应用性能监控 APM

用户操作指南

天翼云科技有限公司

一、产品概述

1.1、产品定义

应用性能监控（Application Performance Monitoring，APM）是天翼云可观测体系中的一款子产品，帮助您进行应用性能管理。

背景：传统监控面临诸多问题

随着云和云原生技术的发展，软件基础设施和架构正在向着更加轻量、灵活和高效的方向演进。这虽然为企业提供了更多的选择和可能性，但也暴露出传统监控的诸多问题：

- 传统监控割裂分散，运维监控手段和工具的多样化和碎片化。在这种模式下，各个系统、设备或服务之间的**数据无法实现有效的整合和关联**，这使得故障的定位变得困难，也使得事件分析和数据的应用变得更加复杂。
- 传统监控主要关注于基础设施层面的指标型数据，忽略了对应用服务层的关注。这种做法导致运维人员难以全面了解和评估整体业务运行的健康状态，**无法进行有效的全局监测和判断**。
- 传统监控在面对故障时，往往**难以精确判断故障对业务的具体影响**，从而无法迅速采取有效的应对措施，这不仅包括了对故障影响的**误判误报**，还包括了对某些实际已经影响到业务运行的故障的**漏报**。同时，在故障定位方面，由于现场还原困难、故障定位和恢复效率低下等问题，也严重影响了问题的解决效率。

APM 为您的应用健康保驾护航

可观测性面对云原生架构下的大规模集群以及海量灵活的微服务应用，可以明确知道集群中运行的详细信息，清晰地发现和记录主机快速变化的应用行为，清晰地观察到应用之间复杂的调用关系。

而**应用性能监控**（Application Performance Monitoring，APM）作为天翼云可观测体系中的关键产品。可以基于分布式应用、容器等不同可观测环境与场景提供全链路一体化监控解决方案。帮助您实现全栈性能监控与端到端追踪诊断，为您的应用健康保驾护航。

关键特性

- **可视化监控**：无需配置，自动监控 JVM、基础资源、URL、Exception、SQL 等各类监控指标，并提供可视化图表展示。
- **全链路追踪**：自动发现应用的上下游依赖关系，捕获计算并立体展示不同应用之间组成的调用链，进行全链路拓扑化追踪，轻松发现异常调用。
- **灵活告警**：提供几十项告警指标配置，具备静默策略等告警收敛能力，支持短信、邮件、IM 应用等多通知渠道，满足客户各场景的灵活告警诉求。

客户价值

支撑云上业务的可观测诉求，保障应用健康。

- **全局一站式管理**：深度融合 OpenTracing 标准，拥抱开源生态，支持 Java、Golang、Python、NodeJs 等多语言应用接入，方便客户对多个应用以全局视角进行一站式管理。
- **降低运维成本**：全托管式监控服务，免基础人力运维，按真实用量付费，降低成本。
- **提升运维效率**：立体化监控，隐患主动告警，故障快速诊断，化被动为主动，高效运维！

1.2、产品优势

应用性能监控 APM 的核心产品优势如下：

无入侵式接入

使用 Java agent 进行接入，您无需修改应用代码，不会影响到应用本身的运行。

实时监控

实时获取、处理和分析应用各项数据，方便您及时获取最新的应用健康信息，从而对可能存在的应用性能问题进行更快预防和处理。

数据可视化

提供包括应用实例、接口、数据库在内多个纬度的统计图表，满足开发者在多类场景下进行深入的多角度分析需求。

调用链分析

通过分析和追踪服务之间的调用顺序和层次关系，以拓扑和方法栈的形式进行呈现，帮助您理解和追踪代码的执行流程，方便您快速定位异常问题以及进行性能优化。

高性能&高稳定

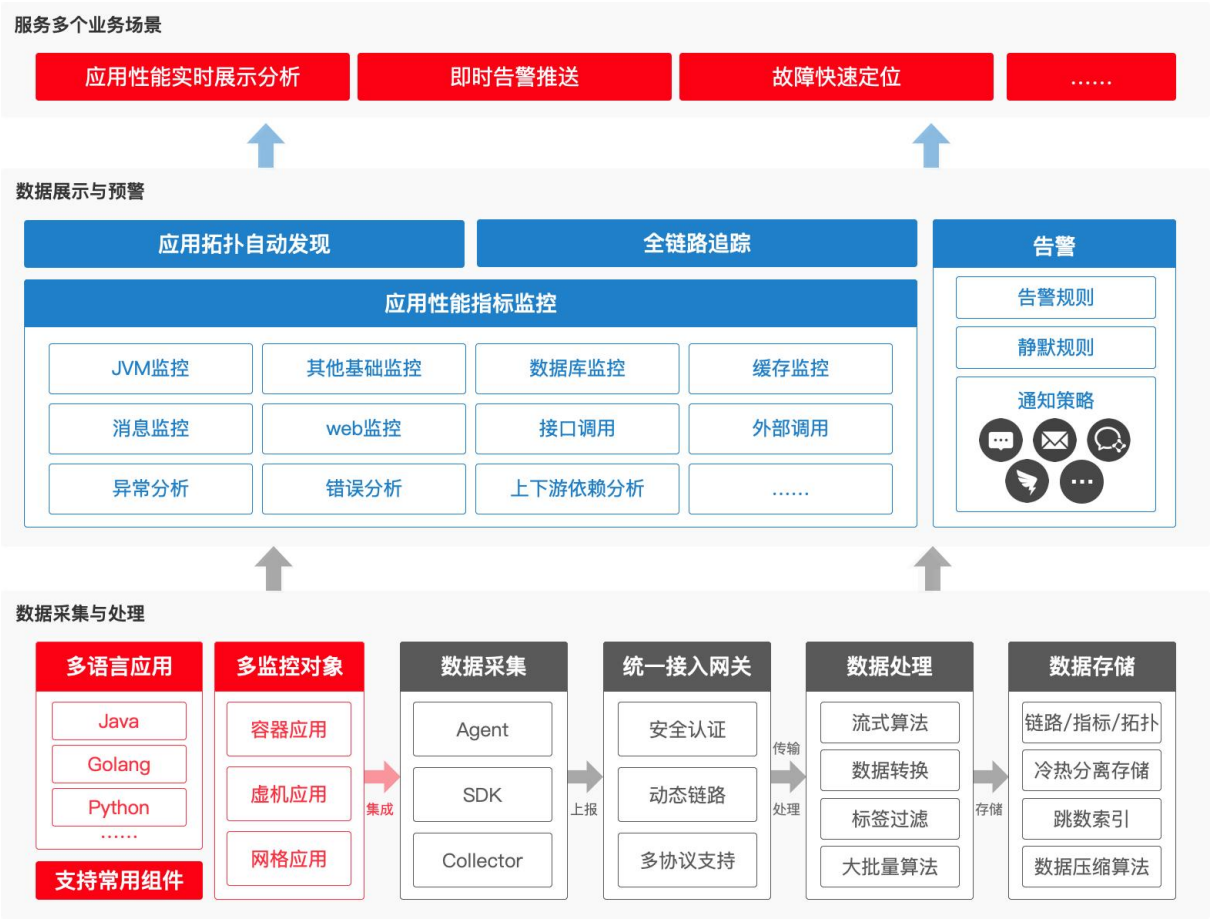
优化探针性能损耗，提供低消耗应用探针；采用自研的 ETL 服务等可提高系统稳定性和可靠性的技术架构和服务，共同确保监测平台稳定可用。

灵活告警

自研智能降噪技术和指纹算法，避免告警通知中的无效告警和重复告警，提供静默策略，支持人为过滤业务上不需要的告警信息。

1.3、功能特性

应用性能监控 APM 是一款云原生可观测产品，包含应用监控、应用拓扑和链路追踪、告警管理等一系列能力。帮助您提高监控效率，减少运维工作量。



应用接入

- **支持多语言接入**：支持包括 java 在内的更多语言，诸如常见的 go、python、node.js 等等。
- **兼容多协议接入**：基于 OpenTelemetry 标准，全面兼容 Jaeger、SkyWalking 等多种开源产品。
- **Java 应用无侵入式接入**：无需客户修改应用代码，只需为应用安装一个探针，不会影响到应用本身运行。

应用监控

- **JVM 监控分析**：包含内存、GC、线程等监控。
- **其他基础监控**：包含资源监控、Netty 内存和 Java 方法（即将上线）。
- **数据库监控**：提供各类数据库的监控展示。包含 MySQL、ClickHouse、Postgresql、Elasticsearch、MongoDb、DBCP 连接池、Druid 连接池、C3P0 连接池等。
- **缓存监控**：提供各类缓存的监控展示。包含 Redis、Jedis、Lettuce 等。
- **消息监控**：提供各类消息的监控展示。包含 kafkaConsumer、KafkaProducer、RabbitMqConsumer、RabbitMqProducer 等。
- **web 容器监控**：包含 tomcat 监控。
- **各类调用监控**：包含接口调用、外部调用监控等。
- **异常错误分析**：综合分析应用异常和错误情况，提供趋势图及问题明细，对错、慢 SQL 等常见问题进行更细致的分析。
- **上下游依赖分析**：对链路上下游进行监控展示。

链路追踪

- **自动发现应用拓扑**：通过拓扑图更加直观地看到应用的上下游组件以及与它们的调用关系，同时提供应用请求总量、请求耗时等关键指标，帮助客户更快速地找出应用的瓶颈。
- **链路分析**：一次请求将会被记录为一条调用链路数据，每条调用链数据都包含着详细的调用数据，包括服务名称、接口名称、方法名称、调用类型、平均耗时等。通过调用栈逐层查看异常发生的具体节点，可以更快确认问题可能原因。

告警管理

- **多指标告警配置**：提供接口调用、HTTP 返回码、异常、JVM 内存与线程、数据库等几十项告警指标配置。
- **多通知渠道**：支持短信、邮件、翼连、webhook（企业微信、钉钉、飞书等）。

- **静默策略**：支持组合配置且或条件，设置静默时间段。

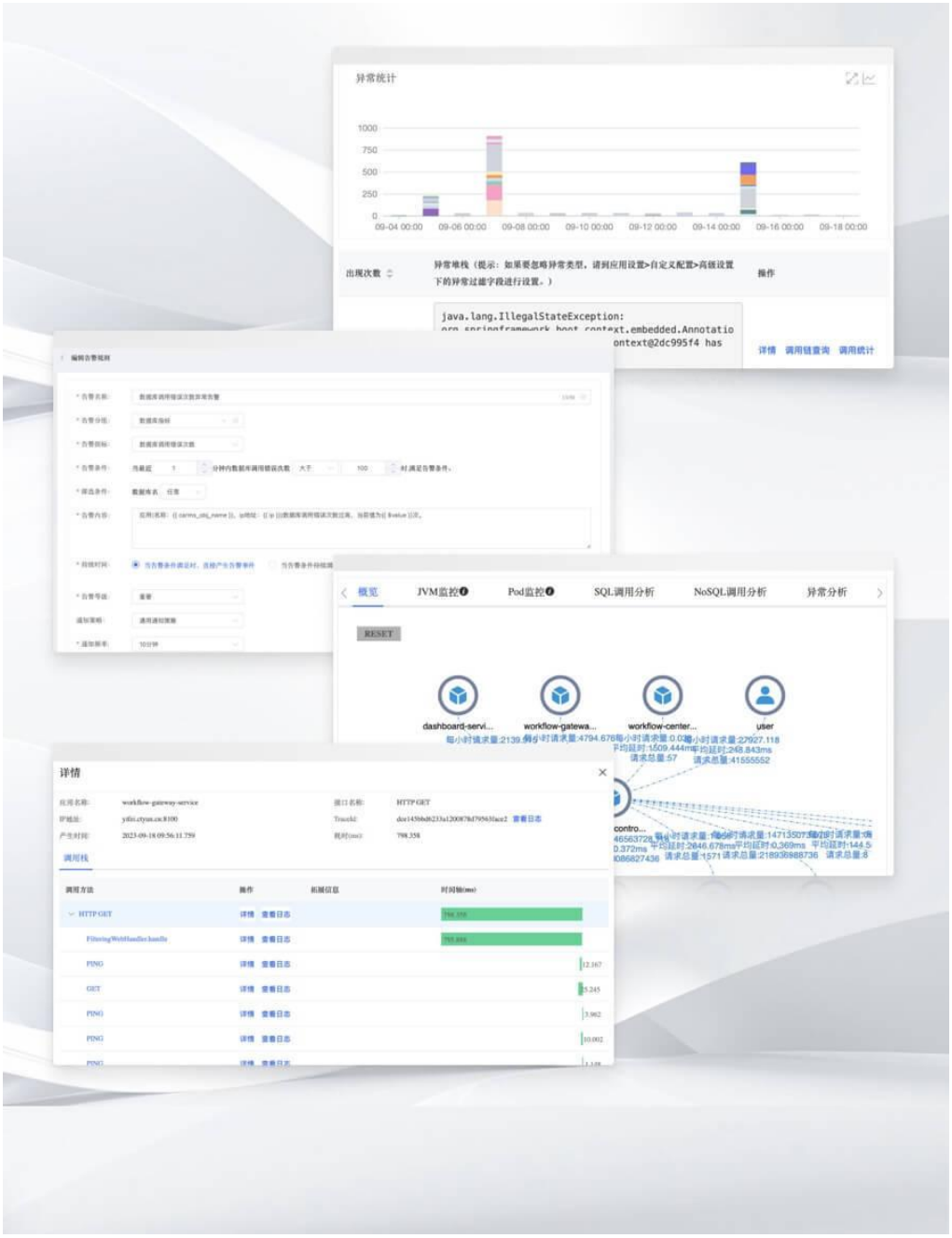
1.4、应用场景

应用场景如下：

场景 1:微服务架构应用性能监控

场景说明

使用分布式微服务架构的应用，虽然研发交付效率高，但处理架构梳理、性能分析、异常定位等问题的难度却陡增。APM 提供了大型分布式应用的异常诊断能力，当应用出现请求失败或性能下降时，通过应用拓扑、调用链、性能指标监控等能力组合，可以帮助用户快速定位问题。



业务价值

- **多语言接入**：同一套标准，支持 Java、Go、Python、node.js 等多语言应用接入，同时以拓扑图的方式全面展示相关应用的调用关系。
- **丰富的指标监控**：提供包括 JVM、java 方法、主机/Pod 等基础监控；Kafka、RocketMQ 等消息监控；Mysql，redis，es 等数据库监控；httpClient、grpc 等调用监控。
- **慢 SQL 分析**：通过自定义的慢查询阈值、结合 SQL 的调用频次，获取导致数据库性能下降的不规范的 SQL 语句。

- **告警**：针对接口响应时间、异常调用、数据库、JVM 等性能指标做一定阈值的告警，先于客户之前发现并解决问题。

场景 2:定位系统故障异常

场景说明

微服务架构下的分布式应用在提供更灵活组合的同时也带来了更复杂的监控诊断问题。一个服务出现问题，可能其上的多个应用都会出现异常。通过链路追踪能力，可以帮助用户快速追溯异常原因，定位本质问题。

业务价值

- **应用拓扑展示**：自动梳理业务应用，以拓扑图的方式全面展示相关应用调用关系。
- **链路层级展示**：当出现异常调用，通过调用链能力查看完整调用链路详情，支持链路层层拆解，方便追踪异常。

- **查看日志**：对接“云日志服务”，需要在应用设置中开启日志，即可在链路详情中查看对应调用方法的日志信息，异常可定位到代码级。

1.5、指标总览

1.5.1、JVM 监控

（1）内存指标说明表

指标类别	指标	指标说明	数据类型
jvm 内存使用量	PS Old Gen	老年代大小	long
	Code Cache	代码缓存区大小	long
	PS Eden Space	伊甸区大小	long
	PS Survivor Space	servivor 区大小	long
	Metaspace	元空间大小	long
	Compressed Class Space	类压缩空间大小	long
jvm 最大可用内存量	PS Old Gen	老年代大小	long
	Code Cache	代码缓存区大小	long
	PS Eden Space	伊甸区大小	long
	PS Survivor Space	servivor 区大小	long
	Metaspace	元空间大小	long
	Compressed Class Space	类压缩空间大小	long

非堆内存使用量	jvm.non_heap.used	非堆内存使用量	long
非堆内存最大容量	jvm.non_heap.max	非堆内存最大容量	long
jvm 映射池大小	used	已使用内存	long
	count	总量	long
	capacity	初始化值	long
堆内存最大值	jvm.heap.max	堆内存最大值	long
堆内存已使用量	jvm.heap.used	堆内存已使用量	long
jvm 内存提交量	PS Old Gen	老年代大小	long
	Code Cache	代码缓存区大小	long
	PS Eden Space	伊甸区大小	long
	PS Survivor Space	servivor 区大小	long
	Metaspace	元空间大小	long
	Compressed Class Space	类压缩空间大小	long
jvm 非堆内存初始化大小	jvm.non_heap.init	jvm 非堆内存初始化大小	long
堆内存提交数	jvm.heap.committed	堆内存提交数	long
非堆内存提交数	jvm.non_heap.committed,	非堆内存提交数	long
堆内存初始化大小	jvm.heap.init	堆内存初始化大小	long

整体内存初始化大小	PS Old Gen	老年代大小	long
	Code Cache	代码缓存区大小	long
	PS Eden Space	伊甸区大小	long
	PS Survivor Space	servivor 区大小	long
	Metaspace	元空间大小	long
	Compressed Class Space	类压缩空间大小	long

(2) GC 指标说明表

指标类别	指标	指标说明	数据类型
GC 详情	dataSendSize	发送到网关数据大小	long
	jvm.gc.count.delta	最近发生的收集次数	long
	jvm.gc.count	已发生的收集数	long
	jvm.gc.time.delta	最近发生的 gc 时间	long
	jvm.gc.time	已发生的 gc 时间	long
	jvm.direct.pool	JVM 直接内存池大小	long

(3) 线程指标说明表

指标类别	指标	指标说明	数据类型

线程池大小	new	创建线程数	long
	blocked	blocked 线程数	long
	waiting	waitting 线程数	long
	total	线程数总量	long
	runable	runable 线程数	long
	live	live 线程数	long
	deamon	守护线程数	long
	timed_waiting	timed_waiting 线程数	long
	deadlock	死锁线程数	long

1.5.2、其他基础监控

（1）资源监控指标说明表

指标类别	指标	指标说明	数据类型
内存	host.memory.totalMemory	节点级别内存大小	long
	host.memory.usage	节点级别已使用内存大小	long
	host.memory.freeMemory	节点级别空闲内	long

		存大小	
磁盘	host.disk.total	节点级别磁盘大小	double
	host.disk.usage	节点级别磁盘已使用大小	double
	host.disk.free	节点级别未使用磁盘大小	double
cpu	process.runtime.jvm.cpu.utilization	进程级别最近的 cpu 使用率	double
	process.runtime.jvm.system.cpu.utilization	整个系统最近的 cpu 利用率	double
	process.runtime.jvm.system.cpu.load_1m	一分钟整个系统的平均 CPU 负载	double
类	process.runtime.jvm.classes.unloaded	自 JVM 启动以来卸载的类数	long
	process.runtime.jvm.classes.loaded	自 JVM 启动以来加载的类数	long
	process.runtime.jvm.classes.current_loaded	当前加载的类数	long

正在运行的线程数	process.runtime.jvm.threads.count	正在运行的线程数	long
pod 级别网络 IO	rx_dropped	接收丢包大小	long
	tx_bytes	发送总大小	long
	tx_errors	发从错误数大小	long
	tx_dropped	发送丢包大小	long
	tx_packets	发送包数	long
	rx_bytes	接收总大小	long
	rx_packets	接收包数	long
节点级别网络 IO	rx_dropped	接收丢包大小	long
	tx_bytes	发送总大小	long
	tx_errors	发从错误数大小	long
	tx_dropped	发送丢包大小	long
	tx_packets	发送包数	long
	rx_bytes	接收总大小	long
	rx_packets	接收包数	long

(2) Netty 内存指标说明表

指标类别	指标	指标说明	数据类型
Netty 内存	netty.max.direct.memory	netty 直接内存最大量	double
	netty.used.direct.memory	netty 直接内存使用量	double

1.5.3、数据库

（1）MySQL 指标说明表

指标类别	指标	指标说明	数据类型
异常（exception，sql 调用发送的异常统计信息）	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT
sql 监控(sql,以 sql 为维度统计 sql 调用详情)	sql	sql	STRING
	failedCount	错误数	INT
	count	请求数	INT
	totalCost	总耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT

	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
汇总（total，sql 调用的汇总数据统计）	count	请求数	INT
	totalCost	总耗时	LONG
	failedCount	错误数	INT
数据库连接（connection，以数据库为维度统计 sql 调用详情）	name	数据库名称	STRING
	failedCount	错误数	INT
	count	请求数	INT
	maxCost	最大耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT

	totalCost	总耗时	LONG

(2) ClickHouse 指标说明表

指标类别	指标	指标说明	数据类型
异常（exception，sql 调用发送的异常统计信息）	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT
sql 监控(sql,以 sql 为维度统计 sql 调用详情）	sql	sql	STRING
	failedCount	错误数	INT
	count	请求数	INT
	totalCost	总耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT

	msMorethan10000Count	10s 以上次数	INT
汇总（total，sql 调用的汇总数据统计）	count	请求数	INT
	totalCost	总耗时	LONG
	failedCount	错误数	INT
数据库连接（connection，以数据库为维度统计 sql 调用详情）	name	数据库名称	STRING
	failedCount	错误数	INT
	count	请求数	INT
	maxCost	最大耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	totalCost	总耗时	LONG

（3）Elasticsearch 指标说明表

指标类别	指标	指标说明	数据类型
异常(exception，EsRestClient 调用的异常信息统计)	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT
url 监控(esClient，以被调用的 url 为维度统计接口调用信息)	url	url	STRING
	total_count	该 url 的调用次数	INT
	error_count	错误次数	INT
	duration_max	该 url 在采集周期内最大响应时间	LONG
	total_cost	总响应时间	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT

（4）MongoDb 指标说明表

指标类别	指标	指标说明	数据类型
异常（exception，sql 调用发送的异常统计信息）	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT
sql 监控（sql，以 sql 为维度统计 sql 调用详情）	sql	sql	STRING
	failedCount	错误数	INT
	count	请求数	INT
	totalCost	总耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
汇总（total，sql 调	count	请求数	INT

用的汇总数据统计）	totalCost	总耗时	LONG
	failedCount	错误数	INT
MongoDb 调用监控（client，MongoDb 调用监控）	name	数据库名称	STRING
	failedCount	错误数	INT
	count	请求数	INT
	maxCost	最大耗时	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	totalCost	总耗时	LONG
MongoDb 集群调用监控（cluster，MongoDb 集群调用监控）	hostName	ip 和 port	STRING
	failedCount	错误数	INT
	count	请求数	INT
	maxCots	最大耗时	LONG
	totalCost	总响应时间	LONG

	ms100To500Count	100-500ms 次数	INT
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT

(5) DBCP 连接池指标说明表

指标类别	指标	指标说明	数据类型
数据源（dataSource，数据源）	db_dbcp_initialSize	初始化连接数	INT
	db_dbcp_minIdle	连接池最小空闲数	INT
	db_dbcp_maxIdle	连接池最大空闲数	INT
	db_dbcp_numIdle	最大响应时间	INT
	db_dbcp_timeBetweenEvictionRunsMillis	验证连接是否有效的时间周期	INT

	db_dbcp_numActive	活跃连接数	INT
	db_dbcp_maxWaitMillis	在抛出异常之前，池等待连接被回收的最长时间（当没有可用连接时）。	INT

(6) Druid 连接池指标说明表

指标类别	指标	指标说明	数据类型
数据源（dataSource,数据源）	URL	jdbcURL	STRING
	dbType	数据库类型	STRING
	driverClassName	驱动	STRING
	db_druid_initialsize	初始化连接数	INT
	db_druid_minIdle	连接池最小空闲数	INT
	db_druid_maxIdle	连接池最大空闲数	INT
	db_druid_maxActive	连接池大小上限	INT

	db_druid_waitThreadCount	等待线程数	INT
	db_druid_MaxwaitThreadcount	等待线程数上限	INT
	db_druid_PoolingCount	池中连接数	INT
	db_druid_PoolingPeak	最大池中连接数	INT
	db_druid_ActiveCount	活跃连接数	INT
	db_druid_ActivePeak	最大活跃连接数	INT
	db_druid_ConnectCount	获取连接总数	INT
	db_druid_Maxwait	获取连接最大等待时间	INT
	db_druid_RemoveAbandonedcount	超时连接回收次数	INT
	db_druid_RemoveAbandonedTimeoutMillis	如果池中连接被获取且超过该时长未被还回，则回收该连接	INT

	db_druid_TimeBetweenEvictionRunsMillis	检查池中连接空闲周期	INT
	db_druid_MinEvictableIdleTimeMillis	池中连接可空闲的时间	INT

(7) C3P0 连接池指标说明表

指标类别	指标	指标说明	数据类型
数据源（dataSource,数据源）	db_c3p0_numIdleConnections	空闲连接数	INT
	db_c3p0_numBusyConnections	活跃连接数	INT
	db_c3p0_numConnections	获取连接总数	INT
	db_c3p0_maxIdleTime	连接最大空闲时间	INT
	db_c3p0_idleConnectionTestPeriod	空闲连接检查周期	INT
	db_c3p0_acquireRetryAttempts	获取连接重试次数	INT
	db_c3p0_acquireRetryDelay	获取连接重试间隔	INT
	db_c3p0_acquireIncrement	无连接可用时创	INT

		建连接数	

1.5.4、缓存

（1）Redis 指标说明表

指标类别	指标	指标说明	数据类型
REDIS 调用详情（detail，调用详情指标集）	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	count	请求数	INT
	SUM	总响应时间	INT
	maxCost	最大响应时间	INT
	failedCount	错误数	INT
主机汇总（host，	ms0To10Count	0-10ms 次数	INT

主机汇总指标集）	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	count	请求数	INT
	SUM	总响应时间	INT
	maxCost	最大响应时间	INT
	failedCount	错误数	INT

（2）Jedis 指标说明表

指标类别	指标	指标说明	数据类型
连接池信息	maxTotal	最大连接数	INT
	maxIdle	最大空闲数	INT
	minIdle	最小空闲数	INT
	numActive	当前激活个数	INT

	numIdle	当前空闲个数	INT
	numWaiters	等待个数	INT
	maxWaitMillis	最大等待时间（单位：ms）	INT
	maxBorrowWaitTimeMillis	borrow 最大等待时间（单位：ms）	INT
	meanActiveTimeMillis	平均激活时间（单位：ms）	INT
	meanBorrowWaitTimeMillis	平均 borrow 等待时间（单位：ms）	INT

（3）Lettuce 指标说明表

指标类别	指标	指标说明	数据类型
lettuce 主备切换（switch，主备切换指标集）	maxTotal	最大连接数	INT
	maxIdle	最大空闲数	INT
	minIdle	最小空闲数	INT
lettuce 客户端信息（clientInfo，客户	numActive	当前激活个数	INT
	numIdle	当前空闲个数	INT

端信息指标集）	numWaiters	等待个数	INT
	maxWaitMillis	最大等待时间（单位：ms）	INT
	maxBorrowWaitTimeMillis	borrow 最大等待时间（单位：ms）	INT
	meanActiveTimeMillis	平均激活时间（单位：ms）	INT
	meanBorrowWaitTimeMillis	平均 borrow 等待时间（单位：ms）	INT

1.5.5、消息队列

（1）kafkaConsumer 监控指标说明表

指标类别	指标	指标说明	数据类型
主题（topic，kafka 的 topic 监控数据)	id	clientid 和 ip 信息	ENUM
	topic	kafka 的 topic 名称	ENUM
	kafka_consumer_bytes_consumed_rate	每秒消费字节	INT

	kafka_consumer_fetch_size_avg	请求获取平均字节	INT
	kafka_consumer_fetch_size_max	请求获取最大字节	INT
	kafka_consumer_records_consumed_rate	每秒消费消息数	INT
	kafka_consumer_records_per_request_avg	单次请求平均消息数	INT
	kafka_consumer_records_consumed_rate	总消费次数	INT
	kafka_consumer_records_per_request_avg	总消费字节数	INT
kafka 消费方法监控（consumer，kafka 消费方法监控)	errorCount	错误数	INT
	invokeCount	调用次数	INT
	maxTime	最慢调用	INT
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT

	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	totalTime	总响应时间	INT
汇总（total，KafkaConsumer 汇总信息统计）	recordConsumedTotal	总消费次数	INT
	bytesConsumedTotal	总消费字节数	INT
异常(exception，kafka 消费异常信息)	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT

（2）KafkaProducer 监控指标说明表

指标类别	指标	指标说明	数据类型
异常	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING

	exception_count	错误数	INT
topic (topic，kafka 的 topic 监控数据)	id	clientid 和 ip 信息	ENUM
	topic	kafka 的 topic 名称	ENUM
	kafka_producer_byte_rate	每秒发送字节	INT
	kafka_producer_record_error_rate	每秒错误数	INT
	kafka_producer_record_retry_rate	每秒重试数	INT
	kafka_producer_record_send_rate	每秒发送数	INT
	kafka_producer_record_send_total	总发送次数	INT
	kafka_producer_byte_total	总发送字节数	INT
汇总（total，KafkaProducer 汇总信息统计）	kafka_producer_record_send_total	总发送次数	INT
	kafka_producer_byte_total	总发送字节数	INT
发送方法（doSendMethod，发送消息方法监控）	topic	kafka 的 topic 名称	ENUM
	errorCount	错误数	INT
	invokeCount	调用次数	INT

	maxTime	最慢时延	INT
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	totalTime	调用总耗时	INT

（3）RabbitMqConsumer 监控指标说明表

指标类别	指标	指标说明	数据类型
异常	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING
	exception_count	错误数	INT
推模式消费维度监控	message_error_count	消费消息的错误次数	INT
	ms0To10Count	0-10ms 次数	INT

	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	message_count	调用次数	INT
connection 监控	not_acknowledged_published	该连接中未确认的消息数	INT
total 监控	message_error_count	消费消息的错误次数	INT
	message_count	消费消息数	INT
	message_total_payload	消费字节数	INT
	Max(message_total_payload)	单次消费最大字节数	INT
	acknowledged	ack 消息数	INT
	rejected	reject 消息数	INT
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT

	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	not_acknowledged_published	未确认的消息数	INT
	message_max_cost	最大响应时间	INT
	sum(message_cost)	消费消息的总响应时间	INT

（4）RabbitMqProducer 监控指标说明表

指标类别	指标	指标说明	数据类型
异常	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_count	错误数	INT
exchange 监控	message_destination	exchange 名	ENUM
	message_error_count	推送消息的错误次数	INT
	message_count	推送消息数	INT
	message_total_payload	推送字节数	INT

	Max(message_total_payload)	单次推送最大字节数	INT
	message_max_cost	推送消息的最大响应时间	INT
	sum(message_cost)	推送消息的总响应时间	INT
total 监控	message_total_payload	推送字节数	INT
	message_count	推送消息数	INT
	message_error_count	推送消息的错误次数	INT
	message_max_cost	推送消息的最大响应时间	INT
	sum(message_cost)	推送消息的总响应时间	INT
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT

1.5.6、接口调用

（1）URL 指标说明表

指标类别	指标	指标说明	数据类型
汇总（total ，url 接口调用汇总统计数据）	error_count	总错误次数	INT
	total_count	总调用次数	INT
	total_cost	总响应时间	LONG
状态码（statuscode,以接口返回的状态码维度统计接口调用数据）	code	状态码	STRING
	total_count	该状态码的发生次数	INT
url 监控（url，以 url 维度统计接口调用数据）	error_count	该 url 的错误数	INT
	total_count	该 url 的调用次数	INT
	duration_max	该 url 在采集周期内最大响应时间	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT

	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
	total_cost	总响应时间	LONG
集群调用（user，以调用方集群 id 维度统计接口调用数据）	HTTP_CODE_4XX	4xx 数量	INT
	HTTP_CODE_5XX	5xx 数量	INT
	total_cost	总响应时间	LONG
	error_count	该 url 的错误数	INT
	max_cosgt	最大耗时	LONG

1.5.7、外部调用

（1）HttpClient 指标说明表

指标类别	指标	指标说明	数据类型
异常（exception，httpclient 调用的异	exception_stacktrace	异常产生的堆栈信息	STRING
	exception_type	异常类型	STRING

常信息统计）	exception_count	错误数	INT
url 监控（invocation，以被调用的 url 为维度统计接口调用信息）	url	url 信息	STRING
	total_count	该 url 的调用次数	INT
	error_count	错误次数	INT
	duration_max	该 url 在采集周期内最大响应时间	LONG
	total_cost	总响应时间	LONG
	ms0To10Count	0-10ms 次数	INT
	ms10To100Count	10-100ms 次数	INT
	ms100To500Count	100-500ms 次数	INT
	ms500To1000Count	500-1000ms 次数	INT
	ms1000To10000Count	1-10s 次数	INT
	msMorethan10000Count	10s 以上次数	INT
汇总（total ，httpclient 接口调用的汇总信息统计）	total_count	总的调用次数	LONG
	total_cost	总响应时间	LONG

1.5.8、Web 容器

（1）tomcat 指标说明表

指标类别	指标	指标说明	数据类型
tomcat 信息	http_server_tomcat_threads	当前线程数	INT
	http_server_tomcat_errorCount	错误数	INT
	http_server_tomcat_requestCount	请求数	INT
	http_server_tomcat_maxTime	请求处理最大时间(ms)	INT
	http_server_tomcat_processingTime	请求处理时间(ms)	INT
	http_server_tomcat_traffic	请求吞吐量	INT
	http_server_tomcat_sessions_activeSessions	活跃会话数	INT

1.5.9、上下游分析

（1）上游应用指标说明表

指标类别	指标	指标说明	数据类型
上游应用详	queueSize	请求排队数量	long

情	exceptionTransactionCountMapSize	异常事务计数大小	long
	appTransactionDurationCountMapSize	应用事务持续时间	long
	appTransactionCountMapSize	应用事务次数	long
	exceptionTransactionDurationCountMapSize	异常事务持续时间	long
	appInCallDurationCountMapSize	应用内部请求持续时间	long
	appOutCallCountMapSize	应用外部请求持续次数	long
	appTransactionErrorCountMapSize	应用事务错误次数	long
	appOutCallDurationCountMapSize	应用外部请求耗时	long
	appInCallErrorCountMapSize	应用内部调用错误次数	long
	app_outcall_count	应用请求数量	long

	app_outcall_duration	应用请求耗时	double
	app_incall_count	应用提供调用请求数量	long
	app_incall_duration	应用提供调用请求耗时	long
	app_outcall_error_count	应用请求错误量	long
	app_incall_error_count	应用提供调用错误量	long
	appOutCallErrorCountMapSize	应用提供调用请求耗时	long
	appInCallCountMapSize	应用提供调用请求数量	long
	httpCodeTransactionCountMapSize	http 请求返回码统计	long
	aggregationQueueSize	聚合队列大小	long
	aggregationExporterFailMetrics	聚合输出失败指标大小	long
	arms_heart_beat_deployment	心跳	long

(2) 下游应用指标说明表

指标类别	指标	指标说明	数据类型
下游应用详情	db_outcall_duration	db 请求排队数量	double
	db_outcall_count	db 请求数量	double
	db_outcall_error_count	db 请求错误数据	-
	dbOutcallDurationMapSize	db 外部调用持续时间	long
	dbOutcallCountMapSize	db 外部调用数量	long
	dbOutcallErrorCountMapSize	db 外部调用错误数量	long

1.6、使用限制

本节主要介绍应用性能监控的各语言使用限制

1.6.1、自研 JAVA 探针使用限制

类型	名称	版本
工具	JDK	jdk8、jdk11、jdk17

通讯协议	httpclient	apache httpclient2.0+、apache asynchttpclient1.9+
Java 框架	spring	3.1.x ~ 5.0.x
	springboot	1.2.x~1.5.x、2.0.4 ~ 2.0.9
	Dubbo	2.7+
	gRPC	1.6+
数据库	MySQL	mysql-connector-java 5.1.X
	Oracle	ojdbc5、ojdbc6、ojdbc14
	druid	1.0
	c3p0	0.9.2+
	dbcp2	2.0+
	JDBC	java8+
web 服务器	Tomcat	7.0.x, 8.5.x, 9.0.x, 10.0.x
消息队列	RabbitMQ	spring-rabbit1.0+、amqp-client 2.7+
	Kafka	kafka-producer 0.11+、kakfa-consumer 0.11+、kafka-stream 0.11+
NoSQL	Redis	jedis 1.4+，lettuce 4.0+
	Mongodb	3.1+

	ElasticSearch	5.0+
Rest Client	Common HTTP	java http client java11+、 HttpURLConnection java8+

1.6.2、Go 使用限制

（1）Opentelemetry 支持的框架列表

OpenTelemetry 提供了若干半自动埋点插件，支持为常见的框架自动创建 Span。支持的框架详情请参见 Opentelemetry 官方文档

（2）Skywalking 支持的框架列表

支持自动埋点的依赖库和框架请参考 skywalking-go

框架	版本
gin	1.7.0~1.9.0
http	1.17~1.20
go-restfulv3	3.7.1~3.10.2
mux	1.7.0~v1.8.0
fiber	2.49.0~2.50.0
echov4	4.0.0~4.11.4
dubbo	3.0.1~3.0.5
kratosv2	2.3.1~v2.6.2

microv4	4.6.0~4.10.2
gRPC	1.55.0~1.57.0
gorm	1.22.0~1.25.1
mongo	1.11.1~1.11.7
Native SQL	1.17~1.20
MYSQL Driver	1.4.0~1.7.1
go-redisv9	9.0.3~9.0.
rabbitMQ	2.1.2
logrus	1.8.2~1.9.3

（3）Jaeger 支持的框架列表

目前 OpenTracing 社区已有许多组件可支持各种 Go 框架，详情请参见 OpenTracing API Contributions。

1.6.3、Python 使用限制

（1）OpenTelemetry 支持的框架

OpenTelemetry 提供了若干自动埋点插件，支持为常见框架自动创建 Span，支持的框架列表如下，完整信息请参见 OpenTelemetry 官方文档。

说明：`版本~= V.N` 表示 `≥ V.N` 且 `== V.*`，例如，`aiohttp ~= 3.0` 表示 aiohttp 版本要求 `aiohttp ≥ 3.0` 且 `aiohttp == 3.*`。

