

# **AppTrace Server 应用业务性能管理**

## **技术白皮书 v2.1.0**

时间 : 2018 年 09 月 03 日

单位 : 神州灵云 ( 北京 ) 科技有限公司

1.前言 .....	4
2.AppTrace 业务性能解决方案 .....	5
3.部署示意 .....	5
3.1.数据采集 .....	5
3.2.资源消耗 .....	6
3.3.部署图示意 .....	6
4.核心功能 .....	7
4.1.业务分析 .....	7
4.1.1    业务请求建模 .....	7
4.1.2    业务组合分析 .....	9
4.1.3    业务错误定位 .....	11
4.1.4    快速排查分钟内的业务请求 .....	12
4.2.SQL 分析 .....	14
4.3.程序异常 .....	16
4.4.线程剖析 .....	17
4.5.服务器 .....	19
4.6.Agent 动态配置 .....	20
4.7.智能告警 .....	20
4.2.1    自定义精准告警 .....	20
4.2.2    实时告警通知 .....	22
4.2.3    告警信息快速分析 .....	23
4.8.应用报表 .....	24

5.产品理念 .....	28
5.1.业务精益管理，持续优化 .....	28
5.2.三层维度覆盖解析应用系统 .....	29
5.3.数据自动关联 .....	31
5.4.端到端关联分析 .....	33
5.4.1.用户统计 .....	33
5.4.2.业务全链路分析 .....	35
5.5.客户端随时关注应用状态 .....	36
6.应用场景 .....	39
6.1.30 秒定位问题根源 .....	39
6.2.业务端到端关联分析 .....	41
6.3.持续业务优化 .....	42
6.4.突发问题处理 .....	46
7.技术指标 .....	48
7.1.运行环境 .....	48
7.2.数据精度 .....	48

# 1. 前言

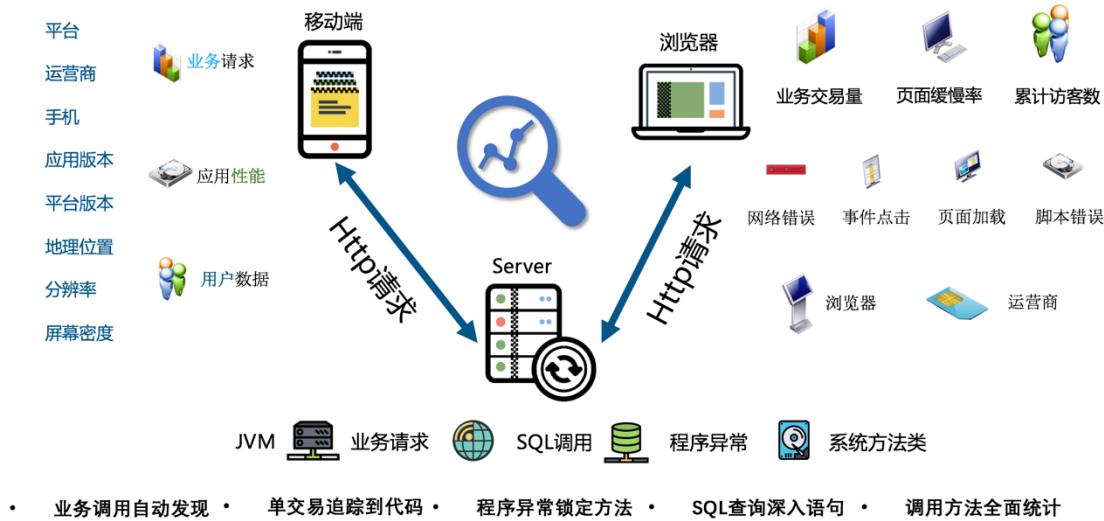
随着电脑的普及和不断深入人民的生活，上网的人们不断的增多，对网络的好奇和各种需求不断扩大，国内迎来了一波又一波的互联网创业潮，在 PC 端创业不断摸索前进中，又迎来了移动互联网的变革，一批批的移动互联网的创业者纷纷的投入到这个大潮中。时到如今国内的 IT 环境已经发生了翻天覆地的变化，琳琅满目的互联网产品到处都是，一个行业大大小小拥挤着几十个甚至几百个产品的存在，在这种环境下大家从争夺需求、圈用户的时期已经渡过，随着性能体验、响应时间、请求缓慢等代表着产品性能指标的名词不断流入我们的互联网行业中，产品在用户体验、产品性能的稳定性等上面的竞争已经拉开帷幕。

很难想象一款体验极差的产品如何在互联网行业中生存，如今的生活中各类同质化产品到处都是，不会那一个产品拥有着同类产品无法提供服务和功能，在这样的环境下如何才能让产品在同类产品中赢得优势，所以应该从自身的产品性能出发，通过性能的不断优化来提高用户体验从而实现产品的竞争优势。

AppTrace 应用性能管理平台是神州灵云专门为企业用户提供产品性能优化解决方案的平台，平台包含了 Mobile、Browser、Server 多端产品服务，不仅仅满足了各端性能优化的功能数据需求，而且在传统 APM 的基础上做到两大突破，一做到了真正的业务监控分析，二多维度数据自动关联分析。通过这两大突破，不仅帮助企业产品的性能优化，而且帮助他们实现业务交易和流程的梳理，并解决前后端业务无法关联分析的困境。

## 2.AppTrace 业务性能解决方案

AppTrace 是下一代智能易用的应用业务性能管理解决方案，立足业务，关注用户体验，从业务系统的核心组件包括移动、浏览器以及服务器端，自动获取业务运行相关的性能数据并实时计算多达几十种 KPI，实现对业务系统 360 度无死角监控，在建模业务的基础上实现分钟级故障定位、精准告警、实现应用，业务持续优化。



## 3. 部署示意

### 3.1. 数据采集

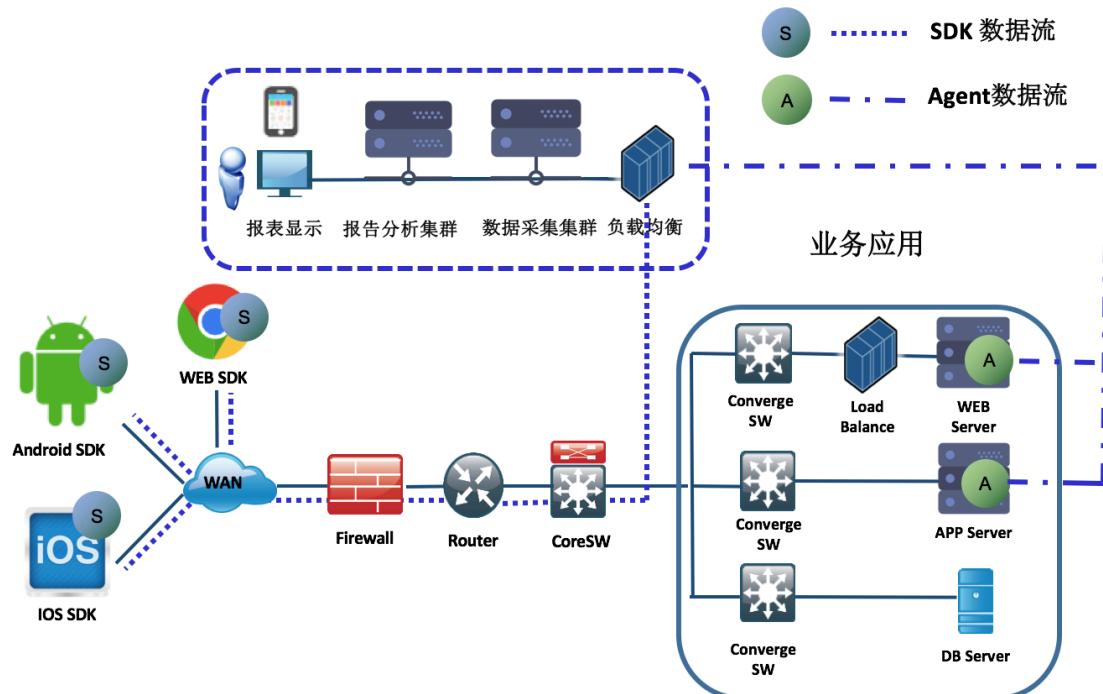
AppTrace 采用嵌码的方式进行数据采集，在移动应用嵌入 SDK 采集数据，浏览器应用嵌入 JS 采集数据，服务器应用嵌入 Agent 采集数据，采集的数据通过安全网络传输到服务器，在嵌入的过程中无须改动应用程序代码。通过嵌入式的采集方式实现代码级分析、SQL 脚本、服务等进行深入的分析，实现从用户端到服务层的针对用户真实行为的端到端应用性能监控。

