard配置对应的熔断规则，如果熔断需要主动探测，还需要配置探测接口及请求体和恢复判断条件，配置后规则会主动推送到SDK。
- (2) 当请求到达SDK时，SDK会优先判断对应的接口是否有熔断相关配置，如果没有按照正常流程进入到下一个拦截器或者业务代码。
- (3) 熔断配置了熔断，则需要判断当前的熔断开关，如果熔断时间没有到，则直接拒绝。
- (4) 如果熔断时间到，则放行部分请求，如果请求依然失败，则继续进入熔断，重新设置熔断市场。
- (5) 如果探针全部成功，则关闭熔断，放行业务请求。
- (6) 当处于熔断时，如果配置了主动探测，则会启动一个任务主动探测下游接口，如果主动探测成功，则也可以直接关闭熔断，在探测成功或者探针成功后都会主动管杯探测接口。

Mock管理

微服务架构下，服务的验证需要多个依赖环境，然而当所依赖服务服务没有就绪时会严重阻碍QA的测试进度。gmter主要是提供一种快速Mock数据的方法，提高QA的验证效率。



Mock场景

Mock一般发生在以下场景下：

场景1：QA验证环节，依赖方没有Ready

项目提测后，QA在验证过程中往往会由于依赖方没有Ready或者处理有BUG时会阻塞项目的进度，此时比较适合使用Mock返回预期的任务，以便达到流程流转的效果。

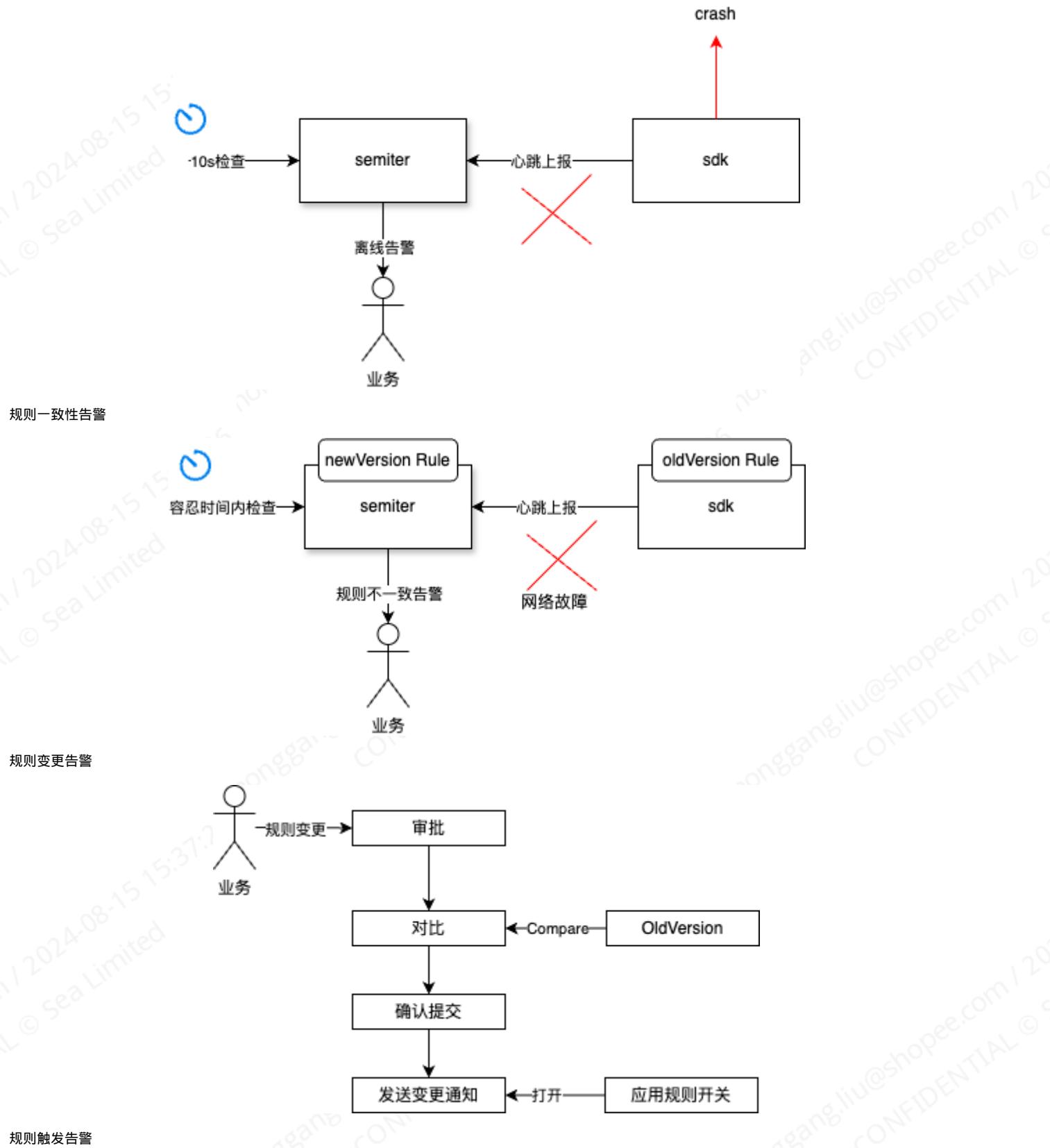
场景2：QA测试自动化

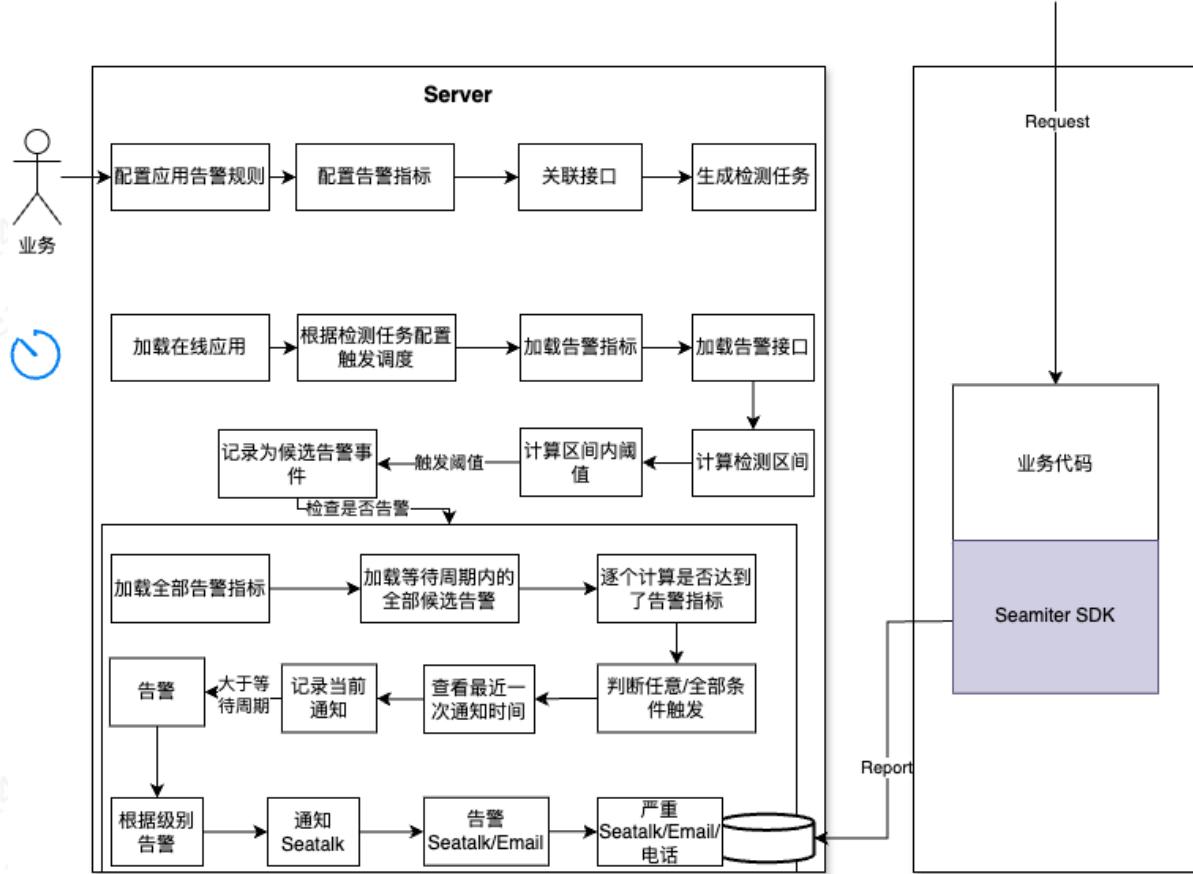
QA的自动化测试是我们项目日常防护的手段，这个流程的目的主要是验证业务自身系统的可靠性。然而由于测试环境并不稳定，同时部分依赖方不一定有对应的环境，因此此时Mock对应的接口，可以帮助QA构建稳定的自动化脚本

节点管理

节点管理主要是指对所管理节点的内存、CPU、负载等信息进行管理，包含配置对应的限流、监控等。同时节点信息可以和SRE维护的CMDB信息进行交互，以便对机房的整体进行把握。

