19 | 端到端Trace：消息收发链路的监控体系搭建

2019-10-09 袁武林

即时消息技术剖析与实战

进入课程 >

讲述：袁武林
时长 19:24　大小 15.56M

你好，我是袁武林。

前面的大部分课程，我基本都是围绕“如何开发和设计一个 IM 系统”的技术点，来进行分析和讲解的，但在实际项目的工程落地实践中，IM 系统的监控和保障也是极其重要的一环。

只有通过对消息收发链路的监控，我们才能够实时地了解到链路是否可用，后端服务是否足够健康；如果监控体系不够完善，我们的业务即使上线了，也是处于“蒙眼狂奔”的状态。所以，我们要在工程上线时有一个**“服务上线，监控先行”**的意识和原则。

今天，我们就一起来聊一聊，消息收发链路中监控体系搭建的问题。

在 IM 场景中，常见的监控模式大概可以分为两种：一种是基于数据收集的被动监控，一种是基于真实探测的主动监控。下面我们来分别看一下这两种监控模式的实现细节和区别。

基于数据收集的被动监控

“基于数据收集的被动监控”，应该是我们日常开发保障中，最常见的服务和系统监控方式了。

一般来说，被动监控可以粗略地分成几个子类型：系统层监控、应用层监控及全链路 Trace 监控。

系统层监控是整个监控体系中最基础的监控方式，一般来说，它主要监控的是操作系统维度的一些核心性能指标。

举个例子：我们要对上线的业务进行监控，可以通过 Nagios、Zabbix 等类似的系统监控工具，来实时收集机器相关的性能数据，如 CPU、内存、IO、负载、带宽、PPS、Swap 等数据信息。

由于足够通用，系统监控相关的具体细节，我在这里就不展开了，你可以在留言区与我一起探讨。

应用层监控

除了系统层面的监控，我们非常依赖的另一种被动监控是应用自身的监控。

在即时消息场景中，我们需要实时监控消息收发接口的 QPS、耗时、失败数、消息在线推送到达率等，如果出现耗时、失败率增长或者推送到达率降低等情况，我们就要引起注意并对其进行分析了。

比如在微博平台的场景里，就用到了基于 Graphite 的监控体系，来对微博的应用状态进行监控。

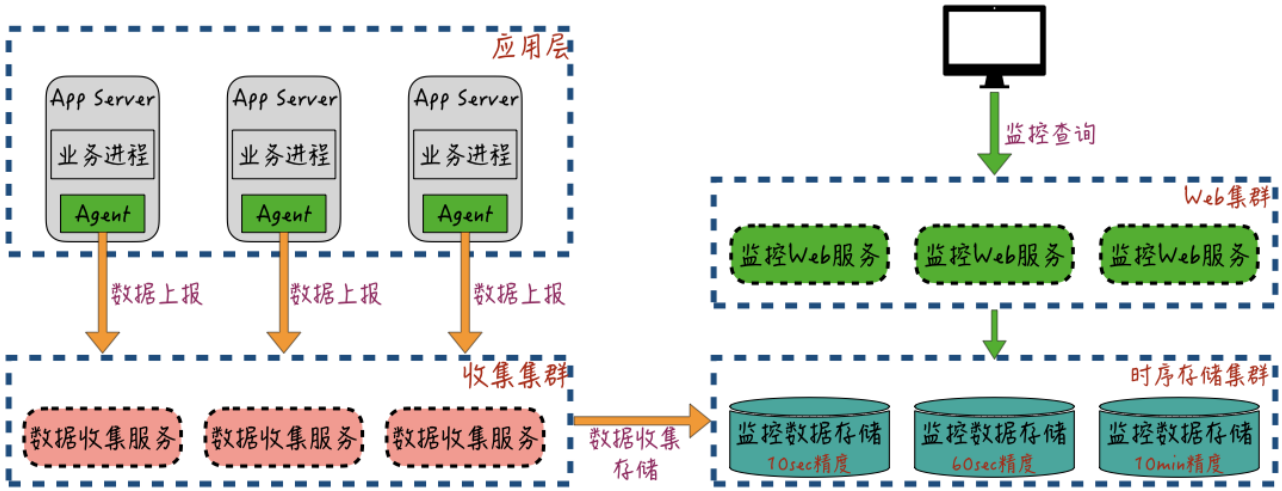
在对应用层业务直接相关的 API 接口进行监控之外，我们还需要知道在消息收发的核心链路中，这些业务直接相关的 API 接口所依赖的其他 API 或资源的性能情况，以便于我们在业务接口出现失败率高或者耗时增长的时候，能够通过监控系统，快速找到导致这个接口出现问题时，所依赖的资源或者其他依赖接口。