## 3.2.资源消耗

AppTrace 的采集器包含移动 SDK、浏览器 JS、服务器 Agent 对应用的影响都在 3%以内。在同一测试环境下分别收集注入采集器的应用 cpu、内存和未注入采集器的应用 cpu、内存进行对比 得出采集器对应用的整体消耗低于 3%。

## 3.3.部署图示意

AppTrace “应用探针” 工作原理是对应用中的“真实生产数据”进行分析。从前端设备操作应用，点击按钮发出请求，后台服务器收到请求，服务器对请求做出处理，判断请求参数或报文内容做出操作动作，调用堆栈、数据库或访问第三方和缓存，AppTrace “应用探针” 统一采集处理上传到 AppTrace 数据中心进行处理展示。AppTrace 应用性能管理平台的部署方式如下图：



移动端：Android SDK、IOS SDK 负责移动端应用的应用性能、业务数据及用户行为的数据采集处理。

浏览器：Web SDK 负责浏览器端网站的网站性能、业务数据、页面处理及用户行为的数据采集处理。

服务器：Server Agent 负责服务器端服务器应用的服务器性能、请求数据处理及堆栈和 SQL 等数据的采集处理。

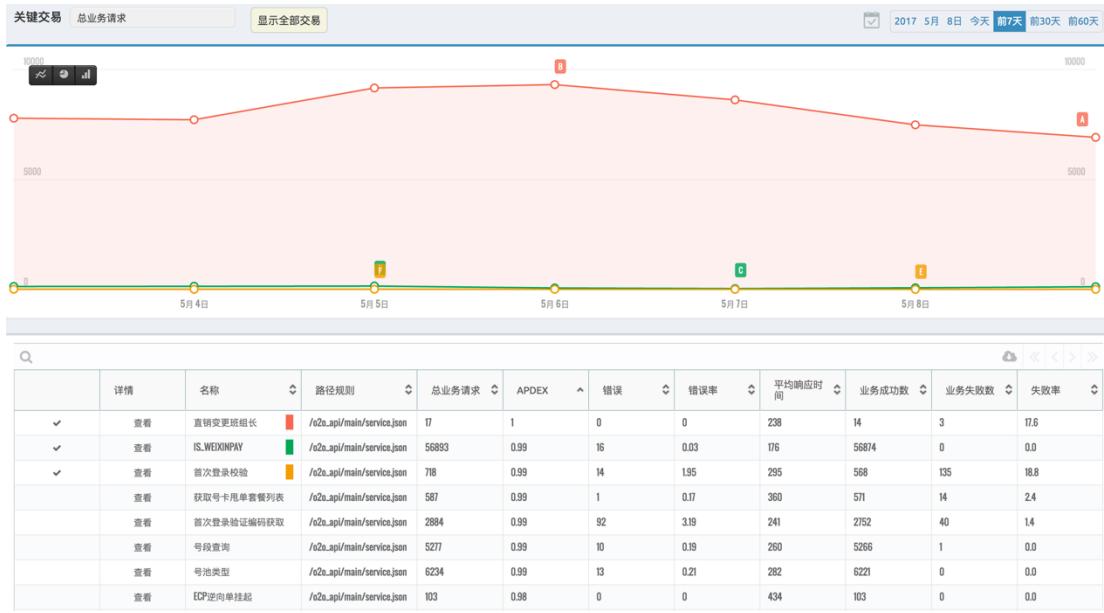
## 4.核心功能

### 4.1.业务分析

我们对业务的处理主要分三步法：一首先将业务请求跟业务指标一对一对应时间，二根据业务去做组合分析，找出业务之间的转换关系，三对业务中的错误异常进行纠正排查。前两步主要是帮助企业梳理和优化业务逻辑，第三步是帮助企业对应用业务发生的异常快速定位处理。

#### 4.1.1 业务请求建模

应用的业务通常利用 URL 请求来进行业务的处理，如何区分 URL 中的不同业务，我们有三步法，来确定 URL 所代表的具体业务。



## ● 一、确定唯一 URL 请求

如果要确定唯一的业务请求，必须要知道业务请求的服务器和路径，在我们的平台上对每一个 URL 请求做到全面采集，其中包含服务器和路径，并可以对 URL 请求进行路径搜索方便您快速找到需要的 URL 请求。

业务名称	服务器	路径
获取用户信息	192.168.1.32	/app/user/get-user
订单成功提交	192.168.1.32	/app/order
选择支付方式	192.168.1.32	/app/preorder/select-pay-type
预购订单详情	192.168.1.32	/app/preorder
加入预购订单	192.168.1.32	/app/preorder/add
成功加入购物车	192.168.1.32	/app/cart
产品列表	192.168.1.32	/app/product
登录	192.168.1.32	/app/user/login-post
加入购物车	192.168.1.32	/app/cart/add

## ● 二、锁定唯一业务请求

单凭服务器和 URL 路径还是不能做到确定唯一业务请求的，要想确定唯一的业务请求就要去深入到数据包中区分不同的内容来甄别具体的业务，我们已经解决了最后挡在我们前后的拦路虎，通过对报文信息的获取并分析为确定唯一业务请求奠定了基础。

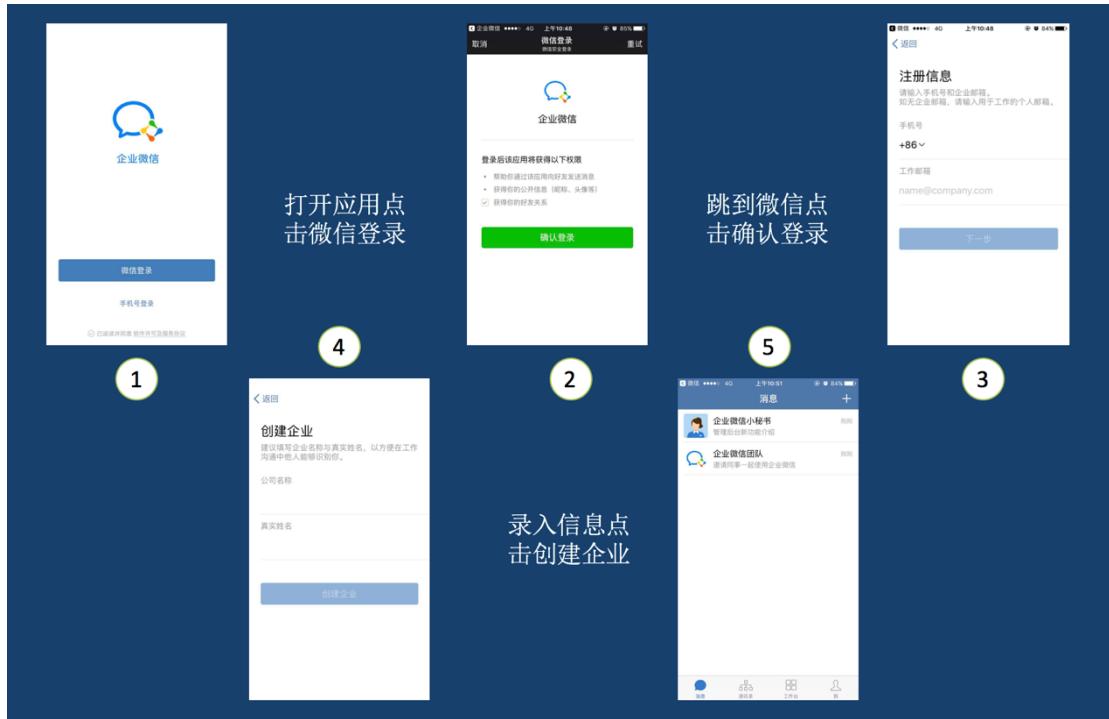
业务名称	URL	方法	事件请求	交易结果
获取用户信息	<a href="http://192.168.1.32:38080/app/user/get-user">http://192.168.1.32:38080/app/user/get-user</a>	POST	我的	成功
成功加入购物车	<a href="http://192.168.1.32:38080/app/cart">http://192.168.1.32:38080/app/cart</a>	POST	购物车	成功
加入预购订单	<a href="http://192.168.1.32:38080/app/preorder/add">http://192.168.1.32:38080/app/preorder/add</a>	POST	去结算(3)	
预购订单详情	<a href="http://192.168.1.32:38080/app/preorder/0d38fe4cc6d64c7baf24a967df547e89">http://192.168.1.32:38080/app/preorder/0d38fe4cc6d64c7baf24a967df547e89</a>	POST	填写订单	成功

### ● 三、模型规则算法最终完成业务锁定

理论上能够确定唯一业务了，但是一个应用每天发生着不可计数的请求，而且每一个应用中承载着几千几万个不同内容不同路径的请求，我们怎样快速通过一套规范化的流程就可以对这些 URL 请求进行处理，并对关心业务进行处理，我们就是独创了这套规范化的流程实现和简化了业务请求的快速锁定和处理。