框架&版本	插件

aio_pika >= 7.2.0, < 10.0.0	opentelemetry-instrumentation-aio-pika
aiohttp ~= 3.0	opentelemetry-instrumentation-aiohttp-client
aiohttp ~= 3.0	opentelemetry-instrumentation-aiohttp-server
aiopg >= 0.13.0, < 2.0.0	opentelemetry-instrumentation-aiopg
asgiref ~= 3.0	opentelemetry-instrumentation-asgi
asyncio	opentelemetry-instrumentation-asyncio
asyncpg >= 0.12.0	opentelemetry-instrumentation-asyncpg
aws_lambda	opentelemetry-instrumentation-aws-lambda
boto~=2.0	opentelemetry-instrumentation-boto
boto3 ~= 1.0	opentelemetry-instrumentation-boto3sqs
botocore ~= 1.0	opentelemetry-instrumentation-botocore
cassandra-driver ~= 3.25,scylla-driver ~= 3.25	opentelemetry-instrumentation-cassandra
celery >= 4.0, < 6.0	opentelemetry-instrumentation-celery

confluent-kafka >= 1.8.2, <= 2.4.0	opentelemetry-instrumentation-confluent-kafka
dbapi	opentelemetry-instrumentation-dbapi
django >= 1.10	opentelemetry-instrumentation-django
elasticsearch >= 6.0	opentelemetry-instrumentation-elasticsearch
falcon >= 1.4.1, < 4.0.0	opentelemetry-instrumentation-falcon
fastapi ~= 0.58	opentelemetry-instrumentation-fastapi
flask >= 1.0	opentelemetry-instrumentation-flask
grpcio ~= 1.27	opentelemetry-instrumentation-grpc
httpx >= 0.18.0	opentelemetry-instrumentation-httpx
jinja2 >= 2.7, < 4.0	opentelemetry-instrumentation-jinja2
kafka-python >= 2.0	opentelemetry-instrumentation-kafka-python
logging	opentelemetry-instrumentation-logging
mysql-connector-python ~= 8.0	opentelemetry-instrumentation-mysql

mysqlclient < 3	opentelemetry-instrumentation-mysqlclient
pika >= 0.12.0	opentelemetry-instrumentation-pika
psycopg >= 3.1.0	opentelemetry-instrumentation-psycopg
psycopg2 >= 2.7.3.1	opentelemetry-instrumentation-psycopg2
pymemcache >= 1.3.5, < 5	opentelemetry-instrumentation-pymemcache
pymongo >= 3.1, < 5.0	opentelemetry-instrumentation-pymongo
PyMySQL < 2	opentelemetry-instrumentation-pymysql
pyramid >= 1.7	opentelemetry-instrumentation-pyramid
redis >= 2.6	opentelemetry-instrumentation-redis
remoulade >= 0.50	opentelemetry-instrumentation-remoulade
requests ~= 2.0	opentelemetry-instrumentation-requests
scikit-learn ~= 0.24.0	opentelemetry-instrumentation-sklearn
sqlalchemy	opentelemetry-instrumentation-sqlalchemy
sqlite3	opentelemetry-instrumentation-sqlite3

starlette ~= 0.13.0	opentelemetry-instrumentation-starlette
psutil >= 5	opentelemetry-instrumentation-system-metrics
threading	opentelemetry-instrumentation-threading
tornado >= 5.1.1	opentelemetry-instrumentation-tornado
tortoise-orm >= 0.17.0	opentelemetry-instrumentation-tortoiseorm
urllib	opentelemetry-instrumentation-urllib
urllib3 >= 1.0.0, < 3.0.0	opentelemetry-instrumentation-urllib3
wsgi	opentelemetry-instrumentation-wsgi

(2) SkyWalking 支持的框架

Skywalking-python 是 SkyWalking 的 Python Agent 官方库，可以通过接入 Skywalking-python 实现对 Python 应用的监控，Skywalking-python 支持 Kafka、HTTP、AIOHTTP、Redis、WebSockets 等多种第三方库的自动埋点。

完整信息请参见 skywalking-python。

框架	Python 版本 - 库版本	插件名
aiohttp	Python >=3.7 - [‘3.7.*'];	sw_aiohttp

aioredis	Python >=3.7 - [‘2.0.*'];	sw_aioredis
aiormq	Python >=3.7 - [‘6.3’, ‘6.4’];	sw_aiormq
amqp	Python >=3.7 - [‘2.6.1’];	sw_amqp
asyncpg	Python >=3.7 - [‘0.25.0’];	sw_asyncpg
bottle	Python >=3.7 - [‘0.12.23’];	sw_bottle
celery	Python >=3.7 - [‘5.1’];	sw_celery
confluent_kafka	Python >=3.7 - [‘1.5.0’, ‘1.7.0’, ‘1.8.2’];	sw_confluent_kafka
django	Python >=3.7 - [‘3.2’];	sw_django
elasticsearch	Python >=3.7 - [‘7.13’, ‘7.14’, ‘7.15’];	sw_elasticsearch
hug	Python >=3.11 - NOT SUPPORTED YET; Python >=3.10 - [‘2.5’, ‘2.6’]; Python >=3.7 -	sw_falcon

	[‘2.4.1’，‘2.5’，‘2.6’];	
fastapi	Python >=3.7 - [‘0.89.’，‘0.88.'];	sw_fastapi
flask	Python >=3.7 - [‘2.0’];	sw_flask
happybase	Python >=3.7 - [‘1.2.0’];	sw_happybase
http_server	Python >=3.7 - ['*'];	sw_http_server
werkzeug	Python >=3.7 - [‘1.0.1’，‘2.0’];	sw_http_server
httpx	Python >=3.7 - [‘0.23.’，‘0.22.'];	sw_httpx
kafka-python	Python >=3.7 - [‘2.0’];	sw_kafka
loguru	Python >=3.7 - [‘0.6.0’，‘0.7.0’];	sw_loguru
mysqlclient	Python >=3.7 - [‘2.1.*'];	sw_mysqlclient
neo4j	Python >=3.7 - [‘5.*'];	sw_neo4j

psycopg[binary]	Python >=3.11 - [‘3.1.']; Python >=3.7 - [‘3.0.18’, ‘3.1.'];	sw_psycopg
psycopg2-binary	Python >=3.10 - NOT SUPPORTED YET; Python >=3.7 - [‘2.9’];	sw_psycopg2
pulsar-client	Python >=3.8 - [‘3.3.0’];	sw_pulsar
pymongo	Python >=3.7 - [‘3.11.*'];	sw_pymongo
pymysql	Python >=3.7 - [‘1.0’];	sw_pymysql
pyramid	Python >=3.7 - [‘1.10’, ‘2.0’];	sw_pyramid
pika	Python >=3.7 - [‘1.2’];	sw_rabbitmq
redis	Python >=3.7 - [‘3.5.*’, ‘4.5.1’];	sw_redis
requests	Python >=3.7 - [‘2.26’, ‘2.25’];	sw_requests

sanic	Python >=3.10 - NOT SUPPORTED YET; Python >=3.7 - [‘20.12’];	sw_sanic
tornado	Python >=3.7 - [‘6.0’, ‘6.1’];	sw_tornado
urllib3	Python >=3.7 - [‘1.26’, ‘1.25’];	sw_urllib3
urllib_request	Python >=3.7 - ['*'];	sw_urllib_request
websockets	Python >=3.7 - [‘10.3’, ‘10.4’];	sw_websockets

（3）Jaeger 支持的框架

目前 Jaeger 支持 Flask、Django 和 Grpc 等框架进行上报，更多请参见：

- jaeger-client-python
- OpenTracing API Contributions

1.6.4、Node.js 使用限制

（1）OpenTelemetry 支持的框架

OpenTelemetry 提供了若干自动埋点插件，支持为常见框架自动创建 Span，支持的框架列表如下，完整信息请参见 OpenTelemetry 官方文档。

框架	插件
amqplib	@opentelemetry/instrumentation-amqplib
aws-lambda	@opentelemetry/instrumentation-aws-lambda
aws-sdk	@opentelemetry/instrumentation-aws-sdk
bunyan	@opentelemetry/instrumentation-bunyan
cassandra-driver	@opentelemetry/instrumentation-cassandra-driver
connect	@opentelemetry/instrumentation-connect
cucumber	@opentelemetry/instrumentation-cucumber
dataloader	@opentelemetry/instrumentation-dataloader
dns	@opentelemetry/instrumentation-dns
express	@opentelemetry/instrumentation-express
fastify	@opentelemetry/instrumentation-fastify
generic-pool	@opentelemetry/instrumentation-generic-pool
graphql	@opentelemetry/instrumentation-graphql

grpc	@opentelemetry/instrumentation-grpc
hapi	@opentelemetry/instrumentation-hapi
http	@opentelemetry/instrumentation-http
ioredis	@opentelemetry/instrumentation-ioredis
knex	@opentelemetry/instrumentation-knex
koa	@opentelemetry/instrumentation-koa
lru-memoizer	@opentelemetry/instrumentation-lru-memoizer
memcached	@opentelemetry/instrumentation-memcached
mongodb	@opentelemetry/instrumentation-mongodb
mongoose	@opentelemetry/instrumentation-mongoose
mysql	@opentelemetry/instrumentation-mysql
mysql2	@opentelemetry/instrumentation-mysql2
nestjs-core	@opentelemetry/instrumentation-nestjs-core
net	@opentelemetry/instrumentation-net

pg	@opentelemetry/instrumentation-pg
pino	@opentelemetry/instrumentation-pino
redis	@opentelemetry/instrumentation-redis
restify	@opentelemetry/instrumentation-restify
socket.io	@opentelemetry/instrumentation-socket.io
winston	@opentelemetry/instrumentation-winston

(2) SkyWalking 支持的框架

skywalking-nodejs 是 SkyWalking 的 Node.js Agent 官方库，可以通过接入 skywalking-nodejs 实现对 Node.js 应用的监控，skywalking-nodejs 支持 MySQL、Redis、RabbitMQ 等多种第三方库的自动埋点。

完整信息请参见 skywalking-nodejs。

框架	插件名
built-in http and https module	http / https
Express	express
Axios	axios

MySQL	mysql
MySQL	mysql2
PostgreSQL	pg
pg-cursor	pg-cursor
MongoDB	mongodb
Mongoose	mongoose
RabbitMQ	amqplib
Redis	ioredis
AWS2DynamoDB	aws-sdk
AWS2Lambda	aws-sdk
AWS2SNS	aws-sdk
AWS2SQS	aws-sdk

(3) Jaeger 支持的框架

更多信息请参见

- jaeger-client-node
- OpenTracing API Contributions

1.7、隐私与敏感信息保护声明

应用性能监控会将采集到的指标信息在控制台页面进行展示，如果您有隐私敏感数据请注意加密保护，必要时不建议上传。

1.8、基本概念

实例

1 个应用可以部署在多台机器上，实例指的是被监控的应用所部署的机器。

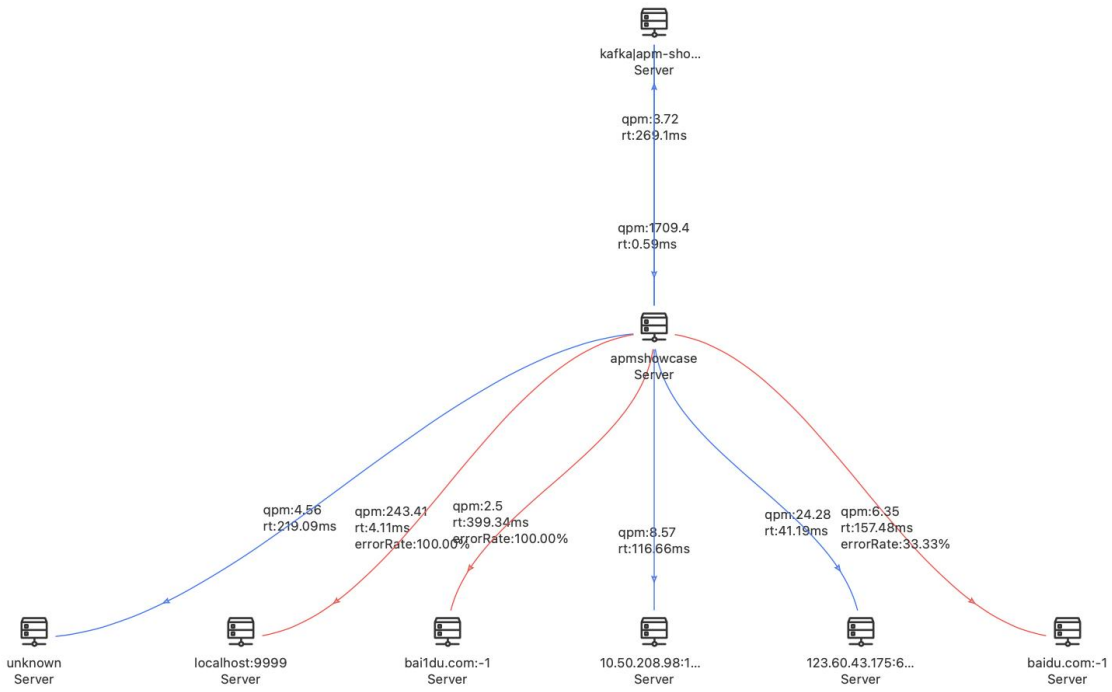
例如在下图中，service 是一个应用，下方的每一行都是该应用所部署的一台机器，即一个实例。

探针时（agent*hour）

1 个探针可以监控 1 个应用实例，1 探针时（agent*hour）表示一个探针使用了 1 小时。3600 探针时可支持 1 个探针使用 150 天，也可以是 5 个探针使用 30 天。

拓扑图

拓扑图是一种图形工具，可以用于描述和图形化各个应用之间的连接和依赖关系。如下图，圆圈代表一个服务，可通过 icon 和名称识别具体服务；虚线代表服务之间存在请求，显示每小时请求量、平均延时和请求总量。



二、产品计费

目前天翼云应用性能监控 APM 提供“按量付费”一种计费模式。

“按量付费”是一种后付费形式，按您每小时的实际消耗进行结算。根据您选择的不同应用接入方式，我们提供两种不同的计费方式。

2.1、计费方式

一：以 Java-agent 方式接入

计费公式

总费用 = Agent 数量 * 按需计费单价*时间

计费项

计费项	单位	价格
探针时	agent*hour	0.2 元

注：

- 一个探针使用一小时收费 0.2 元，一天需支付 4.8 元。调用链数据默认存储 1 个月，指标数据默认存储 1 个月。
- Agent 实例使用时长不足 1 小时不计费。

探针时（agent*hour）：1 个探针可以监控 1 个应用实例，如一个 Tomcat 实例或一个 Java 进程。1 探针时（agent*hour）表示一个探针使用了 1 小时。

二：以其他方式接入

计费公式

总费用 = 上报量总费用 + 链路存储时长总费用

上报量总费用 = 上报数量（百万） × 上报数量单价

链路存储时长总费用 = 上报数量（百万） × 存储时长（天） × 链路储存单价

计费项

计费项	单位	价格
Span 上报量	百万个	0.1 元
Span 存储量	百万个*天	0.06 元

注：

- 产生 1 百万条 span 上报量，收费 0.1 元；1 百万 Span 存储 1 天，收费 0.06 元，存储 1 小时，收费。调用链数据默认存储 1 个月，指标数据默认存储 1 个月。
- Span 上报量不足 1 百万不计费
- Span 存储量不足 1 百万不计费

Span：trace 通常指的是一个完整的调用序列，包括所有函数的调用和返回 Span 则更侧重于单个函数调用或其内部逻辑的一部分，可以被视为 trace 的一个子集，代表一个具体的逻辑运行单元。

开通方式

您可以在官网找到应用性能监控 APM 产品，点击「立即开通」。

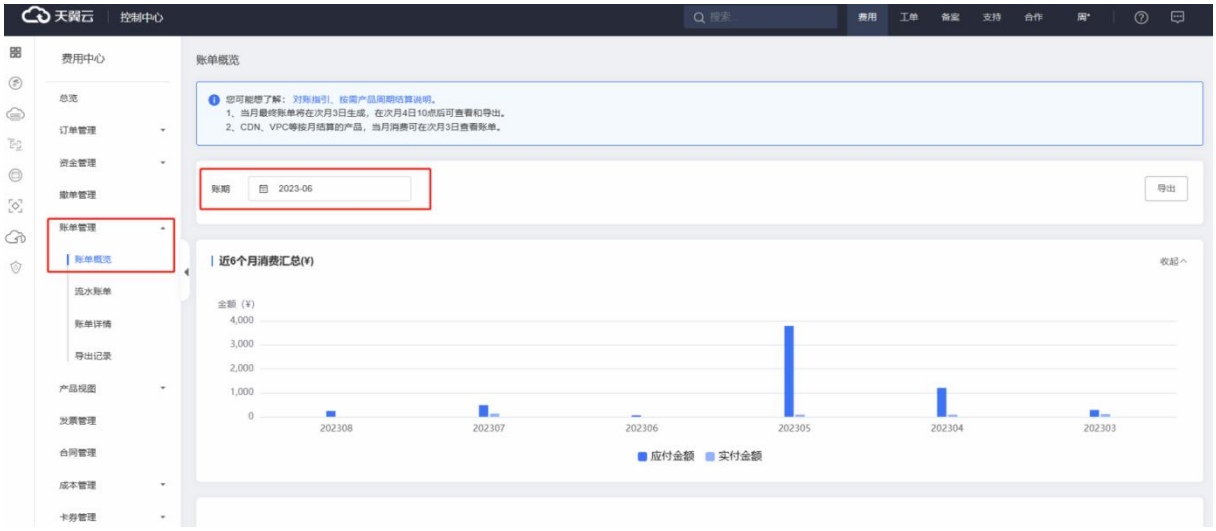
2.2、费用账单

如果您需要查看费用账单信息，可以在费用中心–账单管理进行查看。

查看账单概览

1、进入“费用中心＞账单管理＞账单概览”页面；

2、点击“账期”下拉框，设置想要查看的账期月份。

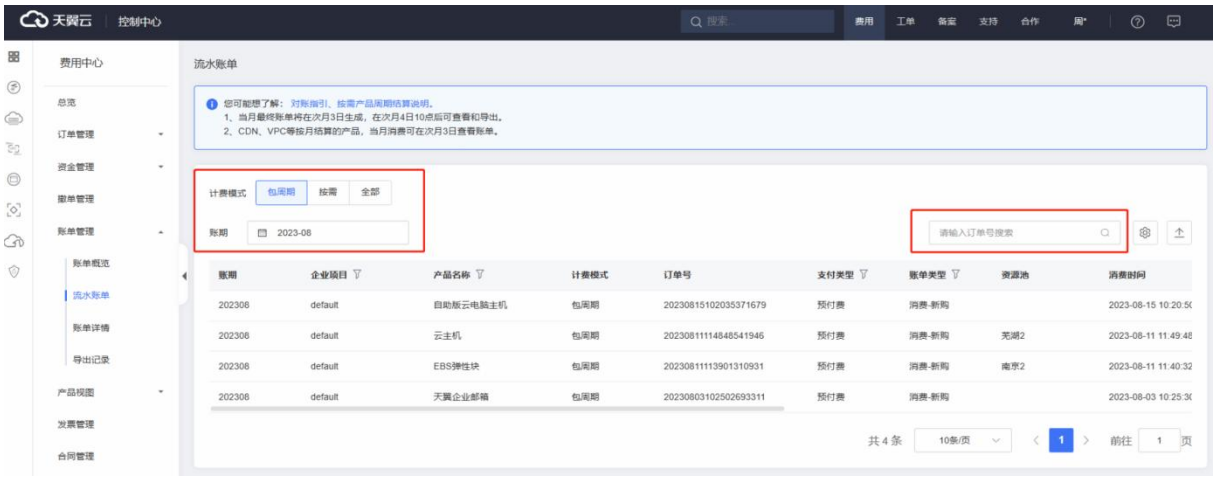


详情见账单概览。

查看账单流水

1、进入“费用中心＞账单管理＞流水账单”页面；

2、设置计费模式、账期、订单号等查询条件，查看流水账单数据；



详情见流水账单。

2.3、欠费说明

如果您的账户余额不足以支付当前账单，系统将判断为欠费，需要您在保留期内完成缴费。

欠费原因

按量计费会根据资源的结算周期进行结算，应用性能监控按量计费的结算周期是小时，在达到结算周期时，系统会生成账单，进行扣费。如果您的账户余额不足以支付当前账单，系统将判断为欠费。

欠费影响

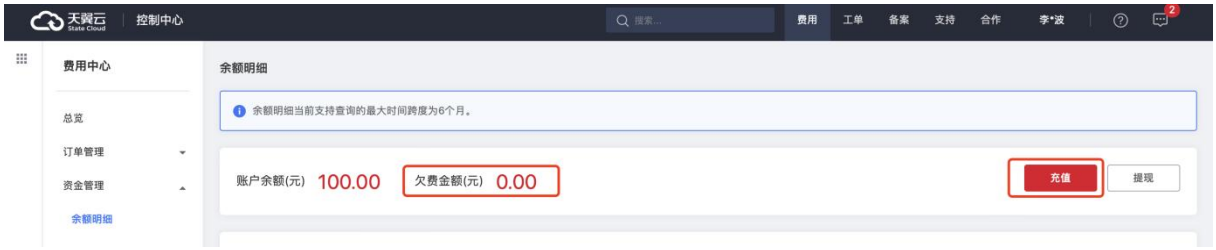
欠费后，您的资源将进入保留期（15 天），您将不能正常使用应用性能监控服务，无法继续上报新的链路、指标数据，但对于您已上报的数据会按照存储周期在保留期内继续存储。

- 若您在保留期内操作充值，充值后系统会自动扣减欠费金额，若扣减完后余额不低于 0，系统将恢复正常使用。
- 若保留期到期您未操作充值或是充值扣减后仍存在欠费，所使用资源将被关停并收回资源。

处理方式

如已处于欠费状态，您可以点击右上角费用-充值前往费用中心。

再点击余额明细确认欠费额度并操作充值。



详情见账户充值，如您对于扣费有疑问可在余额明细和流水账单中查看具体流水。

2.4、停止计费

您如果不再使用 APM，请取消所有应用接入，停止数据上报，并删除存储在 APM 上的数据，此时将会停止计费。

计费项

应用性能监控在不同的应用接入方式下有不同的计费项，具体收费及停止计费详情见下表。

应用接入方式	计费项	说明
以 Java-agent 方式接入	探针时（agent*hour）	只要 agent 生效中就会收费。如想暂停收费，可以在应用设置-agent 开关设置中关闭 agent 总开关。 如您不再使用该应用，可取消应用接入，并删除存储在 APM 上的数据。
以其他方式接入	Span 上报量（百万个）	只要产生 span 上报就会产生费用。

		如想停止收费，请取消应用接入，停止数据上报。
	Span 存储量（百万个*天）	只要有 span 在存储就会收费。 如想停止收费，请删除存储在 apm 上的 span 数据，并停止数据上报。

三、快速入门

3.1、如何开通 APM

本小节讲述如何开通 APM。

前提条件

您已注册天翼云帐号并完成实名认证。

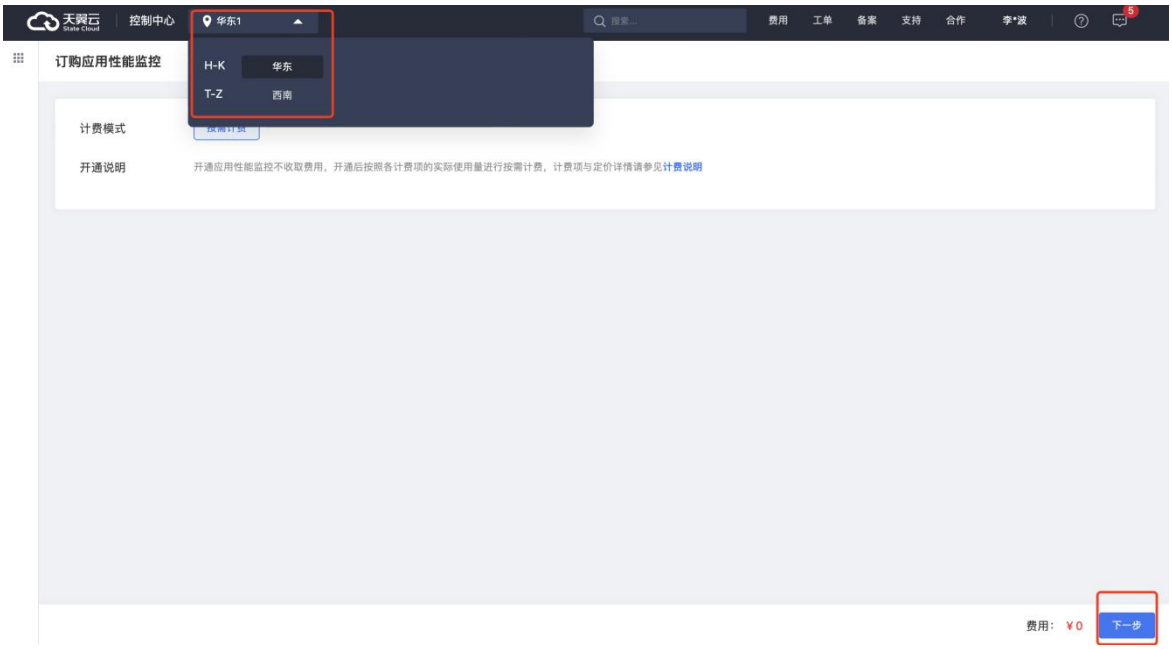
如您未注册，可参考账号中心>操作指南>注册天翼云账号完成注册。

如您未认证，可参考帐号中心>操作指南>实名认证完成认证。

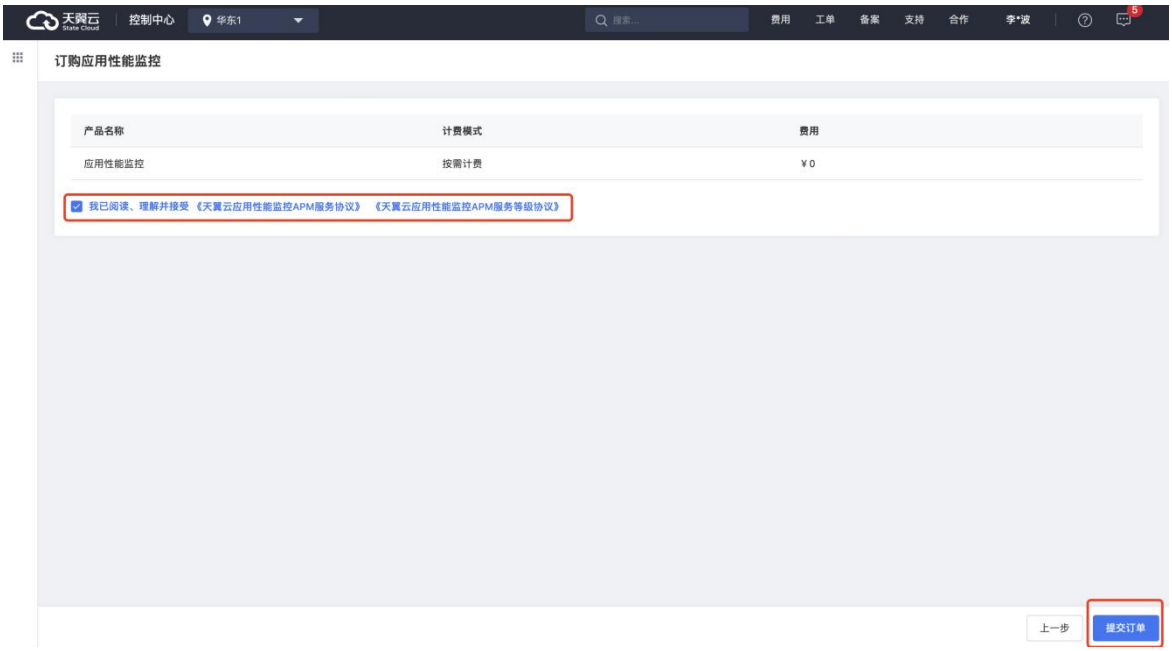
开通 APM

1、 点击应用性能监控产品详情页的立即开通按钮。

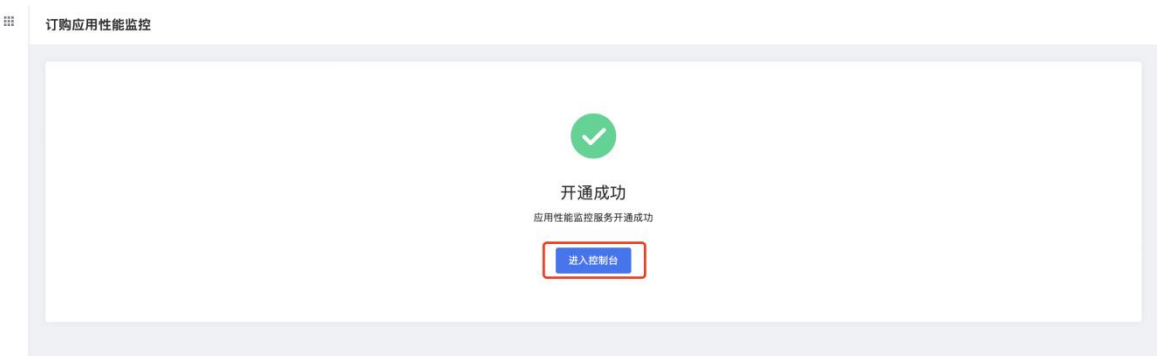
2、 在已登陆的前提下，选择你想开通的资源池，并点击下一步。



3、 阅读确认并勾选协议，点击下一步。



4、 稍等片刻，开通成功后，点击进入控制台即可开始使用。



3.2、接入 JAVA 应用

为 Java 应用手动安装 Agent

为 Java 应用安装 Agent 后，APM 即可开始监控 Java 应用，您可以查看应用拓扑、调用链路、异常事务、慢事务和 SQL 分析等一系列监控数据。您可以选择以手动方式或脚本方式安装 Agent，本小节介绍如何为 Java 应用手动安装 Agent。

下载 Agent

各个资源池的下载地址都不一样，在**接入应用**面板的 **STEP1** 区域下载对应的 Agent。

安装 Agent

1．进入 Agent 压缩包所在目录并将其解压至任意工作目录下。

```
unzip ctyunArmsAgent.zip -d /{user.workspace}/
```

“{user.workspace}”是示例路径，此处及以下均请根据具体环境替换为正确的路径 。

2.复制以下 JVM 参数，添加到应用服务的启动脚本中。

```
-javaagent:/{user.workspace}/ctyunArmsAgent/ctyunArmsAgent.jar

-Dotel.resource.attributes=service.name=xxx

-Darms.licenseKey='Jnyb84Wi@857657ULhP2G6B'

-Dotel.exporter.otlp.endpoint=http://100.64.2.29:27149
```

license.key 和 endpoint 是我们为您自动生成的。

service.name 是您的应用名称，请将 my-service 替换成您自己的应用名。

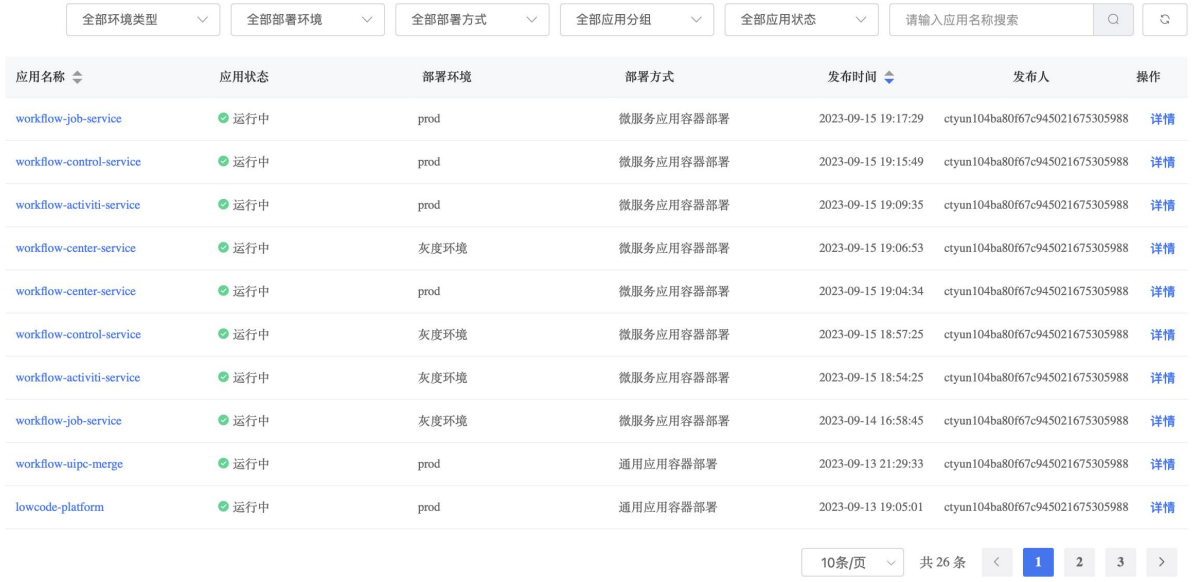
如需在同一台服务器为一个应用部署多个进程实例，则应在-Dotel.resource.attributes 内容中增加配置：instance.id=逻辑实例名，使用逗号与其他配置隔开，如若无此配置，则多个进程实例数据将合并为同一个实例。

启动应用

请在重新启动您的应用，大约 5 分钟后 Agent 将会安装完毕。

结果验证

约 5 分钟后，若 Java 应用出现在**应用列表**页面中且有数据上报，则说明接入成功。



为部署在容器环境的 Java 应用安装探针

借助应用性能监控 APM，您可以对容器环境的应用进行应用拓扑、接口调用、异常事务和慢事务监控、SQL 分析等监控。本小节将帮助您将容器环境中的应用接入应用性能监控 APM。

前提

1、该账号有可用的 ccse 容器集群。

2、准备好 java 应用的镜像包。

安装 cubems

安装应用性能监控 APM 插件 cubems

1.登录天翼云 CCSE 控制中心

2.在左侧菜单点击**集群**，在右侧页面进入集群详情。

3.在左侧导航栏选择**插件 > 插件市场**，在右侧页面找到 cubems 插件进行安装。

开启 APM

如需在创建新应用的同时开启应用性能监控 APM，请按以下步骤操作：

1.在天翼云 CCSE 控制中心左侧导航栏选择**集群**，在右侧页面进入集群详情。

2.在左侧导航栏选择**工作负载 > 有状态/无状态**，点击**新增**。

3.在新增页面，填写 Deployment 相关信息，点击**>“显示 高级设置” > Pod 标签**，新增 Pod 标签，设置标签名 **armsCubeMsAutoEnable**，标签值 **on**，点击确定。

添加工作负载后，可以在 yaml 文件中查看对应的标签

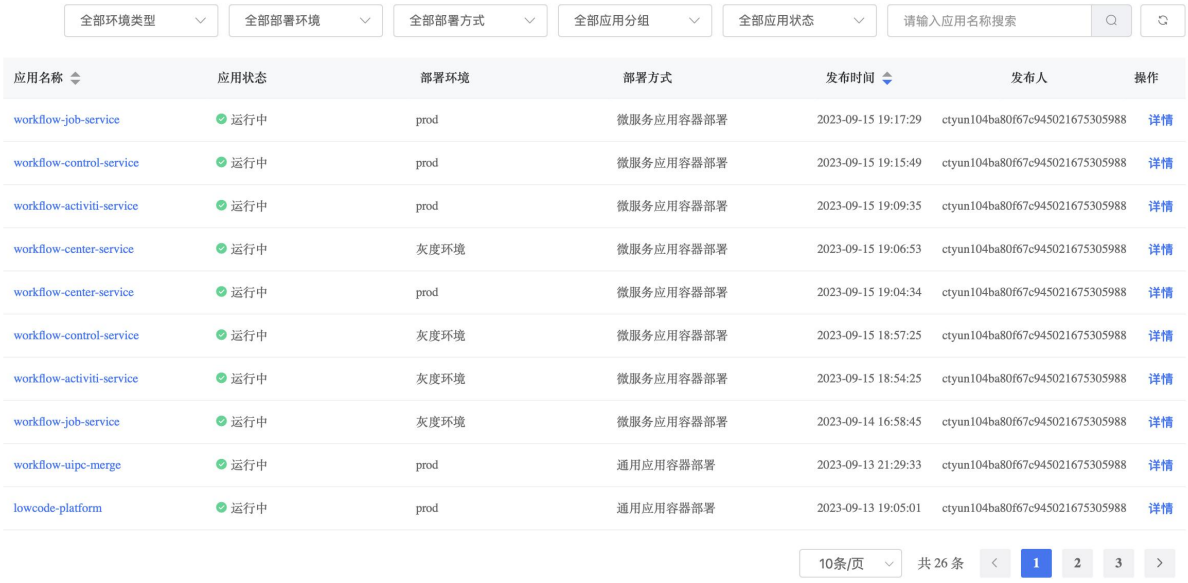
```
labels:

armsCubeMsAutoEnable: "on" //接入 AMS 的标签
```

如下图：

```yaml
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  labels:
    name: "5954aaf6-46cc-4220-9fa0-f9fa36c93571"
  name: "test-test-9f8j6b-3"
  namespace: "test1102"
spec:
  replicas: 1
  selector:
    matchLabels:
      name: "5954aaf6-46cc-4220-9fa0-f9fa36c93571"
  template:
    metadata:
      labels:
        name: "5954aaf6-46cc-4220-9fa0-f9fa36c93571"
        source: "CCSE"
        armsCubeMsAutoEnable: "on"
```

结果验证

约 5 分钟后，若 Java 应用出现在**应用列表**页面中且有数据上报，则说明接入成功。



3.3、接入 GO 应用

通过 OpenTelemetry 上报 Go 应用数据

在监控 Go 应用之前，您需要通过客户端将应用数据上报至 APM 服务端。本小节介绍如何通过 OpenTelemetry Go SDK 上报 Go 应用数据。

前提条件

完成 vpce 接入。

背景信息

OpenTelemetry Go SDK 提供了 Go 语言的分布式链路追踪能力，您可以直接使用 OTLP gRPC 或者 HTTP 协议向 APM 服务端上报数据。