比如，我们还需要监控离线 Buffer 用到的 Redis 的使用量、相应的读写 QPS、耗时等。

除了监控应用层整体的情况，当业务层直接相关的 API 接口在整体层面上出现性能问题时，由于可能只是某一两台机器的 API 接口出现了性能问题，并且由于统计的原因，导致该 API 接口在整体上看失败率升高或者耗时增加，因此为了便于排查和分析，我们一般还需要对单机的应用状态分别进行监控。

比如，某一台机器由于内存不够吃 Swap 了，或者网络发生抖动导致接口耗时增长了，我们就需要针对这台单机进行监控，来快速发现问题并处置。

应用层的监控数据收集和使用的架构，你可以参考下图：



应用服务进程通过本地套接字的方式，把服务自身的 Metrics 数据发送给本机的“数据收集代理器”Agent，或者通过本地日志记录的方式，记录服务的 Metrics 数据。

“数据收集代理器”Agent 从本地日志里流式获取这些日志数据，最终收集到的监控数据，由“数据收集代理器”上报给远程的数据收集服务集群；收集集群对数据进行缓存和预聚合，然后再提交给存储集群。

监控数据的存储集群出于数据规模和数据聚合查询能力的考虑，一般会采用“时序数据库”来进行多精度的存储，比如 OpenTSDB、InfluxDB 等；然后通过时序数据库的高压缩比存储和聚合计算功能，来解决监控数据规模大和查询效率低的问题；最终存储到“时序数据库”中的监控数据，通过 Web 服务对用户提供时间维度的界面查询功能。

对于系统层监控和应用层监控，目前业界都有非常成熟的解决方案。比如常见的 Statsd + Graphite + Grafana 方案和 ELK（ElasticSearch + Logstash + Kibana）方案，它们的使

用度都非常高。在实现上，也基本和上面图中展现的监控数据收集与架构方式差不多，所以具体的细节实现，我在这里就不展开了，你可以自行了解一下。

全链路 Trace 监控

除了系统监控和应用服务监控外，在严重依赖网络可用性的即时消息场景里，很多时候，我们需要关心的不仅仅是服务端的性能，还要从用户自身的体验角度出发，来全局性地监控 IM 服务的可用性和性能。

另外，由于各个微服务都是独立部署并且互相隔离的，很多时候，我们在排查消息收发失败的原因时，很难查询到具体的异常是由哪一个依赖的服务或者资源引起的，问题定位和处理效率也就非常低。