#### 4.1.2 业务组合分析

下面图是企业版微信的首次登录路径，从打开应用到登录成功进入首页，共需经过 5 个步骤，一选择登录方式，二第三方确认登录，三录入手机和邮箱信息，四录入企业信息，五确认创建企业。这 5 个步骤组成了完整的首次登录业务，其中包含了多个唯一的业务请求，我在任何一个环节出现错误、异常或放弃对首次登录业务都是失败的。



将每一个单独业务组合在一起，构成完整的业务链，对于分析业务增长与下降有重要意义。以下面购物流程为例每一个环节的用户数量和请求数，一个环节到下一个环节的请求错误率和用户流失率，有了这个数据不仅对业务进行了梳理，而且可以分析那个环节错误率高导致流失率高，又或者是错误率不高流失率高是不是功能设计中存在问题。



在上图中发现“加入购物车订单”流失率最大，查看下面列表数据分析，“加入购物车订单”缓慢率达到 83%，平均请求时间 1871.3ms，很多用户无法忍耐长时间等待而放弃“加入购物车订单”。

	用户数	业务请求数	APDEX	失败率	缓慢率	错误率	平均响应时间
1.加入购物车	18(100%)	349(100%)	0.21	0%	21%	55.5%	803.5
2.加入预购订单	13(72.2%)	142(40.7%)	0.86	0%	83%	0%	1871.3
3.选择支付方式	10(55.6%)	72(20.6%)	0.99	0%	16.7%	0%	516.5
4.获取用户信息	11(61.1%)	148(42.4%)	0.96	17%	78.3%	0%	462.4

### 4.1.3 业务错误定位

URL 请求发生错误，我们在这里有单独的分析展示，可以通过网络错误与 HTTP 错误区分错误类型，并对发生错误的用户进行统计，实时了解请求错误都是什么类型和影响了多少用户。从应用层面对请求错误有一个量的评估。



对发生错误的请求支持从关键交易、出现次数、URL、方法、上次出现时间、应用版本进行筛选。从请求错误层面去发现错误的请求集中发生的关键交易、版本、时间、请求。

网络错误							
关键交易	出现次数	URL	方法	上次出现	应用版本	操作	筛选
文件上传	2162	http://202.98.113.246:6001/v2x_api/main/service.json	POST	3秒前	3.0		
记录RFID使用情况	538524	http://202.98.113.246:6001/v2x_api/main/service.json	POST	6秒前	3.0		
栏目配置查询	17440	http://202.98.113.246:6001/v2x_api/main/service.json	POST	1分钟前	3.01		
蓝牙设备接口	38063	http://202.98.113.246:6001/v2x_api/main/service.json	POST	2分钟前	3.01		
	13523	http://api.share.mob.com:80/log4	POST	2分钟前	3.01		
	26278	https://ddc.networkbench.com/uploadMobileData?version=2.19&token=gypQW3GItY4ff9jc3a	POST	2分钟前	3.01		
O2O.GET_FLOORINFO	76968	http://202.98.113.246:6001/v2x_api/main/service.json	POST	2分钟前	3.0		
登录验证接口	86982	http://202.98.113.246:6001/v2x_api/main/service.json	POST	2分钟前	3.0		
全屏弹窗消息	150596	http://202.98.113.246:8003/v2x_api/main/service.json	POST	2分钟前	3.0		
	73843	http://202.98.113.246:8003/v2x_api/main/service.json	GET	2分钟前	3.01		
	35716	http://wapsc.189.cn/njwebview/agressproxy/complete	GET	2分钟前	3.01		
版本升级	37538	http://202.98.113.246:6001/v2x_api/main/service.json	POST	2分钟前	3.01		

对单个业务请求的错误分析，每个错误发生的时间、用户、设备信息、触发的事件和状态码、请求报文、错误信息的统计分析，帮助企业快速了解错误发生的原因和位置。单个错误层面搜集错误信息。

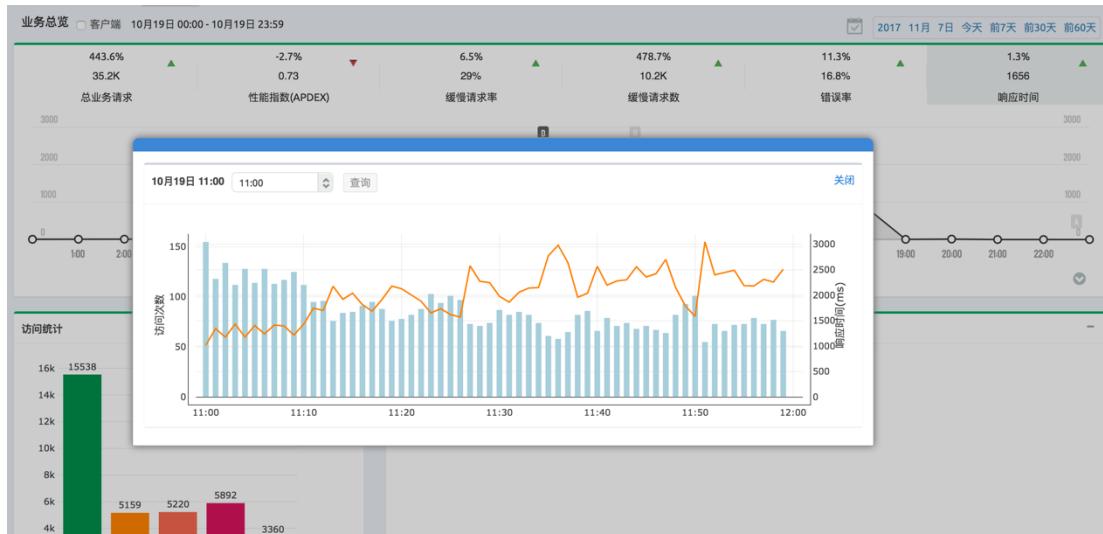
请求错误	URL	方法	状态码	设备	用户编号	应用版本	事件请求	关键交易
2017年05月11日 14:47:07	http://202.98.113.246:6001/o2o_api/main/service.json	POST	500	KW-CLD0 a6.0.1	50R	3.2	com.tydic.o2o.activity.AddImageNewActivity	文件上传
2017年05月11日 14:44:47	http://202.98.113.246:6001/o2o_api/main/service.json	POST	4001	SM-J319 a5.1.1	c0P	3.2	com.sec.android.app.camera.AttachActivity	文件上传
2017年05月11日 14:44:46	http://202.98.113.246:6001/o2o_api/main/service.json	POST	4001	SM-J319 a5.1.1	c0P	3.2	com.sec.android.app.camera.AttachActivity	文件上传
2017年05月11日 14:44:45	http://202.98.113.246:6001/o2o_api/main/service.json	POST	4001	SM-J319 a5.1.1	c0P	3.2	com.sec.android.app.camera.AttachActivity	文件上传
2017年05月11日 14:44:47	http://202.98.113.246:6001/o2o_api/main/service.json	POST	500	SM-J319 a5.1.1	c0P	3.2	com.sec.android.app.camera.AttachActivity	文件上传
2017年05月11日 14:41:05	http://202.98.113.246:6001/o2o_api/main/service.json	POST	4001	OPPO R7m a5.1.1	44I	3.2	com.tydic.o2o.activity.quickorder.QuickOrderActivity	文件上传
2017年05月11日 14:41:00	http://202.98.113.246:6001/o2o_api/main/service.json	POST	4002	vivo X5Max V a4.4.4	a4	3.2	com.tydic.o2o.activity.quickorder.QuickOrderActivity	文件上传