接入步骤

1. 添加 OpenTelemetry Go 依赖。

```
package main

import (
    "context"
    "flag"
    "go.opentelemetry.io/contrib/instrumentation/net/http/httptra
ce/otelhttptrace"
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhtt
p"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/baggage"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptraceh
ttp"
    "go.opentelemetry.io/otel/sdk/resource"
    sdktrace "go.opentelemetry.io/otel/sdk/trace"
    semconv "go.opentelemetry.io/otel/semconv/v1.17.0"
    "go.opentelemetry.io/otel/trace"
    "io"
    "log"
    "net/http"
```

```
    "net/http/httptrace"
    "time"
)
```

2. 查看接入点信息。

应用列表的接入指引会根据您所在资源池提供“通过 HTTP 上报数据”和“通过 gRPC 上报数据”的 ENDPOINT(天翼云 vpc 网络接入点)、鉴权 TOKEN 信息。

3. 初始化 OpenTelemetry Go SDK。

方式 1：使用 HTTP 协议上报数据。（注意需将<token>和<endpoint>替换成相应地域的接入点信息。）

```
func initProvider() func() {
  ctx := context.Background()
  auth := map[string]string{
      "x-ctg-authorization": "<token>", // 替换成 token 信息
  }
  traceClient := otlptracegrpc.NewClient(
      otlptracegrpc.WithInsecure(),
      otlptracegrpc.WithEndpoint("<endpoint>"),    //替换成 otel http 的上报地址。
      otlptracehttp.WithURLPath("/api/traces"),
      otlptracegrpc.WithHeaders(auth),
      otlptracegrpc.WithDialOption(grpc.WithBlock()))
  log.Println("start to connect to server")
```

```
traceExp, err := otlptrace.New(ctx, traceClient)
handleErr(err, "Failed to create the collector trace exporter")
res, err := resource.New(ctx,
    resource.WithFromEnv(),
    resource.WithProcess(),
    resource.WithTelemetrySDK(),
    resource.WithHost(),
    resource.WithAttributes(
          // 在可观测链路 OpenTelemetry 版后端显示的服务名称。
          semconv.ServiceNameKey.String("otel-go-client-demo"),
          semconv.HostNameKey.String(""),
    ),
)
handleErr(err, "failed to create resource")
bsp := sdktrace.NewBatchSpanProcessor(traceExp)
tracerProvider := sdktrace.NewTracerProvider(
    sdktrace.WithSampler(sdktrace.AlwaysSample()),
    sdktrace.WithResource(res),
    sdktrace.WithSpanProcessor(bsp),
)
otel.SetTracerProvider(tracerProvider)
return func() {
```

```
        cxt, cancel := context.WithTimeout(ctx, time.Second)
        defer cancel()
        if err := traceExp.Shutdown(cxt); err != nil {
              otel.Handle(err)
        }
  }
}
```

方式 2：使用 gRPC 协议上报数据。（注意需将<token>和<endpoint>替换成相应地域的接入点信息。）

```
func initProviderGrpc() func() {
  ctx := context.Background()
  auth := map[string]string{
        "x-ctg-authorization": "<token>", //接入流程里面获取 token 信息
  }
  traceClient := otlptracegrpc.NewClient(
        otlptracegrpc.WithInsecure(),
        otlptracegrpc.WithEndpoint("<endpoint>"),  //替换成 grpc 接入信息
        otlptracegrpc.WithHeaders(auth),
        otlptracegrpc.WithDialOption(grpc.WithBlock()))
  log.Println("start to connect to server")
  traceExp, err := otlptrace.New(ctx, traceClient)
  handleErr(err, "Failed to create the collector trace exporter")
```

```
    res, err := resource.New(ctx,
        resource.WithFromEnv(),
        resource.WithProcess(),
        resource.WithTelemetrySDK(),
        resource.WithHost(),
        resource.WithAttributes(
            // 在可观测链路 OpenTelemetry 版后端显示的服务名称。
            semconv.ServiceNameKey.String("otel-go-client-demo_prov
ider"),
            semconv.HostNameKey.String(""),
        ),
    )
    handleErr(err, "failed to create resource")
    bsp := sdktrace.NewBatchSpanProcessor(traceExp)
    tracerProvider := sdktrace.NewTracerProvider(
        sdktrace.WithSampler(sdktrace.AlwaysSample()),
        sdktrace.WithResource(res),
        sdktrace.WithSpanProcessor(bsp),
    )
    otel.SetTracerProvider(tracerProvider)
    return func() {
        cxt, cancel := context.WithTimeout(ctx, time.Second)
```

```
        defer cancel()
        if err := traceExp.Shutdown(cxt); err != nil {
            otel.Handle(err)
        }
    }
}
```

4. client 端上报例子。

```
func sendReq(url string, ctx context.Context) {
// 构造一个 trace client
    client := http.Client{
        Transport: otelhttp.NewTransport(
            http.DefaultTransport,
            otelhttp.WithClientTrace(func(ctx context.Context) *h
ttptrace.ClientTrace {
                return otelhttptrace.NewClientTrace(ctx)
            }),
        ),
    }

    tr := otel.Tracer("example/client")
    err := func(ctx context.Context) error {
        ctx, span := tr.Start(ctx, "say hello", trace.WithAttribute
```

```
s(semconv.PeerService("ExampleService")))
      defer span.End()
      ctx = httptrace.WithClientTrace(ctx, otelhttptrace.NewClien
tTrace(ctx))
      req, _ := http.NewRequestWithContext(ctx, "GET", url, nil)
      res, err := client.Do(req)
      if err != nil {
            panic(err)
      }
      _, err = io.ReadAll(res.Body)
      _ = res.Body.Close()
      return err
  }(ctx)

    if err != nil {
        log.Fatal(err)
    }
  }

  func doClient(url string, ctx context.Context) {
    // 不断请求数据
    for {
        sendReq(url, ctx)
```

```
        time.Sleep(3 * time.Second)
    }
  }

  func main() {
    shutdown := initProvider()
    defer shutdown()
    url := flag.String("server", "http://localhost:8070/hello", "
server url")
    flag.Parse()
    bag, _ := baggage.Parse("username=xxx")
    ctx := baggage.ContextWithBaggage(context.Background(), bag)
    doClient(*url, ctx)
}
```

5. 服务端上报例子。

```
func handler() {
  uk := attribute.Key("username") //加点属性

  helloHandler := func(w http.ResponseWriter, req *http.Request)
{
      ctx := req.Context()
    //继续调用下一个服务最终效果是 client-» hello-» hello1
      sendReq("http://localhost:8071/hello1", ctx)
```

```
  }

  helloHandler1 := func(w http.ResponseWriter, req *http.Request)
 {
      ctx := req.Context()
      span := trace.SpanFromContext(ctx)
      bag := baggage.FromContext(ctx)
      span.AddEvent("handling this...", trace.WithAttributes(uk.S
tring(bag.Member("username").Value())))
      _, _ = io.WriteString(w, "Hello, world!\n")
  }

  otelHandler := otelhttp.NewHandler(http.HandlerFunc(helloHandle
r), "Hello")
  otelHandler1 := otelhttp.NewHandler(http.HandlerFunc(helloHandl
er1), "Hello1")

  http.Handle("/hello", otelHandler)
  http.Handle("/hello1", otelHandler1)
}

  func main() {
    flag.Parse()
    shutdown := initProvider()
```

```
    defer shutdown()
    handler()
    err := http.ListenAndServe(":" + *serverPort, nil) //nolint:g
osec // Ignoring G114: Use of net/http serve function that has no
 support for setting timeouts.
    if err != nil {
      log.Fatal(err)
    }
}
```

6、通过以上步骤，最后就在 APM 控制台的**应用列表**页面选择目标应用，查看监控数据。具体参考示例。

通过 SkyWalking SDK 上报 Go 应用数据

在监控 Go 应用之前，您需要通过客户端将应用数据上报至 APM 服务端。本小节介绍如何通过 SkyWalking SDK 上报 Go 应用数据。

接入步骤

1. 添加 SkyWalking Go 依赖。

```
package main

import (
    "context"
    "flag"
```

```
    "fmt"
    "github.com/SkyAPM/go2sky"
    httpPlugin "github.com/SkyAPM/go2sky/plugins/http"
    "github.com/SkyAPM/go2sky/reporter"
    "io/ioutil"
    "log"
    "net/http"
    "time"
)
```

2. 查看接入点信息。

应用列表的接入指引会根据您所在资源池提供 v3 版本接入点(Skywalking 8.*)的 ENDPOINT(天翼云 vpc 网络接入点)、鉴权 TOKEN 信息。

3. 初始化 SkyWalking Go SDK。

使用 gRPC 协议上报数据。（注意需将<token>和<endpoint>替换成相应地域的接入点信息。）

```
serverPort := flag.String("port", "9891", "server port")
    token := flag.String("token", "<token>", "token") // 鉴权信息
    endpoint := flag.String("endpoint", "<endpoint>", "endpoint")
 // grpc 端口
    authMap := map[string]string{
        "x-ctg-authorization": *token,
    }
```

```
    flag.Parse()
    report, err := reporter.NewGRPCReporter(*endpoint, reporter.W
ithAuthHeader(authMap))
    //report, err := reporter.NewLogReporter()
    tracer, err := go2sky.NewTracer(service, go2sky.WithReporter(
report))
```

4. client 端上报例子。

```
func do(tracer *go2sky.Tracer, serverPort string, client *http.Cl
ient) {
  for {
      time.Sleep(5*time.Second)
      doClient(tracer, client, "client", "http://localhost:" + se
rverPort + "/helloserver")
  }
}

func doClient(tracer *go2sky.Tracer, client *http.Client, spanNam
e string, url string) {
  clientReq, err1 := http.NewRequest(http.MethodGet, url, nil)
  parentSpan, newCtx, _ := tracer.CreateLocalSpan(context.Backgro
und())
  parentSpan.SetOperationName(spanName)
```

```
  defer parentSpan.End()
  clientReq = clientReq.WithContext(newCtx)
  res, err1 := client.Do(clientReq)
  if err1 != nil {
      fmt.Println(err1, "send req error")
      return
  }
  defer res.Body.Close()
  body, err1 := ioutil.ReadAll(res.Body)
  fmt.Println(string(body))
}
```

5. 服务端上报例子。

```
route := http.NewServeMux()
    route.HandleFunc("/helloserver", func(writer http.ResponseWri
ter, request *http.Request) {
    upstreamURL := "http://localhost:9891/hello1"
        client, err := httpPlugin.NewClient(tracer)
        clientReq, err1 := http.NewRequest(http.MethodGet, upstre
amURL, nil)
        if err1 != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("unable to create http request error: %v \
```

```
n", err)
            return
        }
        subSpan, newCtx, _ := tracer.CreateLocalSpan(request.Cont
ext())
        subSpan.SetOperationName("server2")
        defer subSpan.End()
        clientReq = clientReq.WithContext(newCtx)
        res, err1 := client.Do(clientReq)
        if err1 != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("unable to do http request error: %v \n",
err)
            return
        }
        defer res.Body.Close()
        body, err1 := ioutil.ReadAll(res.Body)
        if err1 != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("read http response error: %v \n", err)
            return
        }
```

```
        writer.WriteHeader(res.StatusCode)
        _, _ = writer.Write(body)
    })

    sm, err := httpPlugin.NewServerMiddleware(tracer)
    if err != nil {
        log.Fatalf("create server middleware error %v \n", err)
    }
    err = http.ListenAndServe(":" + *serverPort, sm(route))
    if err != nil {
        log.Fatal(err)
    }
```

6. 通过以上步骤，最后就在 APM 控制台的应用列表页面选择目标应用，查看监控数据。

通过 SkyWalking SDK 上报 Go 应用数据

在监控 Go 应用之前，您需要通过客户端将应用数据上报至 APM 服务端。本小节介绍如何通过 SkyWalking SDK 上报 Go 应用数据。

接入步骤

1. 添加 SkyWalking Go 依赖。

```
package main

import (
```

```
    "context"
    "flag"
    "fmt"
    "github.com/SkyAPM/go2sky"
    httpPlugin "github.com/SkyAPM/go2sky/plugins/http"
    "github.com/SkyAPM/go2sky/reporter"
    "io/ioutil"
    "log"
    "net/http"
    "time"
)
```

2. 查看接入点信息。

应用列表的接入指引会根据您所在资源池提供 v3 版本接入点(Skywalking 8.*)的

ENDPOINT(天翼云 vpc 网络接入点)、鉴权 TOKEN 信息。

3. 初始化 SkyWalking Go SDK。

使用 gRPC 协议上报数据。(注意需将<token>和<endpoint>替换成相应地域的接入点信息。)

```
serverPort := flag.String("port", "9891", "server port")
    token := flag.String("token", "<token>", "token") // 鉴权信息
    endpoint := flag.String("endpoint", "<endpoint>", "endpoint")
 // grpc 端口
    authMap := map[string]string{
```

```
        "x-ctg-authorization": *token,
    }
    flag.Parse()
    report, err := reporter.NewGRPCReporter(*endpoint, reporter.W
ithAuthHeader(authMap))
    //report, err := reporter.NewLogReporter()
    tracer, err := go2sky.NewTracer(service, go2sky.WithReporter(
report))
```

4. client 端上报例子。

```
func do(tracer *go2sky.Tracer, serverPort string, client *http.Cl
ient) {
  for {
      time.Sleep(5*time.Second)
      doClient(tracer, client, "client", "http://localhost:" + se
rverPort + "/helloserver")
  }
}

func doClient(tracer *go2sky.Tracer, client *http.Client, spanNam
e string, url string) {
  clientReq, err1 := http.NewRequest(http.MethodGet, url, nil)
  parentSpan, newCtx, _ := tracer.CreateLocalSpan(context.Backgro
```

```
und())
  parentSpan.SetOperationName(spanName)
  defer parentSpan.End()
  clientReq = clientReq.WithContext(newCtx)
  res, err1 := client.Do(clientReq)
  if err1 != nil {
      fmt.Println(err1, "send req error")
      return
  }
  defer res.Body.Close()
  body, err1 := ioutil.ReadAll(res.Body)
  fmt.Println(string(body))
}
```

5. 服务端上报例子。

```
route := http.NewServeMux()
    route.HandleFunc("/helloserver", func(writer http.ResponseWri
ter, request *http.Request) {
    upstreamURL := "http://localhost:9891/hello1"
        client, err := httpPlugin.NewClient(tracer)
        clientReq, err1 := http.NewRequest(http.MethodGet, upstre
amURL, nil)
        if err1 != nil {
```

```
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("unable to create http request error: %v \
n", err)
            return
        }
        subSpan, newCtx, _ := tracer.CreateLocalSpan(request.Cont
ext())
        subSpan.SetOperationName("server2")
        defer subSpan.End()
        clientReq = clientReq.WithContext(newCtx)
        res, err1 := client.Do(clientReq)
        if err1 != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("unable to do http request error: %v \n",
err)
            return
        }
        defer res.Body.Close()
        body, err1 := ioutil.ReadAll(res.Body)
        if err1 != nil {
            writer.WriteHeader(http.StatusInternalServerError)
            log.Printf("read http response error: %v \n", err)
```

```
            return
        }
        writer.WriteHeader(res.StatusCode)
        _, _ = writer.Write(body)
})

sm, err := httpPlugin.NewServerMiddleware(tracer)
if err != nil {
    log.Fatalf("create server middleware error %v \n", err)
}
err = http.ListenAndServe(":" + *serverPort, sm(route))
if err != nil {
    log.Fatal(err)
}
```

6. 通过以上步骤，最后就在 APM 控制台的应用列表页面选择目标应用，查看监控数据。

3.4、接入 Python 应用

通过 OpenTelemetry 上报 Python 应用数据

在监控 Python 应用之前，您需要通过客户端将应用数据上报至 APM 服务端。本小节介绍如何通过 OpenTelemetry Python SDK 上报 Python 应用数据。

前提条件

- 完成 vpce 接入。

- python 版本不低于 3.8。

接入步骤

1. 下载所需包。

```
pip install opentelemetry-api

pip install opentelemetry-sdk

pip install opentelemetry-exporter-otlp

pip install flask

pip install requests
```

2. 查看接入点信息。

应用列表的接入指引会根据您所在资源池提供“通过 HTTP 上报数据”和“通过 gRPC 上报数据”的 ENDPOINT(天翼云 vpc 网络接入点)、鉴权 TOKEN 信息。

3. 创建服务端。

请将代码中的<token>和<endpoint>替换成前提条件中获取的接入点信息。

请根据实际情况替换代码中的<service-name>(服务名)和<host-name>(主机名)。

```
# server.py

from flask import Flask

from opentelemetry import trace, baggage

from opentelemetry.trace import SpanKind

from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import
 OTLPSpanExporter as OTLPSpanGrpcExporter
```

```
from opentelemetry.exporter.otlp.proto.http.trace_exporter import
 OTLPSpanExporter as OTLPSpanHttpExporter
from opentelemetry.sdk.resources import SERVICE_NAME, Resource, H
OST_NAME
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

app = Flask(__name__)

def init_opentelemetry():
  resource = Resource(attributes={
    SERVICE_NAME: "<service_name>",
    HOST_NAME: "<host_name>"
  })

  span_processor = BatchSpanProcessor(OTLPSpanGrpcExporter(
    endpoint="<endpoint>",
    headers=[("x-ctg-authorization", "<endpoint>")]
  ))

  trace_provider = TracerProvider(resource=resource, active_span_
processor=span_processor)
  trace.set_tracer_provider(trace_provider)

@app.route('/trace_demo')
```

```
def trace_demo():
  tracer = trace.get_tracer(__name__)
  with tracer.start_as_current_span("server_span", kind=SpanKind.
SERVER):
    return "Server traced this request."

@app.route('/baggage_demo')
def baggage_demo():
  tracer = trace.get_tracer(__name__)
  with tracer.start_as_current_span("server_span_baggage", kind=S
panKind.SERVER):
    baggage = baggage.set_baggage("key", "value_from_server")
    return f"Server traced this request with baggage: {baggage.ge
t_baggage('key', baggage)}"

if __name__ == '__main__':
  init_opentelemetry()
  app.run(port=5000, debug=True)
```

4. 创建客户端。

请将代码中的<token>和<endpoint>替换成前提条件中获取的接入点信息。

请根据实际情况替换代码中的<service-name>(服务名)和<host-name>(主机名)。

```
# client.py
```

```
import requests
from opentelemetry import trace
from opentelemetry.sdk.resources import SERVICE_NAME, Resource, H
OST_NAME
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import
 OTLPSpanExporter as OTLPSpanGrpcExporter

def init_opentelemetry():
  resource = Resource(attributes={
    SERVICE_NAME: "<service_name>",
    HOST_NAME: "<host_name>"
  })

  span_processor = BatchSpanProcessor(OTLPSpanGrpcExporter(
    endpoint="<endpoint>",
    headers=[("x-ctg-authorization", "<token>")]
  ))

  trace_provider = TracerProvider(resource=resource, active_span_
processor=span_processor)
  trace.set_tracer_provider(trace_provider)
```

```
def make_request(path):
  tracer = trace.get_tracer(__name__)
  with tracer.start_as_current_span("client_span"):
    response = requests.get(f"http://localhost:5000{path}")
    print(f"Response from server: {response.text}")

if __name__ == '__main__':
  init_opentelemetry()
  make_request('/trace_demo')
  make_request('/baggage_demo')
```

5. 运行项目。

```
python server.py
python client.py
```

6. 通过以上步骤，最后就在 APM 控制台的**应用列表**页面选择目标应用，查看监控数据。

3.5、接入 Node.js 应用

通过 OpenTelemetry 上报 Node.js 应用数据

在监控 Node.js 应用之前，您需要通过客户端将应用数据上报至 APM 服务端。本文介绍了如何通过 OpenTelemetry 将 Node.js Express 应用接入 APM。

前提条件

完成 vpce 接入。

接入步骤

1. 引入 opentelemetry 相关依赖。

```
npm install --save @opentelemetry/api

npm install --save @opentelemetry/auto-instrumentations-node

npm install --save @opentelemetry/sdk-node
```

2. 查看接入点信息。

应用列表的接入指引会根据您所在资源池提供“通过 HTTP 上报数据”和“通过 gRPC 上报数据”的 ENDPOINT(天翼云 vpc 网络接入点)、鉴权 TOKEN 信息。

3. 初始化 Node.js Provider。

请将<url>和<token>替换成相应地域的接入点信息。

```
const opentelemetry = require('@opentelemetry/sdk-node');

const { Resource } = require('@opentelemetry/resources');

const { SemanticResourceAttributes } = require('@opentelemetry/se

mantic-conventions');

const {

  getNodeAutoInstrumentations,

} = require('@opentelemetry/auto-instrumentations-node');

const {

  OTLPTraceExporter,

} = require('@opentelemetry/exporter-trace-otlp-proto');
```

```
const {
  OTLPMetricExporter,
} = require('@opentelemetry/exporter-metrics-otlp-proto');
const { PeriodicExportingMetricReader } = require('@opentelemetry
/sdk-metrics');
const { diag, DiagConsoleLogger, DiagLogLevel } = require('@opent
elemetry/api');

// 打印日志
diag.setLogger(new DiagConsoleLogger(), DiagLogLevel.INFO);

const resource = new Resource({
  [SemanticResourceAttributes.SERVICE_NAME]: 'nodejs-demo', // 在
这里设置您的服务名称
});

const sdk = new opentelemetry.NodeSDK({
  resource: resource, // 添加 resource 配置
  traceExporter: new OTLPTraceExporter({
    url: '<url>', // 通过 otel http 方式上报的地址
    headers: {'x-ctg-authorization': '<token>'},//上鉴权 token
  }),
  instrumentations: [getNodeAutoInstrumentations()],
});
```

```
sdk.start();
```

4. client 上报 demo。

```
// 引入 OpenTelemetry SDK 和自动 instrumentation
require('./tracing'); // 确保在代码最顶部引入

const fetch = require('node-fetch');

async function makeRequest() {
  const response = await fetch('http://localhost:3001/test');
  const data = await response.text();
  console.log(data);
}

makeRequest();
```

5. 服务端上报 demo。

```
// 引入 OpenTelemetry SDK 和自动 instrumentation
require('./tracing'); // 确保在代码最顶部引入

const express = require('express');
const app = express();
const port = 3001;

app.get('/test', (req, res) => {
```

```
  res.send('Hello from the server!');
});

app.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});
```

6. 通过上面步骤就可以在控制台查看 node.js 产生的监控数据了。更详细的接入 demo 请参考示例。

四、用户指南

4.1、应用列表

4.1.1、应用列表

APM 的应用列表展示当前租户当前资源池下已接入（有过数据上报）的所有应用。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台

2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。

功能说明

接入应用

当前提供四种语言应用接入，详情见应用接入。

应用列表展示

APM 的应用列表展示当前租户当前资源池下已接入（有过数据上报）的所有应用。

- 显示基础的应用名称、语言信息。
- 显示每秒请求数、错误率、平均响应时间、响应时间趋势图等关键指标信息。
- 支持根据应用名称搜索。
- 支持筛选时间段。
- 支持操作删除。（注：如果操作删除后应用仍然进行数据上报，在获取到上报数据时，列表页将会显示该应用。）
- 支持操作查看详情，点击应用名称，进入对应的应用详情页，详见应用监控。

4.1.2、监控概览

以应用为维度，统计整个应用的关键指标，帮助您快速掌握应用的整体状况。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用总览**」-「**概览**」页签查看相应信息。

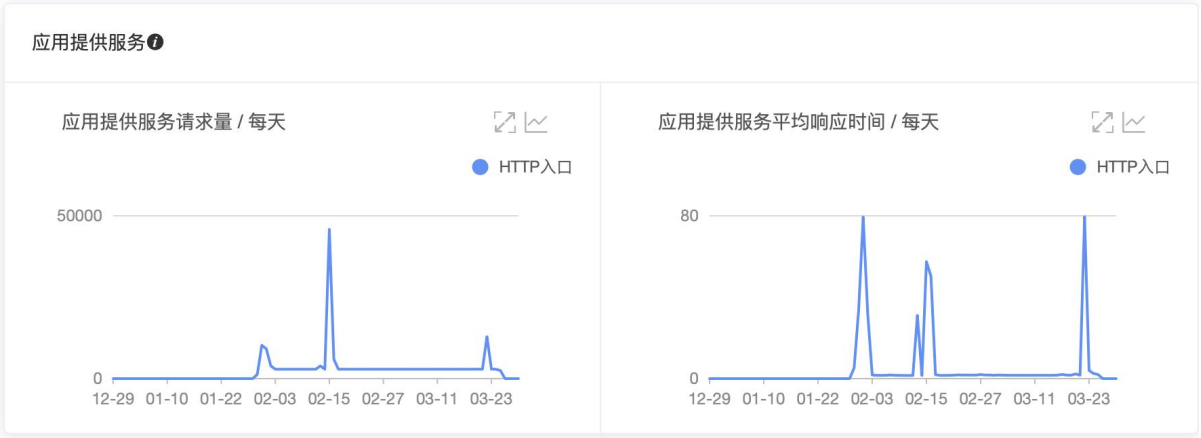
功能说明

总览指标

- **总请求量**：筛选时间段内，应用提供服务请求量+应用依赖服务请求量。
- **平均响应时间**：筛选时间段内，（所有应用提供服务响应时间+所有应用依赖服务响应时间）/总请求量。
- **错误数**：error，筛选时间段内，请求出错的数量，通常指 http 状态码为 4xx、5xx 的请求。
- **FullGC**：筛选时间段内，整堆垃圾回收的次数，回收的区域包括年轻代、老年代以及方法区。
- **慢 SQL**:筛选时间段内,执行时间大于等于慢 SQL 阈值的 SQL 数量,默认 500ms，您可根据实际情况在「应用设置」中修改。
- **异常**：exception，筛选时间段内，该应用报的异常数。

- **慢调用**：筛选时间段内，响应时间大于等于慢调用阈值的调用数量，默认 500ms，您可根据实际情况在「应用设置」中修改响应时间阈值。

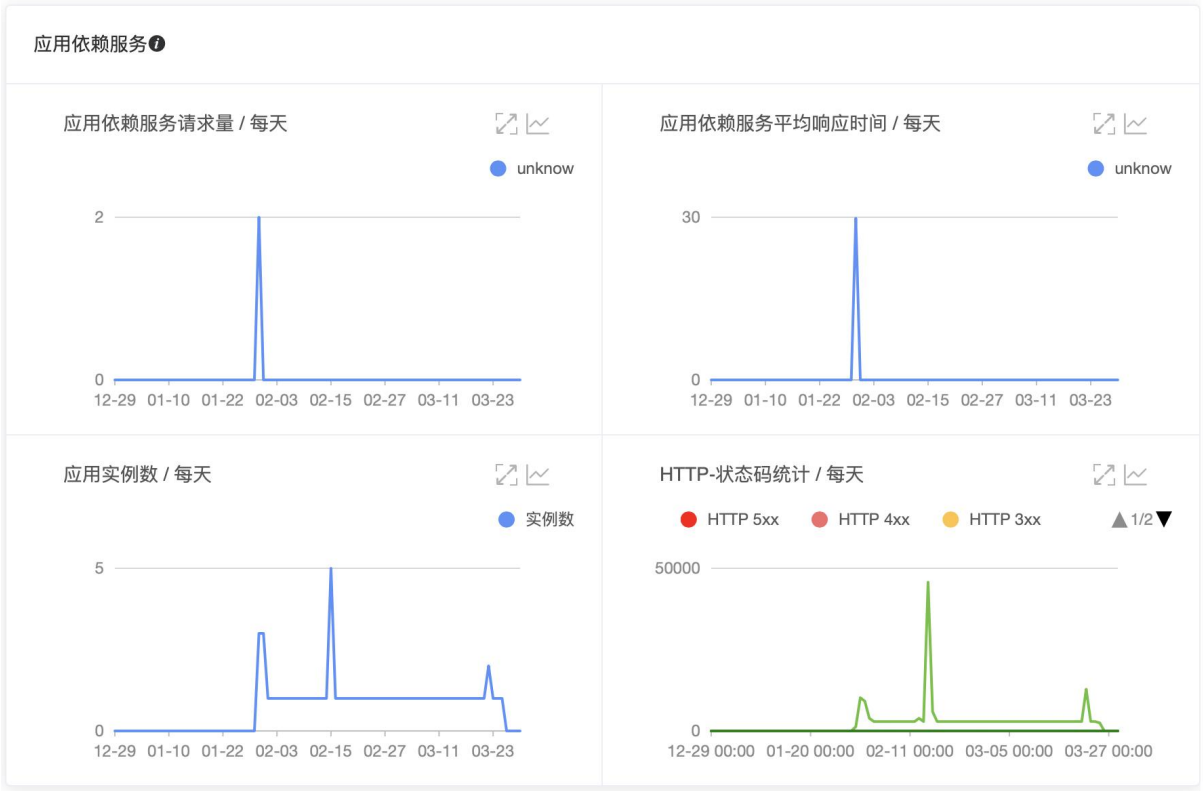
应用提供服务



因用户访问该应用而产生的数据，例如用户在浏览器中访问该应用

- **应用提供服务请求量**：筛选时间段内，用户向该应用发起的请求数量
- **应用提供服务平均响应时间**：响应时间是指从用户发起请求到服务端给予反馈的时长，平均响应时间是筛选时间段内，所有请求的响应时间的平均值。

应用依赖服务



因该应用访问其他服务而产生的数据，例如该应用访问数据库

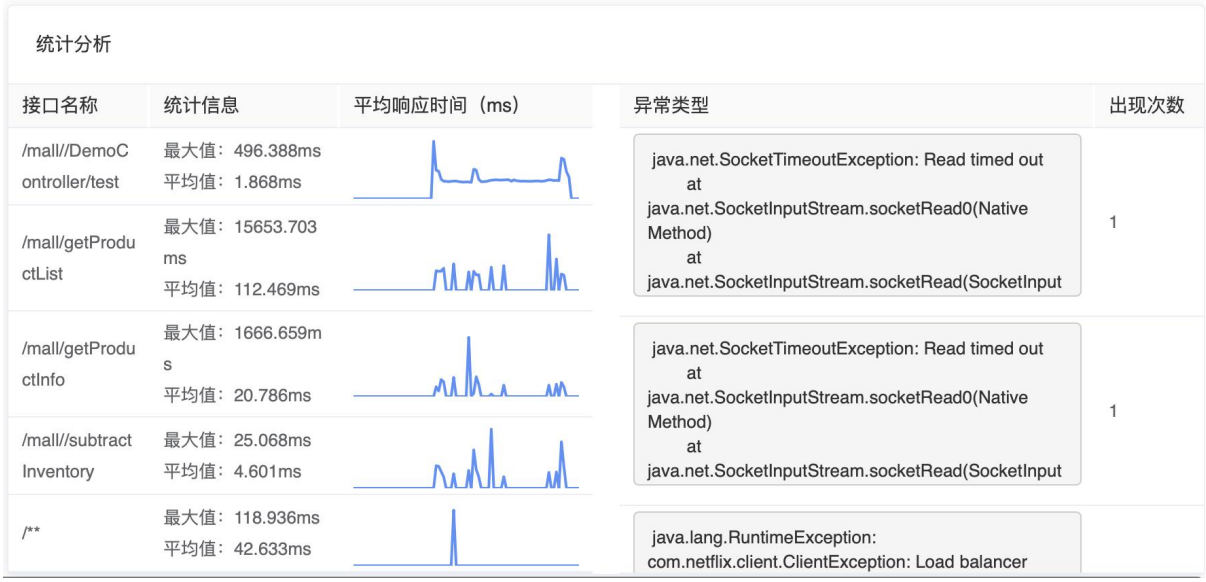
- **应用依赖服务请求量**：筛选时间段内，该应用向其他服务发起的请求数量
- **应用依赖服务平均响应时间**：响应时间是指从该应用发起请求到其他服务给予反馈的时长，平均响应时间是筛选时间段内，所有请求的响应时间的平均值。
- **应用实例数**：筛选时间段内，有调用行为的应用实例数量。
- **HTTP-状态码统计**
  - 5xx：服务器异常，服务器在处理请求的过程中发生错误
  - 4xx：客户端异常，请求包含语法错误或无法完成请求
  - 3xx：重定向问题，需要进一步操作
  - 2xx：成功，服务器成功接收请求并执行
  - 200：请求成功

慢调用

该应用访问其他服务时，其他服务响应时间大于等于 500ms（默认 500ms，可在应用设置中修改阈值）的调用，定义为慢调用。显示饼图和详情表，表头显示如下

- **时间**：判定为慢调用的时间点
- **服务名**：被调用的服务名称
- IP：被调用的服务的 IP 地址
- **耗时（ms）**：具体响应时间
- **响应码**：200 表示请求成功，03 表示调用时长超过最大监听时长 15 秒
- TraceID：Trace 表示一个完整的请求链路，一个 Trace 包含了多个调用过程 span，TraceID 是该请求链路的唯一标识。

统计分析



以接口维度来统计调用的情况

- **接口名称**：被调用的接口的名称
- **最大值**：筛选时间段内，该接口被调用的响应时间的最大值
- **平均值**：筛选时间段内，该接口被调用的平均响应时间
- **平均响应时间**：筛选时间段内，每天的平均响应时间的趋势图
- 异常情况
- **异常类型**：显示异常明细，与点击详情按钮看到的内容一致
- **出现次数**：筛选时间段内，此类异常出现的次数

4.1.3、应用拓扑

以应用为维度，进行可视化拓扑展示。

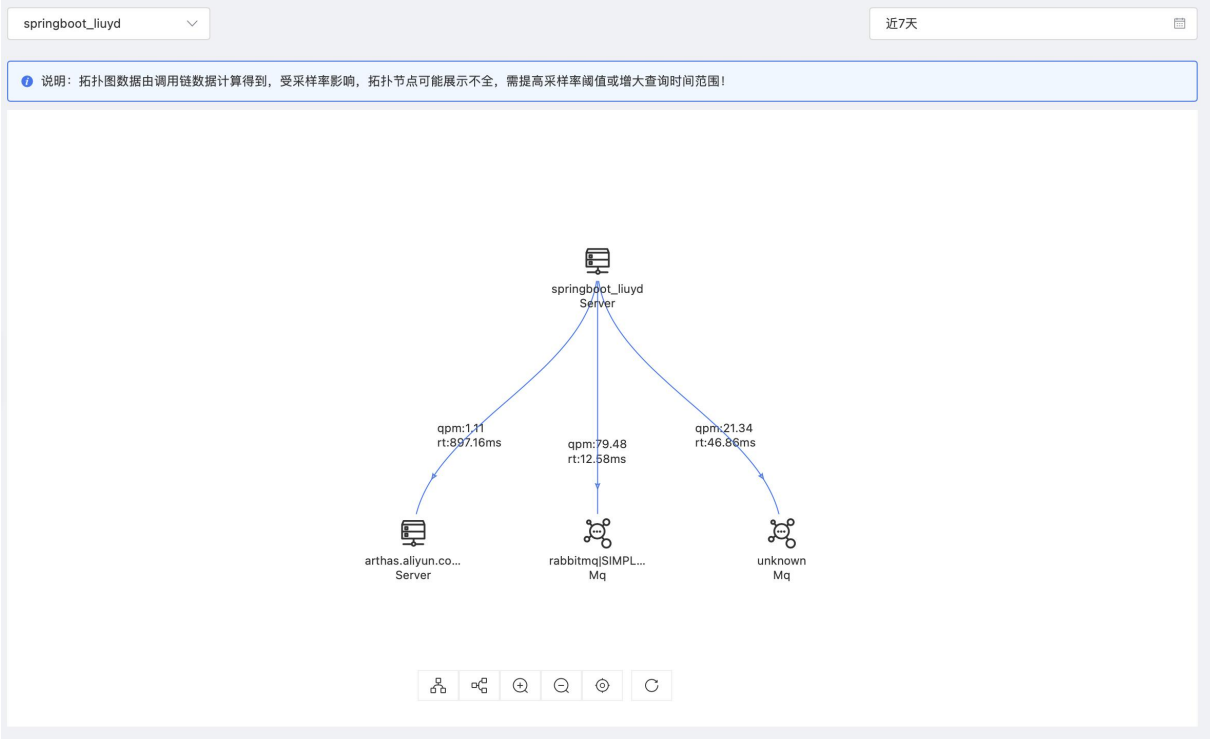
功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。

3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用拓扑**」查看相应信息。

功能说明



拓扑图是一种以图形化方式展示应用之间关系的图表，帮助开发人员或运维人员了解应用程序的整体结构和运行状况。

信息

拓扑图通常包括以下信息：

- **应用或服务的组成部分**：例如数据库、缓存、消息队列、Web 服务器等，当前版本支持的插件见下表，暂未支持的插件会显示为 unknown。

类别	支持的服务
其他基础服务	JavaMethod
	Netty
数据库	Mysql

	ClickHouse
	EsRestClient
	MongoDb
缓存	Redis
	Jedis
	Lettuce
消息	KafkaConsumer
	KafkaProducer
	RabbitMqConsumer
	RabbitMqProducer
Web 容器	Tomcat
外部调用	HttpClient

- **组件之间的依赖关系**：例如一个组件调用另一个组件的接口等，包含各项基础指标

指标名	说明
客户端	显示当前链路的客户端名称。
服务端	显示当前链路的服务端名称。
请求总量	显示当前链路在筛选时间段内的请求次数。
吞吐量	显示当前链路在筛选时间段内平均每分钟的请求量。

平均响应时间	显示当前链路在筛选时间段内所有请求的平均响应时间。
错误数	显示当前链路在筛选时间段内请求错误次数。
错误率	显示当前链路在筛选时间段内请求错误率，即：错误数/请求总量。

通过分析拓扑图，开发/运维人员可以快速定位应用中的问题，并进行及时的排查和修复。拓扑图还可以帮助开发/运维人员进行容量规划和性能优化，以提高应用程序或服务的性能和可靠性。

操作

操作	说明
纵向图	点击纵向图操作按钮，修改拓扑图布局为纵向树状图。
横向图	点击横向图操作按钮，修改拓扑图布局为横向树状图。
放大	点击放大按钮或者鼠标滚轮放大，放大拓扑图。
缩小	点击缩小按钮或者鼠标滚轮缩小，缩小拓扑图。
复位	点击复位按钮，重新复位拓扑图，将之前的放大缩小拖拉拽各种操作复原。
刷新	点击刷新按钮，刷新拓扑图数据。
拖拉拽	按住元素可以进行拖拉拽。

4.1.4、监控详情

4.1.4.1、监控项视图

APM 的应用详情包括各类应用性能指标集的各种可视化图表展示。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。

2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。

3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用详情**」，您可以在应用详情页面切换至不同页签，查看该应用实例相应的指标信息。

指标集

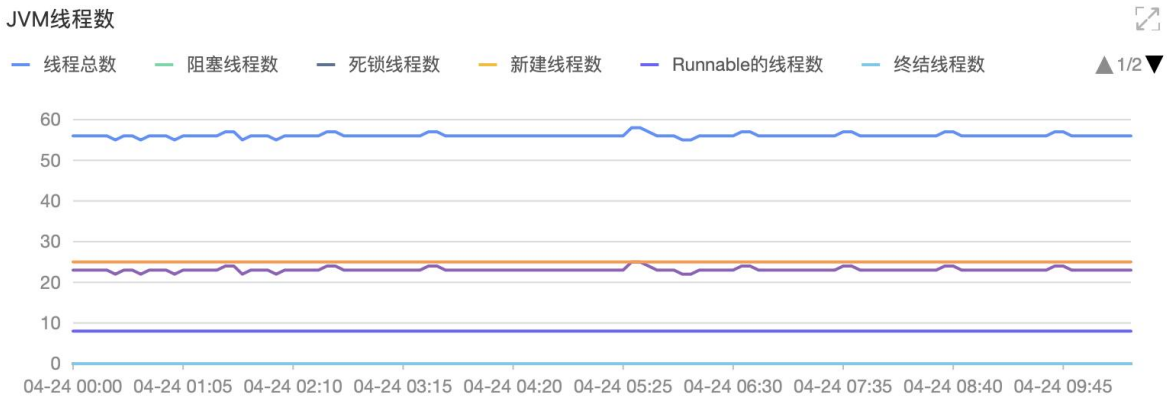
类型	内容
JVM 监控	内存、GC、线程
其他基础监控	资源监控、Netty 内存、Java 方法
数据库监控	Mysql、es、MongDB、ClickHouse、Druid 连接池、C3P0 连接池、DBCP 连接池
缓存监控	Redis、Jedis、Lettuce
消息监控	Kafka、RabbitMQ
Web 容器	Tomcat
异常错误分析	异常分析、错误分析
上下游分析	上游应用、下游应用

显示类型

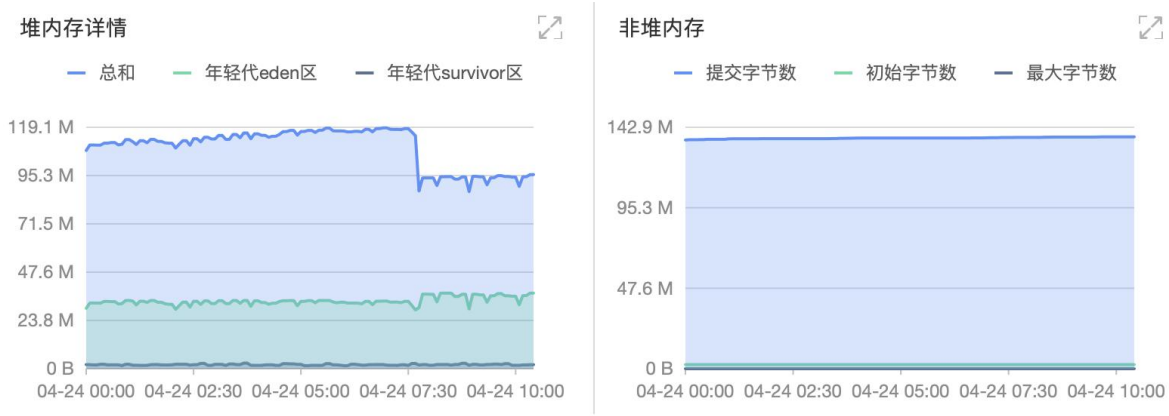
图

通常用于显示趋势图，表示某些指标在筛选时间段内的趋势变化。

（1）单图



（2）双图



- 如果某个周期内未采集到数据，将会进行补 0 显示，效果呈现为横坐标为 0 的横线（或断点）。
- 支持放大操作。

表

通常用于显示一段时间内某类指标基于某个纬度的汇总统计/明细。

产生时间	接口名称	所属应用	耗时(ms)	状态	TraceId
2024-04-24 10:14:53.773	/hello	apm-ccse	0.485		22887a348028a7f...
2024-04-24 10:10:11.187	/hello	apm-ccse	0.484		227a164f643b1102...
2024-04-24 10:05:59.753	/hello	apm-ccse	0.962		63d2f9de7e62e5c...
2024-04-24 10:05:51.709	/hello	apm-ccse	0.549		fbcbae112fe7c6b6...
2024-04-24 10:02:54.712	/hello	apm-ccse	0.532		49e519583babbb...
2024-04-24 10:01:55.359	/hello	apm-ccse	0.493		b6b6310e9427df4...
2024-04-24 10:00:44.960	/hello	apm-ccse	0.502		7ab21dc8b267cb0...
2024-04-24 09:57:39.898	/hello	apm-ccse	0.516		7c3b84bdcd1d334...
2024-04-24 09:56:06.367	/hello	apm-ccse	0.546		ce3d8e281ea553b...
2024-04-24 09:55:59.328	/hello	apm-ccse	0.523		616aaf420f2eef3d...