怎样才能把某次消息收发的各环节的性能数据，以及整个访问链路的情况聚合起来，以便于我们来定位问题呢？

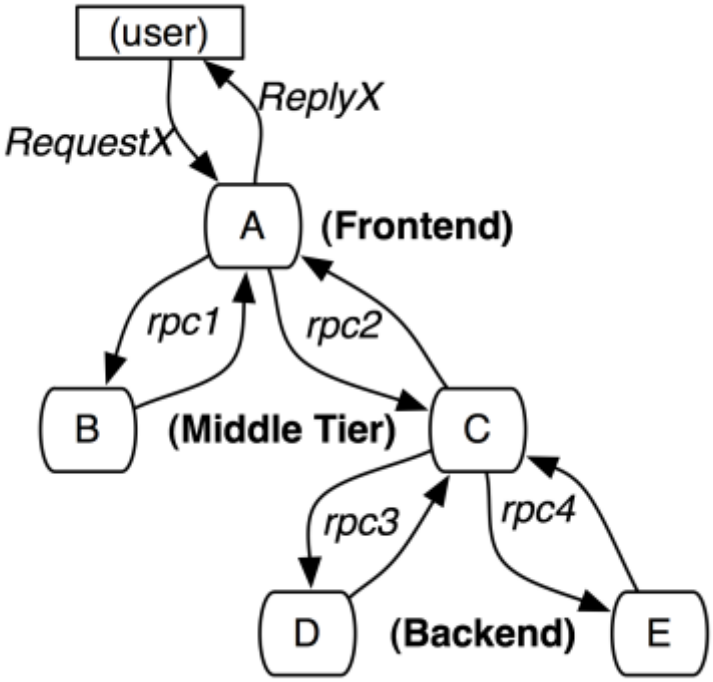
一个比较好的解决方案就是：基于 Trace 服务，对消息的收发行为进行“全链路监控数据收集”。

那么，接下来我们就来了解一下，这个 Trace 服务到底是什么？

Trace 一词的出现，起源于 Google 的一篇官方论文“Dapper，a Large-Scale Distributed Systems Tracing Infrastructure”。

在这篇论文中，介绍了 Google 的 Dapper 系统，并首次定义了什么是分布式跟踪系统，以及介绍了分布式跟踪系统的三大设计要点：低开销、对应用透明、高可扩展性。

为了实现分布式链路追踪，Dapper 论文提出了 Trace、Span、Annotation 的概念，并给出了一个 Trace 调用的示例，如下图：



A~E 分别表示五个服务，用户发起一次请求到 A，然后 A 分别发送 RPC 请求到 B 和 C，B 处理请求后返回，C 还要发起两个 RPC 请求到 D 和 E，最终服务 A 将请求结果返回给用户。

我们再分别来看一下 Trace、Span、Annotation 的概念。

Trace表示对一次请求完整调用链的跟踪，每一条链路都使用一个全局唯一的 TraceID 来标识。类似于上图中的整个一次调用链路就是一次 Trace。

Span是指在链路调用中，两个调用和被调用服务的请求 / 响应过程叫做一次 Span。一条 Trace 可以被认为是由多个 Span 组成的一个有向无环图（DAG 图）。

比如上图的示例中，用户对服务 A 的请求和响应过程就是一个 Span（假设叫 Span 1），服务 A 对服务 B 的调用和响应过程是另一个 Span（假设叫 Span 2）。Span 支持父子关系，比如这里的 Span 1 就是 Span 2 的父 Span，这些 Span 通过同一个 TraceID 来串联。

一个 Span 会记录 4 个时间戳：“客户端发送时间（Client Send）” “服务端接收时间（Server Receive）” “服务端发送时间 (Server Send)” “客户端接收时间（Client Receive）。

通过这 4 个时间戳，我们就可以在一次请求完成后，计算出整个 Trace 的执行耗时、服务端处理耗时和网络耗时，以及 Trace 中每个 Span 过程的执行耗时、服务端处理耗时和网络耗时。