通过三个层面的错误统计，应用层得到错误的规模，请求错误层得到错误的规律，单个错误层得到错误信息，将三个汇总分析定位锁定错误的原因和错误解决方向。

#### 4.1.4 快速排查分钟内的业务请求

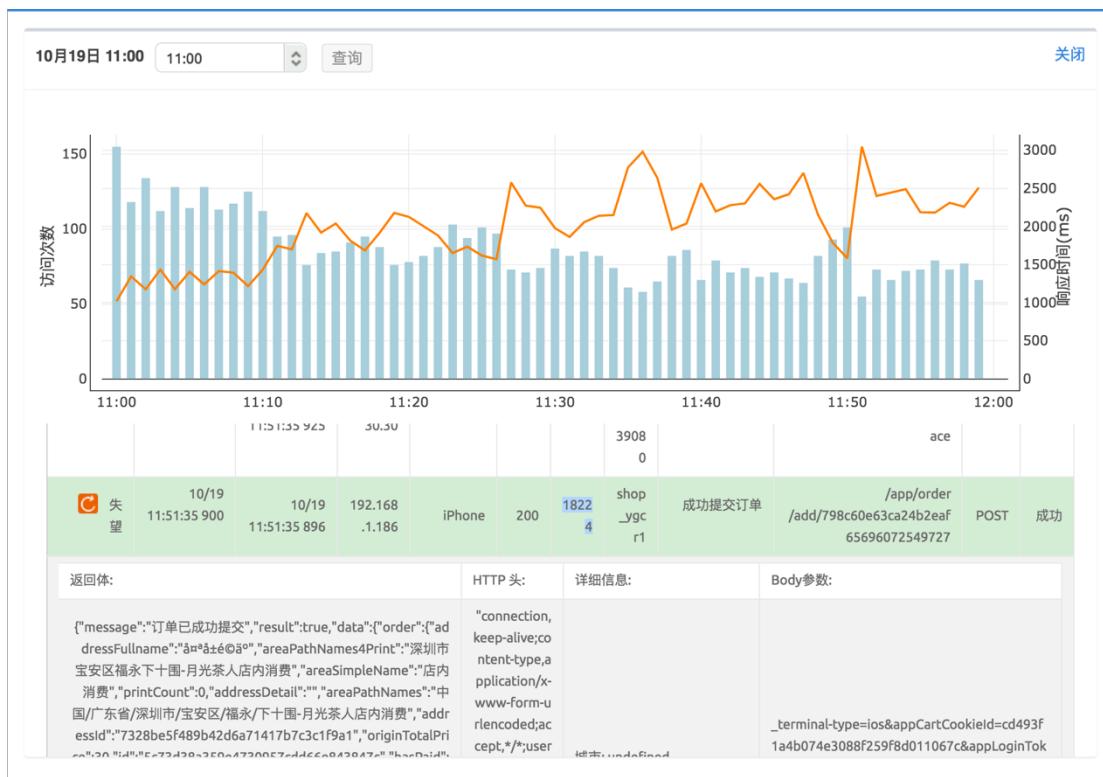


在业务总览中发现 10.19 日的响应时间高，通过观察折线图发现在 11 点的时候，响应时间达到了 2.3 秒，点击该小时的折线图，弹出 11 点的业务请求数和响应时间的对比图。



柱状图代表访问次数，折线图代表响应时间，通过访问次数与响应时间的对比，发现随着访问次数的减少响应时间在逐渐的升高，在 11:51 分达到最大值 3 秒，我们想要知道 11:51 是所有业务请求响应时间高 还是个别的或者是哪一个高。

点击 11:51 的柱状图或者折线图

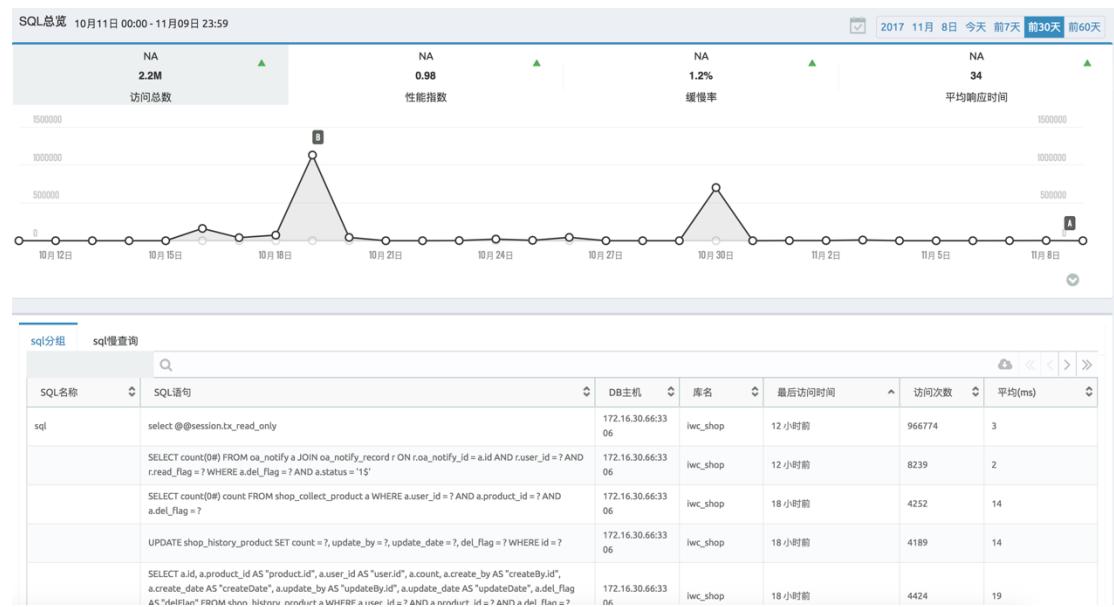


通过对 11:51 的发生的所有业务请求分析，发现 11 : 51 中并不是所有的业务响应时间都高，只是个别比较高，其中最高的是在 11:51 : 35 秒发生的“成功提

交订单”业务。这样快速分析问题的模式在业务详情中同样可行。

## 4.2.SQL 分析

SQL 调用可以通过建模的方式，将重要的 SQL 进行集中重点关注，通过中文的方式让其更容易监控与分析。



前 30 天中共调用了 2.2 万次数据库，性能指数为 0.98，缓慢率为 1.2%，平均响应时间是 34ms。点击列表中的 SQL 慢查询找到缓慢的 SQL 调用，并进行分析。

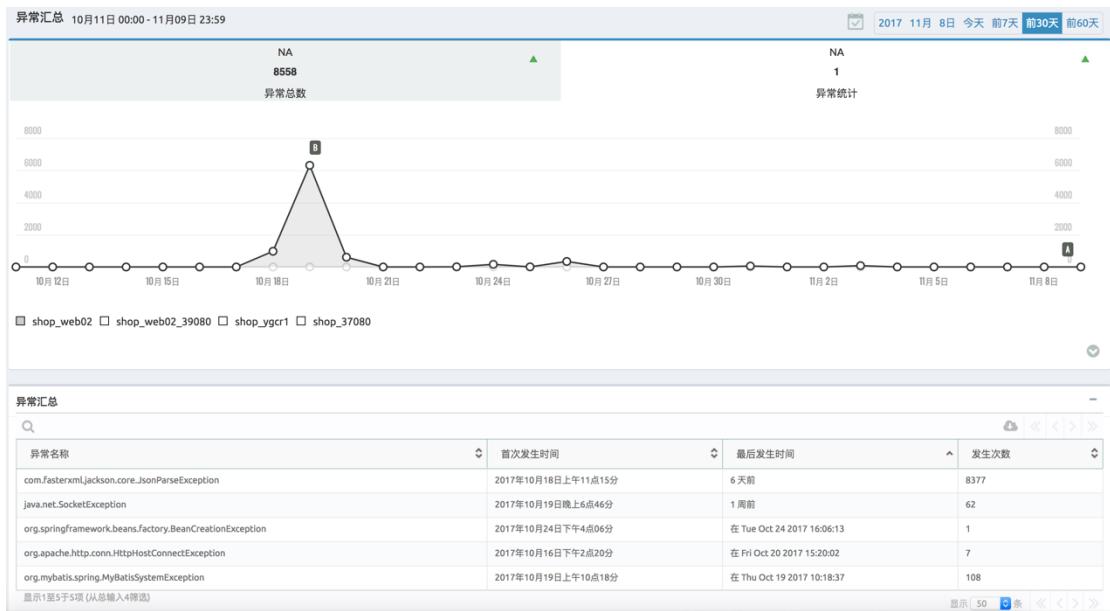
sql慢查询									
	SQL名称	响应时间	sql信息	请求时间	DB客户端	DB主机	库名	绑定的数据	
新增模型	14503	2017年10月19日 10:20:14	SELECT a.id AS "id", a.cart_id AS "cart_id", a.cookie_id AS "cookie_id", a.preorder_id AS "preorderId", a.area_id AS "area_id", a.area.name AS "areaName", a.area.parent_id AS "areaParentId", a.area.path_ids AS "areaPaths", a.area.path_names AS "areaPathNames", a.area.simple_name AS "areaSimpleName", a.area.zip_code AS "areaZipCode", a.store_id AS "storeId", a.ip AS "ip", a.total_count AS "totalCount", a.total_price AS "totalPrice", a.address_fullname AS "addressfullname", a.address.telephone AS "addressTelephone", a.address_detail AS "addressDetail", a.serial_no AS "serialNo", a.print_count AS "printCount", a.user_id AS "user.id", a.coupon_user_id AS "couponUser.id", a.coupon.user_total_price AS "couponUserTotalPrice", a.coupon.user_id AS "addressId", a.notice, a.has_paid AS "hasPaid", a.paid_date AS "paidDate", a.pay_type AS "payType", a.rough_pay_type AS "roughPayType", a.status_id AS "orderStatus.id", a.status_union AS "statusUnion", a.op.transaction_id AS "opTransactionId", a.min_total_price_label AS "minTotalPriceLabel", a.create_by AS "createBy.id", a.create_date AS "createDate", a.update_by AS "updateBy.id", a.update_date AS "updateDate", a.de_flag AS "deFlag" FROM shop_order WHERE 0=1 AND a.user_id = e19fd14f3ebf48bcbc79d09d6775ff04 AND (a.status_union = '25' OR a.status_union = '35' OR a.status_union = '45' OR a.status_union = '55') AND a.de_flag = 0 ORDER BY a.create_date DESC	交易分析	172.16.30.66:3306	shop_ygr1	iwc_shop	e19fd14f3ebf48bcbc79d09d6775ff04, 0	true
	11524	2017年10月30日 11:31:43	SELECT a.id AS "id", a.cart_id AS "cart_id", a.cookie_id AS "cookie_id", a.preorder_id AS "preorderId", a.area_id AS "area_id", a.area.name AS "areaName", a.area.parent_id AS "areaParentId", a.area.path_ids AS "areaPaths", a.area.path_names AS "areaPathNames", a.area.simple_name AS "areaSimpleName", a.area.zip_code AS "areaZipCode", a.store_id AS "storeId", a.ip AS "ip", a.total_count AS "totalCount", a.total_price AS "totalPrice", a.address_fullname AS "addressfullname", a.address.telephone AS "addressTelephone", a.address_detail AS "addressDetail", a.serial_no AS "serialNo", a.print_count AS "printCount", a.user_id AS "user.id", a.coupon_user_id AS "couponUser.id", a.coupon.user_total_price AS "couponUserTotalPrice", a.address_id AS "addressId", a.notice, a.has_paid AS "hasPaid", a.paid_date AS "paidDate", a.pay_type AS "payType", a.rough_pay_type AS "roughPayType", a.status_id AS "orderStatus.id", a.status_union AS "statusUnion", a.op.transaction_id AS "opTransactionId", a.min_total_price_label AS "minTotalPriceLabel", a.create_by AS "createBy.id", a.create_date AS "createDate", a.update_by AS "updateBy.id", a.update_date AS "updateDate" a.de_flag AS "deFlag" FROM shop_order WHERE 0=1 AND a.user_id = e19fd14f3ebf48bcbc79d09d6775ff04 AND (a.status_union = '25' OR a.status_union = '35' OR a.status_union = '45' OR a.status_union = '55') AND a.de_flag = 0 ORDER BY a.create_date DESC	交易分析	172.16.30.66:3306	shop_ygr1	iwc_shop	e19fd14f3ebf48bcbc79d09d6775ff04, 0	true