应用离线告警





使用指南

Dashboard

dashboard包含三个部分：概览、调用链路和服务监控

概览

概览主要是展示系统的统计数据，如服务的实例个数、是否触发了防护事件、服务请求概览和时延概览，同时可以展示从自身业务请求下游服务的占比统计服务占比等。

对于服务列表，可以直接对整个服务或者单个节点设置节点的限流阈值。

对于实例列表，可以直接对其设置离线监控，如邮件/seatalk告警等。

对于请求数，可以对全局的请求数、熔断数等设置告警。

对于状态码，可以针对单个状态码设置监控告警。

服务调用监控

概览 调用链路 服务监控

服务名称 下拉列表 时间范围 30min 1h 1d 3d 7d 自定义7天内的时间 下拉列表

应用实例数 3

实例列表			
机器名称	IP地址	心跳时间	状态
Host1	127.0.0.1:15000	2024-06-07 15:41:00	健康
Host2	127.0.0.2:15000	2024-06-07 15:41:00	监控
Host3	127.0.0.3:15000	2024-06-07 15:41:00	离线

防护事件

事件类型	值	发生时间	接口名称
限流	10	2024-06-07 15:41:00	CheckStatus
熔断	3	2024-06-07 15:41:00	SellerWallet
Mock	5	2024-06-07 15:41:00	OrderTag

服用请求数

设置QPS阈值监控

总请求数	成功请求数	限流请求数	熔断请求数	Mock请求数
1.5W	1.4W	800	90	10

总请求数 成功请求数 限流请求数 熔断请求数

服用请求时延

均值	最大值	最小值	P99	P95
80ms	100ms	40ms	79ms	78ms

均值 最大值 最小值 P99 P95

上游调用占比

设置状态码值监控

上游服务列表

上游调用占比

上游请求时延分布

设置RT监控

服务列表

时延分布

状态码占比

设置节点限流

状态调用表

状态码占比

下游调用占比

下游服务列表

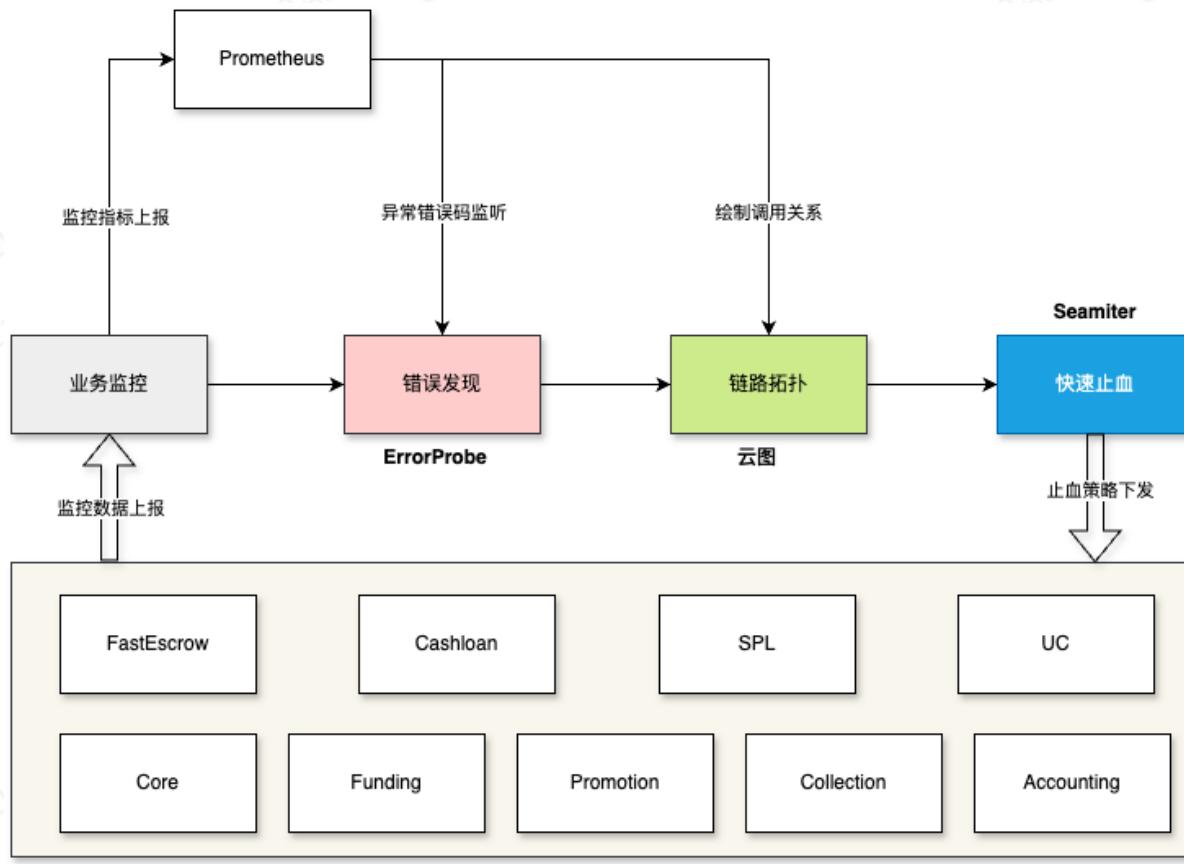
下游调用占比

设置节点限流

机器名称	IP地址	总请求数-下游	失败请求数	熔断请求数	Mock数	平均时延(ms)	成功率
Host1	127.0.0.1:15000	1000	1	0	0	20	99%
Host2	127.0.0.2:15000	8000	5	10	2	30	98%
Host3	127.0.0.3:15000	5000	3	1	0	25	95%