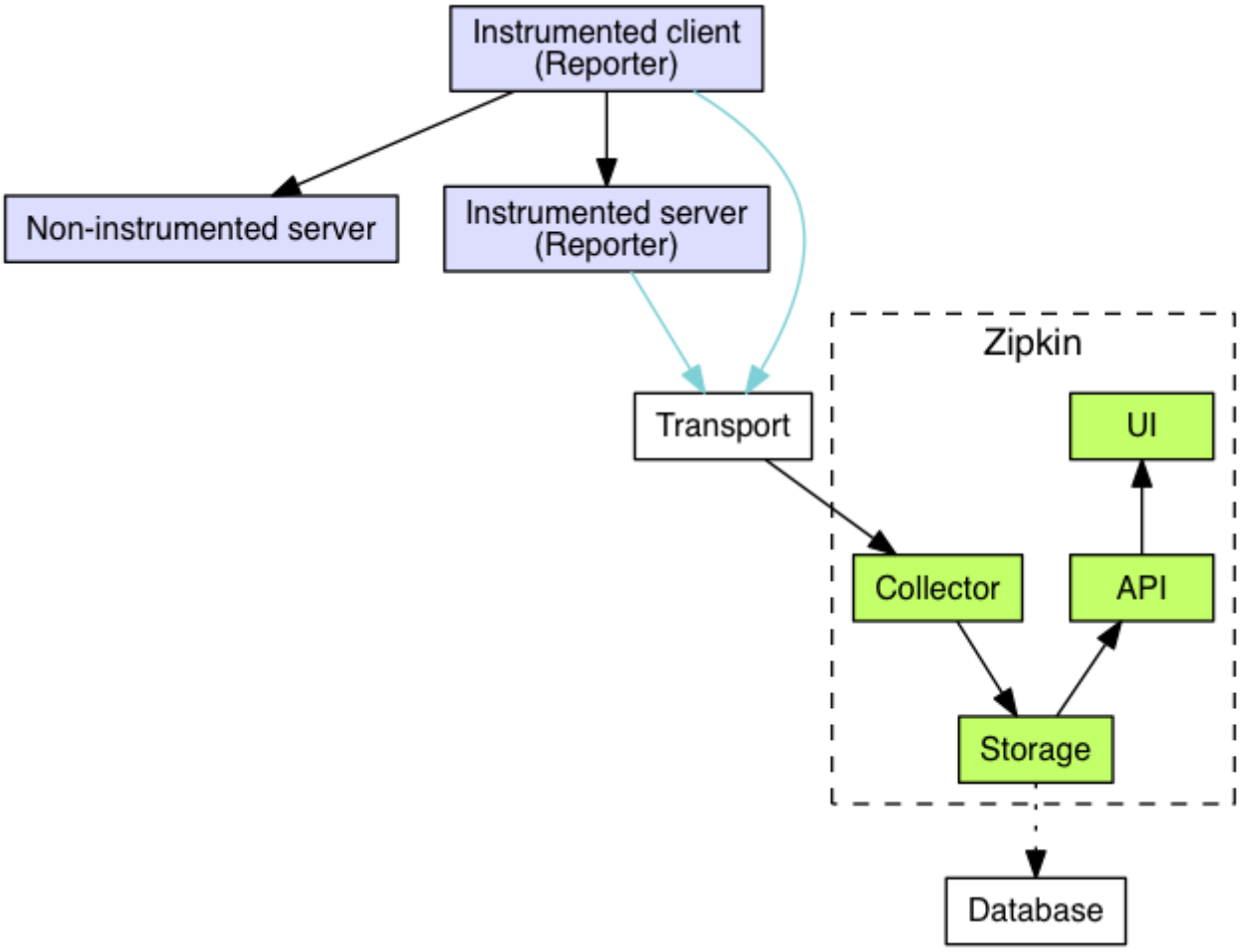
比如，客户端整体调用耗时 =Client Receive-Client Send，服务端处理耗时 =Server Send-Server Receive，那么，这一次请求的网络耗时 = 客户端整体调用耗时 - 服务端处理耗时。

Annotation主要用于用户自定义事件，Annotation 可以携带用户在链路环节中的自定义数据，用来辅助定位问题。

比如在直播互动场景中，记录发弹幕的 Trace 的 Span 里，还可以利用 Annotation，通过 KV 键值对的方式把房间 ID、发送人 UID 等信息也一起记录下来，便于我们后续根据这些 KV 键值对，进行业务维度的查询。