发现“新增模型”关键 SQL 比较缓慢，查看 SQL 信息内容排查缓慢原因，如果没有得到结论，可以点击列表中的交易分析，关键到该次 SQL 调用的业务中。

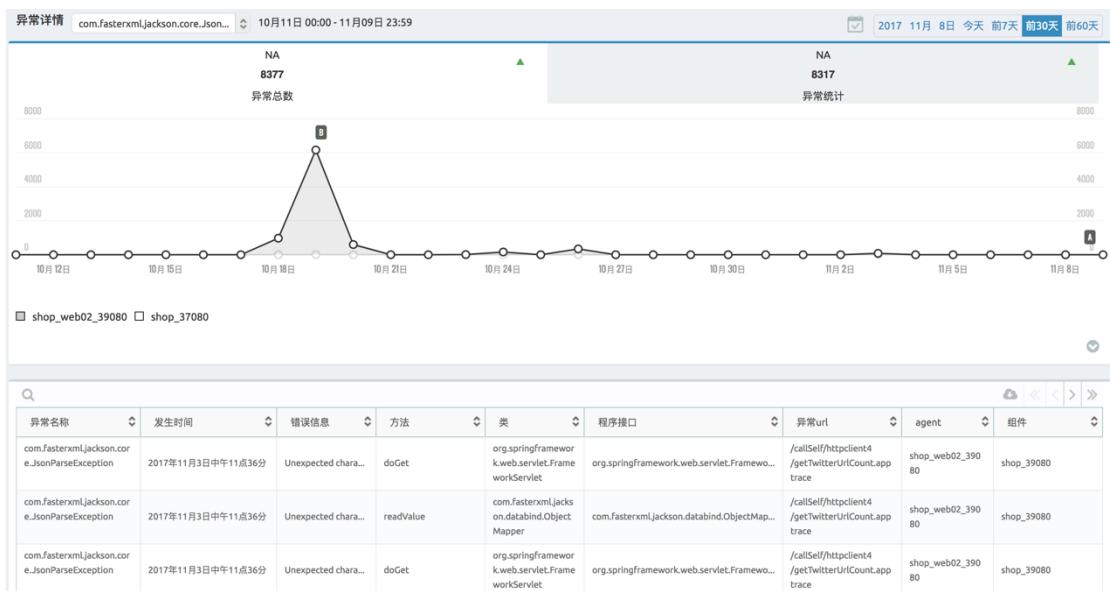
序号	方法	类	参数	开始时间	间隙	执行时间(m...)	执行时间(%)	自身>=
-	Tomcat Servlet Process		/app/product/60f09f82a8f948f9bfcce9917551973	15:41:00 457		179		0 TOMCAT
-	REMOTE_ADDRESS		192.168.1.166					
-	invoke(Request request, Response response)	org.apache.catalina.core.StandardHostV...		15:41:00 457	0	179		1 TOMCAT_ME...
-	@ http.requestHeader.uid		8CC1A155-9226-455D-B8DC-1F0997F11924					
-	@ http.requestHeader.id		M2017110815411f16aba1837df5079ce2d3f00d23e2bb829					
-	@ http.status.code		200					
-	@ http.method		POST					
1	doPost(HttpServletRequest request, HttpServletResponse response)	org.springframework.web.servlet.Frame...		15:41:00 458	1	178		9 SPRING_MVC
2	@ setAutoCommit(boolean autoCommitFlag)	com.mysql.jdbc.ConnectionImpl		15:41:00 462	4	0		0 MYSQL(iwc_...
-	@ args[]		null					
3	@ isAppLoggedIn(String userId, String token)	com.iwc.shop.modules.sys.service.User...		15:41:00 462	0	2		0 SPRING_BEAN
4	@ selectOne(String statement, Object parameter)	org.mybatis.spring.SqlSessionTemplate		15:41:00 462	0	2		1 MYBATIS
-	@ cached_args[]		com.iwc.shop.modules.sys.dao.UserDao.isAppLoggedIn					
5	@ prepareStatement(String sql)	com.mysql.jdbc.ConnectionImpl		15:41:00 463	1	0		0 MYSQL(iwc_...
6	@ execute()	com.mysql.jdbc.PreparedStatement		15:41:00 463	0	1		1 MYSQL_EXE...
-	@ SQL		SELECT COUNT(0) FROM sys_user WHERE id = ? AND e19fd14f3ebf48bcbc79d09d6775ff04, cda2fd42ea1d0a5d98					
7	@ SQL-BindValue			15:41:00 464	0	0		0 MYSQL(iwc_...
8	@ commit()	com.mysql.jdbc.ConnectionImpl		15:41:00 464	0	1		1 MYSQL(iwc_...
-	@ setAutoCommit(boolean autoCommitFlag)	com.mysql.jdbc.ConnectionImpl		15:41:00 464	0	1		
-	@ args[]		true					
9	@ executeQuery(String sql)	com.mysql.jdbc.StatementImpl		15:41:00 465	0	0		0 MYSQL_EXE...
-	@ SQL		select @@session.tx_read_only					
10	@ setAutoCommit(boolean autoCommitFlag)	com.mysql.jdbc.ConnectionImpl		15:41:00 466	1	0		0 MYSQL(iwc_...
-	@ args[]		null					
11	@ countByUserId(String userId, String isSelected)	com.iwc.shop.modules.shop.service.Car...		15:41:00 466	0	122		0 SPRING_BEAN

