美团点评技术团队

深度剖析开源分布式监控CAT

尤勇 ·2016-10-23 14:00

CAT（Central Application Tracking）是一个实时和接近全量的监控系统，它侧重于对Java应用的监控，基本接入了美团点评上海侧所有核心应用。目前在中间件（MVC、RPC、数据库、缓存等）框架中得到广泛应用，为美团点评各业务线提供系统的性能指标、健康状况、监控告警等。自2014年开源以来，除了美团点评之外，CAT还在携程、陆金所、猎聘网、找钢网等多家互联网公司生产环境应用，项目的开源地址是 http://github.com/dianping/cat (http://github.com/dianping/cat)。

本文会对CAT整体设计、客户端、服务端等的一些设计思路做详细深入的介绍。

背景介绍

CAT整个产品研发是从2011年底开始的，当时正是大众点评从.NET迁移到Java的核心起步阶段。当初大众点评已经有核心的基础中间件、RPC组件Pigeon、统一配置组件Lion。整体Java迁移已经在服务化的路上。随着服务化的深入，整体Java在线上部署规模逐渐变多，同时，暴露的问题也越来越多。典型的问题有：

- 大量报错，特别是核心服务，需要花很久时间才能定位。
- 异常日志都需要线上权限登陆线上机器排查，排错时间长。
- 有些简单的错误定位都非常困难（一次将线上的库配置到了Beta，花了整个通宵排错）。
- 很多不了了之的问题怀疑是网络问题（从现在看，内网真的很少出问题）。

虽然那时候也有一些简单的监控工具（比如Zabbix，自己研发的Hawk系统等），可能单个工具在某方面的功能还不错，但整体服务化水平参差不齐、扩展能力相对较弱，监控工具间不能互通互联，使得查找问题根源基本都需要在多个系统之间切换，有时候真的是靠“人品”才能找出根源。

适逢在eBay工作长达十几年的吴其敏加入大众点评成为首席架构师，他对eBay内部应用非常成功的CAL系统有深刻的理解。就在这样天时地利人和的情况下，我们开始研发了大众点评第一代监控系统——CAT。