目前业界比较成熟的分布式 Trace 系统有：Twitter 的 Zipkin、Uber 的 Jaeger、阿里的鹰眼、美团的 Mtrace，等等。

在这里，我以使用比较广泛的**Zipkin**为例，其整体的实现架构你可以参考下面的这张官网图：



Reporter 模块通过 AOP 探针或者硬编码的方式嵌入到业务代码中，负责 Span 的生成和上报。

Transport 模块是 Trace 数据上报通道，支持 HTTP、Kafka、Scribe 等方式。

Colletor 模块负责 Trace 数据的接收，并将 Trace 数据写入到中央存储中。

Storage 部分为存储组件，默认是 In-Memory 存储，并支持 Cassandra、ElasticSearch、MySQL 等存储系统。

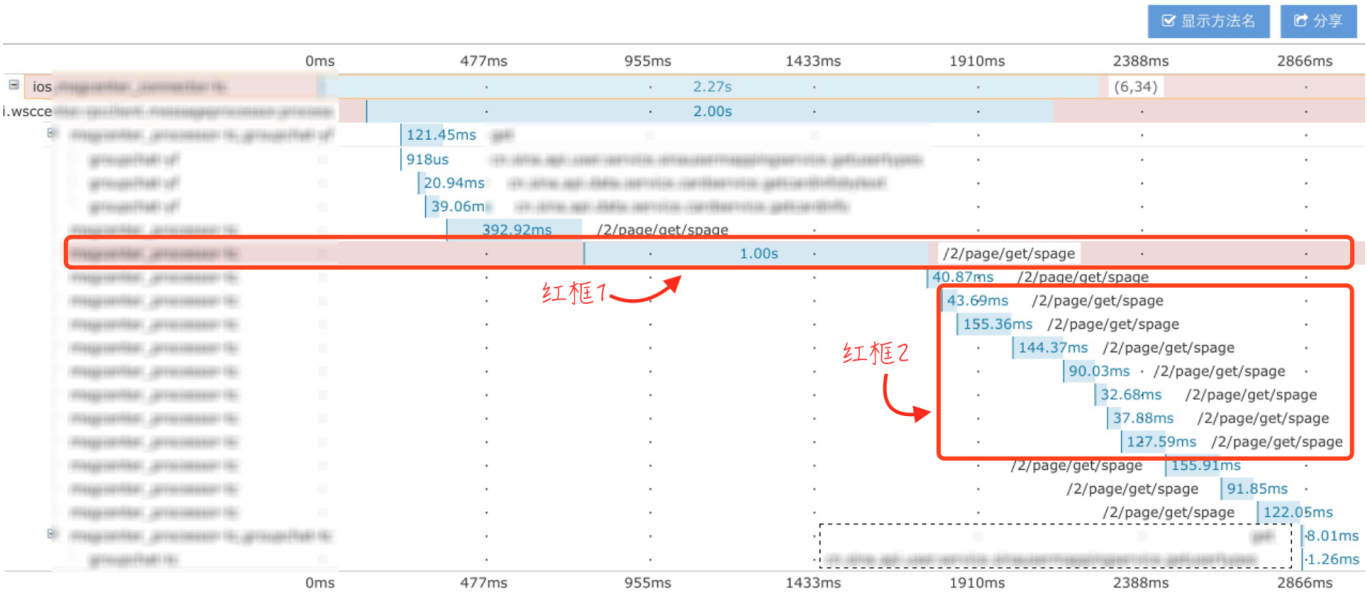
API 层提供 Trace 的查询、分析和上报链路的对外接口。

UI 部分主要用于页面展示。

可见，通过应用类似 Zipkin 的这种分布式全链路 Trace 系统，我们不仅能做到快速排查消息收发链路中出现的问题，而且还能根据 Trace 数据，来分析各个调用环节的性能，并可以结合实时数据分析工具（如 Flink），多维度地进行业务维度的监控和报警。

在微博的线上业务中，就通过基于 Zipkin 优化定制的 Trace 系统，来定位消息收发的故障点，以及用于链路优化的分析支撑。

以下图中出现的故障点为例：



这一次群聊消息查询失败的原因是，调用一个“富文本”解析服务 1 秒超时失败（红框 1，调用一个叫 spage 接口）；而且还发现群聊服务对“富文本消息”解析是串行调用的（红框 2），这里的“串行调用”就是一个待优化点。