通过完整的业务调用链去分析慢调用的原因。

## 4.3.程序异常

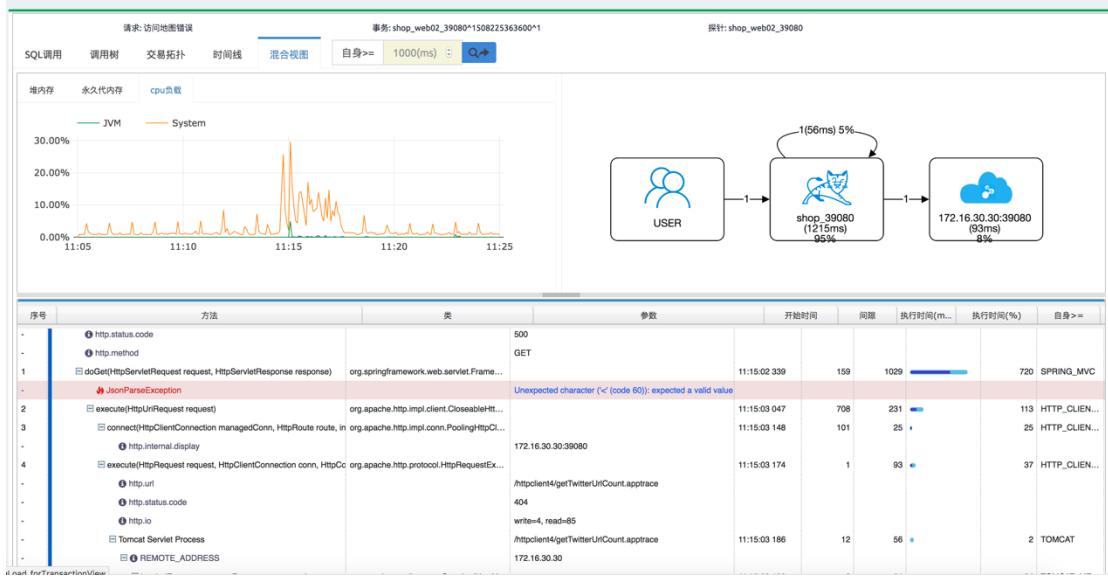


过去的 30 天内累计发生了 8558 次程序异常，通过选择不同的服务器查看每一个服务发生异常的数量。分析列表中的异常种类，发现列表第一条 com.fasterxml.jackson.core.JsonParseException 异常发生最多达到 8377 次，点击查看该异常的详情。



com.fasterxml.jackson.core.JsonParseException 异常都发生在 shop\_web02\_39080 这个服务器上，点击列表中的交易分析，查看异常发生时

的实际状态。



通过还原业务交易情况中异常的发生准确判断异常的原因。

## 4.4.线程剖析

采集选中服务器一段时间内的线程信息，分析服务器线程是否发生死锁，展示服务器拥有线程的信息，包括线程的堆栈信息和持有锁信息。



支持多个服务器同时采集，根据应用场景选择剖析时长。

探针名称	开始时间	剖析时长	采集次数	状态
shop_ygr1	2018年3月29日 15:59:31	1分钟	13	已完成
采集任务详情				
序号	采集时间	是否死锁	线程数	占用CPU时长
1	2018年3月29日 15:59:40	否	28	1847084.0毫秒
2	2018年3月29日 15:59:45	否	28	1522527.2毫秒
3	2018年3月29日 15:59:50	否	40	1946367.7毫秒
4	2018年3月29日 15:59:55	否	24	1149570.7毫秒
5	2018年3月29日 16:00:00	否	12	924114.7毫秒
6	2018年3月29日 16:00:05	否	30	1431572.9毫秒
7	2018年3月29日 16:00:10	否	6	417471.6毫秒
8	2018年3月29日 16:00:15	否	24	1307374.9毫秒
9	2018年3月29日 16:00:20	否	18	936373.7毫秒
10	2018年3月29日 16:00:25	否	24	1621251.3毫秒
11	2018年3月29日 16:00:30	否	42	2471389.7毫秒
12	2018年3月29日 16:00:35	否	12	581754.0毫秒
13	2018年3月29日 16:00:40	否	18	858379.3毫秒
shop_37080	2018年3月14日 14:44:09	2分钟	25	已完成
shop_37080	2018年3月14日 14:44:09	2分钟	25	已完成
shop_web02_39080	2018年3月13日 17:13:11	1分钟	13	已完成
shop_web02_39080	2018年3月13日 17:13:11	1分钟	13	已完成
shop_ygr1	2018年3月13日 15:53:10	5分钟	61	已完成
shop_ygr1	2018年3月12日 18:10:42	1分钟	13	已完成
shop_ygr1	2018年3月6日 15:07:52	2分钟	25	已完成

记录每次剖析的采集时间、是否死锁、采集到的线程数、线程占用 CPU 时长，将详情按钮查看每一次采集的详细数据。

探针名称	采集时间	线程数	是否死锁	占用CPU时长
shop_ygr1	2018年3月29日 15:59:40	28	否	1847084.0毫秒
显示1至1项				
显示 50 条 << < > >>				
<b>剖析列表</b> <input type="checkbox"/> 只显示耗时占比大于5%				
线程名称	线程组	占用CPU时长	线程状态	等待锁信息
%004fder%0053status%0050rocess.data	main	102.8毫秒	TIMED_WAITING	java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@3f143f93 Object@2de8af59
显示 50 条 << < > >>				
线程调用栈	方法	行号	源文件	
序号	类名			
0	sun.misc.Unsafe	park	-	
1	java.util.concurrent.locks.LockSupport	parkkhilos	215	
2	java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject	awaitNanos	2078	
3	java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue	take	1093	
4	java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue	take	809	
5	java.util.concurrent.ThreadPoolExecutor	getTask	1067	
6	java.util.concurrent.ThreadPoolExecutor	runWorker	1127	
7	java.util.concurrent.ThreadPoolExecutor\$Worker	run	617	
8	java.lang.Thread	run	745	
004fder%0053status%0050rocess.data	main	1017.5毫秒	TIMED_WAITING	java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@2de8af59 Object@2de8af59
				-
				5
				107

记录每一个线程的线程名称、线程组、占用 CPU 时长、线程状态、等待锁信息、持有锁信息、优先级、线程 ID，以及每一个线程的堆栈信息，堆栈类名、方法、行号、源文件。

## 4.5.服务器



服务器的资源是应用系统的基础，所以要时刻关注服务器的系统 CPU、JVM CPU、堆内存利用率、永久代内存率的资源变化。



在这里可以看到所有服务器的状态和信息，及时了解每个服务器的堆、内存使用量和 CPU 的状态，发生大规模异常时可以看看是不是某个服务器发生异常，导致无法正常提供服务，致使大规模业务失败。

## 4.6.Agent 动态配置

The screenshot shows the 'Probe Configuration' section of the AppTrace platform. On the left, a sidebar lists various probes: shop\_39080, aptrace30, shop\_web02\_39080 (selected), shop\_webbygr, aptrace30, shop\_ygcr1, test37080, aptrace30, and shop\_37080. The main area is divided into 'Data Reception' and 'Global Parameters'. In 'Data Reception', there are four sections: Collector (IP: 127.0.0.1, Port: 9996), Business Data (UDP) (IP: 127.0.0.1, Port: 9996), Status Data (UDP) (IP: 127.0.0.1, Port: 9995), and Template Data (TCP) (IP: 127.0.0.1, Port: 9994). In 'Global Parameters', settings include: '采样HTTP请求' (Sampling HTTP Requests) set to '是' (Yes), 'HTTP请求采样方式' (HTTP Request Sampling Method) set to '1'; '特定HTTP请求采集' (Specific HTTP Request Collection) set to '否' (No); 'URL路径' (URL Path); '探针版本' (Probe Version) set to '1.9.0'; '探针ID' (Probe ID) set to 'shop\_web02\_39080'; '应用名称' (Application Name) set to 'shop\_39080'; and '应用Key' (Application Key) set to '756edfdcd54b57f02b7ff3624f58e14ec6834784'.

通过 AppTrace 操作平台进行 Agent 动态配置功能，配置完成后自动下发到 Agent，无需重启服务器。可配置内容包括：数据接收参数、全局参数、用户类（POJO）参数、中间件参数。

## 4.7.智能告警

监控产品的核心能力之一，通过对应用用户、性能、业务的实时监控能够在发生问题时及时通知到应用管理者，并提供告警数据分析支持协助运维人员快速定位问题。AppTrace 平台通过对告警功能深度挖掘，设计出完善的告警体系，帮助企业快速得知异常信息，极速定位异常位置。

### 4.2.1 自定义精准告警

自定义告警不仅但从告警的组合上实现自定义，是真正的从根本实现告警的

自定义，从告警业务上支持单个业务的业务指标、性能指标告警设置，也可根据一类业务进行告警设置，同时可对整个业务的指标设置告警阀值。且可进行灵活的告警组合，不同指标组合，不同业务组合，不同告警方式组合。

- 真正基于业务告警

首先通过访问路径、服务器 IP 和报文信息锁定业务请求，然后对业务请求设定告警阀值，做到真正的基于业务告警。

业务名称	服务器	路径
成功加入购物车	192.168.1.32	/app/cart
获取用户信息	192.168.1.32	/app/user/get-user
/app/category/list-with-produc	192.168.1.32	/app/category/list-with-product
产品列表	192.168.1.32	/app/product
订单成功提交	192.168.1.32	/app/order
选择支付方式	192.168.1.32	/app/preorder/select-pay-type
预购订单详情	192.168.1.32	/app/preorder
加入预购订单	192.168.1.32	/app/preorder/add
登录	192.168.1.32	/app/user/login-post
加入购物车	192.168.1.32	/app/cart/add
/app/user/logout	192.168.1.32	/app/user/logout