服务用链路

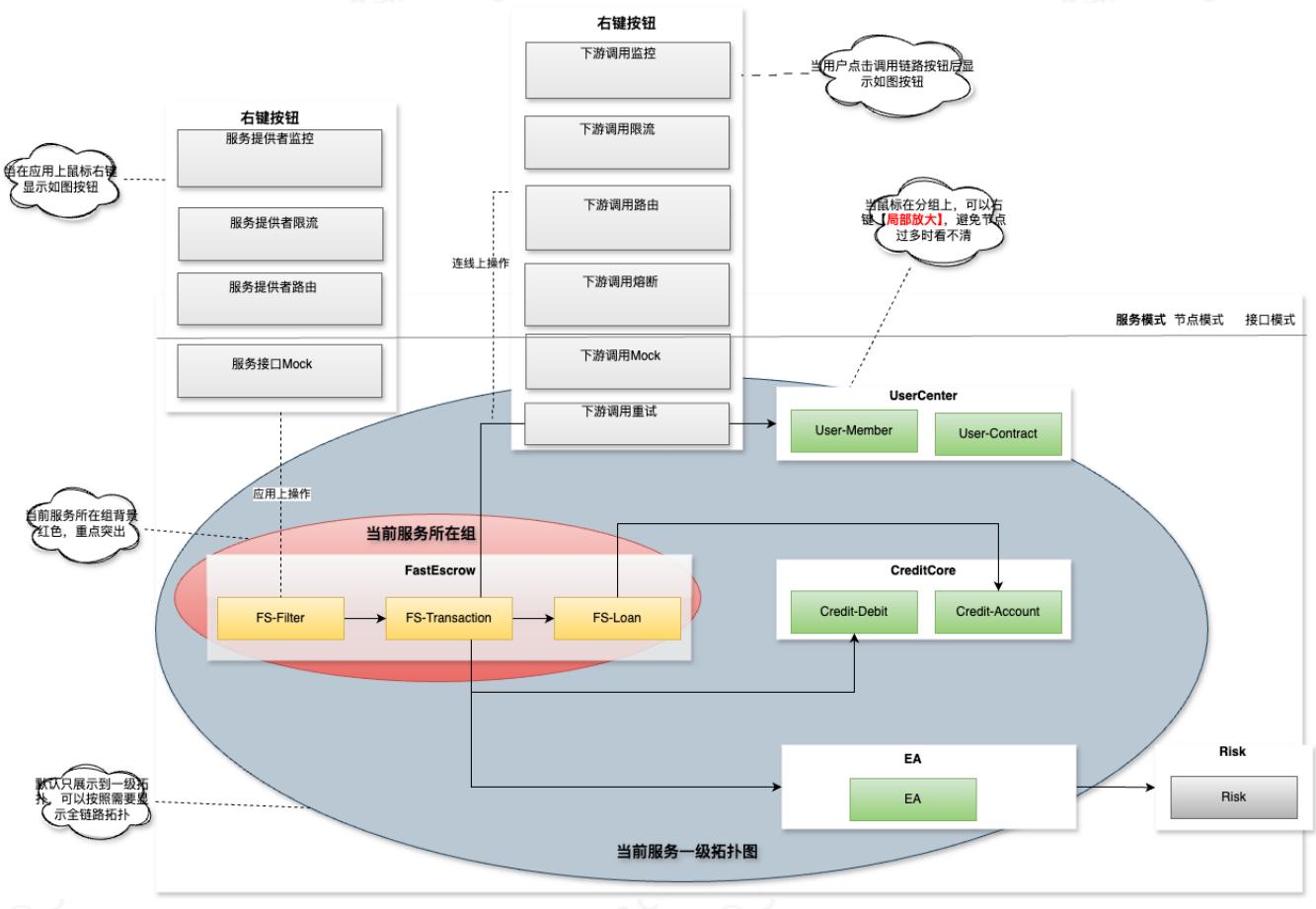
gmriter的目标是可以把业务监控、错误发现、链路拓扑及业务的快速止血融为一体，整体流程如图。



Seameter在整合其他数据的基础上，提供如下三种模式的服务调用链路及链路的操作方法，以便达到快速止血的能力。

- 服务模式

此模式在拓扑图上只展示到服务名称，不展示节点等细节。当从某个应用服务进入后，会把服务所在的组及其他服务框在一个红色背景椭圆中，以便重点突出。服务所直接依赖的服务绘制到一级拓扑图中。同时为用户提供组内服务局部放大功能，便于观察细节。