CAT的原型和理念来源于eBay的CAL系统，最初是吴其敏在大众点评工作期间设计开发的。他之前的CAT不但增强了CAL系统核心模型，还添加了更丰富的报表

前者CAT不仅增强了CAL系统核心模型，还添加了更丰富的报表。

整体设计

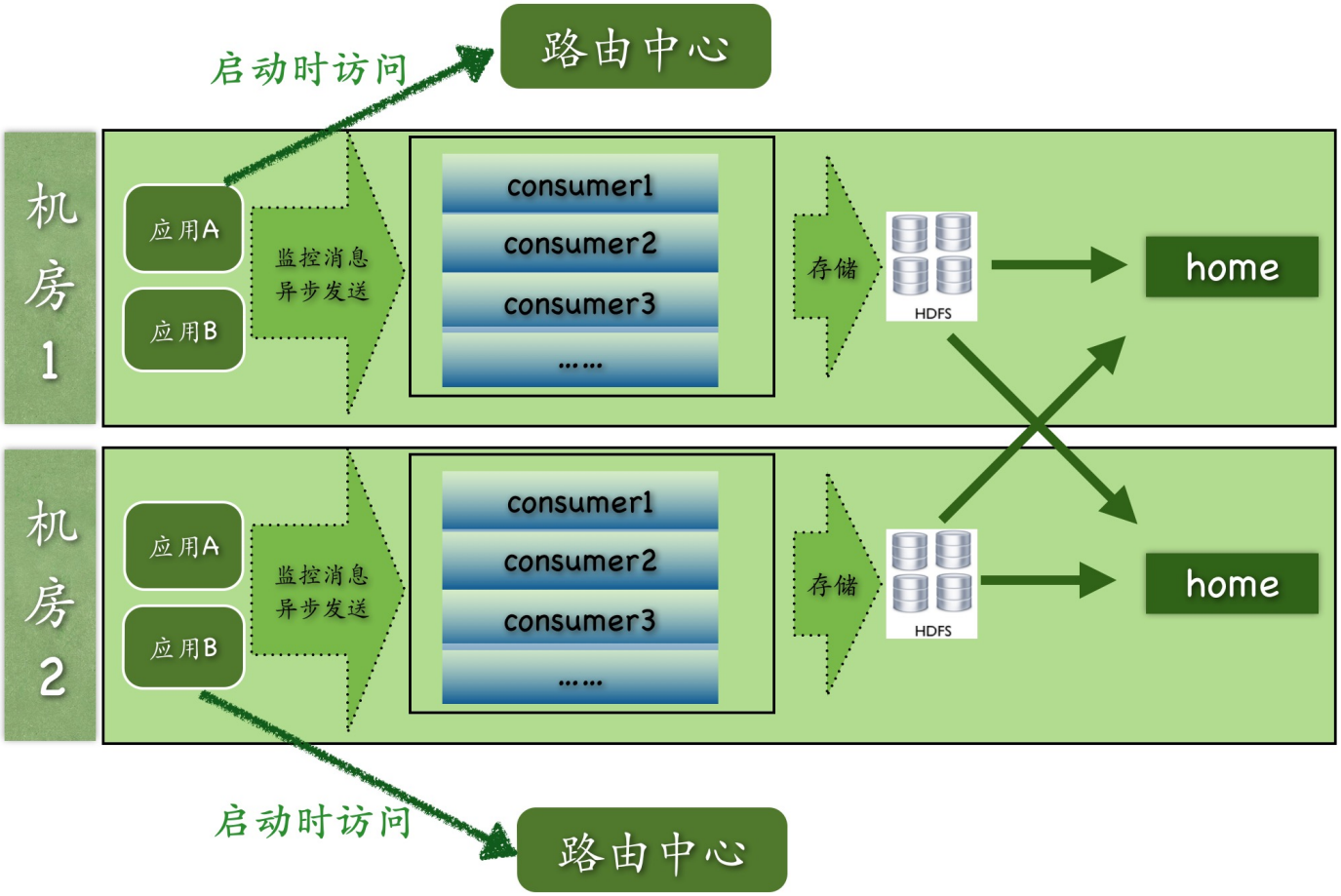
监控整体要求就是快速发现故障、快速定位故障以及辅助进行程序性能优化。为了做到这些，我们对监控系统的一些非功能做了如下的要求：

- 实时处理：信息的价值会随时间锐减，尤其是事故处理过程中。
- 全量数据：最开始的设计目标就是全量采集，全量的好处有很多。
- 高可用：所有应用都倒下了，需要监控还站着，并告诉工程师发生了什么，做到故障还原和问题定位。
- 故障容忍：CAT本身故障不应该影响业务正常运转，CAT挂了，应用不该受影响，只是监控能力暂时减弱。
- 高吞吐：要想还原真相，需要全方位地监控和度量，必须要有超强的处理吞吐能力。
- 可扩展：支持分布式、跨IDC部署，横向扩展的监控系统。
- 不保证可靠：允许消息丢失，这是一个很重要的trade-off，目前CAT服务端可以做到4个9的可靠性，可靠系统和不可靠性系统的设计差别非常大。

CAT从开发至今，一直秉承着**简单的架构就是最好的架构**原则，主要分为三个模块：CAT-client、CAT-consumer、CAT-home。

- Cat-client 提供给业务以及中间层埋点的底层SDK。
- Cat-consumer 用于实时分析从客户端提供的数据。
- Cat-home 作为用户给用户提供展示的控制端。

在实际开发和部署中，Cat-consumer和Cat-home是部署在一个JVM内部，每个CAT服务端都可以作为consumer也可以作为home，这样既能减少整个层级结构，也可以增加系统稳定性。



上图是CAT目前多机房的整体结构图，图中可见：

- 路由中心是根据应用所在机房信息来决定客户端上报的CAT服务端地址，目前美团点评有广州、北京、上海三地机房。
- 每个机房内部都有独立的原始信息存储集群HDFS。
- CAT-home可以部署在一个机房也可以部署在多个机房，在最后做展示的时候，home会从consumer中进行跨机房的调用，将所有的数据合并展示给用户。
- 实际过程中，consumer、home以及路由中心都是部署在一起的，每个服务端节点都可以充当任何一个角色。