10条/页 共 611 条 1 2 3 4 5 6 ... 62

- 对于数字列支持排序。
- 对于首列/首两列关键字段支持搜索。
- 蓝色字体表示可以点击。

4.1.4.2、概览

应用的概览可以提供核心监控数据，帮助您快速掌握某个应用的具体情况。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**接口调用**」、「**外部调用**」等其他维度视角的分析菜单，您可以在应用详情页面切换至「**概览**」页签，在左侧关键指标中选择不同的应用实例/接口，可查看该应用实例/接口相应的概览信息。

功能说明

（1）展示拓扑图

拓扑图是一种以图形化方式展示应用之间关系的图表，帮助开发人员或运维人员了解应用程序的整体结构和运行状况。详情参见“应用总览-2.1 、拓扑图-（1）拓扑图”

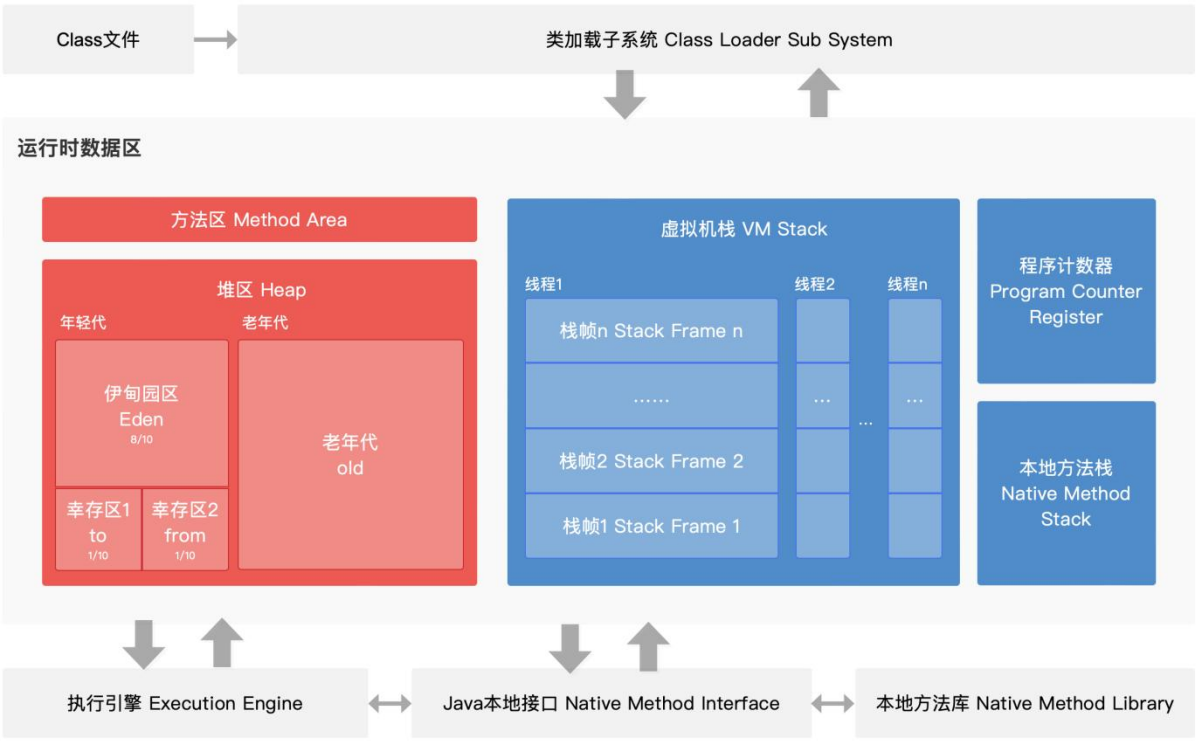
（2）展示请求数、响应时间、错误数、HTTP-状态码统计

- **请求数**：该应用实例/接口在筛选时间段内的 HTTP 入口的总请求数
- **响应时间**：该应用实例/接口在筛选时间段内的 HTTP 入口请求的日均响应时间
- **错误数**：该应用实例/接口在筛选时间段内的 HTTP 入口请求的错误数
- **HTTP-状态码统计**：该应用实例/接口在筛选时间段内的 HTTP 入口请求的状态码
  - 5xx：服务器异常，服务器在处理请求的过程中发生错误
  - 4xx：客户端异常，请求包含语法错误或无法完成请求
  - 3xx：重定向问题，需要进一步操作
  - 2xx：成功，服务器成功接收请求并执行
  - 200：请求成功

4.1.4.3、JVM 监控

Java 虚拟机（Java Virtual Machine，JVM）是一种可以执行 Java 字节码的虚拟机，它是 Java 平台的核心组成部分之一。JVM 负责将 Java 字节码翻译成特定平台上的机器指令，使得 Java 程序可以在各种不同的平台上运行。

JVM 监控即 Java 虚拟机监控，用于监控重要的 JVM 指标。



- JVM 内存包括堆（heap）内存、非堆（non-heap）内存。堆内存用于存储对象实例，而非堆内存用于存储 Java 虚拟机自身使用的数据和代码等。
- 非堆内存包括直接内存中的元空间（sdk1.8 用的元空间，sdk1.7 用的是线程共享的方法区）、线程私有的虚拟机栈、程序计数器、本地方法栈。
- 堆的释放受到 GC 垃圾回收器的管理，GC 会去找那些很久没有引用地址指向的内存块，把它们清理掉。

JVM 相关数据采集间隔为 15s 一次，采集的是瞬时值中的最大值，总数由多个数据加和而成。以堆内存图表为例，1 分钟内我们分别采集了老年代、年轻代各 4 次，取其中的最大值展示，总和为最大值之和（因此总和可能大于用户配置的 jvm 内存大小）。

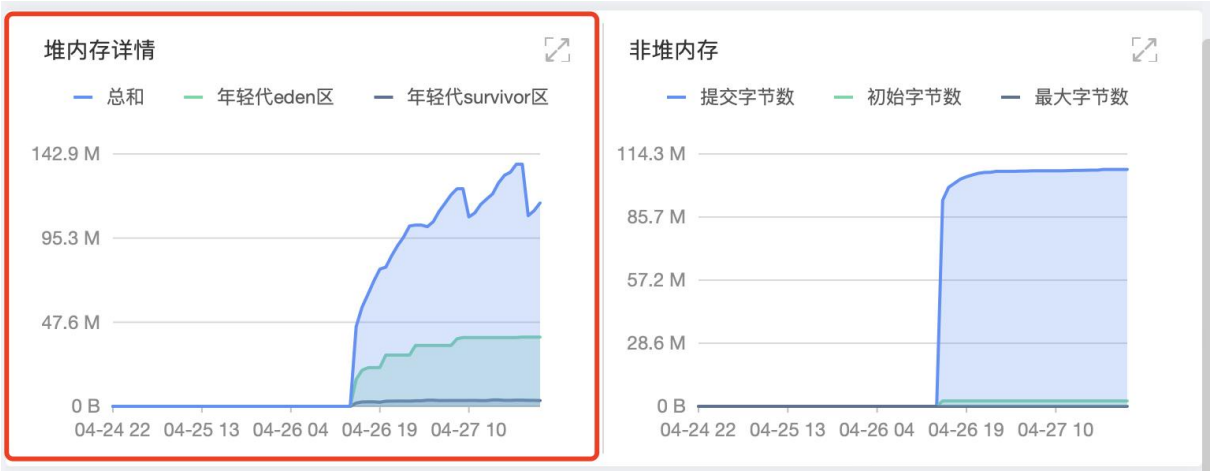
功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用详情**」，您可以在应用详情页面切换至「**JVM 监控**」页签，在左侧关键指标中选择不同的应用实例，可查看该应用实例相应的概览信息。

功能说明

内存

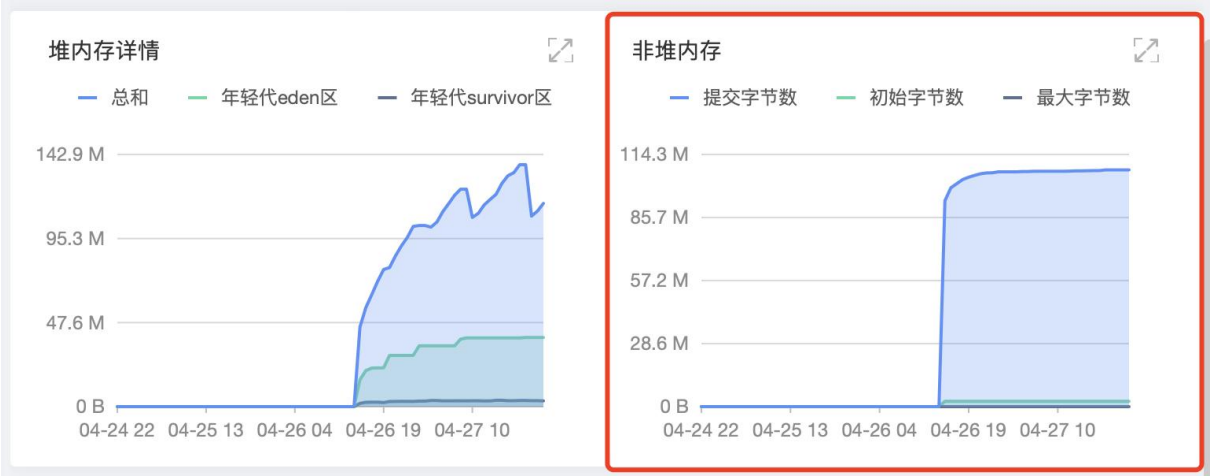
堆内存详情



堆内存用于存储对象实例，包含 1/3 的新生代和 2/3 的老年代。

- **新生代**：是用来存放新生的对象。 分为 Eden 区、 SurvivorFrom、 SurvivorTo 三个区。Minor GC 进行垃圾回收。JVM 每次只会使用 Eden 和其中的一块 Survivor 区域来为对象服务，新生代实际可用的内存空间为 9/10（ 即 90%)的新生代空间。
  - **Eden 区** ：Java 新对象的出生地（如果新创建的对象占用内存很大，则直接分配到老年代）。当 Eden 区内存不够的时候就会触发 Minor GC，对新生代区进行一次垃圾回收。Servivor 区不会触发 Minor GC。
  - **ServivorFrom 区**：上一次 GC 的幸存者，作为这一次 GC 的被扫描者。
  - **ServivorTo 区**：保留了一次 MinorGC 过程中的幸存者。
- **老年代**：主要存放应用程序中生命周期长的内存对象。 Major GC 进行垃圾回收。可能会抛出 OOM（Out Of Memory）异常

非堆内存

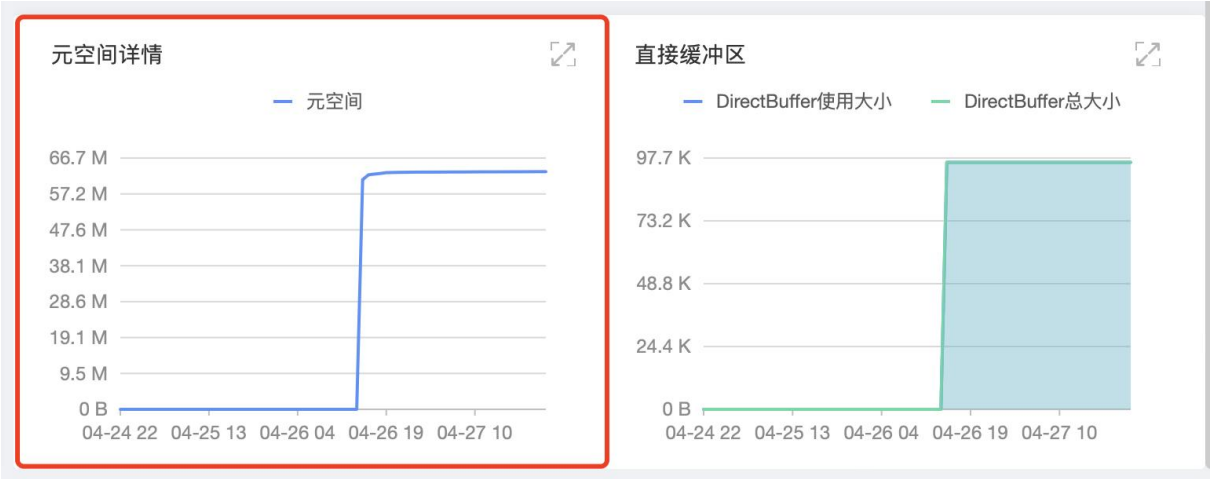


非堆内存用于存储 Java 虚拟机自身使用的数据和代码等等。提交字节数、初始字节数和最大字节数是指非堆内存的三个重要参数，具体含义如下

- **提交字节数（Committed Bytes）**：指已经被操作系统分配给 Java 虚拟机的非堆内存大小，包括已经使用的和未使用的部分，是在 Java 虚拟机运行时动态确定的。
- **初始字节数（Initial Bytes）**：指 Java 虚拟机启动时申请的非堆内存大小，也就是 Java 虚拟机最初分配的非堆内存大小。
- **最大字节数（Max Bytes）**：指 Java 虚拟机能够申请的非堆内存的最大值。当 Java 虚拟机需要更多的非堆内存时，可以向操作系统请求更多的内存，直到达到最大字节数为止。

这些指标可以用来观察非堆内存的大小变化，以便我们根据具体的应用程序调整非堆内存的参数设置（例如使用“-XX:MaxDirectMemorySize”参数来设置非堆内存的最大字节数等等），以保证应用程序能够正常运行，并且不会造成不必要的内存浪费。

元空间详情



- **元空间**（Metaspace）：是 JDK1.8 及以上版本引入的概念，用来替代了永久代（PermGen）的概念。元空间主要用来存储 class 的元数据信息，包括类的名称、父类、接口、字段、方法等信息。元空间不再像永久代那样有固定的大小，而是使用本地内存来存储元数据。通过设置元空间大小参数，可以控制元空间的大小。当元空间不足时，JVM 会自动扩容。元空间的大小仅受本地内存限制。

  永久代（PermGen）是 JDK1.7 及以下版本中的概念，用来存储类信息和常量池。永久代的大小是固定的，由 JVM 启动时指定。当永久代空间不足时，会导致 OutOfMemoryError 异常。

直接缓冲区

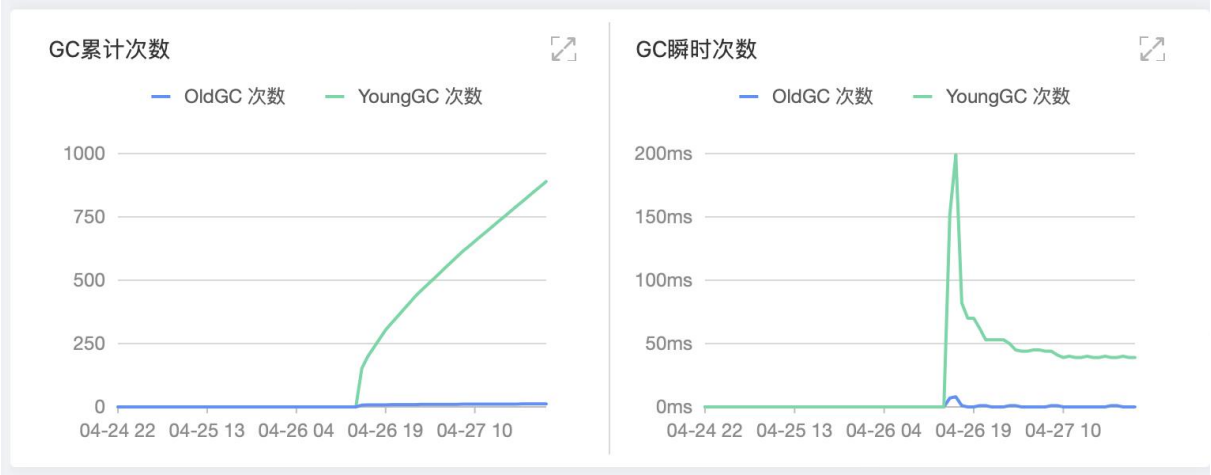
直接缓冲区（Direct Buffer）通常也称为直接内存（Direct Memory），它是一种可以直接访问操作系统内存空间的缓冲区。与普通的 Java NIO 缓冲区不同，直接缓冲区不需要将数据从 Java 堆内存复制到直接内存，而是直接将数据存储在直接内存中，从而避免了数据拷贝的开销，提高了 IO 操作的效率。

- **DirectBuffer 总大小**：指已经被 Java 虚拟机分配的所有 DirectBuffer 缓冲区的总大小，包括已经使用的和未使用的部分。
- **DirectBuffer 使用大小**：指当前正在被使用的 DirectBuffer 缓冲区的总大小。

DirectBuffer 总大小和使用大小都是在运行时动态确定的。这些指标可以用来判断直接缓冲区的使用情况。由于直接内存不受 Java 虚拟机的垃圾回收机制控制，所以如果不及时释放直接内存，可能会导致内存泄漏或者 OutOfMemoryError 等问题。因此，我们需要关注直接内存的使用情况，以便及时释放不再需要的内存空间，可以使用 ByteBuffer 的 cleaner()方法或者手动调用 System.gc()方法来释放直接内存。

GC

GC 累计次数与 GC 瞬时次数



提供累计/瞬时两种视角查看。

- **OldGC 次数**：指的是 Java 虚拟机中的老年代垃圾回收次数。它主要处理老年代中不再使用的对象。
- **YoungGC 次数**：指的是 Java 虚拟机中的新生代垃圾回收次数。它主要处理新生代中不再使用的对象。

GC 累计耗时与 GC 瞬时耗时

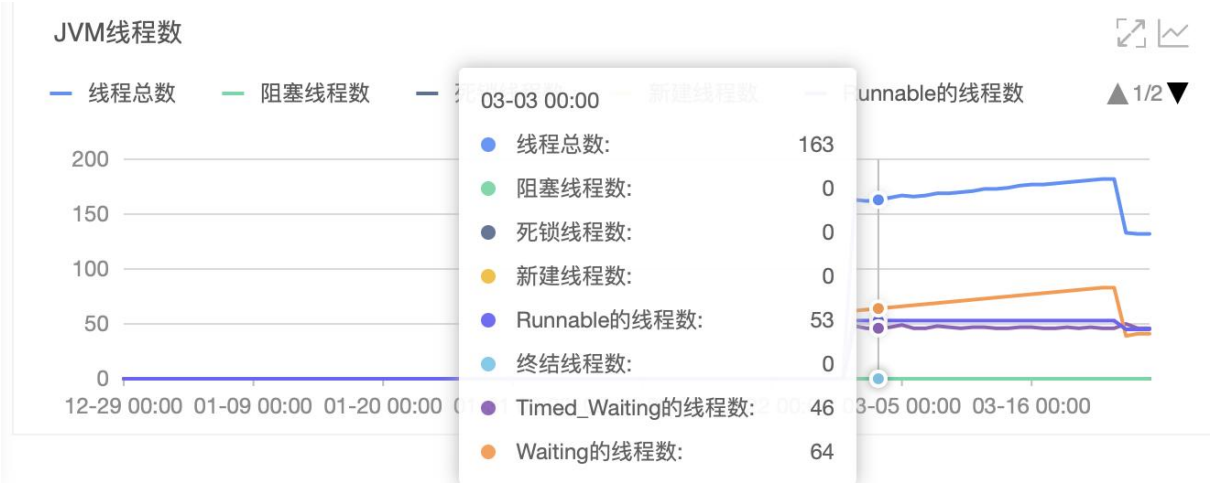
提供累计/瞬时两种视角查看。

- OldGC 耗时：指的是 Java 虚拟机中的老年代垃圾回收累计花费的时间。
- YoungGC 耗时：指的是 Java 虚拟机中的新生代垃圾回收累计花费的时间。

线程

线程（Thread）是操作系统能够进行运算调度的最小单位，是程序执行的基本单元。在 Java 中，每个线程都是一个独立的执行路径，它可以独立地执行代码、访问变量和资源，与其他线程并发执行。

JVM 线程数



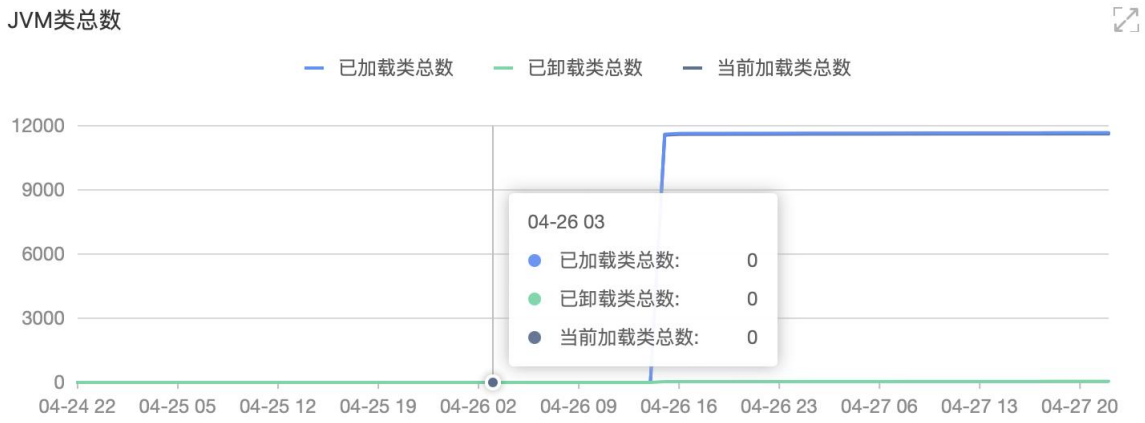
JVM 线程数是指在 Java 虚拟机中活跃的线程数，也就是当前正在执行的 Java 线程数量，包括用户线程（比如计算、IO 操作等等）和守护线程（比如垃圾回收、内存监控等等）。

- **线程总数**：指当前 Java 虚拟机中的线程总数，包括用户线程和守护线程。
- **阻塞线程数**：指当前因为等待某些条件而被阻塞的线程数量，例如等待 IO 操作完成、等待锁释放等。
- **死锁线程数**：指当前处于死锁状态的线程数量，即两个或多个线程相互等待对方释放资源，从而导致所有线程都无法继续执行。
- **新建线程数**：指当前正在创建的线程数量，这些线程尚未开始执行任何任务。
- **Runnable 线程数（可运行线程数）**：指当前处于可运行状态的线程数量，这些线程已经准备好被调度执行，但是可能还没有得到 CPU 的时间片。

- **终结线程数**：指已经被终止但是还没有被垃圾回收的线程数量，这些线程的 run() 方法已经执行完毕，但是线程对象还没有被回收。
- **Timed_Waiting 的线程数（限时等待线程数）**：指当前正在等待一段时间后才能继续执行的线程数量，例如使用 Thread.sleep()方法或 Object.wait(long)方法进行限时等待的线程数量。
- **Waiting 的线程数（等待中线程数）**：指当前正在等待某些条件而被挂起的线程数量，例如使用 Object.wait()方法进行无限等待的线程数量。

这些指标可以用来监控 Java 虚拟机中线程状态的变化，以及分析线程运行情况和性能瓶颈。

JVM 类总数



- **已加载类总数**：指的是当前已经被 Java 虚拟机（JVM）加载的类的总数。当 JVM 加载一个类时，它会对该类进行解析、验证和准备，并将其放入方法区（或称为永久代或元空间，具体取决于 JVM 的版本和配置）。已加载类总数表示了在 JVM 启动以来，已经成功加载到内存中的类的数量。
- **已卸载类总数**：指的是从 JVM 中卸载（或称为卸载垃圾收集）的类的总数。在某些情况下，JVM 可能会卸载已加载的类，例如当类的实例被垃圾收集器判定为不再可达时。已卸载类总数表示自 JVM 启动以来被卸载的类的数量。
- **当前加载类总数**：指的是当前在 JVM 中仍然保留的已加载类的数量。它包括尚未被卸载的类以及仍然被 JVM 使用的类。这个数字可以随着应用程序的进行而变化，因为新的类可以被动态地加载到 JVM 中，而一些不再使用的类可以被卸载。

4.1.4.4、其他基础监控

针对资源、Netty 内存等进行监控。

功能入口

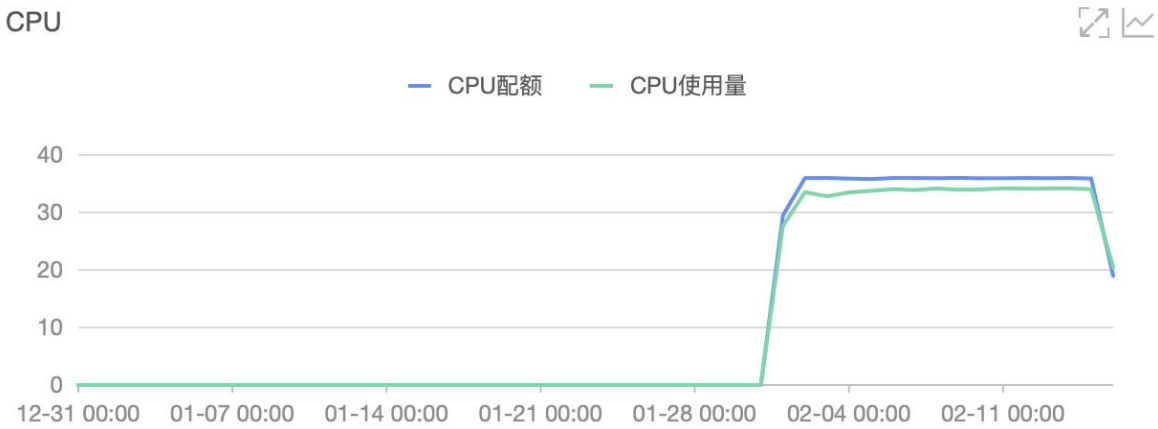
1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用详情**」，您可以在应用详情页面切换至「**其他基础监控**」页签，在左侧关键指标中选择不同的应用实例，可查看该应用实例相应的概览信息。

功能说明

资源监控

包含对 CPU、**物理内存**、**流量**的监控，是保障应用程序稳定性和可靠性的重要任务之一。

CPU



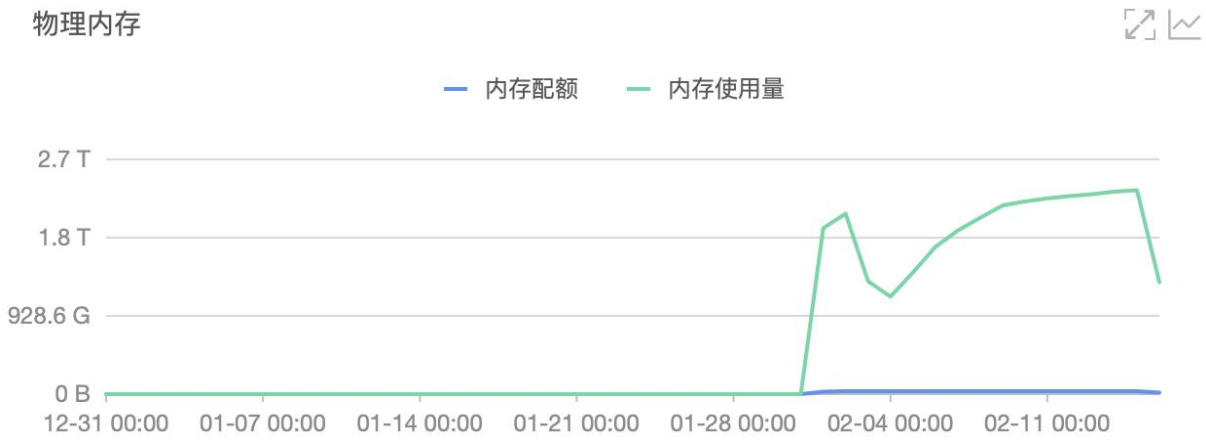
指 Pod 中运行的容器所占用的 CPU 资源，CPU 资源的使用情况可以用来评估容器的负载情况，以及调整容器的资源分配。

- **CPU 配额（CPU Quota）：**指为容器分配的 CPU 资源上限，单位是 CPU 核数。
- **CPU 使用量（CPU Usage）：**指容器实际使用的 CPU 资源量，单位是 CPU 核数。

需注意，使用量不能超过配额。例如，一个 Pod 的 CPU 配额为 1 核，表示该 Pod 中所有容器的 CPU 使用量总和不能超过 1 核。如果容器的 CPU 使用量超过了配额限制，容器将会被限制在配额范围内运行，超出部分的 CPU 使用量将会被抛弃。

通过观察 CPU 配额和 CPU 使用量可以合理分析 CPU 资源使用率，避免资源浪费和不足。

物理内存



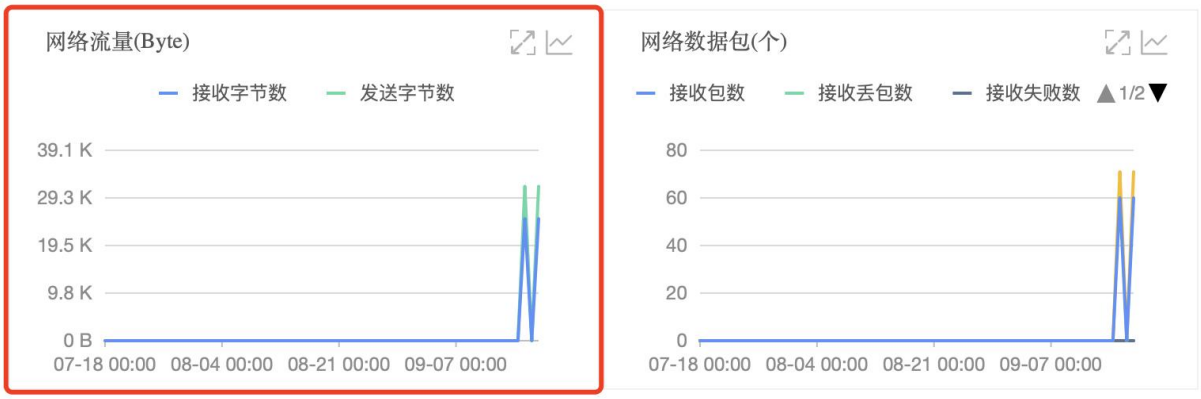
Pod 中运行的容器所占用的物理内存资源，内存资源的使用情况可以用来评估容器的内存使用情况，以及调整容器的资源分配。

- **内存配额（Memory Quota）：**指为容器分配的内存资源上限，通常以字节或者二进制字节（Byte、KB、MB、GB 等）的形式表示。
- **内存使用量（Memory Usage）：**指容器实际使用的内存资源量，通常以字节或者二进制字节（Byte、KB、MB、GB 等）的形式表示。

需注意，使用量不能超过配额。例如，一个 Pod 的内存配额为 1GB，表示该 Pod 中所有容器的内存使用量总和不能超过 1GB。如果容器的内存使用量超过了配额限制，容器将会被限制在配额范围内运行，超出部分的内存使用量将会被抛弃。

通过观察物理内存配额及内存使用量，可以帮助用户根据应用程序的实际内存使用情况，合理设置内存配额，以避免内存资源的浪费和不足。

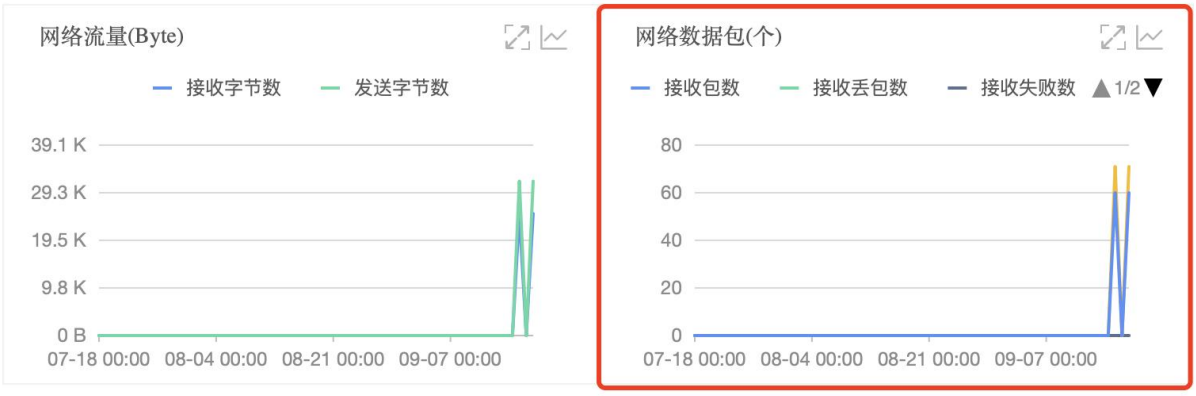
网络流量



- **网络流量**：Pod 中容器的网络数据量，单位是 Byte。
- **接收字节数**：也称作入站流量，是指进入容器 pod 的数据量。
- **发送字节数**：也称作出站流量，是从容器 pod 发送出去的数据量。