- 节点模式

节点模式下，我们可以看到服务调用到的节点分布，及调用节点的失败率和平均RT时间，同时我们可以展示比例细节。

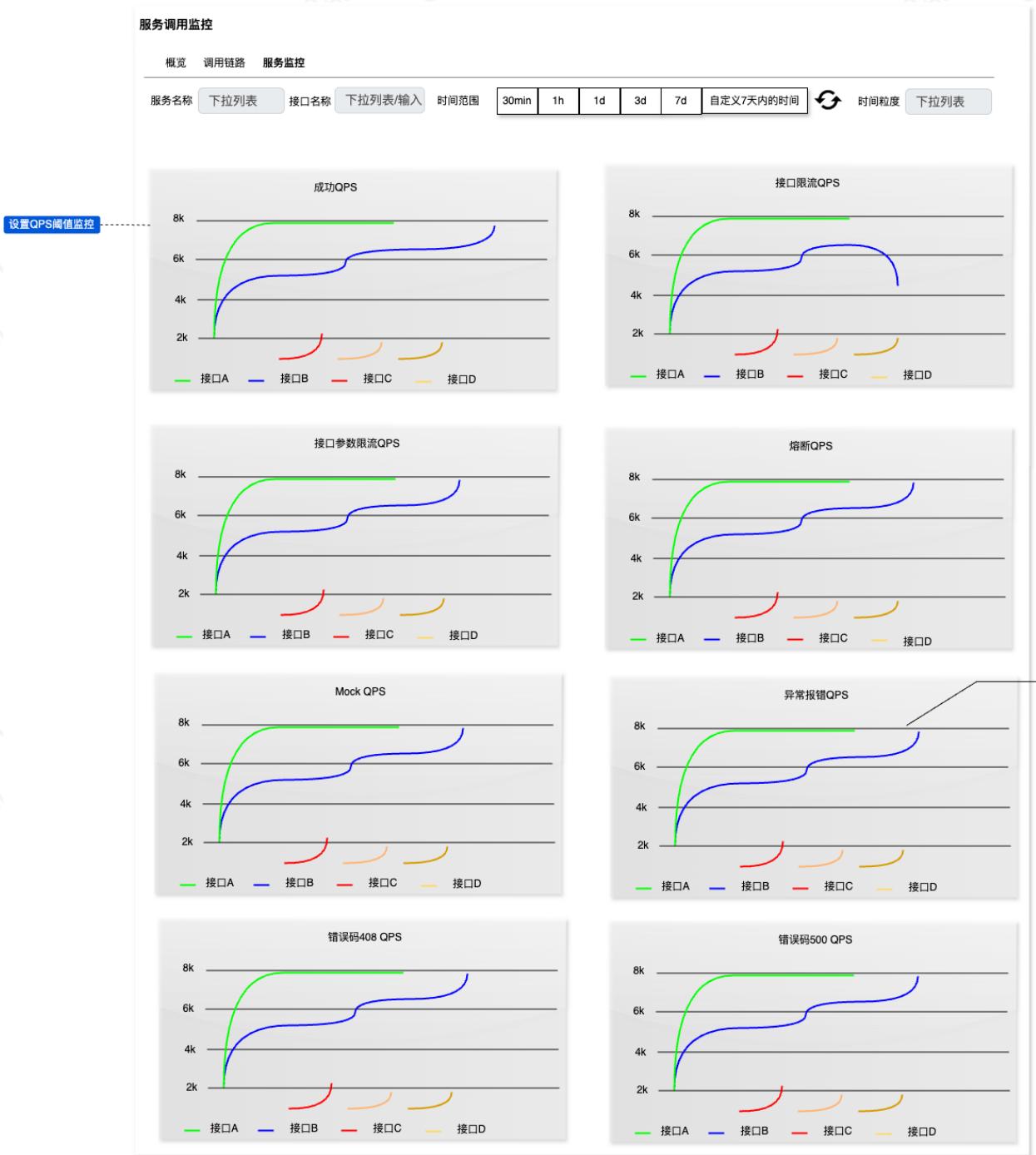
对于服务节点，我们可以展示CPU/内存/负载情况，并设置对应的告警，同时可以了解对应的机房信息及部署节点。

- 接口模式

在接口模式下，我们可以从调用线看到调用的下游接口，及调用接口的最大QPS，失败率，单位时间的统计次数及RT的平均值等信息，同时可以从client或者provider角色设置接口的限流值。