全链路监控 Trace 中，一个值得注意的问题是 Trace 数据采样率。

由于一次消息收发的调用链路 Span 数一般都非常多，对于访问量较大的场景来说，全量的 Trace 数据量太大，所以一般会通过采样的方式来减少 Trace 数据的收集。比如，在 App 启动时，让服务端返回告知客户端采样率，客户端按照约定的采样率对部分请求进行采样。

举个实际的例子：在微博的线上环境中，对上行请求一般是百分百采样，对下行普通用户一般是 1% 采样，对 VIP 用户上下行请求是全量采样。

除了采样率问题，另一个比较麻烦就是 Trace 数据的采集问题。

虽然大部分分布式 Trace 系统支持多语言 Reporter 来上报数据，但由于各系统的完善程度差别比较大，特别是基于 AOP 等探针，来“无感知”地对各种中间件的支持还是不太够，因此在整体上，还需要一定的探针的工作开发量。

另外，针对多个异构系统的对接，除了在各自系统的业务代码中直接上报 Trace 数据外，我们还可以通过本地日志 +Agent 上报的方式，来解耦异构系统对 Trace SDK 的强依赖。

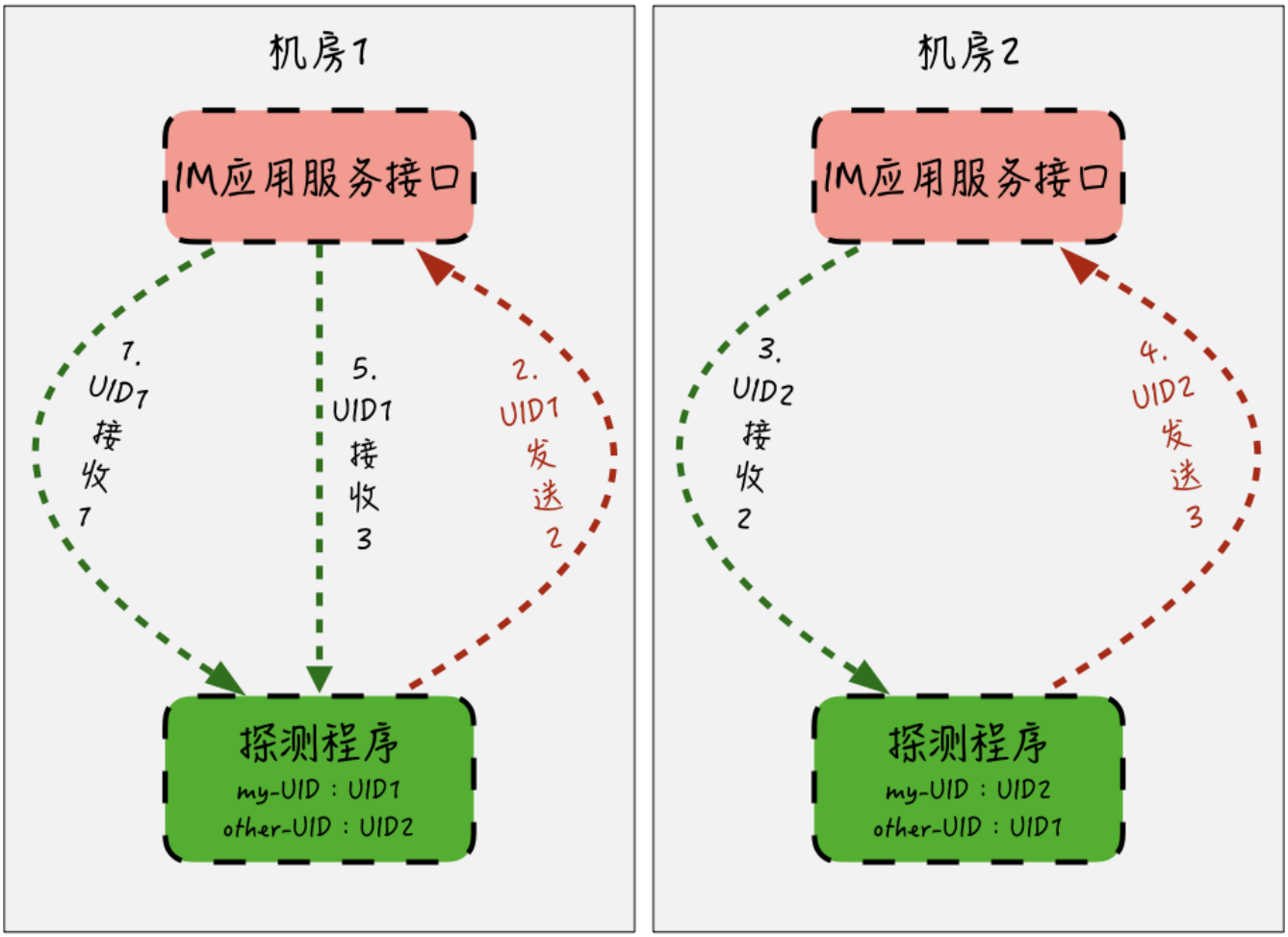
基于回环探测的主动监控

前面讲到，不管是系统监控、应用监控，还是全链路监控，本质上都是基于数据汇报的被动式监控。

正常情况下，基于各种监控数据收集的被动监控，能够协助我们快速发现问题，但当服务出现故障时，被动监控依赖的数据收集就会容易出现上报延迟，甚至上报失败的情况。比如，当系统负载很高时，业务系统就很难再保障监控数据的上报了。

所以，业界另一种实时监控业务系统可用性的方式就是：“基于回环探测的主动监控”。