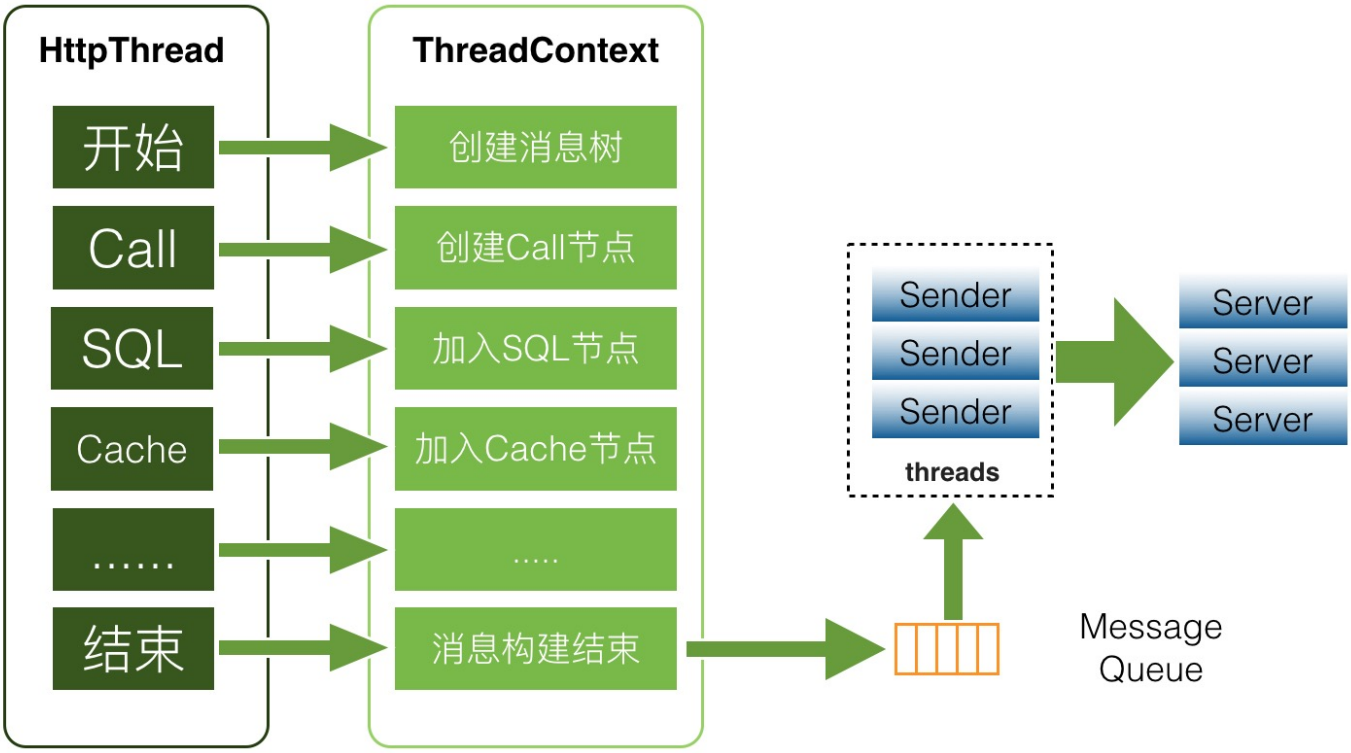
客户端设计

客户端设计是CAT系统设计中最为核心的一个环节，客户端要求是做到API简单、高可靠性能，无论在任何场景下都不能影响客业务性能，监控只是公司核心业务流程一个旁路环节。CAT核心客户端是Java，也支持Net客户端，近期公司内部也在研发其他多语言客户端。以下客户端设计及细节均以Java客户端为模板。

设计架构

CAT客户端在收集端数据方面使用ThreadLocal（线程局部变量），是线程本地变量，也可以称之为线程本地存储。其实ThreadLocal的功用非常简单，就是为每一个使用该变量的线程都提供一个变量值的副本，属于Java中一种较为特殊的线程绑定机制，每一个线程都可以独立地改变自己的副本，不会和其它线程的副本冲突。

在监控场景下，为用户提供服务都是Web容器，比如tomcat或者Jetty，后端的RPC服务端比如Dubbo或者Pigeon，也都是基于线程池来实现的。业务方在处理业务逻辑时基本都是在一个线程内部调用后端服务、数据库、缓存等，将这些数据拿回来再进行业务逻辑封装，最后将结果展示给用户。所以将所有的监控请求作为一个监控上下文存入线程变量就非常合适。