监控网络数据量可以诊断网络性能问题、预测网络容量需求，了解网络的使用情况和资源瓶颈，及时进行网络流量控制和优化，以避免网络拥塞和数据传输失败。

网络数据包



- **网络数据包**：Pod 中容器之间或者容器与外部网络之间进行通信的基本单位。它是在网络中传输的数据的载体，包含了一定的控制信息和有效载荷数据。

- **接收包数**：接收包数是指网络接口（网卡）成功接收到的数据包的总数量。它表示从网络中接收到的数据包的计数。
- **接收丢包数**：接收丢包数指的是在接收过程中丢失的数据包数量。这表示在数据包传输过程中，一些数据包未能成功到达目的地或未能被接收，导致丢包。
- **接收失败数**：接收失败数是指在接收过程中发生的错误数量。这些错误可能与数据包传输、网络协议或硬件设备有关，例如校验错误、帧错误等。
- **发送包数**：发送包数是指通过网络接口成功发送的数据包的总数量。它表示从容器 pod 发送到网络的数据包的计数。
- **发送丢包数**：发送丢包数指的是在发送过程中丢失的数据包数量。这表示在数据包传输过程中，一些数据包未能成功到达目的地或未能被接收，导致丢包。
- **发送失败数**：发送失败数是指在发送过程中发生的错误数量。这些错误可能与数据包传输、网络协议或硬件设备有关，例如校验错误、帧错误等。

监控网络数据包的数量可以帮助管理员诊断网络性能问题和瓶颈，以及进行容器资源分配和负载均衡的调整。例如，如果某个容器接收到的网络数据包数量过多，可能会导致容器负载过高，需要调整容器资源分配或者优化应用程序的网络使用方式。此外，对于网络数据包的监控还可以帮助管理员检测网络攻击和安全漏洞，以及进行网络流量控制和优化。

Netty 内存

监控 netty 内存的使用情况。包括已使用直接内存和最大直接内存的变化趋势。

4.1.4.5、数据库监控

数据库监控通常是指对一段时间内对各类数据库调用情况的统计分析。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」—「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用详情**」或「**接口调用**」或「**数据库调用**」，您可以在应用详情页面切换至「**数据库监控**」页签，在左侧关键指标中**选择不同的应用实例/接口/SQL**，可查看该应用实例/接口/SQL 相应的概览信息。

功能说明

当前支持的数据库包含 MySQL、ClickHouse、Postgresql、Elasticsearch、MongoDb、DBCP 连接池、Druid 连接池、C3P0 连接池。

Mysql

统计图

显示该应用在筛选时间段内的所有 SQL 语句的调用次数和平均耗时变化趋势图，显示不同耗时区间请求数的变化趋势图。

统计表

以 SQL 语句为维度，详细显示不同 SQL 语句各自的调用信息，表头如下。

- **所属应用**：这是指 MySQL 数据库服务于哪个应用，显示应用名称。
- **SQL 语句**：这通常指在数据库操作中执行的具体 SQL 命令，如 SELECT、INSERT、UPDATE、DELETE 等。
- **平均耗时(ms)**：这是指每次查询或操作所需的平均时间，这个指标可以帮助我们了解数据库响应速度的一般水平。
- **调用次数**：这是指在一定时间内，数据库被请求的总次数。这是一个衡量数据库负载的重要指标。
- **慢调用次数**：这是指执行时间超过设定阈值（默认 500ms，在 url 采集设置中可以修改慢调用阈值）的查询次数。这个指标用于识别和解决性能瓶颈。

- **错误次数**：这是指在数据库操作中出现错误的次数，这可能包括违反数据完整性约束、权限问题等。
- **0 到 10ms 请求数**：这是指在 0 到 10 毫秒内完成的查询数量，这反映了数据库响应速度的快速性。
- **10 到 100ms 请求数**：这是指在 10 到 100 毫秒内完成的查询数量，这也是评估数据库性能的一个重要指标。
- **100 到 500ms 请求数**：这是指在 100 到 500 毫秒内完成的查询数量，这个范围通常被视为较慢的响应时间。
- **500 到 1000ms 请求数**：这是指在 500 到 1000 毫秒内完成的查询数量，这个范围通常被视为较慢的响应时间。
- **1 到 10s 请求数**：这是指在 1 到 10 秒内完成的查询数量，这个范围通常被视为非常慢的响应时间。
- **10s 以上请求数**：这是指在 10 秒以上内完成的查询数量，这个范围通常被视为极其慢的响应时间。

这些指标共同构成了对 MySQL 数据库性能的全面评估，有助于识别问题并进行相应的优化措施。

ClickHouse

与 Mysql 监控指标一致，提供趋势统计图和统计表。

Elasticsearch

除 SQL 语句改为 url 外，其他监控指标与 Mysql 一致，提供趋势统计图和统计表。

MongoDb

除 SQL 语句改为指令外，其他监控指标与 Mysql 一致，提供趋势统计图和统计表。

Druid 连接池

统计图

显示该应用在筛选时间段内的所有 druid 连接池各指标的变化趋势图，包括连接总数、初始化连接数、活跃连接数、最大活跃连接数、连接池大小上限、连接池最大空闲数、连接池最小空闲数、池中连接数、最大池中连接数、等待线程数、超时连接回收次数。

统计表

以连接池为维度，详细显示各自的连接信息，表头如下。

- **连接池 ID**：用于标识 Druid 连接池的唯一标识符或 ID。它可以用于区分不同的连接池实例。

- **连接总数**：表示连接池中当前存在的总连接数，包括活跃连接和空闲连接。
- **初始化连接数**：指在连接池初始化时创建的初始连接数量。
- **活跃连接数**：表示当前正在被使用的连接数，即已经被借出但未归还的连接数。
- **最大活跃连接数**：连接池允许的最大活跃连接数。达到此限制后，请求获取连接的线程将进入等待状态。
- **连接池大小上限**：连接池允许的最大连接数，包括活跃连接和空闲连接的总数。
- **连接池最大空闲数**：连接池允许的最大空闲连接数。当连接数超过该限制时，多余的连接将被关闭。
- **连接池最小空闲数**：连接池维持的最小空闲连接数。当连接池中的空闲连接数少于该限制时，将创建新的连接。
- **池中连接数**：当前连接池中存在的连接数量，包括活跃连接和空闲连接。
- **最大池中连接数**：连接池允许的最大池中连接数。达到此限制后，无法再创建新的连接。
- **等待线程数上限**：连接池允许的最大等待获取连接的线程数量。超过此限制的线程将无法获取连接并进入等待状态。
- **等待线程数**：当前正在等待获取连接的线程数量。
- **超时连接回收次数**：连接池中由于超时而被回收的连接次数。
- **池中连接可空闲的时间**：连接在池中可以保持空闲的时间。超过该时间的空闲连接将被回收。
- **获取连接最大等待时间**：获取连接的线程最大允许等待的时间。超过该时间仍未获取到连接的线程将放弃等待。
- **使用时长上限**：连接从借出到归还的最大允许使用时长。超过该时长的连接将被回收。
- **检查池中连接空闲周期**：连接池对池中空闲连接进行检查的周期。用于检测空闲连接的活性和超时。

C3P0 连接池

统计图

显示该应用在筛选时间段内的所有 C3P0 连接池各指标的变化趋势图，包括连接总数、活跃连接数、空闲连接数。

统计表

以连接池为维度，详细显示各自的连接信息，表头如下。

- **连接池 ID**：用于标识 C3P0 连接池的唯一标识符或 ID。它可以用于区分不同的连接池实例。
- **连接总数**：表示连接池中当前存在的总连接数，包括活跃连接和空闲连接。
- **活跃连接数**：表示当前正在被使用的连接数，即已经被借出但未归还的连接数。
- **空闲连接数**：表示当前可用的空闲连接数，即没有被使用的连接数。

- **空闲连接检测周期(ms)**：连接池对空闲连接进行检测的时间间隔。在该周期内，连接池将检测空闲连接的可用性，以决定是否回收或重新创建连接。
- **获取连接重试次数**：在无法获取到连接时，连接池进行重试的次数。当达到重试次数仍未获取到连接时，获取连接操作将失败。
- **获取连接重试间隔(ms)**：在进行连接获取重试时，每次重试之间的时间间隔。该间隔用于控制连接获取的频率。
- **无连接可用时创建连接数**：当连接池中没有空闲连接可用时，连接池将创建的新连接数。这样可以确保在高并发情况下始终有一定数量的连接可供使用。

DBCP 连接池

统计图

显示该应用在筛选时间段内的所有 DBCP 连接池各指标的变化趋势图，包括初始化连接数、活跃连接数、空闲连接数、连接池最大空闲数、连接池最小空闲数。

统计表

以连接池为维度，详细显示各自的连接信息，表头如下。

- **连接池 ID**：用于标识 DBCP 连接池的唯一标识符或 ID。它可以用于区分不同的连接池实例。
- **初始化连接数**：指在连接池初始化时创建的初始连接数量。
- **活跃连接数**：表示当前正在被使用的连接数，即已经被借出但未归还的连接数。
- **空闲连接数**：表示当前可用的空闲连接数，即没有被使用的连接数。
- **连接池最小空闲数**：连接池维持的最小空闲连接数。当连接池中的空闲连接数少于该限制时，将创建新的连接。
- **连接池最大空闲数**：连接池允许的最大空闲连接数。当连接池中的空闲连接数超过该限制时，多余的空闲连接将被关闭。
- **等待连接被回收的最长时间(ms)**：连接在归还到连接池之前可以等待的最长时间。如果连接池中没有空闲连接，请求归还连接的操作将等待一段时间，如果超过该时间仍未归还，则连接会被强制回收。
- **验证连接是否有效的时间周期(ms)**：连接池对连接进行验证的时间周期。在该周期内，连接池会定期验证连接的有效性，以确保连接仍然可用。

4.1.4.6、缓存监控

缓存监控通常是指对一段时间内对各类缓存数据库调用情况的统计分析。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。

2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。

3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用详情**」或「**接口调用**」或「**数据库调用**」，您可以在应用详情页面切换至「**缓存监控**」页签，在左侧关键指标中**选择不同的应用实例/接口/缓存**，可查看该应用实例/接口/缓存相应的缓存信息。

功能说明

当前支持的缓存数据库包含 Redis、Jedis、Lettuce。

Redis

统计图

显示该应用在筛选时间段内的所有操作命令的调用次数和平均耗时变化趋势图，显示不同耗时区间请求数的变化趋势图。

统计表

以 NoSQL 操作命令为维度，详细显示不同操作命令各自的信息，表头如下。

- **所属应用**：显示 NoSQL 操作命令所属的应用名称。
- **NoSQL 类型**：Redis 属于 NoSQL 数据库，因为它不遵循传统关系型数据库的表格模型，而是采用键值对的形式存储数据。
- **操作命令**：Redis 提供了多种命令行命令，用于管理和操作 Redis 数据库，如 SET、GET、DEL、EXPIRE 等。
- **平均耗时**：指每次操作的平均响应时间，通常以毫秒为单位。例如，每隔 1 秒采样一次，得到的平均耗时可能在 0.08 ~ 0.13 毫秒之间。
- **调用次数**：指在一定时间内执行某个操作的次数。
- **慢调用次数**：指响应时间超过某个阈值（默认 500ms，在 url 采集设置中可以修改慢调用阈值）的操作次数。
- **0 到 10ms 请求数**：指在 0 到 10 毫秒内完成的请求数量。
- **10 到 100ms 请求数**：指在 10 到 100 毫秒内完成的请求数量。
- **100 到 500ms 请求数**：指在 100 到 500 毫秒内完成的请求数量。
- **500 到 1000ms 请求数**：指在 500 到 1000 毫秒内完成的请求数量。
- **1 到 10s 请求数**：指在 1 到 10 秒内完成的请求数量。
- **10s 以上请求数**：指在 10 秒以上内完成的请求数量。

Jedis

统计图

显示该应用在筛选时间段内最大连接数、最大空闲数、最小空闲数、当前激活个数、当前空闲个数、等待个数的变化趋势图。

统计表

以实例为维度，详细显示不同实例各自的监控信息，表头如下。

实例 ID：用于标识 Jedis 连接池的唯一标识符或 ID。它可以用于区分不同的连接池实例。

- 最大连接数：连接池允许的最大连接数。超过该限制的连接请求将被阻塞，直到有连接可用或达到最大等待时间。
- 最大空闲数：连接池允许的最大空闲连接数。当连接池中的空闲连接数超过该限制时，多余的空闲连接将被关闭。
- 最大等待时间（ms）：连接池中获取连接时的最大等待时间。如果连接池中没有可用连接，请求获取连接的操作将等待一段时间，如果超过该时间仍未获取到连接，则连接获取操作会失败。
- 平均激活时间（ms）：连接池中连接被激活的平均时间。它表示连接从被借出到归还的平均时间。
- 平均 Borrow 等待时间（ms）：连接池中获取连接时的平均等待时间。它表示连接获取操作在等待可用连接时的平均等待时间。
- Borrow 最大等待时间（ms）：连接池中获取连接时的最大等待时间。如果连接获取操作在等待可用连接时超过该时间，则连接获取操作会失败。
- 最小空闲数：连接池维持的最小空闲连接数。当连接池中的空闲连接数少于该限制时，将创建新的连接。
- 当前空闲个数：连接池中当前可用的空闲连接数，即没有被使用的连接数。
- 等待个数：当前等待获取连接的请求数。它表示连接获取操作当前正在等待可用连接的请求数。
- 当前激活个数：连接池中当前正在被使用的连接数，即已经被借出但未归还的连接数。

Lettuce

与 Jedis 监控指标一致，提供趋势统计图和统计表。

4.1.4.7、消息监控

针对各类消息插件进行监控。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。

2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。

3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用详情**」，您可以在应用详情页面切换至「**消息监控**」页签，在左侧关键指标中选择不同的应用实例，可查看该应用实例相应的概览信息。

功能说明

KafkaProducer

总发送次数&总发送字节数

显示总发送次数和总发送字节数的趋势图。

- **总发送次数**：这是指生产者在其生命周期内成功发送到 Kafka broker 的消息数量。这个数字可以帮助我们了解生产者在实际运行中的消息处理能力和稳定性。例如，如果一个生产者在一段时间内发送了 10000 条消息，那么它的总发送次数就是 10000。
- **总发送字节数**：这是指生产者在其生命周期内发送到 Kafka broker 的所有消息的总字节数。这个数字反映了生产者传输的数据量，是评估生产者负载和资源消耗的一个重要指标。例如，如果生产者发送了 10000 条消息，每条消息大小为 1KB，则总发送字节数为 10000 * 1024 = 10,485,760 字节。

这两个指标通常用于监控和优化 Kafka 生产者的性能，确保其能够高效、稳定地处理消息。通过这些数据，可以发现生产者是否存在瓶颈或问题，并进行相应的调整和优化。

通过消息队列发送消息

显示 topic 列表，表头如下。

- Topic：是指消息发送到消息队列中的特定主题（topic）。主题是消息队列中的逻辑分类，用于将相关的消息归类和分组。每个 topic 可以包含多条消息，每条消息都有一个特定的主题标签。
- **调用次数**：表示当前这个 Topic 在一段时间内，通过消息队列发送消息的总调用次数。每次发送消息都会被计算为一次调用。
- **平均响应时间(ms)**：是指在发送消息的过程中，从发送请求到接收到响应的平均时间。它表示了发送消息的速度和效率。
- **错误数**：表示在发送消息的过程中发生的错误次数。这些错误可能包括发送消息失败、网络连接问题或其他异常情况。
- **最慢调用(ms)**：是指在一段时间内，发送消息过程中最耗时的一次调用的时间。它反映了发送消息中的性能瓶颈或延迟情况。
- **操作**：点击详情，出现弹层显示调用次数、平均响应时间(ms)、错误数在筛选时间段内的趋势图。

这些指标可以帮助您监控和分析通过消息队列发送消息的性能和健康状况。通过追踪调用次数、平均响应时间、错误数和最慢调用时间，您可以了解消息发送的频率、效率、稳定性和延迟情况，从而进行性能优化、故障排除和容量规划等方面的工作。

topic 监控

显示实例纬度监控列表，表头如下。

- **Topic**：这是一个特定的命名空间，用于组织、过滤和路由消息。在 Kafka 中，每个 Topic 代表了一个主题，可以被不同的生产者和消费者使用来发送和接收消息。
- **ID**：每个 Topic 都有一个唯一的 ID，这个 ID 用于标识该 Topic。在监控系统中，通常会通过这个 ID 来追踪和管理 Topic 的相关数据。
- **总发送次数**：这是指在监控周期内，所有生产者向该 Topic 发送消息的总次数。这是一个衡量消息吞吐量的重要指标，可以反映出 Topic 的活跃度和负载情况。
- **总发送字节数**：这是指在监控周期内，所有生产者向该 Topic 发送的数据总量（以字节为单位）。这个指标可以帮助我们了解 Topic 的数据传输量，是评估网络带宽需求的一个重要参数。
- **每秒发送数**：这是指在一秒钟内，成功发送到该 Topic 的消息数量。这个指标可以用来衡量 Topic 的实时吞吐量，是评估系统性能的一个重要指标。
- **每秒发送字节数**：这是指在一秒钟内，所有生产者向该 Topic 发送的数据量（以字节为单位）。这个指标可以用来衡量 Topic 的实时数据传输速率，对于优化网络性能和资源分配非常有帮助。
- **每秒错误数**：这是指在一秒钟内，由于各种原因导致的消息发送失败的次数。这个指标可以帮助我们了解消息传输过程中的问题，比如网络问题或消息格式错误等。
- **每秒重试数**：这是指在一秒钟内，因发送失败而被重试的消息数量。这个指标可以帮助我们了解消息重试机制的效果，以及是否需要调整重试策略以提高系统的稳定性和可靠性。
- **操作**：点击调用链查询切换至调用链查询 tab，并带入相关查询条件，展示当前 topic 的调用链信息。

通过这些监控指标，我们可以全面了解 Topic 的性能和状态，从而进行相应的优化和调整，以确保系统的高效运行。

KafkaConsumer

总消费次数&总消费字节数

显示总消费次数和总消费字节数的趋势图。

- **总消费次数**：这个指标表示消费者从 Kafka 中成功读取并处理的消息数量。它是一个计数器，用于跟踪消费者在其生命周期内接收到的消息总数。这对于监控消费者的工作负载和性能非常有用，可以帮助开发者了解消费者是否有效地从生产者那里获取了数据。
- **总消费字节数**：这个指标表示消费者从 Kafka 中成功读取的数据总量，以字节为单位。它考虑了每条消息的大小，并累加起来给出一个总和。这对于评估网络带宽使

用情况、存储需求以及数据传输效率等方面非常重要。例如，如果一个主题有 20 个分区和 5 个消费者，每个消费者需要至少 4MB 的可用内存来接收记录。因此，通过监控总消费字节数，可以更好地理解数据流的规模和处理速度。

这两个指标共同提供了对 Kafka 消费者行为的全面视图，使得开发者能够优化系统性能，确保数据处理的高效性和稳定性。

通过消息队列接收消息

显示 topic 列表，表头如下。

- topic：是指消息队列中接收到的特定主题（topic）的消息。主题用于将相关的消息进行归类和分组。
- **调用次数**：是指在一段时间内，通过消息队列接收消息的总调用次数。每次调用消息队列的接收操作（例如 KafkaConsumer 的 poll()方法）并成功获取一条消息都会被计算为一次调用。
- **平均响应时间(ms)**：是指从发起接收请求到接收到消息的平均时间。它表示了接收消息的速度和效率。
- **错误数**：表示在接收消息的过程中发生的错误次数。这些错误可能包括接收消息失败、网络连接问题或其他异常情况。
- **最慢调用(ms)**：是指在一段时间内，接收消息过程中最耗时的一次调用的时间。它反映了接收消息中的性能瓶颈或延迟情况。
- **操作**：点击详情，出现弹层显示调用次数、平均响应时间(ms)、错误数在筛选时间段内的趋势图。

这些指标共同构成了对消息队列系统性能和可靠性的全面评估，有助于管理员和开发者优化系统配置，提高系统的稳定性和效率。

topic 监控

显示实例纬度监控列表，表头如下。

- Topic：这是消息队列中的一个概念，通常用于分类和组织消息。每个 Topic 可以包含多个 Partition，每个 Partition 可以被不同的 Consumer Group 消费。
- ID：这是 Topic 的一个唯一标识符，用于在系统中唯一标识该 Topic。
- **总消费次数**：这是指在监控周期内，所有从该 Topic 消费消息的总次数。
- **总消费字节数**：这是指在监控周期内，所有从该 Topic 中消费的数据总量（以字节为单位）。
- **每秒消费数**：这是指在一秒内从特定 Topic 中消费的平均消息数量。这是一个衡量系统吞吐量的重要指标，可以帮助我们了解系统处理消息的能力。
- **每秒消费字节数**：这是指在一秒内从特定 Topic 中消费的平均字节数。这是一个衡量系统吞吐量的重要指标，可以帮助我们了解系统处理数据的能力。
- **请求获取平均字节**：这是指在一次请求中从特定 Topic 中获取的平均字节数。这可以帮助我们了解单次请求的大小，从而评估系统的负载情况。

- **请求获取最大字节**：这是指在一次请求中从特定 Topic 中获取的最大字节数。这可以帮助我们了解系统可能面临的峰值负载情况。
- **单次请求平均消息数**：这是指在一次请求中从特定 Topic 中获取的平均消息数量。这可以帮助我们了解系统在处理消息时的效率。
- **操作**：点击调用链查询切换至调用链查询 tab，并带入相关查询条件，展示当前 topic 的调用链信息。

通过监控这些指标，我们可以及时发现系统中的问题，如消息堆积、延迟等，并进行相应的优化和调整，以确保系统的高性能和稳定性。

RabbitMQProducer

和 KafkaProducer 类似，RabbitMQProducer 主要负责发送消息到 RabbitMQ，监控在这一过程中的各项指标，包括：

- 总发送次数趋势图、总发送字节数趋势图。
- 不同 exchange 的调用次数、平均响应时间、错误数、最慢调用信息和总发送字节数。

这些指标可以帮助识别生产者在消息发送过程中可能遇到的问题，如网络延迟、服务器负载过高或配置错误等。

RabbitMQConsumer

和 KafkaConsumer 类似，RabbitMQConsumer 主要负责处理接收到的消息，监控在这一过程中的各项指标，包括：

- 总消费次数趋势图、总消费字节数趋势图。
- 不同 consumerTag/consumerClass 的调用次数、平均响应时间、错误数、最慢调用信息和总消费字节数。

这些指标有助于了解消费者的健康状况，例如是否存在未确认的消息积压，消费者是否能够及时处理消息，以及队列是否因为消费者处理不及时而积压。

4.1.4.8、Web 容器监控

针对 tomcat 进行监控。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用详情**」，您可以在应用详情页面切换至「**web 容器监控**」页签，在左侧关键指标中选择不同的应用实例，可查看该应用实例相应的概览信息。

功能说明

Tomcat

Tomcat 线程

显示统计表信息，表头如下。

- **实例 id**：用于标识 Tomcat 线程的唯一标识符或 ID。它可以用于区分不同的线程实例。
- **请求数**：表示在某段时间内由 Tomcat 线程处理的总请求数量。每当 Tomcat 线程接收到一个请求并开始处理时，计数器就会增加。
- **错误数**：在某段时间内由 Tomcat 线程处理的请求中发生错误的次数。错误可能包括 HTTP 错误状态（如 404 或 500），处理异常或其他错误条件。
- **请求处理时间(ms)**：表示 Tomcat 线程处理单个请求的平均时间。它衡量了请求处理的效率和性能。
- **请求处理最大时间(ms)**：在某段时间内由 Tomcat 线程处理的请求中，处理时间最长的请求的时间。它反映了请求处理中的最大延迟或性能瓶颈。
- **请求吞吐量**：表示 Tomcat 线程在某段时间内处理的请求数量。它衡量了 Tomcat 服务器的工作负载和处理能力。
- **活跃会话数**：表示当前活动的会话（session）数量。会话是在用户与服务器之间建立的会话状态，用于跟踪用户的状态和信息。
- **当前线程数**：表示当前正在运行的 Tomcat 线程数量。它反映了 Tomcat 服务器的并发处理能力和资源利用情况。

这些指标可以用于监控和评估 Tomcat 服务器的性能和健康状况。

4.1.4.9、异常错误分析

异常错误分析可以帮助您方便了解应用的异常错误情况。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。

2. 在左侧导航栏中选择「**应用监控**」—「**应用列表**」。

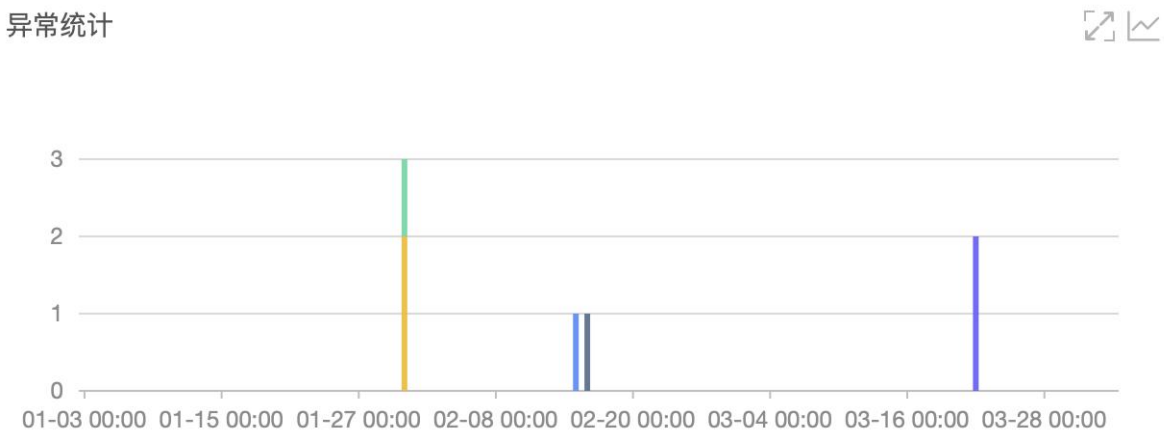
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用详情**」或「**接口调用**」等其他分析视角，您可以在应用详情页面切换至「**异常错误分析**」页签，在左侧关键指标中**选择不同的应用实例/接口等**，可查看该应用实例/接口相应的异常错误信息。

功能说明

异常分析

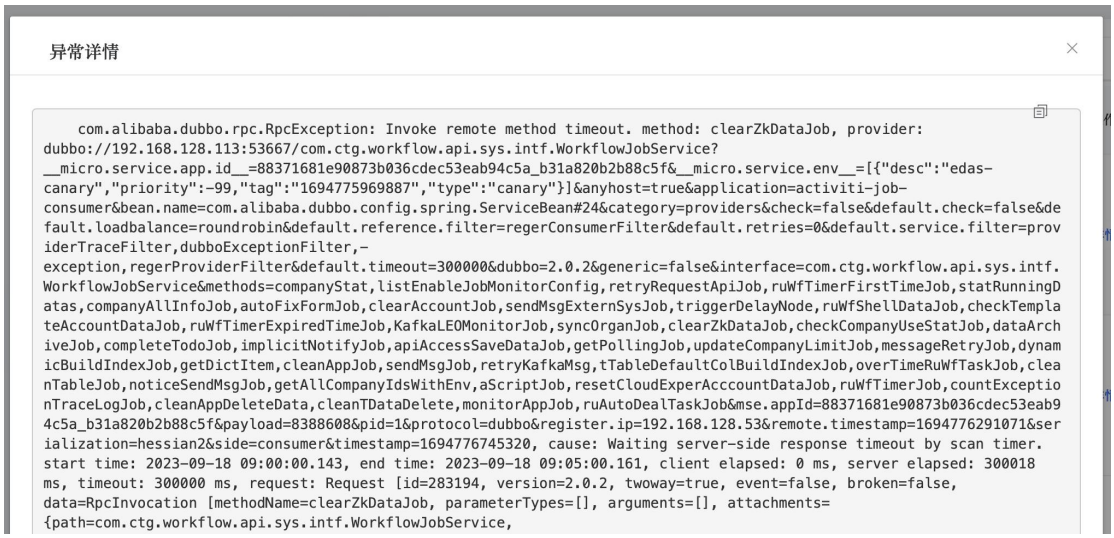
异常分析图



以具体的异常堆栈为维度，展示各个时间段异常出现的次数。

异常明细

- **出现次数**：显示该异常堆栈在筛选时间段内出现的总次数。
- **异常堆栈**：显示异常明细，与点击详情按钮看到的内容一致。
- **操作**
  - **详情**：点击打开异常详情弹窗，显示具体的异常信息，且支持复制。



- **调用链查询：**点击切换至“调用链”tab，查看该异常堆栈对应的调用链的具体信息。

- **调用统计：**点击后，系统会将该异常堆栈作为筛选条件，重新筛选上方异常分析图的结果。

统一交互操作说明：

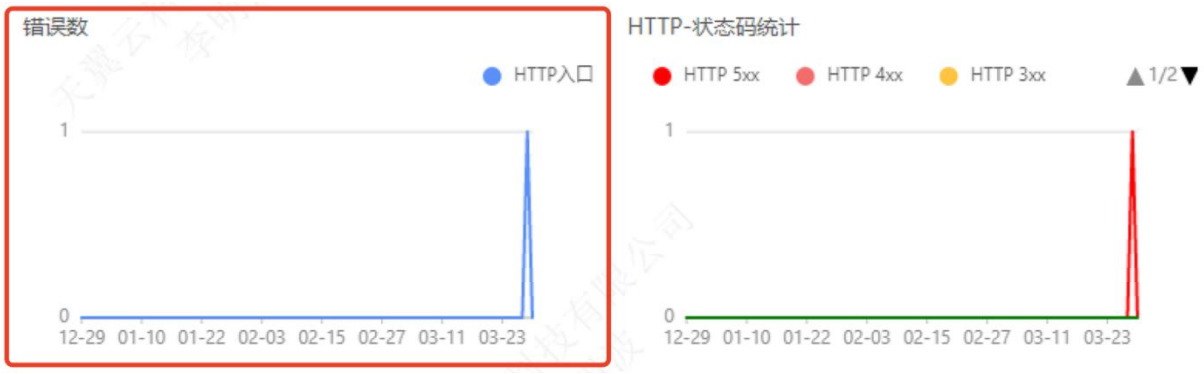
- 将光标移到统计图上，可以查看光标所至时间点的数据详情。
- 单击图标，可以将当前图表放大显示。

错误分析

错误（Error）通常指在应用程序运行过程中出现的严重问题，比如内存溢出、系统崩溃、IO 异常等。错误通常是无法通过编写代码来处理的，因为它们通常是由操作系统、硬件或其他外部因素引起的。

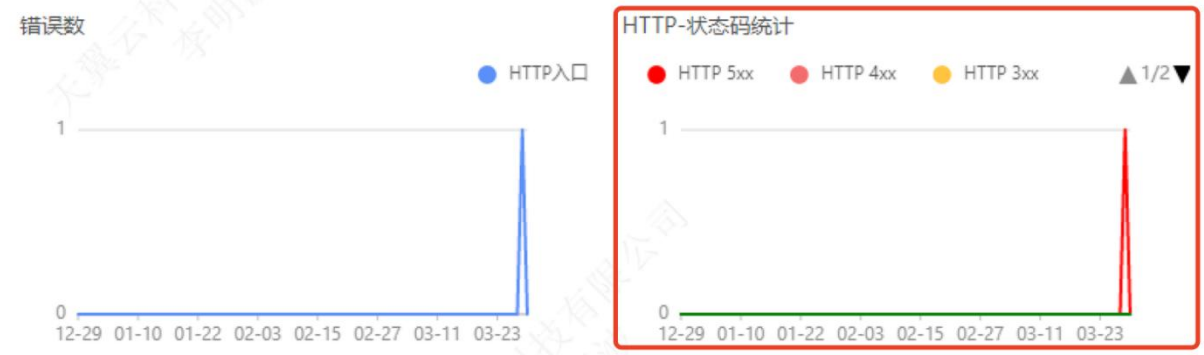
错误分析可以帮助您更快了解应用的错误信息。

错误数



显示筛选时间段内，每天出现的错误次数

Http—状态码统计



显示筛选时间段内 Http 不同状态码出现的次数

- 5xx：服务器异常，服务器在处理请求的过程中发生错误
- 4xx：客户端异常，请求包含语法错误或无法完成请求
- 3xx：重定向问题，需要进一步操作
- 2xx：成功，服务器成功接收请求并执行
- 200：请求成功

错误明细表

产生时间	接口名称	所属应用	耗时(ms)	状态	traceId
2023-03-28 19:03:46.590	http://localhost:8777/exception/test	test_web_liuyd2	8.565		ee91565ec7163bc0ca9e2db56e109cdd
2023-03-28 19:03:44.196	http://localhost:8777/exception/test	test_web_liuyd2	17.47		08ae3a71b8aad2cc6f2eb9579e61b265
2023-03-28 19:02:49.224	http://localhost:8777/users/get?username=12&pageNum=1&pageSize=10	test_web_liuyd2	1989.212		f633b5326268a789e8ddf92ae1497483

显示错误明细

- **产生时间**：发现错误的时间点
- **接口名称**：指发生错误时，调用的 API 接口名称
- **所属应用**：指发生错误时，调用的 API 属于哪个应用
- **耗时（ms）**：指发生错误时，调用 API 的耗时，即 API 调用开始到 API 返回结果的时间间隔。
- **状态**：和状态码对应
- **traceID**：链路 id，指用于跟踪一次 API 调用的唯一标识符，可以跨越不同的应用程序和服务，记录 API 调用的整个调用链路，用于追踪分布式系统中的问题。

统一交互操作说明：

- 将光标移到统计图上，可以查看光标所至时间点的数据详情。
- 单击图标，可以将当前图表放大显示。

4.1.4.8、上下游分析

上下游分析可以查看上下游应用的**请求量**、**平均延时**、**错误数**信息。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用详情**」或「**接口调用**」等其他分析视角，您可以在应用详情页面切换至「**上下游分析**」页签，在左侧关键指标中**选择不同的应用实例/接口等**，可查看该应用实例/接口相应的上下游信息。

功能说明

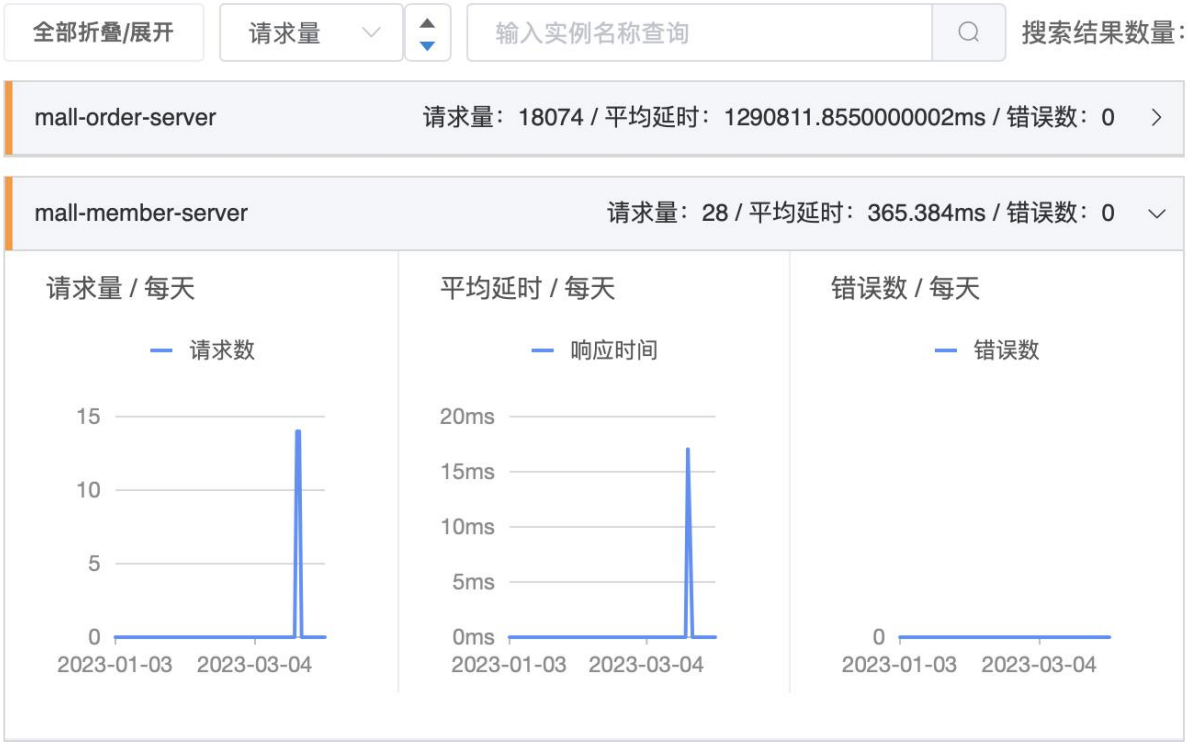
上游应用

上游应用是当前应用/SQL 的调用者。上游应用向当前应用/SQL 发起请求，并等待当前应用/SQL 返回结果。通过上游应用分析可以查看上游应用/SQL 的**请求量**、**平均延时**、**错误数**信息。

- 以卡片形式显示该应用实例在当前筛选时间段内的上游应用，默认全部展开，可以操作展开/收起，每个卡片上显示一个应用的**应用名称、请求量、平均延时**（即：耗时/响应时间）和**错误数**。
- 支持对应用名称进行搜索，支持对请求量/平均延时/错误数进行排序。

下游应用

下游应用指当前应用的被调用方。当前应用程序向下游应用发起请求，并等待下游应用返回结果。通过下游应用分析可以查看下游应用的**请求量、平均延时、错误数**信息。



- 以卡片形式显示该应用实例在当前筛选时间段内的下游应用，默认全部展开，可以操作展开/收起，每个卡片上显示一个应用的应用名称、请求量、平均延时（即：耗时/响应时间）和错误数。
- 支持对应用名称进行搜索，支持对请求量/平均延时/错误数进行排序。

通过记录上游应用和下游应用，可以在应用性能监控系统中查看应用程序与应用/接口之间的调用链路，并追踪请求和响应的流程。

4.1.4.9、调用链查询

展示该**应用实例/数据库等**涉及的**调用链信息**，包括 TraceID、产生时间、接口名称、耗时、状态信息。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择**「应用列表」**。
3. 在应用列表中选择您想查看的应用，点击**「应用名称」**打开新的应用详情链接。
4. 在左侧导航栏中选择**「应用详情」**或**「数据库调用」**等，您可以在应用详情页面切换至**「调用链查询」**页签，在左侧关键指标中**选择不同的应用实例/数据库**，可查看该应用实例/数据库相应的概览信息。

功能说明

选择接口名称

产生时间	接口名称	所属应用	耗时(ms)	状态	TraceId
2023-03-22 21:05:54.773	/mall/getProductList	mall-mall-server	1108.457		78e9d4359bc165177feaff683cba9149
2023-03-22 21:34:32.211	/mall/getProductList	mall-mall-server	312.975		3b2ae9db79f3172200b51d69b9c2797f
2023-03-22 21:14:03.504	/mall/getProductList	mall-mall-server	161.181		93626ea0819a001fff4a8f6970e78ab4