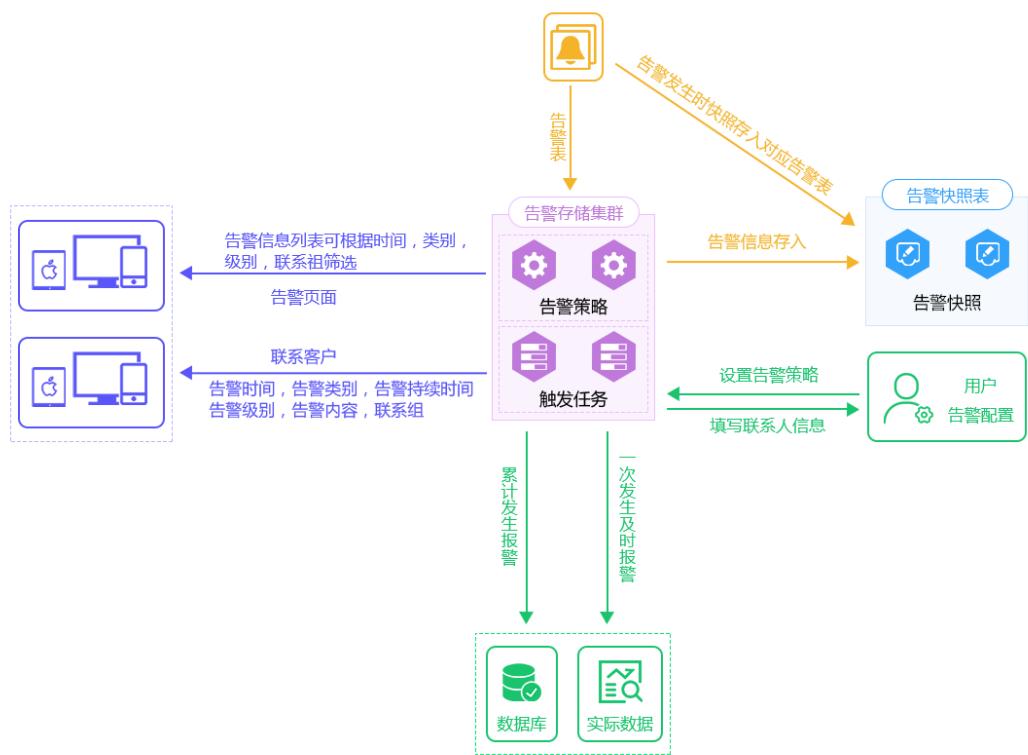
- 告警策略的灵活配置

将告警策略的组成部分模块化，告警监控指标与阀值设计为一个模块，告警通知为一个模块，告警名称与启用状态为一个模块，各模块间可以自由组合，但又各不影响。这种告警策略组合方式，可以根据告警需要去组合各种不同的告警策略，满足不同业务的使用。

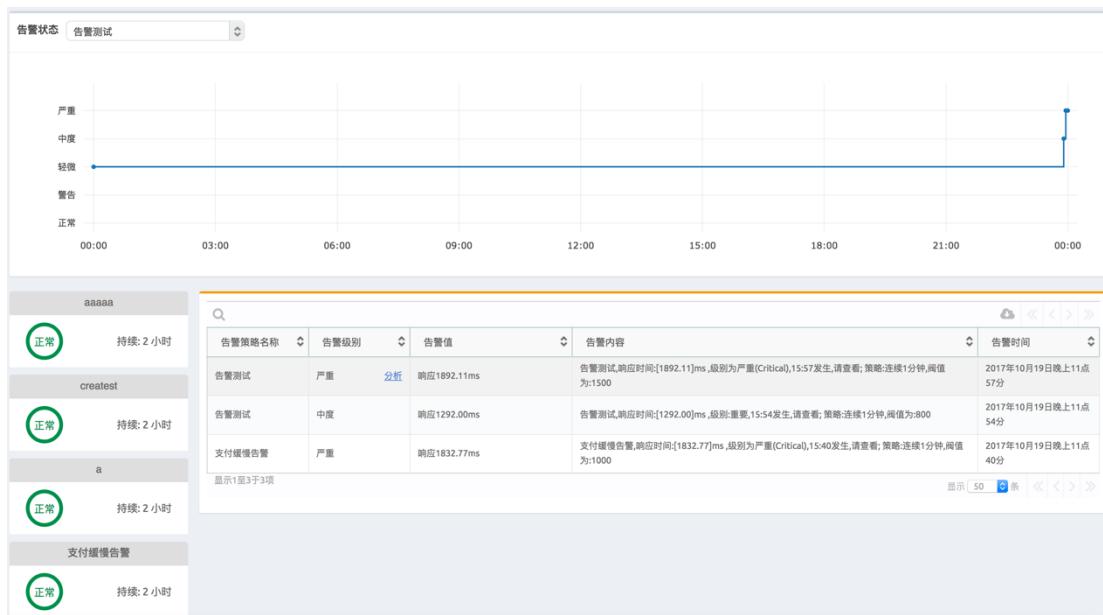


## 4.2.2 实时告警通知

应用发生异常后，触发告警策略的同时向用户发出通知，同时也记录告警发生时的应用信息，同时也将告警信息录入告警库。我们现在已开发出客户端应用，不仅可以实时查看应用数据状态，而且对应用发生的告警能及时收到通知，并可对告警的信息进行查看。

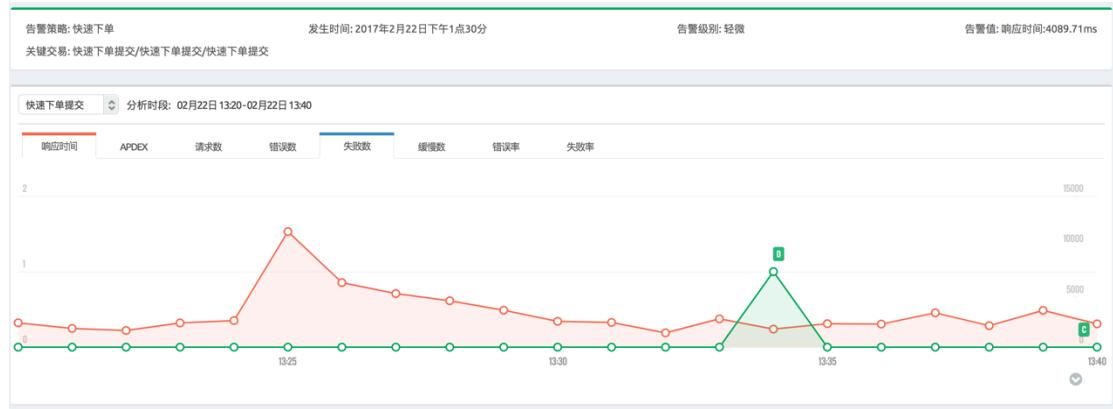


### 4.2.3 告警信息快速分析



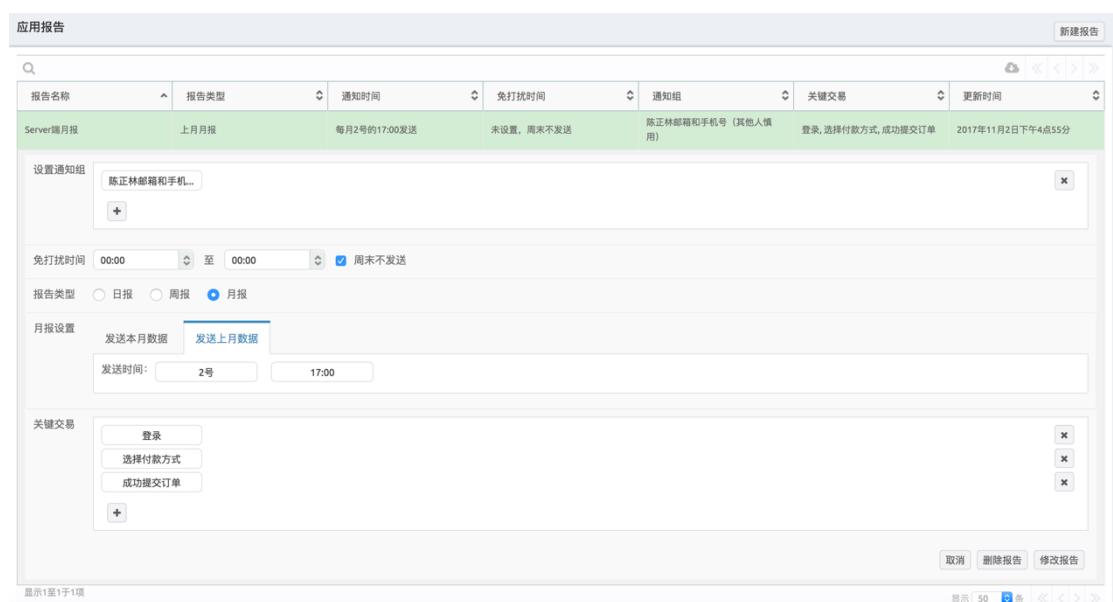
在发生告警后，客户通常情况下要根据告警信息去不同的页面去搜索发生告警的业务。现在我们可以通过一个按钮让客户快速分析发生告警的业务信息。通