对于即时消息场景来说，大部分场景都是基于用户维度的消息收发的业务形态，因此，我们可以通过固定的两个或几个测试用户的 UID，进行消息的互相收发，利用消息收发回环，来实时探测消息链路的可用性。大概的思路如下图：



我们分别在机房 1 和机房 2 部署探测程序，用来监控两个机房的消息收发接口可用性。机房 1 的探测程序设置成“自己的 ID”为 UID1，“对方的 ID”为 UID2，机房 2 的探测程序设置刚好相反。

探测失败的情况有以下两种：

一是探测程序调用本机房的发送消息 API 来发送消息，如果 API 调用失败或者超时，本次探测就会失败然后报警；

另一种情况，比如机房 1 的探测程序发送数字 N 之后，在下一次发送前，会尝试通过服务的 API 接口，来检测是否有接收到来自机房 2 的探测程序发出的数字 N+1，如果没有收到，则说明机房 2 的探测程序可能出现异常，或者机房 2 的应用服务接口有异常，这种情况如果连续出现，就应当提示失败然后报警。

可以发现，通过回环探测的方式，探测程序不仅能检测本机房应用服务接口的可用性，也能通过连续发送的数字，间接地探测出对方机房应用服务的可用性，从而避免了由于两个机房网络间的异常，而无法发现消息收发失败的情况。

这里我再稍微延伸一下，对于多机房或者单机房的回环探测，你可以参考双机房的探测模式，来制定相应的策略。

主动探测监控程序作为一个独立的第三方，通过模拟用户消息收发的行为，弥补了被动监控可能由于应用服务不可用，导致监控数据无法上报的缺陷。但探测报警的缺陷在于可模拟的用户有限，并不能覆盖所有用户的整体情况，存在以偏概全的风险。