显示筛选时间段内，该应用实例/SQL 涉及的调用链信息

- **产生时间**：该调用链产生的时间点
- **接口名称**:显示发起 API 调用时的接口名称,完整调用链中涉及的接口信息在 trace 详情中查看
- **所属应用**：显示当前应用实例所属应用的名称，该调用链涉及的其他应用信息在 trace 详情中查看
- **耗时**：一个完整的链路调用所消耗的时间
- **状态**：与 HTTP 状态码对应
- TraceID：调用链的唯一标识。点击显示详情弹窗如“Trace 详情”

4.1.5、其他分析视角

4.1.5.1、接口调用

提供应用接口级别的监控详情。通过丰富的监控分析报表，展示包括数据库监控、异常错误分析、调用来源及调用链查询。

接口是该应用对外提供的各个接口

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「 **应用列表** 」。
3. 在应用列表中选择您想查看的应用，点击「 **应用名称** 」打开新的应用详情链接。
4. 在左侧导航栏中选择「 **接口调用** 」，您可以在应用详情页面切换不同的页签，诸如「 **概览** 」、「 **数据库监控** 」等等查看相应信息。

功能说明

关键指标

以接口为维度，展示各个接口的接口名称、请求数、响应时间、错误数和异常数。

各维度指标详情

数据库监控

和“应用详情–数据库监控”的指标一致，只是统计维度是接口。

异常错误分析

和“应用详情–异常错误分析”的指标一致，只是统计维度是接口。

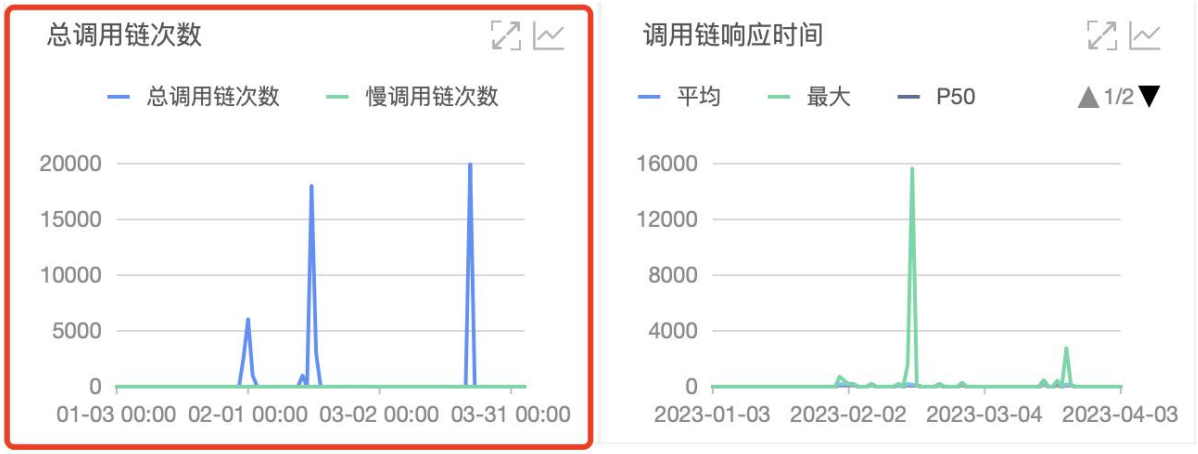
调用来源

和“应用详情–上下游分析”的指标一致，只是统计维度是接口。

调用链查询

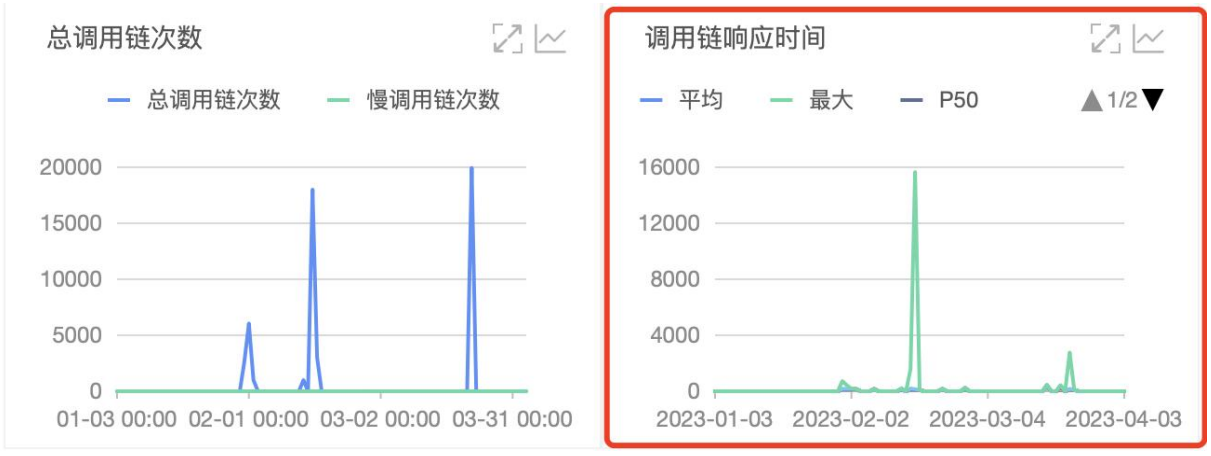
和“应用详情–调用链查询”的指标类似，只是统计维度是接口。

（1）总调用链次数



- **总调用链次数**：指该接口发起的调用链的总次数，包括成功和失败的调用。
- **慢调用链次数**：指调用链路中响应时间超过预设阈值的调用链路次数，默认阈值为500ms，该阈值可以根据实际业务需求在「应用设置」中修改。

（2）调用链响应时间



在筛选时间段内，由该接口发起的调用所属的调用链的耗时

- **平均**：单日响应时间的平均值

- **最大**：单日响应时间的最大值
- **P50**：单日响应时间的中位数（Median），指的是响应时间按从小到大排序后，位于中间的响应时间，它表示一组数据的中间值，也就是 50% 的响应时间低于 P50 值，50% 的响应时间高于 P50 值。
- **P75**：表示 75% 的响应时间低于 P75 值，25% 的响应时间高于 P75 值。
- **P90**：表示 90% 的响应时间低于 P90 值，10% 的响应时间高于 P90 值。
- **P99**：表示 99% 的响应时间低于 P99 值，1% 的响应时间高于 P99 值。

（3）调用链明细表

产生时间	接口名称	所属应用	耗时(ms)	状态	TraceId
2023-03-22 21:34:39.108	/mall/getProductList	mall-mall-server	88.246		110311fb216659edfedd24ffc2399d2a
2023-03-22 21:34:38.411	/mall/getProductList	mall-mall-server	76.199		a620f108842524980088d5b5a20ed9fb
2023-03-22 21:34:32.211	/mall/getProductList	mall-mall-server	312.975		3b2ae9db79f3172200b51d69b9c2797f

显示筛选时间段内，该接口涉及的调用链信息

- **产生时间**：该调用链产生的时间点
- **接口名称**：显示发起 API 调用时的接口名称，完整调用链中涉及的接口信息在 trace 详情中查看

- **所属应用**:显示发起 API 调用时的应用名称,该调用链涉及的其他应用信息在 trace 详情中查看
- **耗时**：一个完整的链路调用所消耗的时间
- **状态**：与 HTTP 状态码对应
- TraceID：调用链的唯一标识。点击显示详情弹窗如“Trace 详情”

4.1.5.2、数据库调用

以关系型数据库为维度，展示当前应用下，各个数据库的监控详情

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「 **应用列表** 」。
3. 在应用列表中选择您想查看的应用，点击「 **应用名称** 」打开新的应用详情链接。
4. 在左侧导航栏中选择「 **数据库调用** 」，您可以在应用详情页面切换不同的页签，诸如「 **概览** 」、「 **数据库监控** 」等等查看相应信息。

功能说明

关键指标

以数据库为维度，展示各个数据库的 IP 端口、数据库名称、请求数、响应时间、错误数和异常数。

各维度指标详情

概览

拓扑图

拓扑图是一种以图形化方式展示应用之间关系的图表，此处展示的是与所选数据库相关的链路信息，可帮助开发人员或运维人员了解应用程序的整体结构和运行状况。详情参见“应用拓扑”。

请求数、响应时间、慢调用次数、性能一览

- **请求数** ： 该 S 数据库在筛选时间段内的 HTTP 入口的总请求数。
- **响应时间** ： 该数据库在筛选时间段内的 HTTP 入口请求的日均响应时间。
- **慢调用次数** ：该数据库在筛选时间段内的 HTTP 入口请求的响应时间超过阈值的次数，默认阈值为 500ms。
- **性能一览** ： 该应用实例在筛选时间段内的 HTTP 入口请求的次数和响应时间。

数据库监控

和“应用详情–数据库监控”的指标一致，只是统计维度是数据库。

异常错误分析

和“应用详情–异常错误分析”的指标一致，只是统计维度是数据库。

调用来源

和“应用详情–上下游分析”的指标一致，只是筛选条件是当前数据库，统计维度是来源接口。

调用链查询

和“应用详情–调用链查询”的指标一致，只是统计维度是数据库。

4.1.5.3、外部调用

指当前应用**向外部应用发起的**调用，以外部应用的 IP 端口为维度进行展示，可用于定位应用外部调用缓慢或出错的问题。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「 **应用列表** 」。
3. 在应用列表中选择您想查看的应用，点击「 **应用名称** 」打开新的应用详情链接。
4. 在左侧导航栏中选择「 **外部调用** 」，您可以在应用详情页面切换不同的页签，诸如「 **概览** 」、「 **上下游分析** 」等等查看相应信息。

功能说明

关键指标

以外部应用的 IP 端口为维度，展示各个外部应用的 IP 端口、请求数、响应时间、错误数和异常数。

支持排序和搜索。

概览

显示当前筛选时间段内对应外部应用调用的请求数、响应时间和错误数。

调用来源

和“应用详情–上下游分析”的指标一致，只是统计条件是当前外部应用，统计维度是来源接口。

提供快捷入口，点击「 **查看详情** 」，会切换到“接口调用–调用链查询”，并选中该来源接口。

4.1.6、应用设置

4.1.4.1、配置说明

应用设置可以让您对具体某个应用进行单独管理，包括 agent 开关、阈值、调用链采样率等等。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」–「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」，您可以在应用设置中修改各项可配置信息。

设置项

设置项	说明
Agent 开关设置	用于控制 Agent 总开关和插件开关，由此控制是否监控该应用，以及对哪些指标集进行监控。
调用链采集设置	用于控制是否采集调用链，采样率是多少，限流阈值是多少。
Java 方法设置	用于设置 Java 拦截方法。
Kafka 设置	用于设置 Kafka 消费类。
URL 采集设置	用于设置 URL 采集。包括 URL 拦截、黑名单、慢请求响应阈值设置、code 定义。
SQL 设置	用于设置慢 SQL 查询阈值。
日志开启设置	用于设置是否关联日志。在已购买云日志服务的情况下，开启关

	联日志可以查看 trace 的日志详情。

设置操作

默认编辑状态，所有修改要点击【保存】按钮后生效。

4.1.4.2、Agent 开关配置

用于控制 Agent 总开关和插件开关，由此控制是否监控该应用，以及对哪些指标集进行监控。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」－「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」－「**Agent 开关设置**」。

功能说明

针对通过 java—agent 接入的应用可进行如下配置：

Agent 总开关

支持控制该应用的 agent 是否生效。默认打开，如关闭，系统将不会监控您的应用，也不会产生费用。

插件开关

也支持单独针对 agent 插件设置开关，用以控制采集哪些指标集信息。需注意，插件开关的修改，需要您手动重启应用后才会生效。

4.1.4.3、调用链采集设置

支持控制是否采集调用链以及设置调用链采样率。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」-「**调用链采集设置**」。

功能说明

是否采集调用链

默认采集，如果关闭开关，将停止调用链采集上报。

采样率设置

支持设置公共采样率，用来控制除慢调用、错误调用之外的采样率。慢调用、错误调用系统默认全采，不支持修改。

采样率设置说明：

- 采样率默认为 10。
- 调大采样率可能会消耗额外的系统资源，有最大每秒采集条数限制。
- 每秒采集 100 条的情况下，开销在 300m 内存左右。如需调整每秒最大采集数，可修改限流阈值。

阈值设置

Agent 端每秒最大可处理请求数，默认 100 条。大于该阈值的调用链，不被收集。

4.1.4.4、URL 采集设置

支持 URL 采集相关的自定义配置。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」-「**URL 采集设置**」。

功能说明

URL 拦截设置

用于设置拦截 header/url 参数。

慢请求响应阈值设置

默认 500ms。当接口响应时间大于该阈值的时候，该接口会被标记为慢调用，下图展示数据会随着配置改变。

阈错误状态码定义

自定义错误状态码，影响错误数展示。

其他采集设置

当前支持设置采集黑名单，提供下图四种匹配方式。

4.1.4.5、Java 方法设置

支持设置 Java 方法相关信息。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」-「**Java 方法设置**」。

功能说明

JAVA 拦截方法

通过拦截方法配置可以影响业务方法的采集，如被拦截，在其他基础监控—Java 方法和调用链查询中就会显示相关的方法信息。

参数名称	说明	样例
拦截类名	配置需要拦截的包名+类名	com.apm.ApmClass
拦截方法名	配置该类下需要拦截的方法名 ● 如存在多个方法需要拦截，则用半角逗号,间隔	apmMethod1,apmMethod2

4.1.4.6、SQL 设置

支持控制 SQL 调用相关信息。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」—「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。
4. 在左侧导航栏中选择「**应用设置**」—「SQL 设置」。

功能说明

慢 SQL 查询阈值

默认 500ms。影响慢 sql 的判断，当 SQL 查询的耗时大于该阈值的时候，该查询会被标记为慢 SQL，下图展示数据会随着配置改变。

4.1.4.6、日志开启设置

支持控制是否开启日志。

功能入口

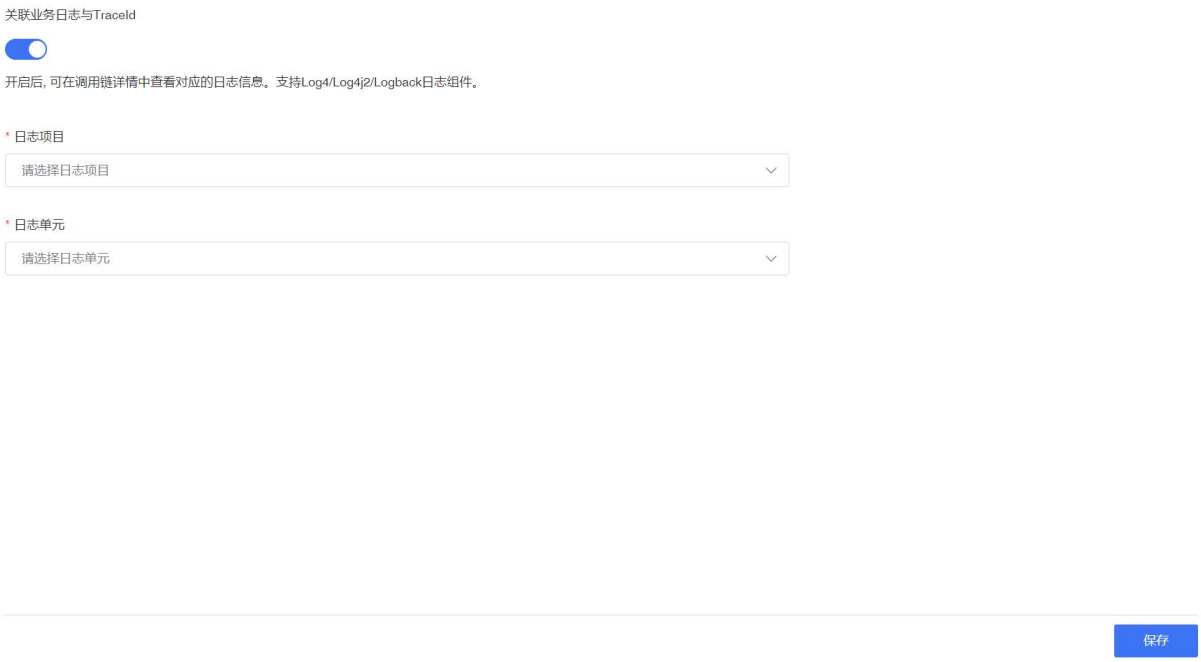
1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**应用监控**」-「**应用列表**」。
3. 在应用列表中选择您想查看的应用，点击「**应用名称**」打开新的应用详情链接。

4. 在左侧导航栏中选择「**应用设置**」-「**日志开启设置**」。

前提条件

需要先开通云日志服务产品，如果您已开通，则可通过配置关联查看调用链路的日志详情。

功能说明



关联业务日志与 TraceId

设置关联云日志服务的具体日志项目和单元，确认日志存储路径，以便查看。开启关联后，在调用链详情会出现查看日志按钮，点击可跳转云日志服务查看具体内容。

4.2、agent 管理

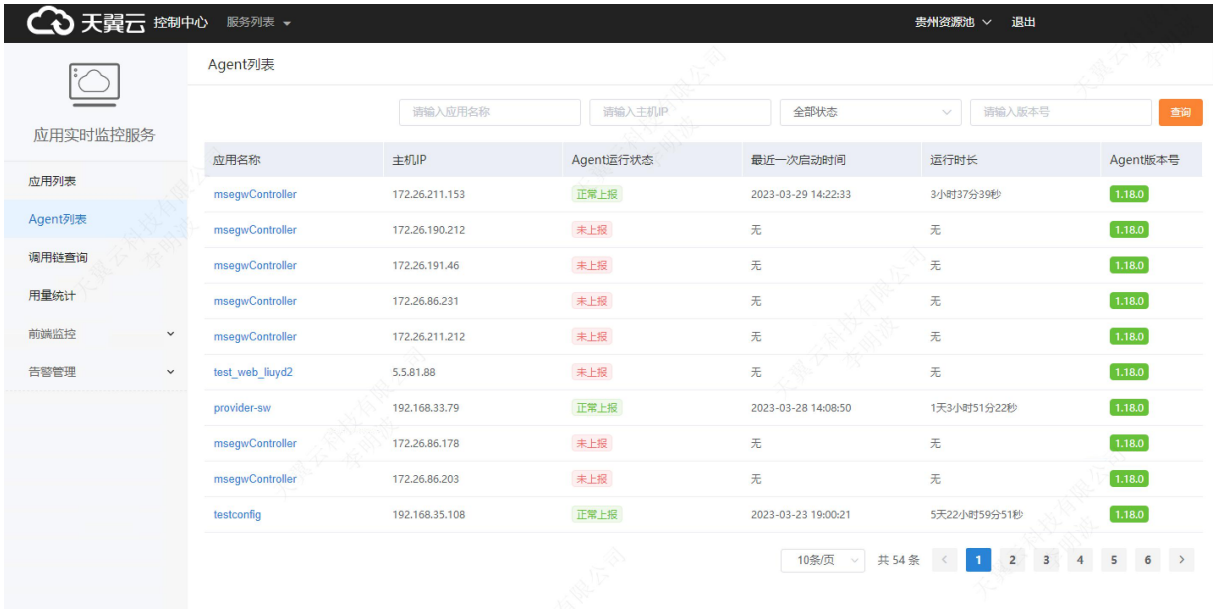
Agent 列表用于管理您安装的所有探针，包括“正常上报“和“未上报”的探针，显示各个探针的关键指标信息，提供查看应用详情的快捷入口和升级探针的功能入口。

功能入口

（1）选择目标资源池，并登录 APM 组件控制台

（2）在左侧导航栏中选择「Agent 列表」

功能说明



（1）关键信息展示与查询

应用名称：显示该探针（Agent）接入的应用的名称

主机 IP：显示该探针（Agent）接入的应用实例所在的主机 IP

Agent 运行状态：根据 APM 是否获取到探针（Agent）采集数据来判断状态为“正常上报/未上报”

最近一次启动时间：正常上报的探针（Agent），最近一次启动时间

运行时长：探针（Agent）最近一次启动后运行的时长

Agent 版本号：显示该探针（Agent）当前版本号

（2）查看应用详情

点击「应用名称」，可以查看具体的应用信息及监控指标明细等等，包括：

应用总览

- 应用详情

接口调用

SQL 调用

NoSQL 调用

外部调用

MQ 监控

调用链查询

（3）升级探针

如果当前探针（Agent）的版本不是最新，那么可在操作列点击「升级」按钮，根据升级指引进行探针（Agent）升级。

4.3、链路信息

4.3.1、调用链查询

展示当前租户下所有调用链路信息。您可以根据多个筛选条件租户查询您想看的调用链，可以点击「TraceID」查看具体的调用链详情。

功能入口

（1）查看该租户下所有调用链路

选择目标资源池，并登录 APM 组件控制台

在左侧导航栏中选择**「调用链查询」**

（2）查看某个应用/agent 相关的调用链

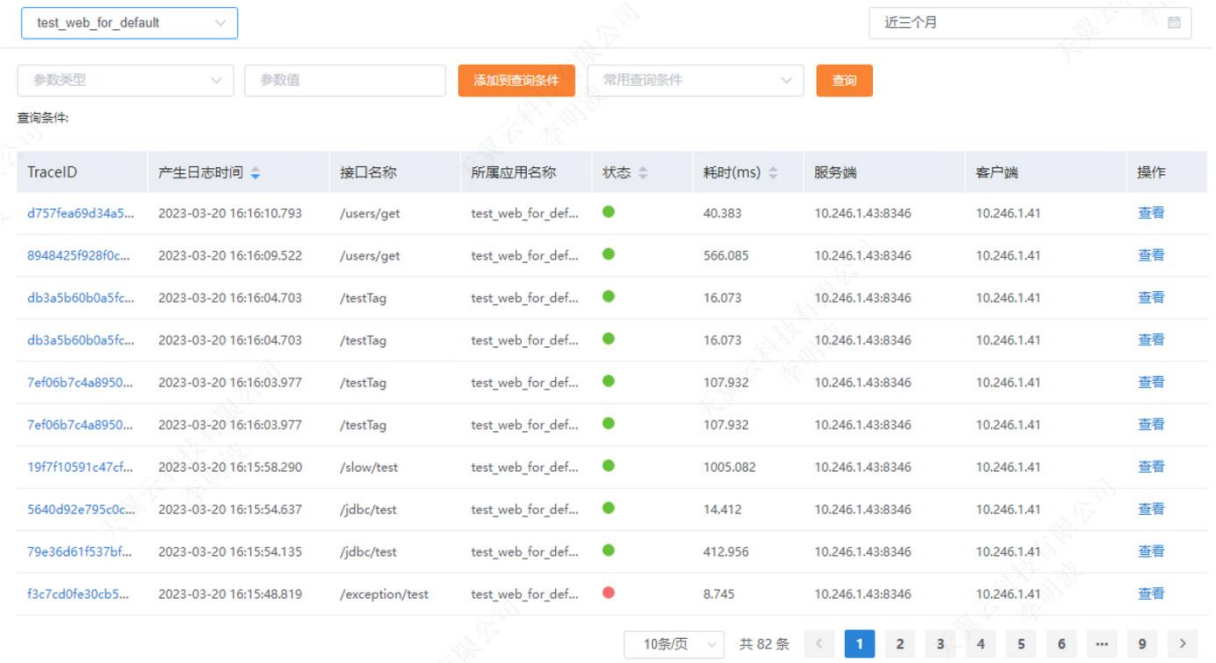
选择目标资源池，并登录 APM 组件控制台

在左侧导航栏中选择**「应用列表」**

在应用列表中选择您想查看的应用，点击**「应用名称」**打开新的应用详情链接

在左侧导航栏中选择**「调用链查询」**查看该应用实例/接口的调用链信息

功能说明



（1）关键信息展示与查询

TraceID：调用链的唯一标识。

产生日志时间：该调用链产生日志的时间点

接口名称：显示发起 API 调用时的接口名称，完整调用链中涉及的接口信息在 trace 详情中查看

所属应用名称：显示当前应用实例所属应用的名称，该调用链涉及的其他应用信息在 trace 详情中查看

状态：显示正常/异常

耗时：一个完整的链路调用所消耗的时间

服务端：调用链中第一段的被调用的应用的 IP 端口

客户端：调用链中第一段的发起调用的应用的 IP 地址

操作：「查看」，点击显示详情弹窗如“Trace 详情”

（2）具体链路详情

详见下章“Trace 详情”

4.3.2、Trace 详情

显示 Trace 的详细信息，包括调用栈及具体每个方法的耗时、日志详情等等。

功能入口

（1）查看该租户下所有调用链路中某个的链路详情

选择目标资源池，并登录 APM 组件控制台

在左侧导航栏中选择**「调用链查询」**

点击**「TraceID」**，打开 Trace 详情弹窗

（2）查看应用/agent 相关的调用链路中某个的链路详情

选择目标资源池，并登录 APM 组件控制台

在左侧导航栏中选择**「应用列表」**

在应用列表中选择您想查看的应用，点击**「应用名称」**打开新的应用详情链接

在左侧导航栏中选择**「调用链查询」**查看该应用实例/接口的调用链信息

点击**「TraceID」**，打开 Trace 详情弹窗

（3）查看应用实例/接口相关的调用链路中某个的链路详情

选择目标资源池，并登录 APM 组件控制台

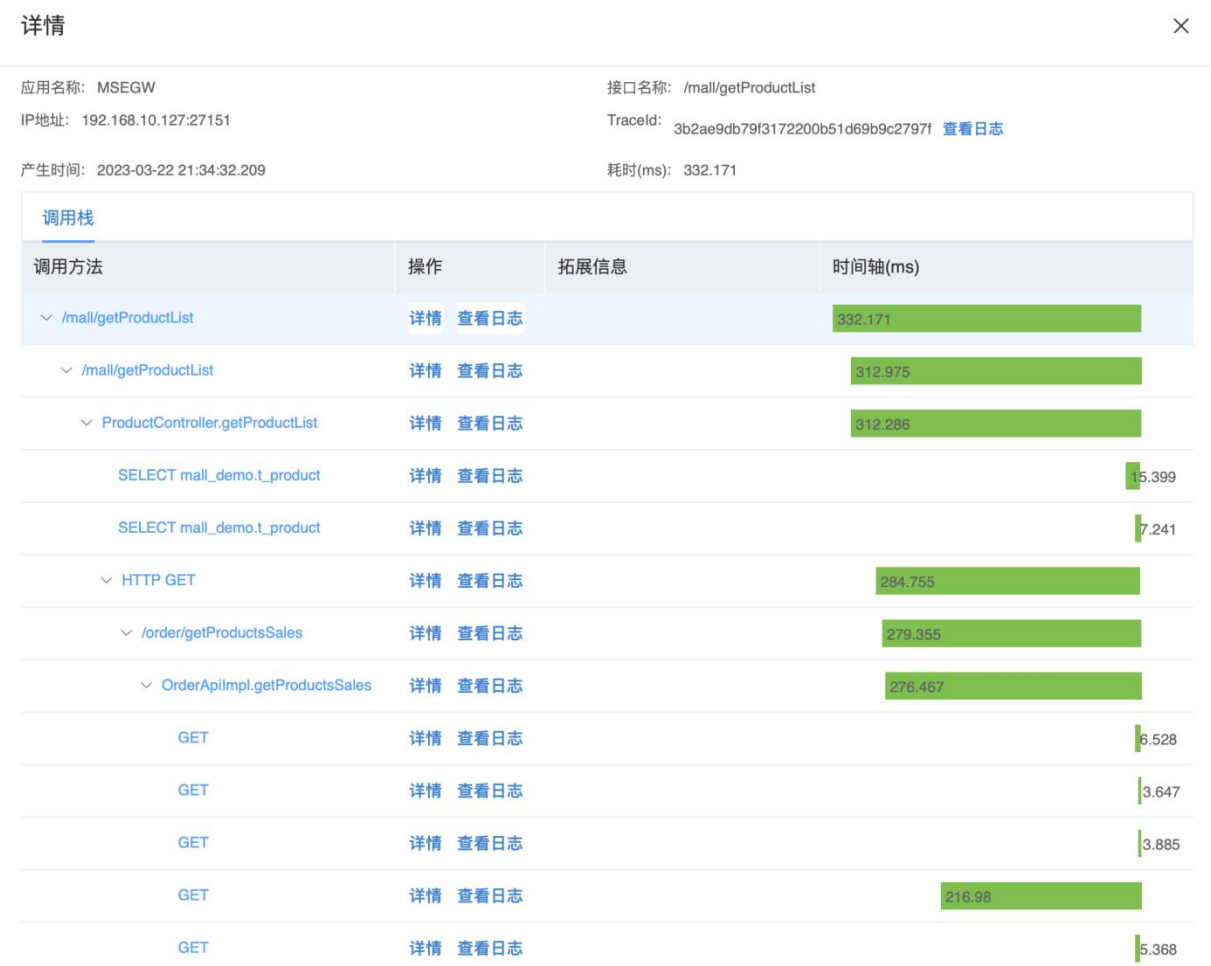
在左侧导航栏中选择**「应用列表」**

在应用列表中选择您想查看的应用，点击**「应用名称」**打开新的应用详情链接

在左侧导航栏中选择**「应用详情」**或**「接口调用」**，您可以在应用详情页面切换至**「调用链查询」**页签，在左侧关键指标中**选择不同的应用实例/接口**，可查看该应用实例/接口相应的概览信息。

点击**「TraceID」**，打开 Trace 详情弹窗

功能说明



（1）基础信息

应用名称：显示当前选中的“调用方法”所属的应用的名称

接口名称：显示当前选中的“调用方法”所属的接口的名称

IP 地址：显示当前选中的“调用方法”所属的应用实例的 IP 地址

TraceID：显示 TraceID，当前调用链的唯一标识。点击「查看日志」打开“云日志服务-检索分析”页面

产生时间：发起调用的时间点

耗时：显示当前选中的「调用方法」从发起调用到返回结果的时间

（2）调用栈

以调用树的方式显示具体的调用信息

调用方法：显示调用方法名称

操作：

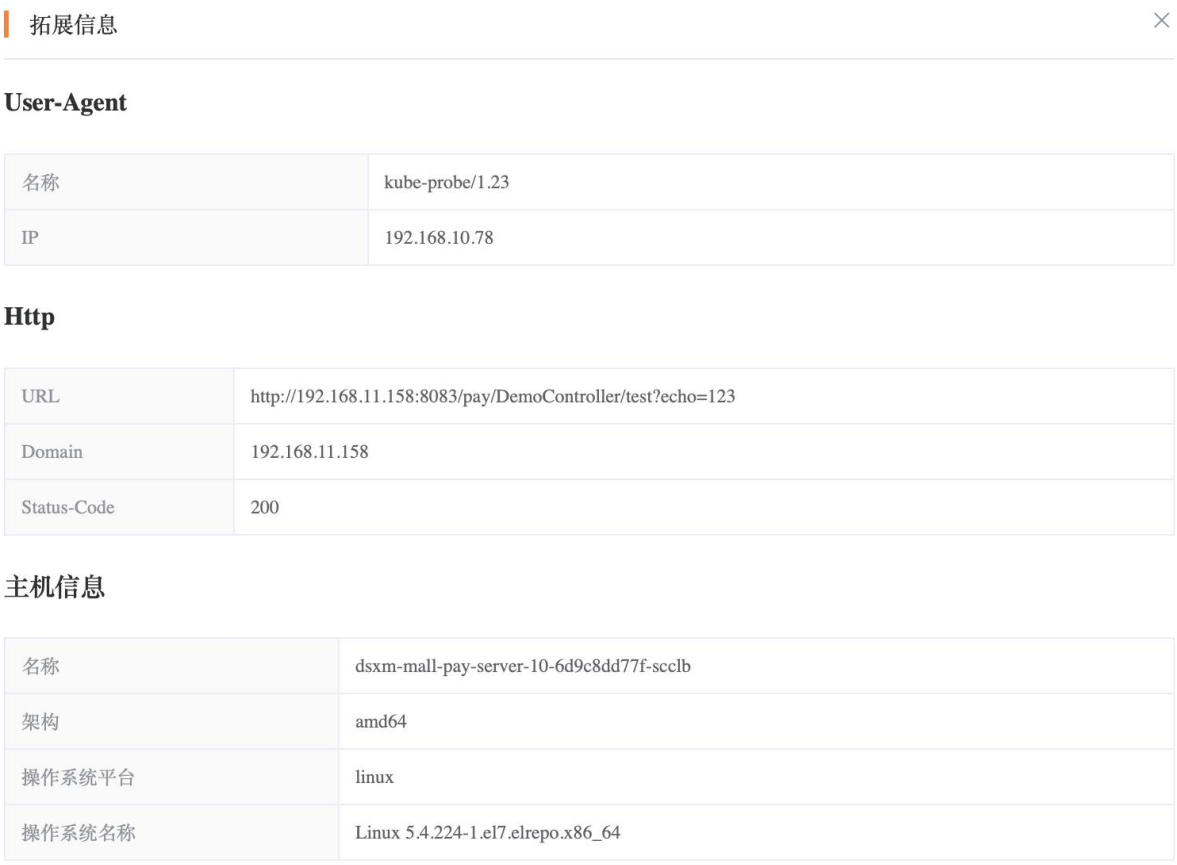
- **详情：**点击详情，打开该调用方法的拓展信息弹窗。显示 Agent、http、主机、Process、Thread、DB、Messaging、目标服务信息。
- **查看日志：**点击打开“云日志服务-检索分析”页

拓展信息：当该调用存在异常/错误信息时，会在此处显示异常堆栈信息

时间轴：以时间轴的形式显示各个调用方法的耗时

（3）拓展信息

点击「详情」按钮，打开弹窗如下



显示该方法其他相关信息，包括

User-Agent

- 名称：接入该应用的探针名称
- IP：该探针接入的应用实例的主机 IP

Http

- URL（Uniform Resource Locator）：统一资源定位符，是一种用于标识互联网上资源的地址，包括协议类型、主机名、端口号、路径和查询参数等信息。
- Domain（域名）：是指互联网中的机器和服务的名称，类似于电话号码的概念。域名由多个部分组成，按照从右到左的顺序，依次表示顶级域名、二级域名、三级域名等。
- Status-Code（状态码）：是指 HTTP 服务器返回给客户端的响应状态码，用于表示请求的处理结果。
  - 5xx：服务器异常，服务器在处理请求的过程中发生错误
  - 4xx：客户端异常，请求包含语法错误或无法完成请求
  - 3xx：重定向问题，需要进一步操作
  - 2xx：成功，服务器成功接收请求并执行
  - 200：请求成功

主机信息

- 名称：主机的名称，通常是由用户指定的一个字符串，用于标识主机的身份。
- 架构：指主机的硬件架构，如 x86、x86_64、ARM 等

- 操作系统平台：指主机所使用的操作系统的平台，如 Windows、Linux、macOS 等。
- 操作系统名称：指主机所使用的具体操作系统的名称和版本号，如 Windows 10、Ubuntu 20.04 LTS、macOS Big Sur 等。

Process（进程）是指正在运行的一个程序的实例

- Pid（Process ID）：进程的唯一标识符，是一个由操作系统分配的整数，用于标识系统中的不同进程，每个进程都有一个不同的 PID。
- Command-Line（命令行）：是指启动进程时指定的命令行参数，包括程序的路径、参数等信息。它是一个字符串，可以通过操作系统提供的接口获取到。

Thread（线程）是指进程中的一个执行单元

- ID（Thread ID）：线程的唯一标识符，是一个由操作系统分配的整数，用于标识系统中的不同线程。每个线程都有一个不同的 ID。
- Name（线程名）：是指用户指定的线程名称，用于标识线程的身份。线程名称可以方便用户在程序中进行调试和监控。

DB（数据库）

- Connection-String（连接字符串）：是指连接到数据库所需要的信息，包括主机名、端口号、用户名、密码等。它是一个字符串，用于在程序中建立到数据库的连接。
- Operation（操作）：是指对数据库执行的操作，包括查询、插入、更新、删除等。它是一个字符串，用于表示当前执行的数据库操作。

- Instance（实例）：是指数据库的实例名，用于标识不同的数据库实例。它是一个字符串，通常是在建立数据库连接时指定的。
- Type（类型）：是指数据库的类型，包括关系型数据库、非关系型数据库等。它是一个字符串，用于表示当前使用的数据库类型。
- Statement（语句）：是指在数据库中执行的 SQL 语句，包括查询语句、插入语句、更新语句、删除语句等。它是一个字符串，用于表示当前执行的 SQL 语句。
- User（用户）：是指对数据库进行操作的用户，包括数据库管理员、应用程序用户等。它是一个字符串，用于表示当前执行操作的用户。

Messaging（消息传递）是指在分布式系统中，通过消息传递实现不同组件之间的通信。

- System（消息系统）：是指消息传递所使用的消息系统，如 Apache Kafka、RabbitMQ、ActiveMQ 等。它是一个软件系统，用于实现异步消息传递。
- Operation（操作）：是指对消息执行的操作，包括发送、接收、确认等。它是一个字符串，用于表示当前执行的操作。
- Destination-Kind（目标类型）：是指消息的目标类型，包括队列（Queue）和主题（Topic）两种。队列用于点对点的消息传递，主题用于发布-订阅模式的消息传递。它是一个字符串，用于表示当前消息的目的地类型。

目标服务（Target Service）是指客户端需要访问的远程服务。

- 名称（Name）：是指目标服务的名称，用于唯一标识服务。它是一个字符串，通常由服务提供方指定，并在服务注册中心中注册。

- 类型（Type）：是指目标服务的类型，用于表示服务的功能类别。例如，Web 服务、消息队列服务、数据库服务等。它是一个字符串，通常由服务提供方指定。
- 实例（Instance）：是指目标服务的实例，用于标识不同的服务实例。在分布式系统中，通常会有多个服务实例提供相同的服务。它是一个字符串，通常由服务注册中心分配。

4.4、系统管理

4.4.1、用量统计

展示当前租户下的探针用量情况，包括实例数、探针时、Span 存储量。

功能入口

1. 选择目标资源池，并登录 APM 组件控制台。
2. 在左侧导航栏中选择「**系统管理**」-「**用量统计**」。

功能说明

关键指标

以 Java-agent 方式接入

- APM Agent*Hour：筛选时间段内，已消耗的探针时

以其他方式接入

- Span 上报量：筛选时间段内，上报过的 Span 总数
- Span 存储量：筛选时间段内，存储过的 Span 总数。

用量趋势

根据应用接入方式不同进行区分。以 Java-agent 方式接入的展示 APM Agent*Hour 的趋势图；以其他方式接入展示 Span 上报量趋势图和 Span 存储量趋势图。