如上图所示，业务执行业务逻辑的时候，就会把此次请求对应的监控存放于线程上下文中，存于上下文的其实是一个监控树的结构。在最后业务线程执行结束时，将监控对象存入一个异步内存队列中，CAT有个消费线程将队列内的数据异步发送到服务端。

API设计

监控API定义往往取决于对监控或者性能分析这个领域的理解，监控和性能分析所针对的场景有如下几种：

- 一段代码的执行时间，一段代码可以是URL执行耗时，也可以是SQL的执行耗时。
- 一段代码的执行次数，比如Java抛出异常记录次数，或者一段逻辑的执行次数。
- 定期执行某段代码，比如定期上报一些核心指标：JVM内存、GC等指标。
- 关键的业务监控指标，比如监控订单数、交易额、支付成功率等。

在上述领域模型的基础上，CAT设计自己核心的几个监控对象：Transaction、Event、Heartbeat、Metric。

一段监控API的代码示例如下：

```
public void sample() {
    String pageName = "";
    String serverIp = "";

    Transaction t = Cat.newTransaction("URL", pageName); // 创建一个Transaction

    try {
        // 记录一个事件
        Cat.logEvent("URL.Server", serverIp, Event.SUCCESS, "ip=" + serverIp + "&...");
        // 记录一个业务指标，主要衡量单位时间内的次数总和
        Cat.logMetricForCount("OrderCount");
        // 记录一个timer类的业务指标，主要衡量单位时间内平均值
        Cat.logMetricForDuration("KeyForTimer", 5);

        yourBusiness();// 自己业务代码

        t.setStatus(Transaction.SUCCESS);// 设置状态
    } catch (Exception e) {
        t.setStatus(e);// 设置错误状态
    } finally {
        t.complete();// 结束Transaction
    }
}
```

序列化和通信

序列化和通信是整个客户端包括服务端性能里面很关键的一个环节。

- CAT序列化协议是自定义序列化协议，自定义序列化协议相比通用序列化协议要高效很多，这个在大规模数据实时处理场景下还是非常有必要的。
- CAT通信是基于Netty来实现的NIO的数据传输，Netty是一个非常好的NIO开发框架，在这边就不详细介绍了。

客户端埋点

日志埋点是监控活动的最重要环节之一，日志质量决定着监控质量和效率。当前CAT的埋点目标是以问题为中心，像程序抛出exception就是典型问题。我个人对问题的定义是：不符合预期的就可以算问题，比如请求未完成、响应时间快了慢了、请求TPS多了少了、时间分布不均匀等等。

在互联网环境中，最突出的问题场景，突出的理解是：跨越边界的行为。包括但不限于：

- HTTP/REST、RPC/SOA、MQ、Job、Cache、DAL;

- 搜索/查询引擎、业务应用、外包系统、遗留系统;
- 第三方网关/银行, 合作伙伴/供应商之间；
- 各类业务指标，如用户登录、订单数、支付状态、销售额。

遇到的问题

通常Java客户端在业务上使用容易出问题的地方就是内存，另外一个是CPU。内存往往是内存泄露，占用内存较多导致业务方GC压力增大； CPU开销最终就是看代码的性能。

以前我们遇到过一个极端的例子，我们一个业务请求做餐饮加商铺的销售额，业务一般会通过for循环所有商铺的分店，结果就造成内存OOM了，后来发现这家店是肯德基，有几万分店，每个循环里面都会有数据库连接。在正常场景下，ThreadLocal内部的监控一个对象就存在几万个节点，导致业务Oldgc特别严重。所以说框架的代码是不能想象业务方会怎么用你的代码，需要考虑到任何情况下都有出问题的可能。

在消耗CPU方面我们也遇到一个case：在某个客户端版本，CAT本地存储当前消息ID自增的大小，客户端使用了MappedByteBuffer这个类，这个类是一个文件内存映射，测试下来这个类的性能非常高，我们仅仅用这个存储了几个字节的对象，正常情况理论上不会有任何问题。在一次线上场景下，很多业务线程都block在这个上面，结果发现当本身这台机器IO存在瓶颈时候，这个也会变得很慢。后来的优化就是把这个IO的操作异步化，所以**客户端需要尽可能异步化，异步化序列化、异步化传输、异步化任何可能存在时间延迟的代码操作。**