服务监控

服务监控主要是对应用的QPS进行监控展示，如限流的QPS，熔断的QPS，错误码的QPS等



服务管理

流量管理

流量管理主要是指对Provider和Consumer设置对应的流量管控及Consumer调用Provider的请求路由。

其中流量管控包含单机的限流配置、集群的限流配置及接口参数的限流配置。同时也可以对节点进行限流配置，如总体RT、CPU使用率等。

请求路由包含Consumer端路由、Provider端路由和A/B Test。

流量管控

1. 单机限流基本配置

规则名称: 输入框

应用名称: 下拉列表

接口名称: 下拉/输入框

限流类型: 单机限流 集群限流

并发类型: QPS

限流阈值: 大于0.001的数

限流效果: 快速失败 匀速排队

高级配置: 折叠

参数限流: 折叠

单机限流模式如图，包含应用名称、规则名称、接口名称等字段，

规则名称：是一个标识Tag，主要为了方便用户分类查看

应用名称：来源于业务在平台的注册列表，同时受权限控制，只有用户有此应用权限才可以查看此应用

接口名称：此处可以是GRPC或者HTTP的接口名，也可以是用户自定义的一个名称

限流类型：包含单机限流和集群限流

阈值类型：QPS或者并发协程数

限流阈值：用户希望打到的限流值

限流效果：快速失败，当被限流后，直接返回BlockError；匀速排队：当被限流后，在内存进行排队，直到排队超时后返回BlockError

是否启用：是：立刻下发到SDK并实时生效；否，此规则不生效。

2. 集群限流-节点均摊基本配置

规则名称: 输入框

应用名称: 下拉列表

接口名称: 下拉/输入框

限流类型: 单机限流 集群限流

集群策略: 节点均摊 Redis集群

全局阈值: 大于0的数

并发类型: QPS

限流阈值: 大于0.001的数

限流效果: 快速失败 匀速排队

高级配置: 折叠

参数限流: 折叠

对于集群限流-节点均摊模式，需要配置【全局阈值】。节点均摊模式下，【限流阈值】会随着业务注册节点的变化而动态调整单机限流值，计算公式为向上取整($\lceil \text{全局阈值} / \text{节点总数} \rceil$)

3. 集群限流 - redis 集群基本配置

规则名称:

应用名称:

接口名称:

限流类型: 单机限流 集群限流

集群策略: 节点均摊 Redis集群

全局阀值:

失败策略: 直接通过 降级为单机限流

并发类型: QPS

限流阈值:

限流效果:

高级配置:

参数限流:

对于此种模式，由于引入了Redis依赖，在依赖不可用的时候需要进行【失败处理】，此处提供了【直接通过】和【降级为单机限流】两种失败处理策略

4. 限流-高级模式

规则名称:

应用名称:

接口名称:

限流类型: 单机限流 集群限流

并发类型: QPS

限流阈值:

限流效果:

高级配置:

流控模式: 直接 关联

Token策略: Direct Warm Up 内存自适应

参数限流:

在高级模式下，新增了【流控模式】和【Token策略】，

流控模式：默认为【直接】也就是针对设置的接口进行限流，【关联】意思是此接口的限流依赖于关联接口的流量情况，如果关联接口的QPS超过配置的阈值，则限制此接口。这在实现优先级相关的接口时非常实用。

Token策略：默认为【Direct】，也就是说如果配置的QPS为100，那么就是每秒直接生成100个Token；WarmUp意思是是不是一下子生成100个Token，而是依赖于预热和当前系统的繁忙情况，这对于大促场景非常实用；

内存自适应：意味着SDK可以依赖Pod的内存使用率自动调整限流值，在内存使用率低时处理更多请求，在内存使用率高时少处理一部分请求。

5. 限流-参数模式

规则名称: 输入框

应用名称: 下拉列表

接口名称: 下拉/输入框

限流类型: 单机限流 集群限流

并发类型: QPS

限流阈值: 大于0.001的数

限流效果: 快速失败 匀速排队

高级配置: 折叠

参数限流: 展开

添加

参数类型	参数Key	参数Val	限流类型	限流阈值	限流效果
Metadata	biz_type	4	单机限流	100	快速失败
Header	user_id	10001	集群限流	500	排队等待
请求体	shopee_user_id	10002	单机限流	800	快速失败

在【参数限流】模式下，我们可以对请求做个性化的限流。比如对Metadata、Header、请求体等。参数限流一般适用于如服务需要对不同的业务进行限流控制，或者是防刷（如固定区间的用户IP）

6. 限流-节点限流

在【节点限流】模式下，我们可以对应用节点的整体做限流保护，当整体的Load或者RT超过阈值时，就执行限流，以便保护应用节点。

节点限流

规则名称: 输入框

应用名称: 下拉列表

阈值类型: Load 平均RT 协程数 入口QPS CPU使用率

限流阈值: 大于0.001的数

动态路由

Consumer权重路由

Consumer端的权重路由是从Consumer的角度出发设置的权重算法，只对这个Consumer节点生效。当被调服务离线后，其设置的权重配置会自动失效，并在1h后自动移除。

Consumer权重路由

规则名称:

客户端应用:

服务端应用:

设置被调节点权重

机器名称	IP地址	类型	当前状态	权重
Host1	127.0.0.1:15000	基准	生效	10000
Host2	127.0.0.2:15000	变更	生效	8000
Host3	127.0.0.3:15000	变更	离线	10000

Provider权重路由

和Consumer权重路由不一样的是，Provider端设置权重，会对所有的Consumer生效。

Provider权重路由

规则名称:

服务端应用:

设置被调节点权重

机器名称	IP地址	类型	当前状态	权重
Host1	127.0.0.1:15000	基准	生效	10000
Host2	127.0.0.2:15000	变更	生效	8000
Host3	127.0.0.3:15000	变更	离线	10000

自定义路由

自定义路由主要用于灰度验证场景。在自定义路由中，业务可以对不同的请求来源匹配到不同的节点，同时支持精准匹配和模糊匹配规则。

自定义路由

规则名称:

客户端应用:

服务端应用:

输入接口名称 查询

设置匹配规则

客户端接口	匹配来源	匹配Key	匹配条件	匹配值	转发至	优先级	状态
接口A	Metadata	bizType	=	4	10.17.11.10	第一优先级	生效中
接口组B	Header	gray	IN	1,2,3,4	10.17.11.11	第二优先级	生效中
接口C	Query	userId	=	100011	10.18.11.1	第三优先级	生效中
接口D	Query	createTime	>	2022.04.03	10.19.11.8, 10.19.11.9	第四优先级	生效中
	Query	createTime	<=	2024.01.01	10.19.11.8, 10.19.11.9	第四优先级	生效中

优先级高优先匹配

- 接口精准匹配
- 接口前缀匹配条件IN匹配
- 参数匹配
- 同一接口AND匹配

故障管理

故障管理主要是指当被调服务 (provider) 出现故障时的客户端处理策略，我们此处主要指熔断和重试的配置。熔断包含2种类型：服务熔断、节点熔断。同时新增主动探测接口，当接口或者服务熔断时可以按照主动探测策略进行接口试探，一旦成功将会主动关闭熔断开关。

1. 服务级别熔断

对于服务级别熔断，我们会统计所配置接口的调用情况，一旦达到阈值后，会对被调服务进行熔断，可以熔断此接口或者此接口所归属的全部服务。熔断后，如果没有配置熔断响应，则会返回一个固定的错误，用户也可以自定义熔断响应。

主动探测是可选的，如果需要主动探测，需要配置一个没有副作用的请求体，及判断成功的条件，匹配后则标记为恢复。主动探测的接口不一定和直接调用的接口一致。只要是同一个服务的就可以。

服务级熔断

规则名称:

应用名称:

被调服务:

被调接口:

熔断类型: 慢比例调用 异常比例 异常数 返回码

最大RT:

熔断时长:

统计时长:

比例阈值:

最小请求数:

探针数量:

熔断粒度: 接口 服务

高级配置:

[服务级熔断基础配置](#)

服务级熔断

规则名称:

应用名称:

被调服务:

被调接口:

熔断类型: 慢比例调用 异常比例 异常数 返回码

最大RT: 比例阈值:

熔断时长: 最小请求数:

统计时长: 探针数量:

熔断粒度: 接口 服务

熔断后降级: 是 否

熔断后响应:

主动探测: 是 否 开启主动探测后，客户端会按照配置的规则主动对目标进行探测，如果探测成功也会关闭熔断

探测接口:

探测周期:

探测请求体:

探测成功匹配: 匹配值:

服务级别熔断高级配置

2. 实例熔断

当被调服务满足熔断条件后，会对整个实例进行熔断。熔断后可以根据接口统计和主动试探结果确认是否可以关闭熔断。注意：实例熔断不能统计配置熔断响应，只能返回固定错误。

实例级熔断

规则名称:

应用名称:

被调服务:

熔断类型: 异常比例 异常数 返回码

最大RT: 比例阈值:

熔断时长: 最小请求数:

统计时长: 探针数量:

熔断粒度: 实例

主动探测: 是 否 开启主动探测后, 客户端会按照配置的规则主动对目标进行探测, 如果探测成功也会关闭熔断

3. 接口重试

为了增加系统的稳定性, 当接口失败后, 我们可以主动增加接口重试。

接口重试

规则名称:

应用名称:

接口名称:

重试策略:

回退策略:

异常匹配: 是 否 开启异常匹配后, 只有匹配了规则后才重试, 否则只要失败就重试

Mock管理

Mock主要包含Mock预跑和Mock匹配。

Mock预跑主要是在Mock生效前, 通过把规则投放到业务节点, 根据请求的匹配情况得到匹配率, 如果匹配率达标, 则允许Mock规则下发, 否则不允许。在设置【预跑通过】后, 系统会每10s会收集并统计匹配率。一旦规则生效, 系统会自动销毁此统计任务, 直到下次变更。

Mock匹配主要是指可以按照匹配来源对请求进行正确的Mock返回, 同时如果多个条件都满足的情况下支持匹配优先级。为了区分不同用户的Mock数据可以进一步对用户数据进行分组, 以便达到隔离目的。