4.5、告警管理

4.5.1、告警规则

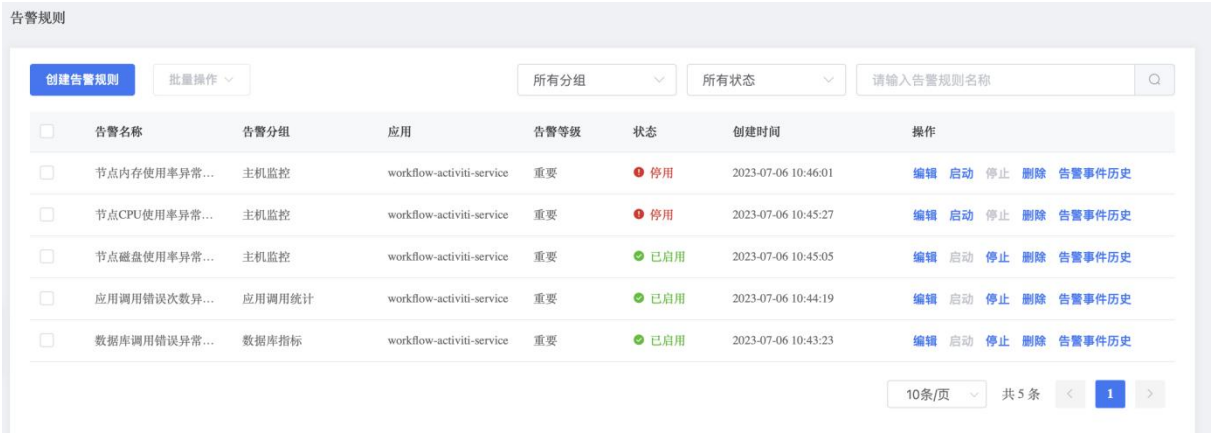
展示当前租户下所有告警规则信息，支持增删改查、起停和查看告警事件历史

功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

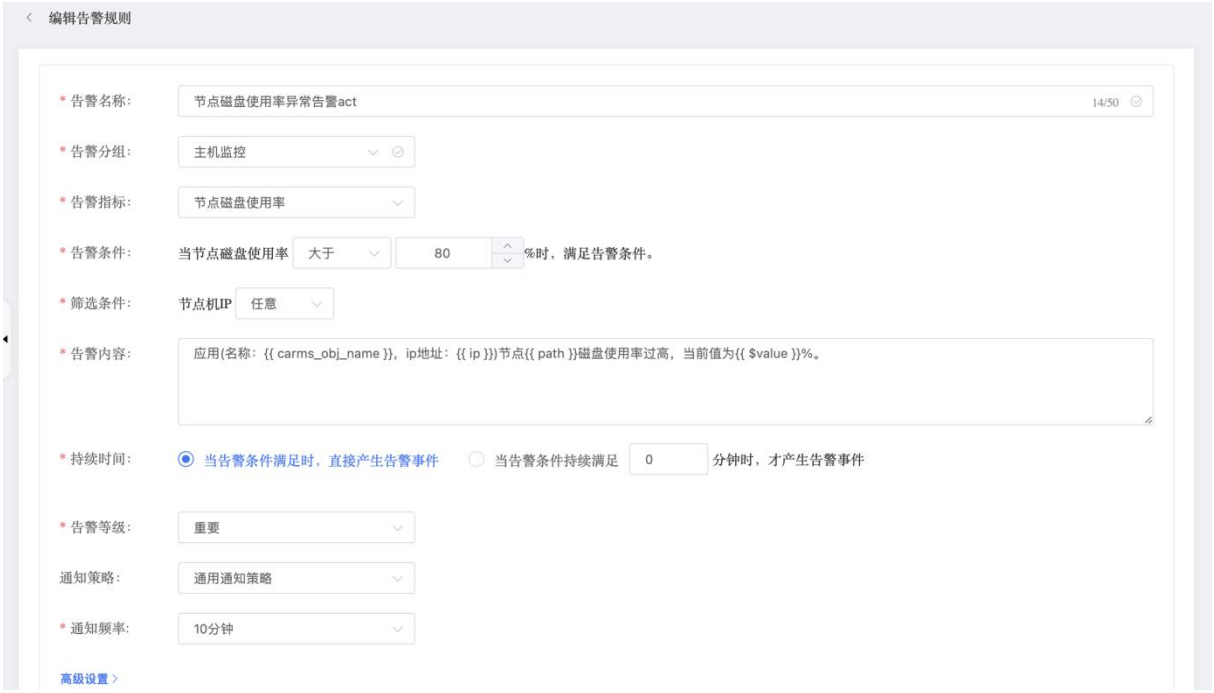
（2）在左侧导航栏中选择**「告警管理」-「告警规则」**。

功能说明



展示当前租户下所有告警规则信息，支持基础的增删改查、起停、查看告警事件历史操作。

（1）创建/编辑告警规则



告警名称：您可自定义告警名称

应用：展示应用列表中，所有已接入应用，支持多选

○ 新增应用自动在此告警规则中追加：开启后，新增应用则自动使用该告警规则

告警详情：包含告警分组、告警指标、告警条件、筛选条件、告警内容

告警分组	告警指标	告警条件
主机监控	节点 cpu 使用率	该指标大于、大于等于、小于、小于等于、等于某个百分比时生效
	节点内存使用率	
	节点磁盘使用率	
	JVM 实例数	
应用调用统计	调用错误次数	某个时间段内，该指标大于、大于等于、小于、小于
	调用错误率	

	调用次数	等于、等于某个数值/百分比时生效
	调用平均响应时间	
HTTP 状态码异常	状态码 5xx 次数	某个时间段内，该指标大于、大于等于、小于、小于等于、等于某个数值/百分比时生效
	状态码 4xx 次数	
异常接口调用	调用次数	
	调用响应时间	
应用调用类型统计	应用提供服务调用总次数	某个时间段内，该指标大于、大于等于、小于、小于等于、等于某个数值/百分比时生效
	应用提供服务调用错误率	
	应用依赖服务调用错误率	
	应用依赖服务调用总次数	
	应用提供服务调用错误次数	
	应用依赖服务调用错误次数	
	应用提供服务调用响应时间	
	应用依赖服务调用响应时间	
数据库指标	数据库调用次数	某个时间段内，该指标大于、大于等于、小于、小于等于、等于某个数值/百分比时生效
	数据库调用响应时间	
	数据库调用错误次数	

JVM 监控	JVM_FullGC 次数瞬时值	该指标大于、大于等于、小于、小于等于、等于某个百分比时生效
	JVM_FullGC 耗时瞬时值	
	JVM_YoungGC 次数瞬时值	
	JVM_YoungGC 耗时瞬时值	
	JVM 堆内使用内存量	
	JVM 非堆使用内存量	
	JVM 死锁线程数	
	JVM 阻塞线程数	
	JVM 等待线程数	
	JVM 可运行线程数	
	JVM 线程总数	

持续时间：支持设置延迟告警

告警等级：一般、次要、重要、等级

通知策略：可以选择通知策略菜单里设置的策略

标签：自定义标签，用于筛选

（2）启动/停止

对于暂时不用的告警策略，您可以操作停止

（3）告警事件历史

点击跳转告警事件历史菜单，显示该告警规则触发的实际告警记录

4.5.2、告警发送历史

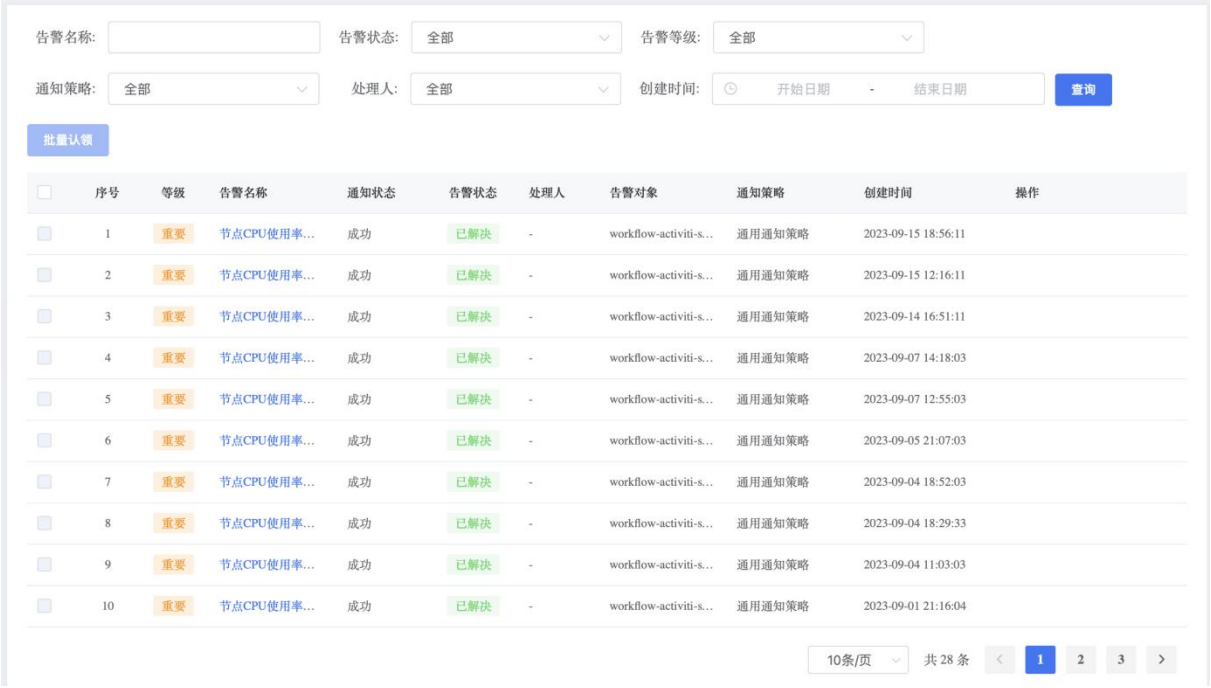
展示当前租户下所有发送的告警历史，支持筛选和对告警信息进行操作。

功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

（2）在左侧导航栏中选择**「告警管理」-「告警发送历史」**。

功能说明



展示当前租户下所有告警发送信息。

（1）支持对告警名称、告警状态、告警等级、通知策略和创建时间进行筛选。

- **告警名称**：显示告警规则名称

- **告警状态**：显示事件当前状态是待认领、已解决、处理中
- **告警等级**：显示告警的重要层级分布是一般、次要、重要、紧急
- **通知策略**：告警对应的通知策略
- **处理人**：告警的最新解决/认领人
- **创建时间**：告警产生的时间

（2）支持认领、解决、指定处理人操作

- **认领**：当告警处于待认领状态时，可主动领取当前告警
- **解决**：当告警处于待认领/处理中状态时，点击解决可更新告警状态为已解决
- **指定处理人**：指定他人处理该告警

（3）支持查看告警详情

- **详情**：显示告警发送的基础信息如告警对象、处理人、解决方案。

- **事件**：显示触发告警的事件名称、状态和触发时间，点击可查看详情。

- **活动**：显示包括认领、取消认领、指派处理人、告警关闭等在内的各种活动信息，支持筛选。

4.5.3、告警事件历史

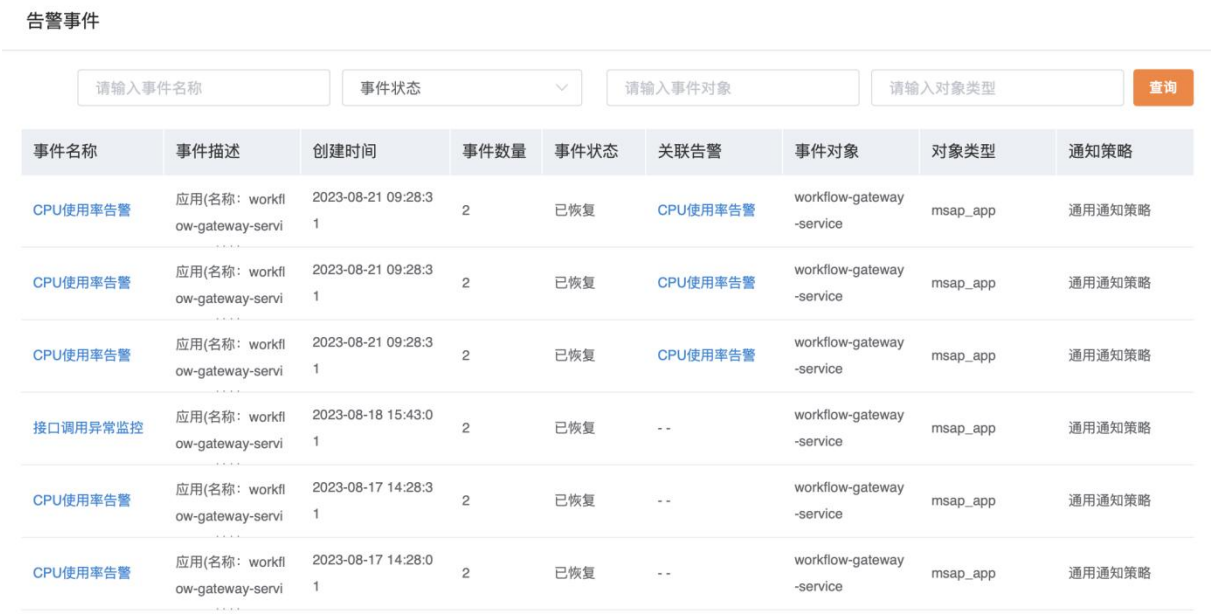
展示当前租户下所有告警事件历史，支持筛选及查看详情。

功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

（2）在左侧导航栏中选择**「告警管理」-「告警事件历史」**。

功能说明

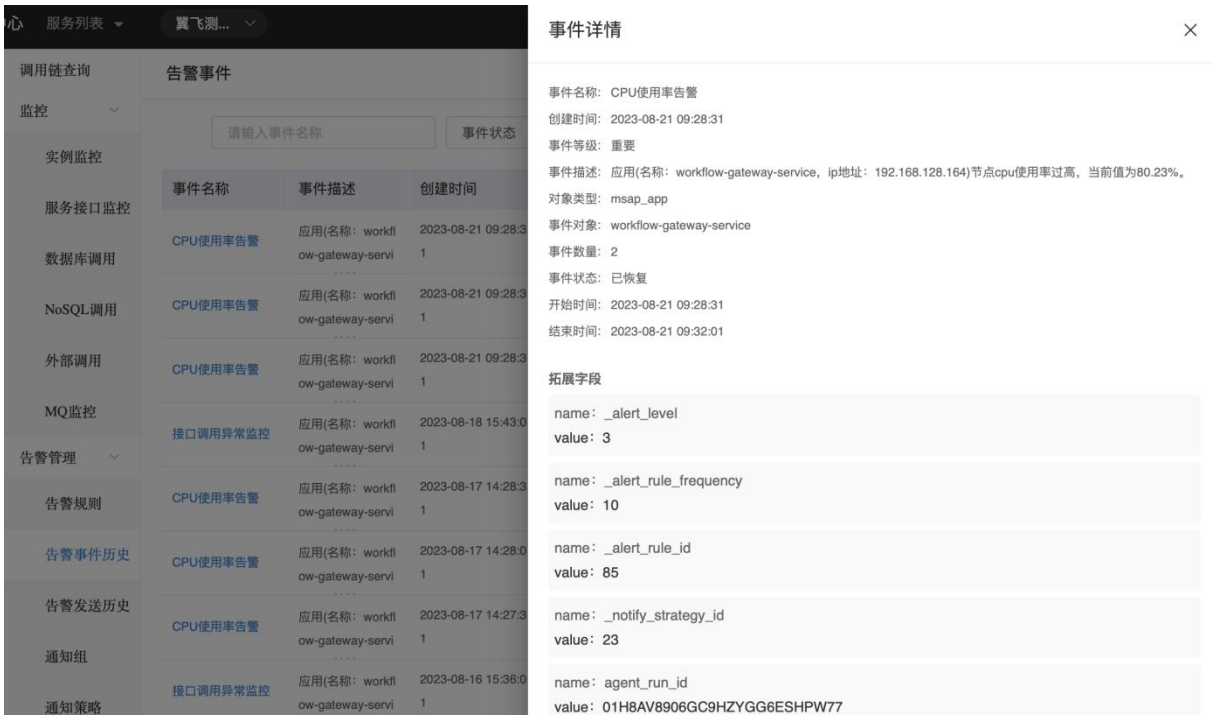


展示当前租户下所有告警事件信息。

（1）支持对事件名称、事件状态、事件对象和对象类型进行筛选。

- **事件名称**：显示告警规则名称
- **事件状态**：显示事件当前状态是告警中、已恢复、静默
- **事件对象**：监控对象，比如应用名称、集群名称等
- **对象类型**：告警事件对象的类型

（2）支持查看事件详情



（3）支持查看告警详情

- **详情**：显示告警发送的基础信息如告警对象、处理人、解决方案。

- **事件**：显示触发告警的事件名称、状态和触发时间，点击可查看详情。

- **活动**：显示包括认领、取消认领、指派处理人、告警关闭等在内的各种活动信息，支持筛选。

4.5.4、通知对象

告警支持通过短信、邮箱、翼连等方式将告警信息通知给对应接收人，我们可以在通知组中设置接收人信息。

功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

（2）在左侧导航栏中选择**「告警管理」-「通知组」**。

功能说明

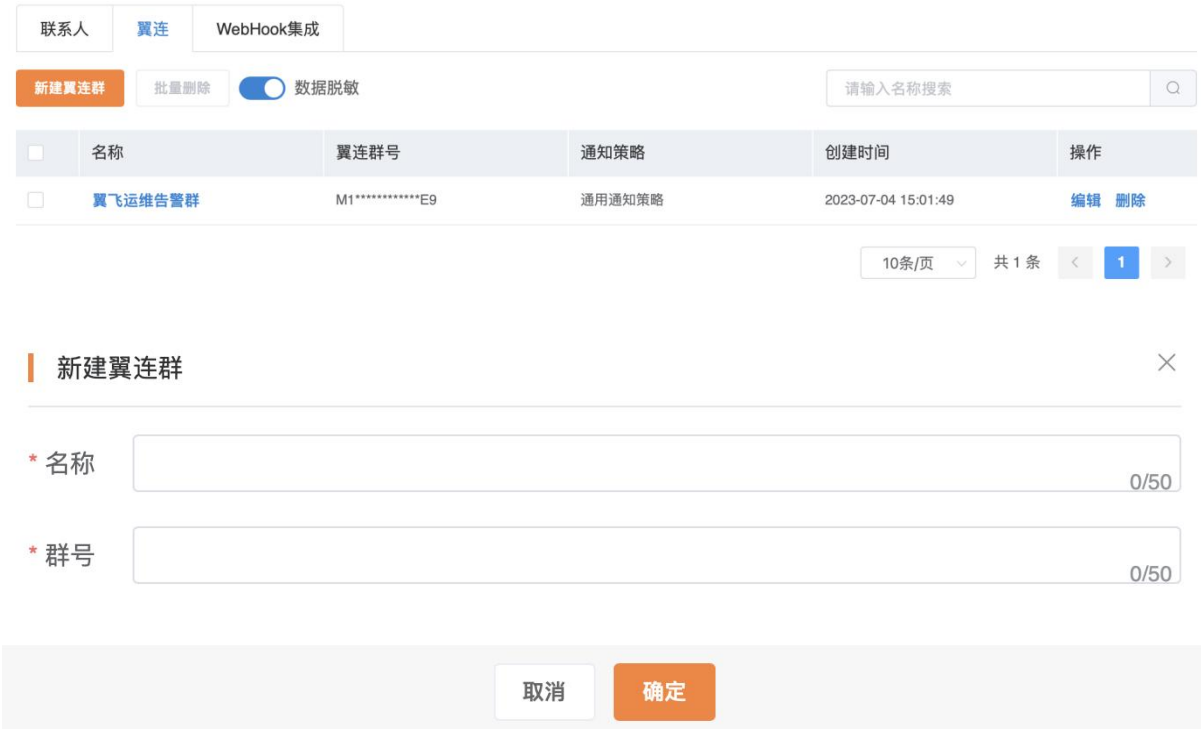
（1）联系人（短信/邮箱）

如需要通过短信/邮箱进行告警，可在此处维护联系人信息。支持对联系人信息进行增删改查，需要录入基础信息。

同时也支持对联系人进行分组，对联系人组进行增删改查。

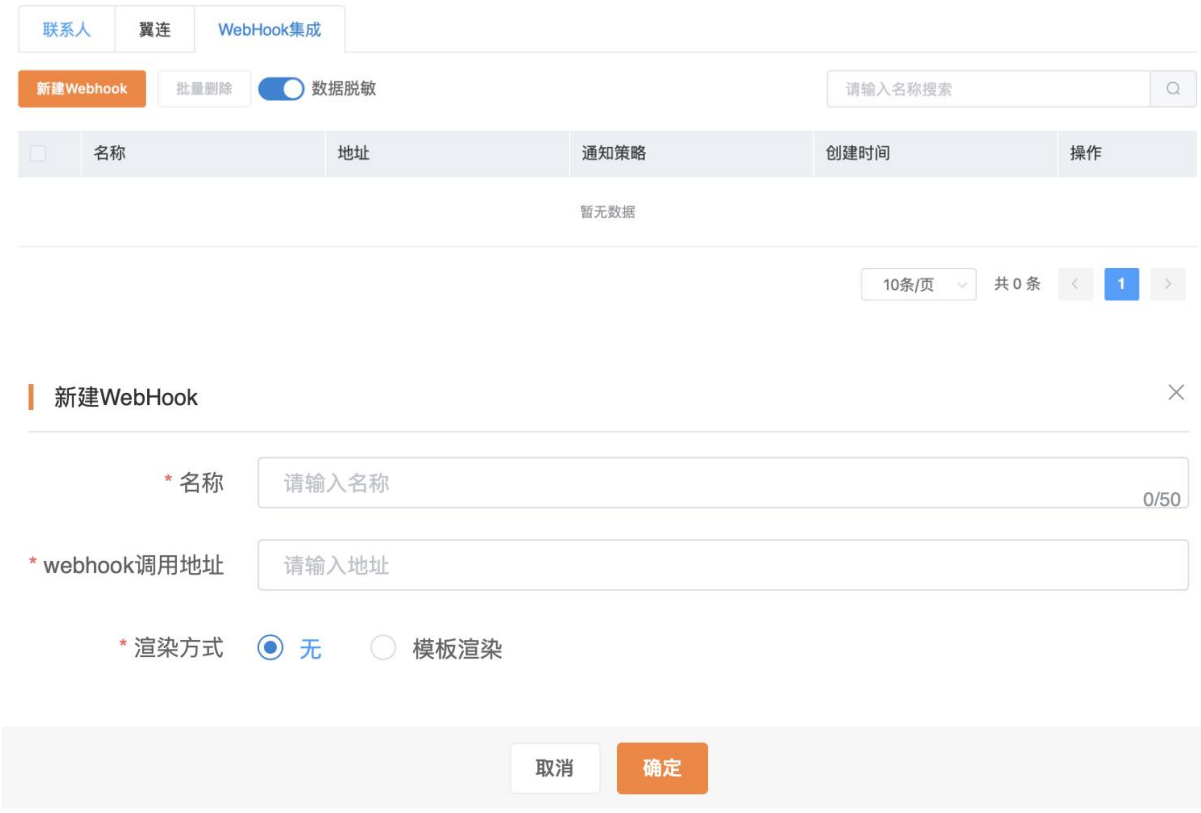
（2）翼连

支持增删改查翼连群信息，将某个翼连群作为一个“联系人组”。



（3）WebHook 集成

支持以 WebHook 的方式对第三方通知对象（钉钉、企业微信、飞书等）发送告警信息。支持增删改查 WebHook 信息。



4.5.5、通知策略

创建告警通知策略，当告警触发时，只要不符合静默策略，就会依照通知策略在对应渠道发送告警信息给对应的通知对象。

功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

（2）在左侧导航栏中选择**「告警管理」-「通知策略」**。

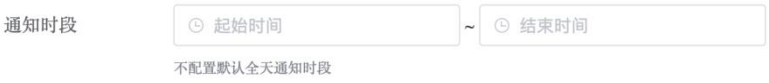
功能说明

支持增删改查告警通知策略信息

- **通知对象**：可从通知组进行选择。

- **通知模板**：可跟进不同通知渠道（邮件、短信、翼连）设置不同的通知模板。

- **通知时段**：可以设置通知时段，默认是告警触发时通知。



4.5.6、静默策略

如果满足静默策略则不会触发告警通知，您可以通过静默策略对告警进行收敛，避免同一时间出现大量重复告警或关联告警。

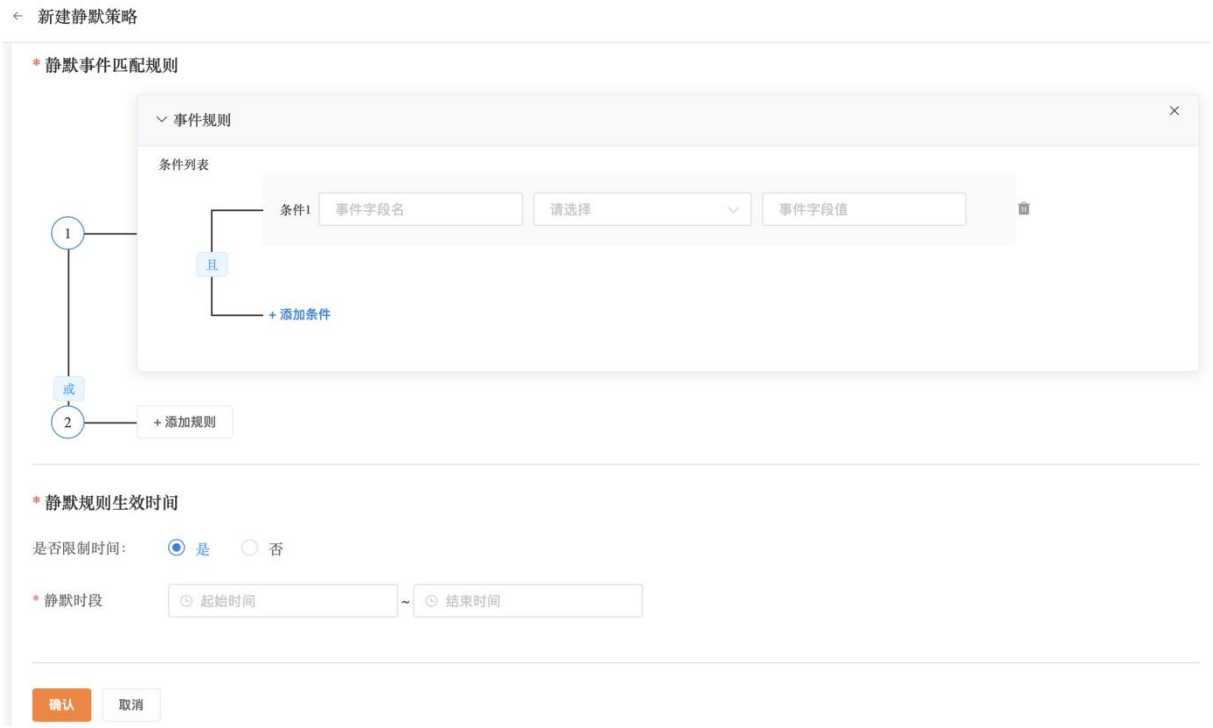
功能入口

（1）选择目标资源池，并登录 APM 组件控制台。

（2）在左侧导航栏中选择**「告警管理」-「静默策略」**。

功能说明

支持增删改查静默策略。



- **事件规则**：支持通过且或关系组合创建事件规则，使得在满足当前事件规则时，触发静默策略。
- **静默时段**：设置静默规则的生效时间段。

五、最佳实践

5.1、使用调用链采样策略

本文介绍 APM 支持的调用链采样模式，帮助您以较低成本获取想要的调用链数据。

对于绝大多数的分布式系统，不是每一条调用链都值得被可观测平台记录，因为其中包含大量重复的、低关注度的信息。因此需要引入采样技术降低整体可观测成本，并过滤对用户没有帮助的噪音。

调用链采样的基本原则是优先记录您最关心、最有可能访问的调用链。APM 提供固定采样率这种采样模式。

固定采样率

固定比例采样就是根据 TraceId 顺序号记录一定比例的调用链数据。例如，固定比例为 10%，则每 10 条调用链数据记录 1 条。固定比例采样不会导致调用链数据本身不完整，要么保留整条链路数据，要么丢弃整条链路数据。

设置固定比例采样的操作步骤如下：

1. 登录 APM 控制台，在左侧导航栏选择**应用监控 > 应用列表**。
2. 在**应用列表**页面单击目标应用名称。
3. 在左侧导航栏中单击**应用设置**，并在右侧页面单击调用链采集设置页签。

4. 在**采样率设置**区域设置采样率。在**采样率设置**字段输入百分比的数字部分，例如输入 1 代表采样 1%，如下图所示。

5.2、诊断服务端报错问题

诊断服务端报错问题

（1）问题描述

网页抛错，尤其是 5xx 错误是互联网应用最常见的问题之一。5xx 错误通常发生于服务端。服务端是业务逻辑最复杂，也是整条网络请求链路中最容易出错、出了错之后最难诊断原因的地方。运维工程师或研发工程师往往需要登录机器查看日志来定位问题。

对于逻辑不太复杂、上线时间不长的应用来说，登录机器查看日志的方式能够解决大部分网站抛错的问题。但在以下场景中，传统的问题诊断方式往往没有用武之地。

1.在一个分布式应用集群中，需知道某一类错误的发生时间和频率。

2.某系统已运行了很长时间，但是不想关心遗留的异常，只想知道今天和昨天相比、发布后和发布前相比多了哪些异常。

3.查看一个异常对应的 Web 请求和相关参数。

4.客服人员提供了一个用户下单失败的订单号，分析该用户下单失败的原因。

（2）解决方案

为应用安装 APM 探针后，即可在不改动应用代码的情况下，利用应用性能监控 APM 的异常自动捕捉、收集、统计和溯源等能力，全面掌握应用的各种错误信息。

步骤一：安装 APM 探针

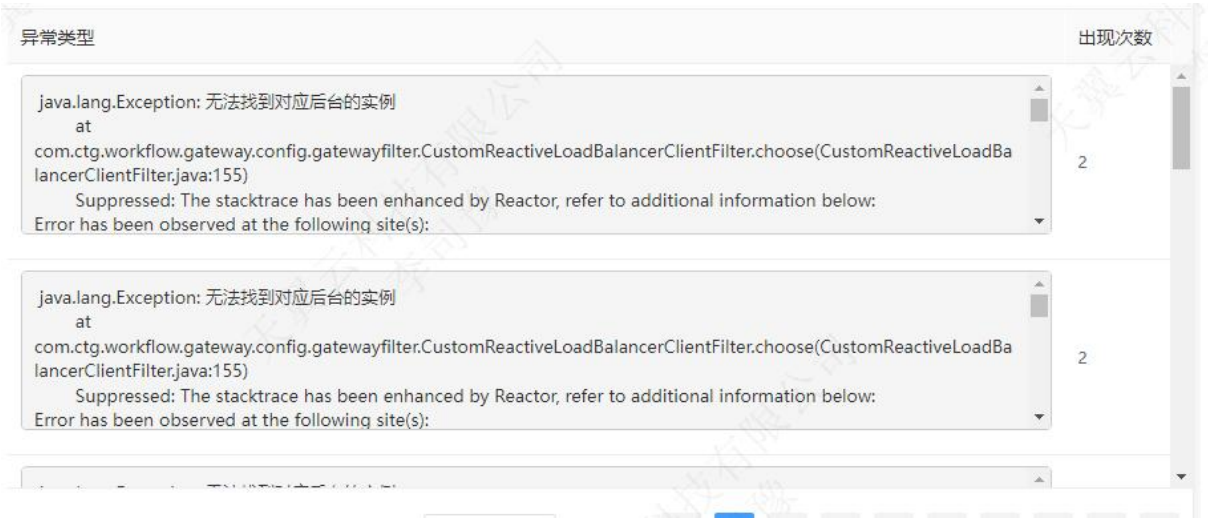
为应用安装 APM 探针后，才能对应用进行全方位监控。请根据实际需求选择一种方式来安装探针。

步骤二：查看关于应用异常的统计信息

为应用安装 APM 探针后，APM 会收集和展示选定时间内应用的总请求量、平均响应时间、错误数、实时实例数、FullGC 次数、慢 SQL 次数、异常次数和慢调用次数，以及这些指标和上一天的环比、上周的同比升降幅度。请按以下步骤查看应用异常的统计信息。

1. 在**应用总览**页面的概览页签下方，查看异常的总数、周同比和日同比数据，如下图所示。

2.滑动页面至**概览**页签底部的**统计分析**区域的**异常类型**，查看各类型异常出现的次数，如下图所示。



3.在左侧导航栏，点击**监控 > 实例监控**，然后在页面右侧单击异常分析页签，查看异常统计图、错误数、异常堆栈等，如下图所示。

步骤三：诊断异常出现的原因

掌握应用异常的统计信息还不足以诊断异常出现的原因。虽然日志中异常堆栈包含调用的代码片段，但并不包含这次调用的完整上下游信息和请求参数。

APM 探针采用了字节码增强技术，让您能够以很小的性能消耗捕获异常上下游的完整调用快照，进而找出导致异常出现的具体原因。

1. 在异常分析页签下，找到要诊断的异常类型，在其右侧操作列，单击调用链查询。调用链查询页签下显示与该异常类型相关的调用链路信息，如下图所示。

2. 在调用链查询页签下，单击某个错误调用的 TraceId，如下图所示。

3. 在弹出的页面，查看异常的调用链路信息。

操作至此，您已发现了应用异常的原因，这将有效地帮助您进行下一步的代码优化工作。您还可以返回调用链查询页签，查看列表中其他异常，逐一解决。

5.3、诊断应用卡顿问题

定位、排查应用卡顿问题的原因有诸多难点。针对这类问题，APM 提供线程剖析、调用链路诊断、接口监控等一套解决方案，帮助您快速准确定位应用中所有慢调用，进而解决应用卡顿问题。

问题分析

网站卡顿、页面加载过慢是互联网应用最常见的问题之一。排查、解决网站卡顿、页面加载过慢等问题过程复杂，耗时较长，原因如下：

应用链路太长从前端页面到后台网关，从 Web 应用服务器到后台数据库，任何一个环节出现故障都有可能导致整体卡顿。

- 采用微服务架构的应用，链路更加复杂，而且不同组件可能由不同的团队和人员维护，加剧了问题排查的难度。

日志不全或质量欠佳应用日志是排查线上问题的主要方法，但出现问题的位置往往无法预期，而且“慢”通常是偶发现象，要真正找到“慢”的原因，需要在每个可能出现问题的地方打印日志，记录每一次调用，但是成 本太高。

监控不足业务发展过快、应用快速迭代导致应用频繁修改接口、增加依赖等情况，进而导致代码质量恶化。应用需要一个完善的监控体系来自动监控应用的每一个接口，自动记录出现问题的调用。

解决方案

为应用安装 APM 探针后，即可在不改动应用代码的情况下，使用 APM 应用监控的线程剖析、调用链路诊断、接口监控等功能，全方位监控应用中所有慢调用。

步骤一：安装 APM 探针

为应用安装 APM 探针后，才能对应用进行全方位监控。请根据实际需求选择一种方式来安装探针。

步骤二：查看慢 SQL 的统计信息

为应用安装 APM 探针后，APM 会收集和展示选定时间内应用的总请求量、平均响应时间、错误数、实时实例数、Full GC 次数、慢 SQL 次数、异常次数和慢调用次数，以及这些指标的周同比和日同比。请按以下步骤查看慢 SQL 的统计信息。

1. 登录 AMRS 控制台，在左侧导航栏选择**应用监控 > 应用列表**。

2. 在**应用列表**页面顶部单击目标应用名称。

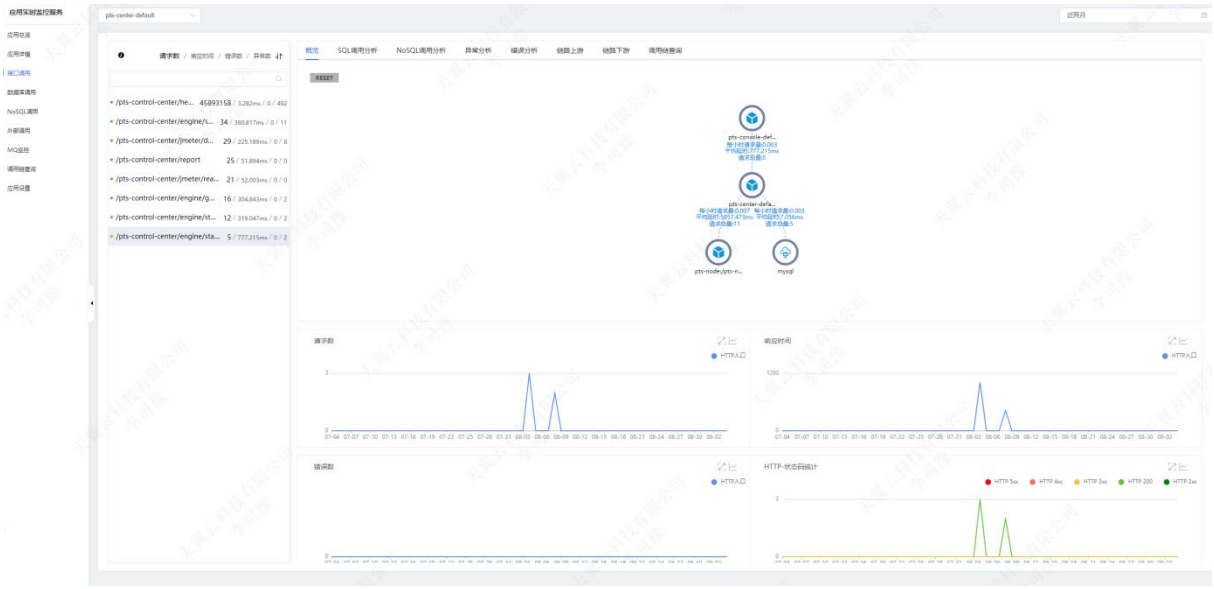
3. 在**应用总览**页面的**概览**页签下，查看慢 SQL 的总数、周同比和日环比数据。