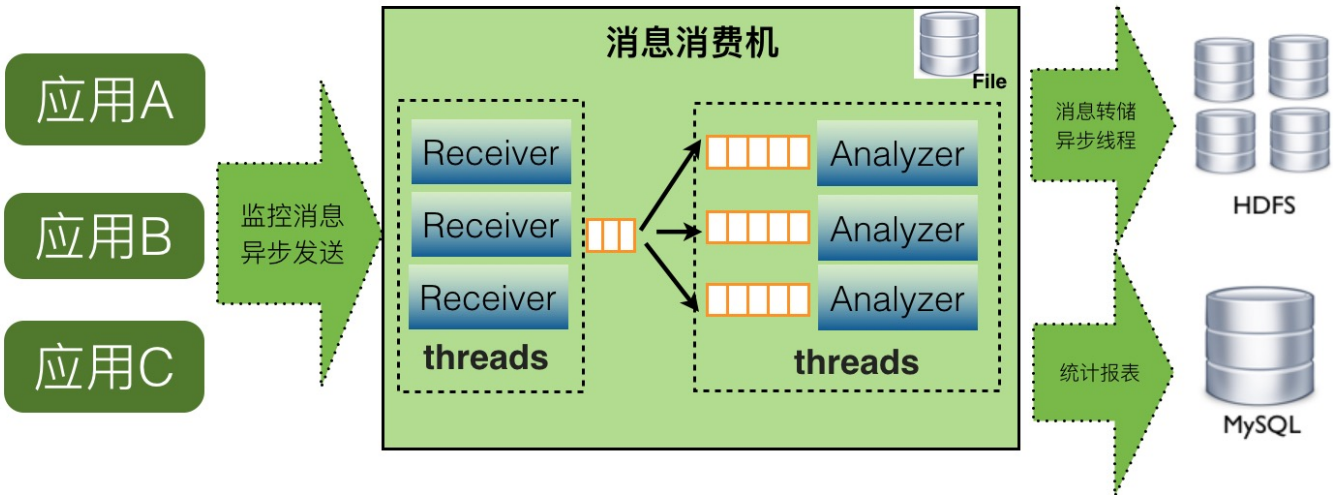
服务端设计

服务端主要的问题是大数据的实时处理，目前后端CAT的计算集群大约35台物理机，存储集群大约35台物理机，每天处理了约100TB的数据量。线上单台机器高峰期大约是110MB/s，接近千兆网打满。

下面我重点讲下CAT服务端一些设计细节。

架构设计

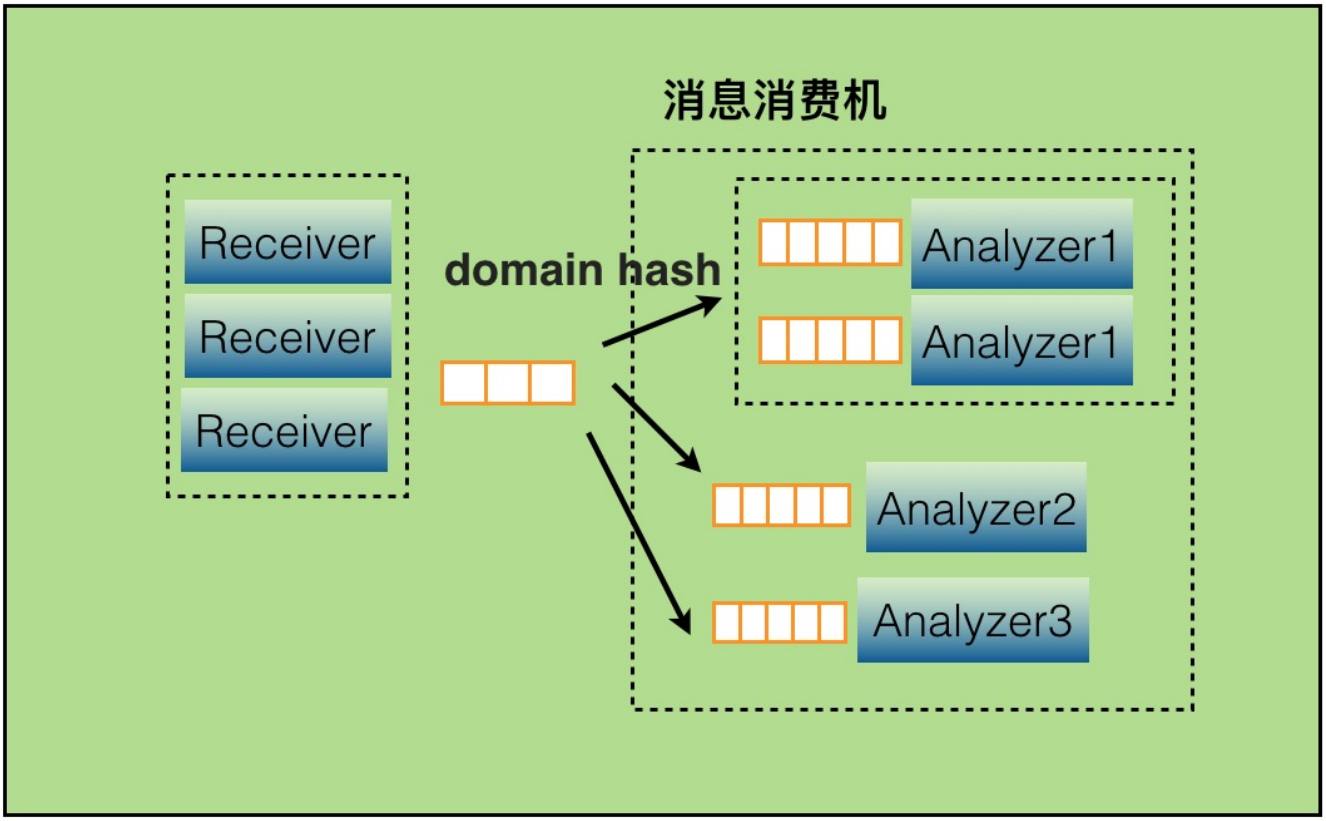
在最初的整体介绍中已经画了架构图，这边介绍下单机的consumer中大概的结构如下：