因此，我们可以将主动探测监控和被动监控一起协同使用，作为即时消息服务的监控双保险。

小结

好，简单回顾一下这一讲的内容。

今天，我们从消息收发链路的监控体系搭建出发，讲解了业界对于服务监控的两种常见模式：**被动监控**和**主动监控**。

被动监控主要依赖服务器或者应用服务的监控数据上报，通过第三方监控系统，来对监控数据进行展示。

被动监控又可以细分为**系统层监控**和**应用层监控**，这两种监控通过实时收集机器层面和应用服务层面的性能数据，协助我们实时掌握机器和应用服务的可用性。

另外，还有一种**全链路 Trace 监控**，也属于被动监控，实际上也属于应用层监控的范畴。

它是基于 Google 的 Dapper 论文衍生出的众多分布式链路追踪系统，进一步通过链路 Trace，将消息收发行为进行整体的端到端的串联，极大地提升了问题排查的效率，而且为链路优化分析和用户访问数据分析，提供了强有力的监控数据支撑。

为了弥补被动监控依赖机器和应用服务的监控数据上报的问题，我们还可以通过第三方的主动探测程序，来实现主动监控。在消息收发场景中，通过模拟用户收发消息行为的回环探测方式，来监控通道的可用性。

我们在即时消息场景中，就可以通过以上这两种监控方式的协同，来更好地监控消息收发服务的可用性。

搭建一套完备的监控体系的重要性是如此之高，特别是对于大规模的分布式应用场景来说，出现这样或那样的问题和故障，已经是一个常态化的情形。如果没有一套可以实时反映系统整体健康状况的监控系统，无异于是盲人摸象，会让我们无法正确及时地评估业务实际受影响的范围，也无法快速定位问题。

实际上，对于今天课程中讲到的这些监控实现的方式，也是前后端普遍通用的方案，不仅适用于 IM 的场景，大部分的业务场景也都是可以参考使用的，也希望你能尝试去了解，然后在自己的业务中实践拓展。

最后给你留一个思考题：**全链路 Trace 系统中，如果被 Trace 服务依赖了其他还没有接入 Trace 的 API，是否追踪还能正常运转?**

以上就是今天课程的内容，欢迎你给我留言，我们可以在留言区一起讨论，感谢你的收听，我们下期再见。

上一篇　18 | Docker容器化：说一说IM系统中模块水平扩展的实现

下一篇　20 | 存储和并发：万人群聊系统设计中的几个难点

精选留言 (6)

写留言

_CountingStars
2019-10-09

trace会中断 无法查看整个链路的情况 主要是因为 traceid 和 parent span id 无法正常传递

1

Z邦
2019-10-12

老师能否开一篇详解应用层监控需要监控的指标、获取方法，异常幅度等细节?

Z邦
2019-10-12

老师能否开一篇详解一下应用层监控收发接口的 QPS、耗时、失败数、消息在线推送到达率

展开

clip
2019-10-10

思考题：
感觉整体可以正常运转，但是会丢失一些细节。
如果被调用服务只是 server 侧没接 trace 那会丢失 server 侧及这个服务内部的其他调用的 trace。
如果被调用服务的 client 也没接 trace 那这次调用会被当做调用它的服务的内部的一个...

展开

> 作者回复: 是的，只是这一个span无法再向后延展，整体trace使用上是ok的，当前这个span也只是丢失ss和sr部分，实际也是属于可用的。

clip
2019-10-10

“对单机的应用状态分别进行监控”是指怎样的监控呢？
是类似监控应用层整体情况那样但改成单机器监控，还是做每台机器的系统层监控呢？
感觉系统层监控的话好像还是不太容易和应用层整体的报错对应起来。

展开

> 作者回复: 实际上分布式应用的整体监控数据的就是依赖单机上报的数据在数据收集端进行聚合的，比如qps是进行累加，耗时是采用平均值等。所以单机的监控只需要对这些单机上报的监控数据和汇总聚合的全局监控数据进行独立存储，单机数据可以根据ip维度进行存储，然后通过api层通过单机ip就能查询到了。

钢
2019-10-10

个人觉得不可用，调用环节已经断层

展开

作者回复: trace多个span的数据组织上本身是一个DAG，当某一个链路的被调用方没有接入trace时，只是从这个分支往下的链路中断了，一般不影响整体trace的呈现，当前相关的这个span也只是没有 ss和sr部分，整体上也是可看。