步骤三：发现并锁定慢调用

APM 在**接口调用**页面展示了被监控的应用提供的所有接口以及这个接口的调用次数和耗时，慢接口会被标注出来，帮助您发现和锁定慢接口。

1. 在左侧导航栏，单击**接口调用**。

2. 在**接口调用**页面的左侧，单击调用次数最多的慢接口，在右侧查看慢接口的详细信息。



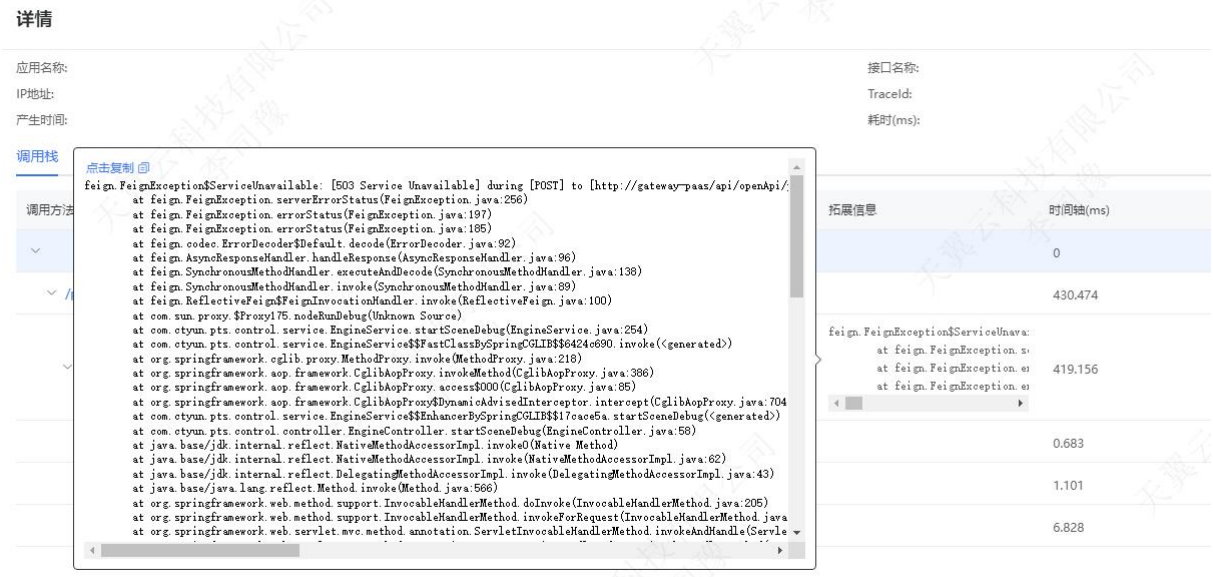
步骤四：查看并锁定问题代码

锁定慢接口后，需要找到问题代码来解决问题。快照是对一次调用的全链路调用的完整记录，包括每一次调用所经过的代码及耗时，可以精准定位问题代码。

1. 在**接口调用**页面右侧，单击**调用链查询**页签。**调用链查询**页签下显示该接口的所有调用链。

2. 在**调用链查询**页签下，单击某个调用链路的 TraceId。

3. 在弹出的页面，查看异常的调用链路信息，将鼠标悬停在拓展信息部分，可以获得异常的具体信息。



操作至此，您已发现了系统中的某个慢调用的原因，这将有效地帮助您进行下一步的代码优化工作。您还可以返回**接口调用**页面，查看列表中其他慢调用，逐一解决。

后续操作

为避免在出现问题后被动诊断错误原因,您还可以使用 APM 的告警功能针对一个接口或全部接口创建告警，即可在出现问题的第一时间向运维团队发送通知。

5.4、业务日志关联调用连的 TraceId 信息

您可以在应用的业务日志中关联调用链的 TraceId 信息，从而在应用出现问题时，能够通过调用链的 TraceId 快速关联到业务日志，及时定位分析、解决问题。

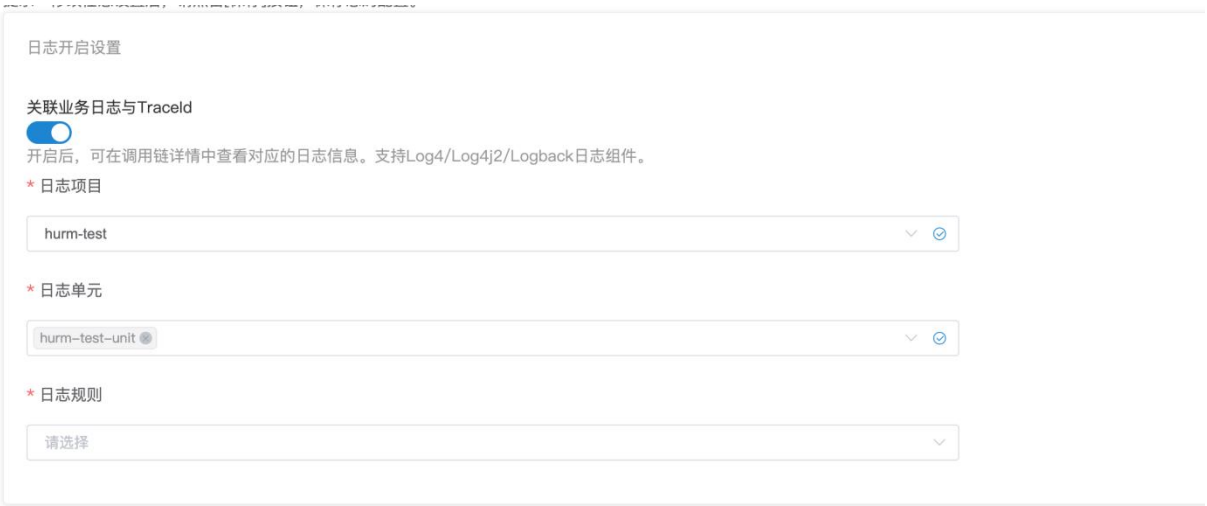
背景信息

APM 在业务日志中关联调用链 TraceId 的功能基于 MDC（Mapped Diagnostic Context）机制实现，支持主流的 Log4j、Log4j2 和 Logback 日志框架。

开启关联业务日志与 TraceId 开关

1. 在左侧导航栏选择**应用监控** > **应用列表**。
2. 在**应用列表**页面顶部选择目标地域，然后单击目标应用名称。
3. 在左侧导航栏中单击**应用设置**，找到日志开启设置栏目。
4. 打开**关联业务日志与 TraceId 的开关**，选择日志项目，日志单元，日志规则。然后保存

如下图:



5.5、通过调用链路和日志分析定位业务异常问题

定位业务异常问题难度大、效率低，一直是应用性能监控的性能瓶颈。应用性能监控通过结合调用链路和日志分析，可以快速、准确地定位业务异常问题，提升微服务框架下的开发诊断效率。

背景信息

在使用调用链路和日志分析定位业务异常问题前，需要先了解 Metrics、Tracing 和 Logging 三个概念。Metrics：应用的关键性能指标，如应用提供服务请求量、应用提供服务平均响应时间、应用依赖服务请求量等。

Tracing：调用链路，应用的任何接口调用、请求响应等动作都会绑定到完整的链路。

Logging：业务日志，应用的任何接口调用、请求响应等动作都会输出完整的业务日志。

当应用出现业务异常问题时，应用指标统计图会出现明显波动，您可据此粗略地分析异常问题；通过完整的调用链路和业务日志分析，可以精准定位业务异常问题。

开启日志设置

开启日志设置的操作步骤如下：

1. 登录 APM 控制台，在左侧导航栏选择**应用监控 > 应用列表**。

2. 在**应用列表**页面单击目标应用名称。

3. 在左侧导航栏中单击**应用设置**，并在右侧页面单击**日志开启设置**页签。

4. 在**日志开启设置**区域开启**关联业务日志与 TraceId**，并设置日志项目、日志单元、日志规则，如下图所示。

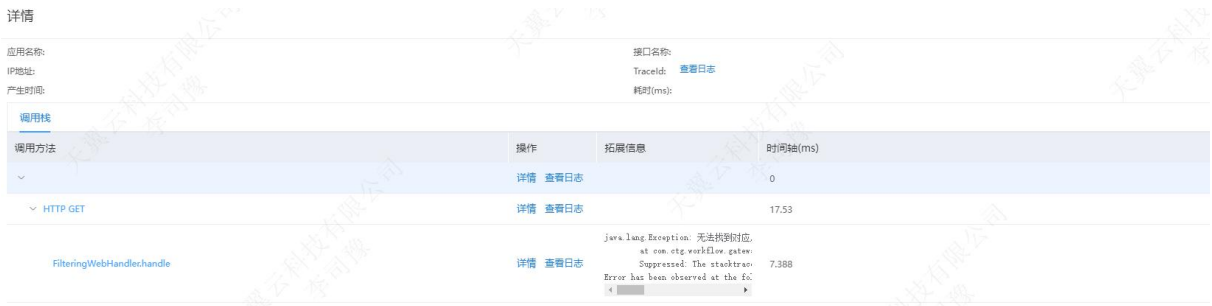
从应用指标的角度排查业务异常问题

在左侧导航栏单击**应用总览**，在顶部选择**概览**，然后在右上角选择或自定义设置目标时间段。

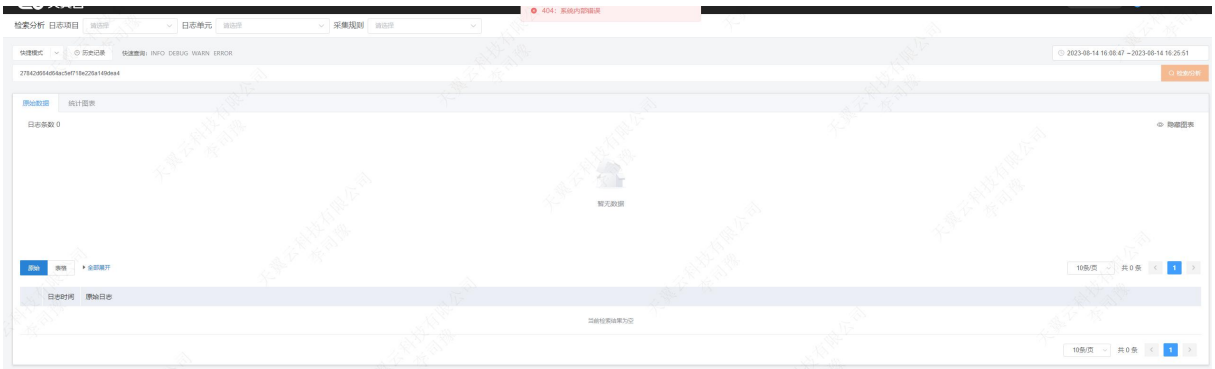
概览页面展示目标应用的关键指标，如**应用提供服务请求量**、**应用提供服务平均响应时间**、**应用依赖服务请求量**等。

1．在调用链路列表面板选择**状态**异常的调用链路记录。

2.单击该调用链路记录 **TraceId** 列下的 TraceId 值。



3.单击查看日志，即可查看日志并定位业务异常原因（目前返回 404）。



从接口调用的角度排查业务异常问题

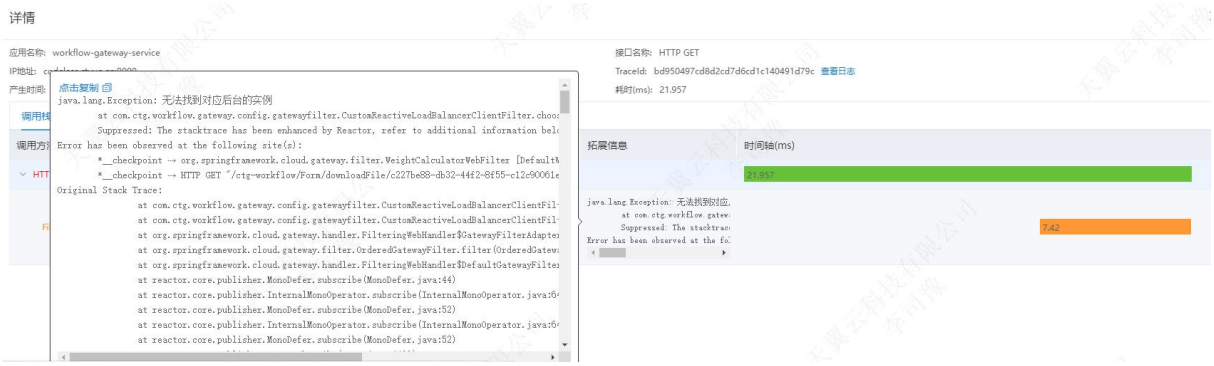
1.登录 APM 控制台，在左侧导航栏选择**监控 > 实例监控**。

2.在**实例监控页面**顶部单击**调用链查询**页签。

3.在**调用链查询**页签选择**状态**异常的接口调用记录，异常状态显示为　　。

4.在目标接口调用记录的 **TraceId** 列下单击 TraceId 的值。

5. 在链路详情信息页面查找错误信息，鼠标悬停在错误信息上可查看异常原因。



6.单击查看日志，即可查看日志并定位业务异常原因。

5.6、通过调用链路和日志分析定位业务异常问题

定位业务异常问题难度大、效率低，一直是应用性能监控的性能瓶颈。应用性能监控通过结合调用链路和日志分析，可以快速、准确地定位业务异常问题，提升微服务框架下的开发诊断效率。

背景信息

在使用调用链路和日志分析定位业务异常问题前，需要先了解 Metrics、Tracing 和 Logging 三个概念。Metrics：应用的关键性能指标，如应用提供服务请求量、应用提供服务平均响应时间、应用依赖服务请求量等。

Tracing：调用链路，应用的任何接口调用、请求响应等动作都会绑定到完整的链路。

Logging：业务日志，应用的任何接口调用、请求响应等动作都会输出完整的业务日志。

当应用出现业务异常问题时，应用指标统计图会出现明显波动，您可据此粗略地分析异常问题；通过完整的调用链路和业务日志分析，可以精准定位业务异常问题。

开启日志设置

开启日志设置的操作步骤如下：

1. 登录 APM 控制台，在左侧导航栏选择**应用监控 > 应用列表**。
2. 在**应用列表**页面单击目标应用名称。

3. 在左侧导航栏中单击**应用设置**，并在右侧页面单击**日志开启设置**页签。

4. 在**日志开启设置**区域开启**关联业务日志与 TraceId**，并设置日志项目、日志单元、日志规则，如下图所示。

从应用指标的角度排查业务异常问题

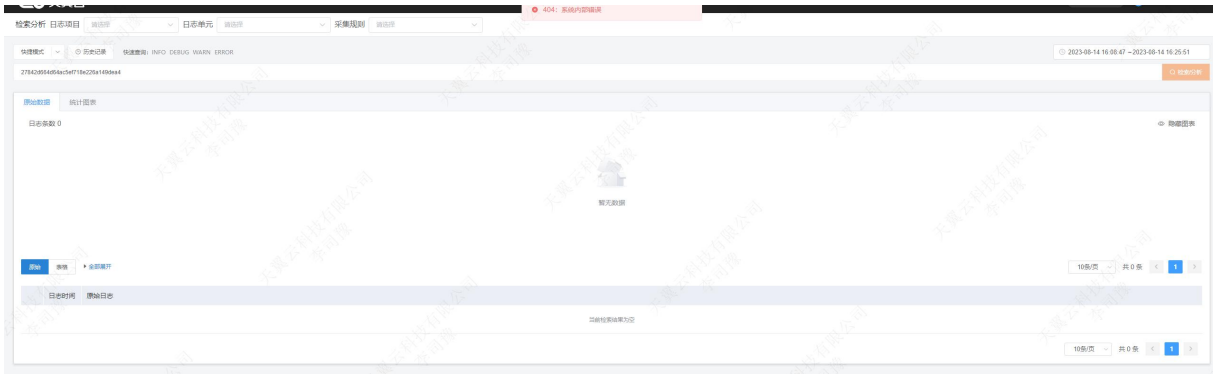
在左侧导航栏单击**应用总览**，在顶部选择**概览**，然后在右上角选择或自定义设置目标时间段。

概览页面展示目标应用的关键指标，如**应用提供服务请求量**、**应用提供服务平均响应时间**、**应用依赖服务请求量**等。

1. 在调用链路列表面板选择**状态**异常的调用链路记录。

2.单击该调用链路记录 **TraceId** 列下的 TraceId 值。

3.单击查看日志，即可查看日志并定位业务异常原因（目前返回 404）。



从接口调用的角度排查业务异常问题

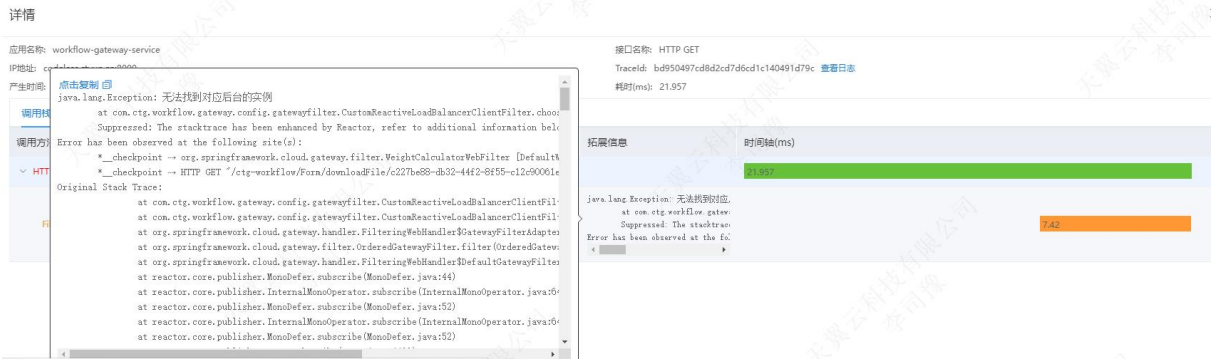
1.登录 APM 控制台，在左侧导航栏选择**监控 > 实例监控**。

2.在**实例监控页面**顶部单击**调用链查询**页签。

3.在**调用链查询**页签选择**状态**异常的接口调用记录，异常状态显示为　　。

4.在目标接口调用记录的 **TraceId** 列下单击 TraceId 的值。

5. 在链路详情信息页面查找错误信息，鼠标悬停在错误信息上可查看异常原因。



6.单击查看日志，即可查看日志并定位业务异常原因。

5.7、使用 APM 监控异步任务

对于通过 Spring @Async 标签或者 Java Executors 实现的异步任务，APM 默认支持对其进行监控。若您的异步任务出现接口超时等异常，可以通过调用链路查看异步任务上下游以便及时处理潜在问题。

前置条件

该功能要求 APM 探针版本为公测版本 v1.1.1 及以上。

方式一：默认支持 Spring @Async 标签

APM 默认支持监控使用 Spring @Async 标签实现的异步任务。对于下列 Spring 框架中默认的 Executor 和 Task，APM 会自动完成增强：

Executor：

- org.springframework.scheduling.concurrent.ConcurrentTaskExecutor
- org.springframework.core.task.SimpleAsyncTaskExecutor
- org.springframework.scheduling.quartz.SimpleThreadPoolTaskExecutor

- org.springframework.core.task.support.TaskExecutorAdapter
- org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor
- org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler
- org.springframework.jca.work.WorkManagerTaskExecutor
- org.springframework.scheduling.commonj.WorkManagerTaskExecutor

Task：org.springframework.aop.interceptor.AsyncExecutionInterceptor$1

- org.springframework.aop.interceptor.AsyncExecutionInterceptor$$Lambda$

方式二：通过增强 Runnable 接口、Callable 接口和 Executor

当您没有使用 spring 框架时，上述场景无法满足需求时，也可以通过自动增强 Runnable 接口、Callable 接口和 Executor，在异步线程中完成调用链串联，实现异步任务监控。

1、**增强 Runnable 接口和 Callable 接口**

```
public class MyRunnable implements Runnable {


    @Override
    public void run() {
        // 在这里定义需要在新线程中执行的任务逻辑
        for (int i = 0; i < 5; i++) {
```

```
            System.out.println("Thread ID: " + Thread.currentThread().getId() + " - Cou
nt: " + i);
        }
    }

    public static void main(String[] args) {
        // 创建一个 Runnable 实例
        Runnable myRunnable = new MyRunnable();

        // 创建线程并将 Runnable 实例传递给线程
        Thread thread1 = new Thread(myRunnable);
        Thread thread2 = new Thread(myRunnable);

        // 启动线程
        thread1.start();
        thread2.start();
    }
}
```

2、 增强 Executor

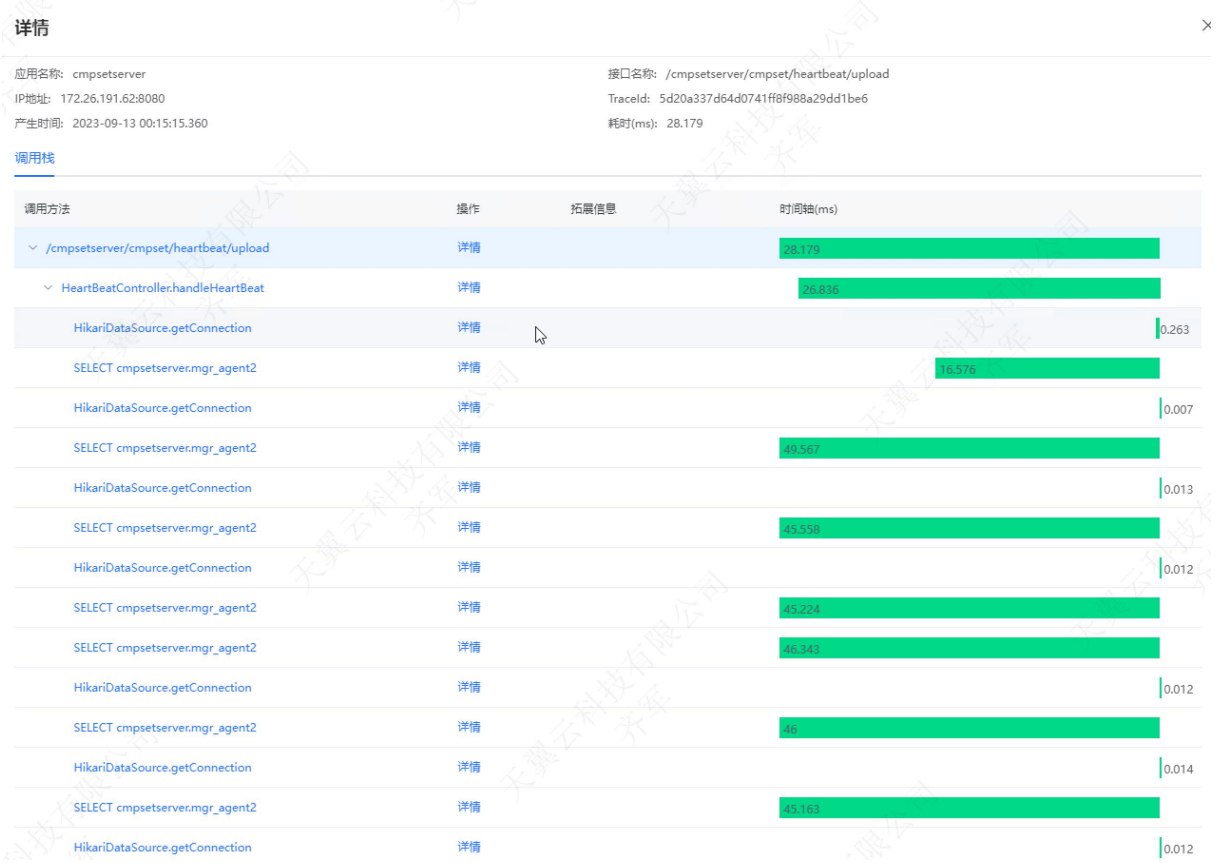
```
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;
```

```
public class ExecutorDemo {

    public static void main(String[] args) {
        // 创建一个线程池，这里使用固定大小的线程池
        Executor executor = Executors.newFixedThreadPool(3);

        // 提交任务给线程池执行
        for (int i = 0; i < 5; i++) {
            final int taskId = i;
            executor.execute(() -> {
                // 在这里定义需要执行的任务逻辑
                System.out.println("Task " + taskId + " is running on thread " + Thread.c
urrentThread().getName());
            });
        }

        // 关闭线程池（通常在应用程序退出时执行）
        if (executor instanceof java.util.concurrent.ExecutorService) {
            ((java.util.concurrent.ExecutorService) executor).shutdown();
        }
```

```
    }

}
```

执行结果

配置完成后，您可以在调用链路详情页查看异步任务的调用链详情。具体详情，请参见下图。



六、常见问题

6.1、计费类

6.1.1、计费相关问题

1、如何停止应用计费

如您后期还需要重新对该应用进行监控,那么在自定义配置中,关闭 agent 总开关即可;

如果您后期不再需要监控该应用以及看历史监控数据,也可在应用列表中直接删除应用。

2、资源包消耗完后如何计费

产生探针时消耗时,优先判断当前时间是否有可用的资源包,如果有,则优先消耗资源包的探针时;如果没有,则按按需付费的方式,根据实际消耗的探针时产生费用。

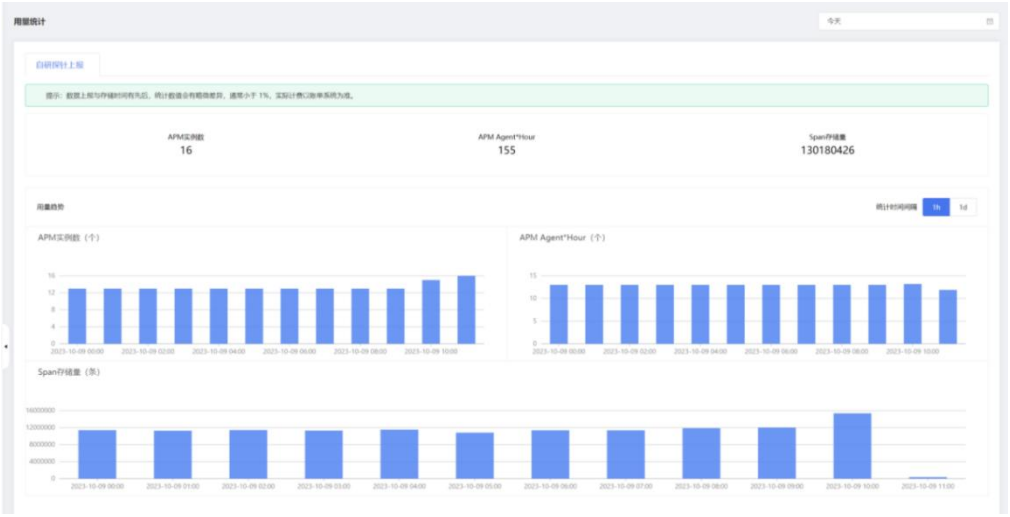
3、公测期间计费

公测期间无论您如何使用都不会产生任何费用,如果公测到期后您选择继续使用,将会按照您选择的使用方式计费。

4、如何查看 agent 用量

应用监控控制台为用户提供查看资源消耗的功能,您可以通过设置过滤条件查看单个或全部应用在特定时间内的资源消耗。

1. 在 AMS 控制台左侧导航栏中选择用量统计。
2. 在用量统计页面右上角设置需要查看的时间段。



- APM 实例数：用户所使用的探针实例个数，一个 APM 实例表示一个正在运行的 Java 服务实例
- APM Agent*Hour：1 个应用实例的 1 天消耗 = 24 Agent×Hour
- Span 存储量：span 数据的存储数

6.2、购买类

公测期间，无需操作购买

6.3、操作类

6.3.1 、网络配置相关问题

如何测试网络连通性

可使用 telnet 命令测试目标主机与 APM 服务器网络是否连通。例如，以测试内蒙 6 地域的连通性为例，请登录应用所部署的机器，并输入以下命令：

```
telnet arms-data-proxy.cq2b.inner.ctyun.cn 27149
```

具体每个资源池的服务器地址参考应用接入

6.3.2、升级探针

对于在 ARMS 中监控的各类型应用，请按照本文所述的相应方法更新 Java 探针版本。

如何为虚拟机部署的应用更新探针？

如果您需要更新虚拟机部署的应用 Java 探针版本，您需要在 应用接入 里面点击下载按钮，下载最新 Java 探针，替换掉旧版本的探针，然后重新启动应用即可。

如何为 ccse 部署的应用更新探针？

如果您需要更新容器服务应用的 Java 探针版本，在开启 APM 的基础上，重新部署工作负载即可。

1. 登录天翼云 CCSE 控制中心，在左侧导航栏选择集群，在右侧页面进入集群详情。
2. 在左侧导航栏选择工作负载 > 有状态/无状态，找到对应的工作负载。
3. 点击重新部署，选择需要的部署方式，点击确定。

如何为其他类型的应用更新探针？

目前仅支持两种接入方式，其他类型正在扩充中，请关注产品动态的更新提醒。

6.3.3、为 Java 应用手动安装 Agent 的常见问题

（1）APM Agent 和其他 APM 产品 Agent（例如 SkyWalking）是否兼容？

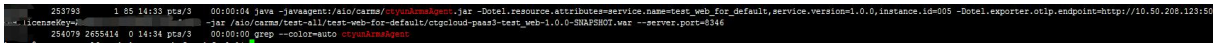
APM Agent 是基于 opentelemetry 开源项目的 Agent 进行的二次开发，和其他 APM 产品 Agent 都不兼容。APM 大多是基于 ASM 框架进行字节码插桩实现的，同时安装两个 Agent 相当于对您的代码插桩两次，由于不同厂家的插装代码实现不同，代码冲突可能造成各类问题（例如应用启动缓慢，类冲突等问题），因此强烈建议您不要同时安装多个 APM Agent。

（2）如何检查 APM Agent 是否安装成功

使用 ps 命令查看命令行启动参数中是否成功安装 APM Agent。

```
ps -ef | grep 'ctyunArmsAgent'
```

成功安装时，如下图所示：



6.3.4、APM Agent 安装成功，为什么控制台上仍无监控数据？

问题现象

APM Agent 安装成功，控制台上仍无监控数据。

可能原因

应用无持续的外部请求访问、或者 APM Agent 安装目录权限不正确等原因都有可能导致控制台无监控数据。

解决方案

- 如果 Agent 日志中出现“send agent metrics. no metrics.”，请确认您的应用是否有持续的外部请求访问，包括 HTTP 请求、HSF 请求和 Dubbo 请求，并确认开发框架是否在 APMAgent 的支持范围内。关于 APM 应用监控对第三方组件和框架的支持情况，请参见 APM 应用监控支持的 Java 组件和框架。
- 确认选择的查询时间范围是否正确。请您将查询时间条件设为最近 5 分钟，然后再次确认是否有监控数据。
- 如果是通过-jar 命令行启动的，请检查命令行设置，确保-javaagent 参数在 -jar 之前。

```
java -javaagent:/{user.workspace}/ctyunArmsAgent.jar -Dotel.exporter.otlp.endpoint=xxx.xxx.xxx.xxx:xxxxx -Darms.licenseKey=xxx -jar testWeb.jar
```

- 如果{user.workspace}/ctyunjavaagent/ctyunarmsagent.log 的日志中出现 transData span is faile 或 transData metrics is faile 异常，请您检查应用与启动参数中的 otel.exporter.otlp.endpoint 指向的地址是否连通。

- 如果{user.workspace}/ctyunjavaagent/ctyunarmsagent.log 的日志中出现 gateway sdk initFail 异常，请您检查应用所属地域与 Agent 所属地域是否一致。
- 如果{user.workspace}/sdklogs/gw-sdk.log 的日志中出现 LICENSE_IS_DISABLE 异常，请您检查 license 是否已过期或是超出配额限制。
- 如果应用启动之后{user.workspace}/目录下无 ctyunjavaagent 和子目录，是由于 ctyunArmsAgent.jar 未被成功加载导致的，请您检查 APMAgent 安装目录的权限是否正确。

6.3.5、安装 Agent 后应用启动时报 OutOfMemoryError 错误怎么办？

请在 Java 启动命令后面加上堆内存大小配置项，以适当调大 JVM 参数。以下示例配置项表示堆内存初始值（Xms）为 512 M，堆内存最大值（Xmx）为 2 G。

说明 请根据实际情况适当调节。对于 Tomcat 等其他环境，请在配置文件的 JAVA_OPTS 中加入此参数。

```
-Xms512M
-Xmx2048M
```

如果出现 OutOfMemoryError: PermGen space 错误，请添加以下配置。

```
-XX:PermSize=256M
-XX:MaxPermSize=512M
```

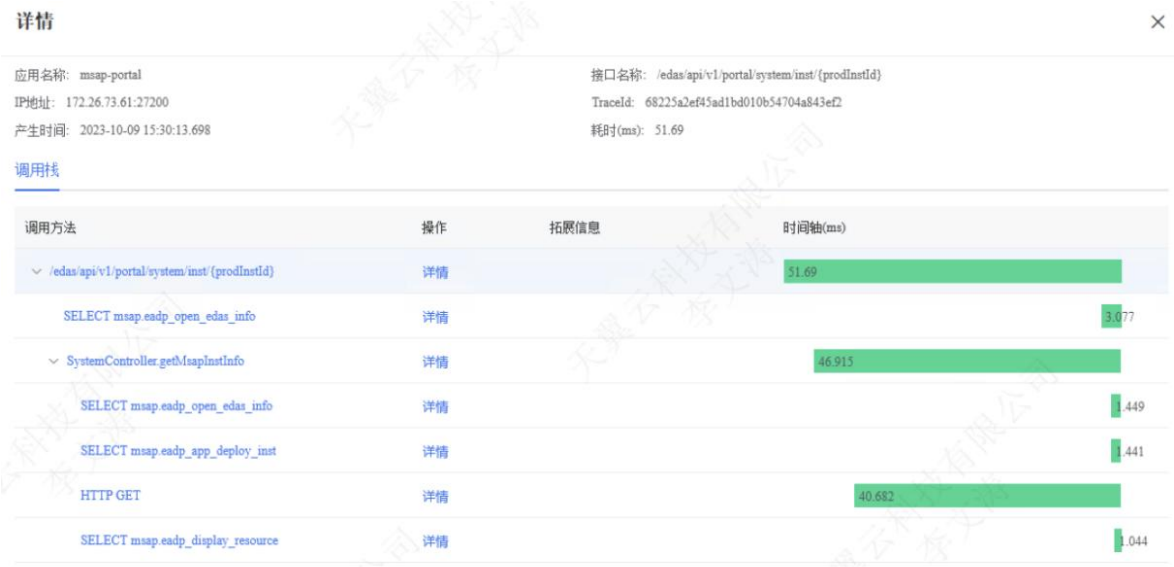
如果出现 OutOfMemoryError: metaspace 错误，请添加以下配置。

```
-XX:MetaspaceSize=256M
-XX:MaxMetaspaceSize=512M
```

6.3.6、调用链的时间线是如何绘制的？

问题现象

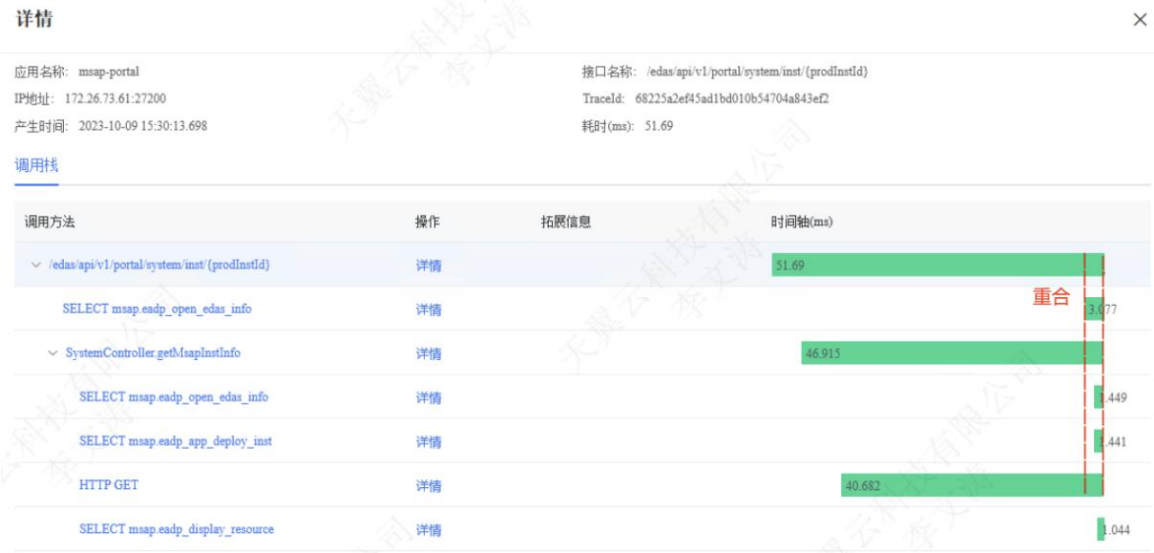
调用链的时间线是如何绘制的？当调用链的时间线类似如下图所示时，是绘制错误了吗？



问题解答

每个方法的时间线是表示该方法的起始位置与总耗时。在调用链界面中，方法线用灰色绘制，每一个方法的耗时用绿色填充。绿色填充段表示总耗时。当您看到上图所示的调用链时，可能会有疑惑：方法 1、2 的时间线有部分重合。

其实，并没有绘制错误，由于 APM 的调用链绘图中，每个方法的时间线是绘制的起始位置与总耗时，所以绘制会出现问题现象所描述的情况。



6.3.7、其他问题

（1）为什么没有看到对应数量的调用链？

问题现象

为什么发起多次请求，但产生的调用链数量较少？

可能原因

调用链数量和设置的调用链采样率有关。

您可以在 APM 控制台目标应用的**应用设置 > 自定义配置**页签的**采样率设置**区域查看调用链采样率。默认的采样率为 1%，即只有 1%的调用链会被采集。

采样规则：错误与异常调用的调用链会被默认采样。

（2）GC 次数为什么有小数？

问题现象

正常不会出现垃圾回收一半就暂停的情况，但 GC 次数出现小数

可能原因

GC 时间的单位为 ms，次数按照平均数计算，所以会出现小数。