如上图，CAT服务端在整个实时处理中，基本上实现了全异步化处理。

- 消息接受是基于Netty的NIO实现。
- 消息接受到服务端就存放内存队列，然后程序开启一个线程会消费这个消息做消息分发。
- 每个消息都会有一批线程并发消费各自队列的数据，以做到消息处理的隔离。
- 消息存储是先存入本地磁盘，然后异步上传到HDFS文件，这也避免了强依赖HDFS。

当某个报表处理器处理来不及时候，比如Transaction报表处理比较慢，可以通过配置支持开启多个Transaction处理线程，并发消费消息。



实时分析

CAT服务端实时报表分析是整个监控系统的核心，CAT重客户端采集的是是原始的logview，目前一天大约有1000亿的消息，这些原始的消息太多了，所以需要在这些消息基础上实现丰富报表，来支持业务问题及性能分析的需要。

CAT是根据日志消息的特点(比如只读特性)和问题场景，量身定做的，它将所有的报表按消息的创建时间，一小时为单位分片，那么每小时就产生一个报表。当前小时报表的所有计算都是基于内存的，用户每次请求即时报表得到的都是最新的实时结果。对于历史报表，因为它是不变的，所以实时不实时也就无所谓了。

CAT基本上所有的报表模型都可以增量计算，它可以分为：计数、计时和关系处理三种。计数又可以分为两类：算术计数和集合计数。典型的算术计数如：总个数（count）、总和（sum）、均值（avg）、最大/最小（max/min)、吞吐（tps）和标准差（std）等，其他都比较直观，标准差稍微复杂一点，大家自己可以推演一下怎么做增量计算。那集合运算，比如95线（表示95%请求的完成时间）、999线（表示99.9%请求的完成时间），则稍微复杂一些，系统开销也更大一点。

报表建模

CAT每个报表往往有多个维度，以transaction报表为例，它有5个维度，分别是应用、机器、Type、Name和分钟级分布情况。如果全维度建模，虽然灵活，但开销将会非常之大。CAT选择固定维度建模，可以理解成将这5个维度组织成深度为5的树，访问时总是从根开始，逐层往下进行。