接口Mock

规则名称:

应用名称:

接口名称:

数据校验: 是 否

预跑通过: 是 否 设置【预跑通过】后, 没有通过前规则不可以【启用】

设置预跑规则

预跑规则名	ref	期望命中率	统计匹配率	状态
预跑1	规则1	100%	100%	开
预跑2	规则2	0%	10%	开
预跑3	规则3	100%	0%	关

设置Mock规则

规则别名	匹配来源	匹配Key	匹配条件	匹配值	分组KV	优先级	Mock数据	状态
risk_reject	Metadata	bizType	=	4	type=WanYan	第一优先级	{"r":"suc"}	生效中
OA打标	Header	gray	IN	1,2,3,4	type=Stress	第二优先级	{}	生效中
系统错误	Query	userId	=	100011	type=Cashloan	第三优先级	systemerror	生效中
压力测试	方法名称	完整接口	=	A	type=Fast	第四优先级	{}	生效中
UC冻结	Cookie	userName	prefix	u1	type=Uc	第四优先级	{"fronzen":true}	生效中

是否启用: 是 否

节点管理

节点管理包含节点所归属应用、节点的内存/CPU信息，节点的机房信息等。

The screenshot displays several components of the Shopee monitoring system:

- 应用列表 (Application List):** Shows a table of applications with columns: 应用名称 (Application Name), 应用类型 (Application Type), 分组 (Group), 健康实例/总实例数 (Healthy Instances/Total Instances), 实例IP (Instance IP), 服务端口 (Service Port), 所在机房 (Data Center), and 操作 (Operations). One row for "cashloan.bif" is highlighted.
- 应用明细 (Application Details):** A detailed view for "输入框" (Input Box) with sections for Application Name, Application Type, Group, Contact Information, Change Notifications, and Monitoring Settings.
- 机房信息 (Data Center Information):** Displays information for "apid1pf81" including归属集群 (Belonging Cluster), 机器IP (Machine IP), and 集群监控地址 (Cluster Monitoring Address).
- 服务器状态 (Server Status):** A table showing CPU and Memory usage for three hosts (Host1, Host2, Host3) over a 15-day period.

场景分析

流量管理

全局限流-节点均摊

网关是Credit TOC/TOB流量的入口，流量较大，网关通过对下游接口配置全局限流-节点均摊保护下游流量。

全局限流-共享Token

网关的部分流量并不是很高，但是对于业务来说为核心接口，如SPL的支付接口，此类接口如果使用均摊模式会由于网关节点流量的不均导致误限制。

如总流量预估500，节点数180个，节点均摊模式下的单机限流值为2，一旦流量分布不均匀将导致限流，此种场景下使用全局共享token模式。

参数限流

UC向Credit的各个业务提供合同生成接口，然而生成合同接口的处理能力有限，需要对各个业务的调用QPS做限制，此时UC根据接口请求参数中的biz_type做限流。

Funding需要和外部银行进行交互，然而银行的处理能力有限，同一个银行的不同下游是通过请求头的不同参数值进行区分的，每个下游的要求不一样，此时Funding按照Header中参数的不同值进行限流。

过载保护

保护业务不被突发流量打垮，同时监控突发流量，做到及时告警

业务防刷

防止恶意用户发送过多流量影响其他正常用户

权重路由

TH某个机房实例由于过保出现故障，导致路由到这个节点的请求全部超时，此时可以配置此类服务节点的权重为0，使调用方不在路由到此节点以便解决故障。

故障管理

接口熔断

新上线接口，由于实现有BUG，导致调用到此接口的请求全部处理失败，通过接口熔断临时下线此接口，并返回特定数据。

UC-Tag服务处理能力有限，如果大量接口请求UC-Tag时有可能会打挂服务，此时需要统计其处理时长，超过某个值后自动熔断。

服务熔断

Credit的所有核心接口都需要调用DR服务进行双写，一旦DR服务出现故障将会影响整个Credit服务。当DR服务出现故障时，快速熔断，当DR服务恢复后继续双写。

UC-Tag服务是一个类白名单服务，如果用户在uc-tag白名单中，会展示Fast服务标签，然而uc-tag是一个可以降级的服务，即便uc-tag挂掉了，让用户看到Fast的激活页面，由于在UC激活流程中会重新校验白名单，所以业务也不会产生问题。

Mock管理

测试自动化

QA通过构造自动化测试用例保证服务的稳定性，然而自身服务需要依赖多个下游服务，下游服务不稳定，QA通过Mock下游接口，保证自身自动化的稳定性。

DR切换验证

DR演练是保证微服务高可用的一个重要手段，然而演练时，依赖的Credit外部服务并不再演练范围内，为了保证验证的正确性，QA Mock非Credit接口，以保证自身业务流程正常。