CAT服务端为每个报表单独分配一个线程，所以不会有锁的问题，所有报表模型都是非线程安全的，其数据是可变的。这样带来的好处是简单且低开销。

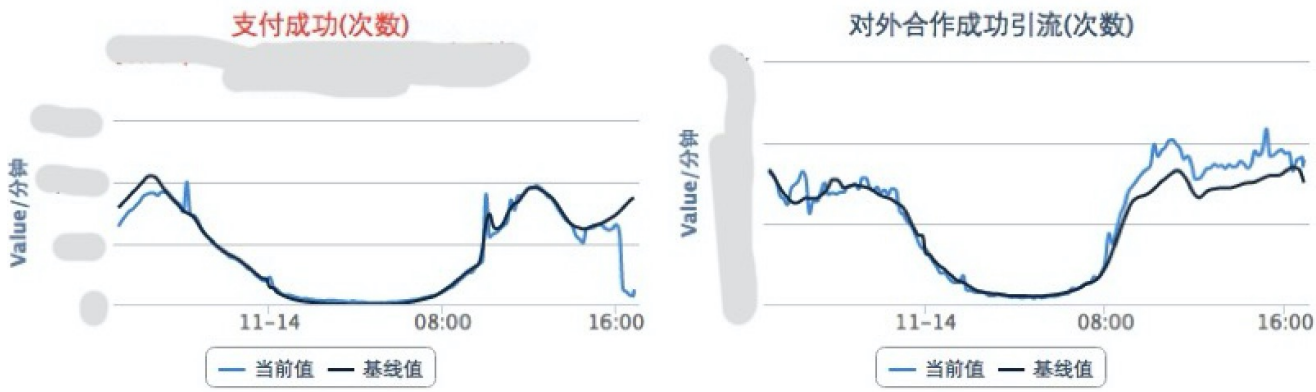
CAT报表建模是使用自研的Maven Plugin自动生成的。所有报表是可合并和裁剪的，可以轻易地将2个或多个报表合并成一个报表。在报表处理代码中，CAT大量使用访问者模式（visitor pattern）。

性能分析报表

报表	说明
Transaction	一段代码运行时间、次数
Event	一行代码的执行次数
Problem	系统可能出现的异常，包括访问较慢的程序等
Hearbeat	JVM内部一些状态信息，Memory，Thread等
Matrix	一个请求调用链路统计
RPC	SOA系统用关于RPC调用的报表
Cache	缓存使用分析统计
Dependency	系统之间实时调用依赖关系等
…	…

故障发现报表

- 实时业务指标监控 ：核心业务都会定义自己的业务指标，这不需要太多，主要用于24小时值班监控，实时发现业务指标问题，图中一个是当前的实际值，一个是基准值，就是根据历史趋势计算的预测值。如下图就是当时的情景，能直观看到支付业务出问题的故障。



- 系统报错大盘。
- 实时数据库大盘、服务大盘、缓存大盘等。

存储设计

CAT系统的存储主要有两块：

- CAT的报表的存储。
- CAT原始logview的存储。

报表是根据logview实时运算出来的给业务分析用的报表，默认报表有小时模式、天模式、周模

式以及月模式。CAT实时处理报表都是产生小时级别统计，小时级报表中会带有最低分钟级别粒度的统计。天、周、月等报表都是在小时级别报表合并的结果报表。

原始logview存储一天大约100TB的数据量，因为数据量比较大所以存储必须要要压缩，本身原始logview需要根据Message-ID读取，所以存储整体要求就是批量压缩以及随机读。在当时场景下，并没有特别合适成熟的系统以支持这样的特性，所以我们开发了一种基于文件的存储以支持CAT的场景，在存储上一直是最难的问题，我们一直在这块持续的改进和优化。

消息ID的设计

CAT每个消息都有一个唯一的ID，这个ID在客户端生成，后续都通过这个ID在进行消息内容的查找。典型的RPC消息串起来的问题，比如A调用B的时候，在A这端生成一个Message-ID，在A调用B的过程中，将Message-ID作为调用传递到B端，在B执行过程中，B用context传递的Message-ID作为当前监控消息的Message-ID。

CAT消息的Message-ID格式ShopWeb-0a010680-375030-2，CAT消息一共分为四段：

- 第一段是应用名shop-web。
- 第二段是当前这台机器的IP的16进制格式，01010680表示10.1.6.108。
- 第三段的375030，是系统当前时间除以小时得到的整点数。
- 第四段的2，是表示当前这个客户端在当前小时的顺序递增号。

存储数据的设计

消息存储是CAT最有挑战的部分。关键问题是消息数量多且大，目前美团点评每天处理消息1000亿左右，大小大约100TB，单物理机高峰期每秒要处理100MB左右的流量。CAT服务端基于此流量做实时计算，还需要将这些数据压缩后写入磁盘。

整体存储结构如下图：