过点击告警列表中(告警总览与告警详情中都有)的分析按钮，进入告警分析页面。



在告警分析页面，展示告警发生时前后 20 分钟的数据，将影响到业务请求的关键指标进行图形化展示，方便用户对告警时业务请求的变化进行分析。

## 4.8. 应用报表



客户不会 24 小时在平台上浏览数据，更多的情况下是想用最短的时间了解到应用的状态，并可以用直观的方式展示出来，现在可以通过应用报表功能实现

这个目标，首先对关心的指标进行配置，选择需要的报表形式，对重要的关键业务进行配置，设置发送时间，客户就可在设置的时间收到应用的报告，并可导出为 PDF 格式，点击链接并可进行到平台对数据进行查看。

应用业务性能管理 | 消息报告

2017-10-30 22:0:0

业务数据概览

用户数: 3771	请求总量: 111088	平均响应时间: 1592
-----		
网络错误率: 5.55%	Http请求错误率: 0.56%	

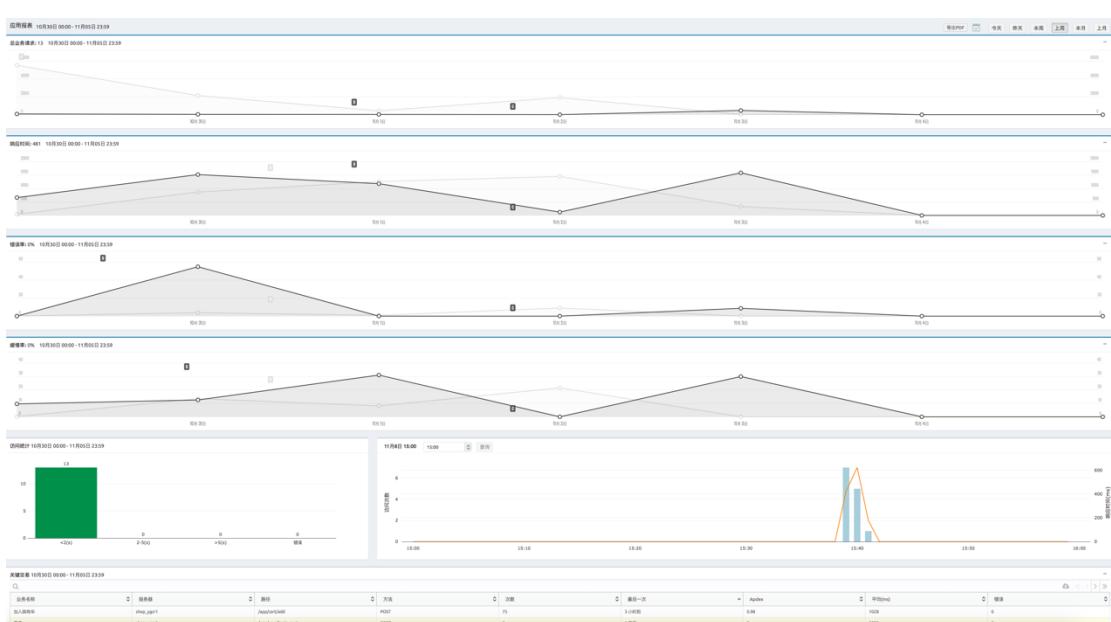
告警信息

无数据

关键业务

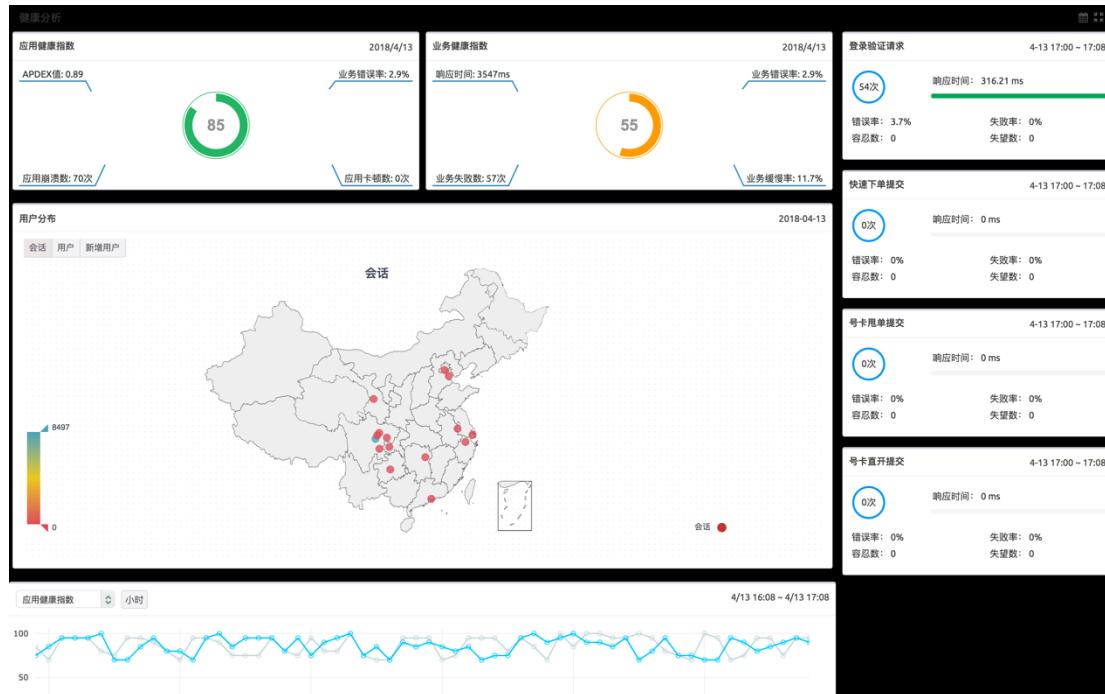
业务名称	请求数量	失败率	错误率	响应时间
登录验证请求	6340	13.83%	2.65%	1636ms
号卡甩单提交	1	0.00%	0.00%	6067ms
号卡直开提交	380	0.00%	0.26%	3744ms
快速下单提交	1259	0.08%	0.08%	2278ms

如果通过邮件还对应用不够放心，可以点击邮件中的链接进入平台进行数据分析。

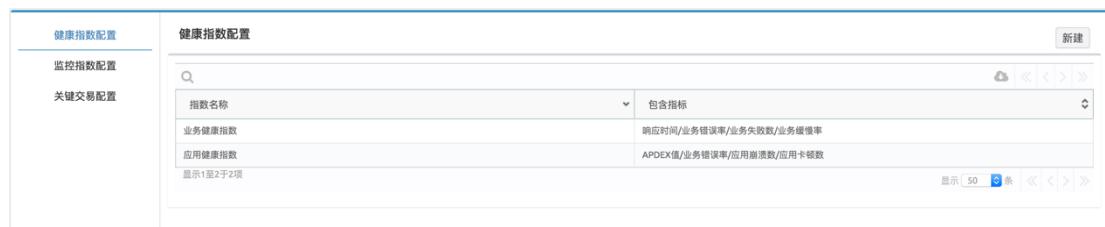


## 4.9.健康评分

健康评分直观了解被测应用的健康度，应用健康分为两个部分一个是应用性能健康指数，应用可用性的表现。另一个是业务健康指数，业务表现力如何。两个指数代表应用的综合健康状态。



对应用的核心指标设置权重，通过权重关系计算出指数分数，可以直观的反映应用当前健康度。满分为一百分，每 5 分钟刷新一次，分数越高应用越健康，分数的边框颜色会根据的分数值进行改变。



设置健康指数，配置指数的指标和权重。依据应用的特性，通过合理的配置让健康分数更准确。

监控指数配置		
监控指标名称 指标样式 数据显示		
业务失败数	折线图	1小时
业务缠售率	柱状图	1小时
业务错误率	折线图	1小时
响应时间	折线图	1小时
应用卡券数	折线图	1小时
应用崩溃数	折线图	1小时

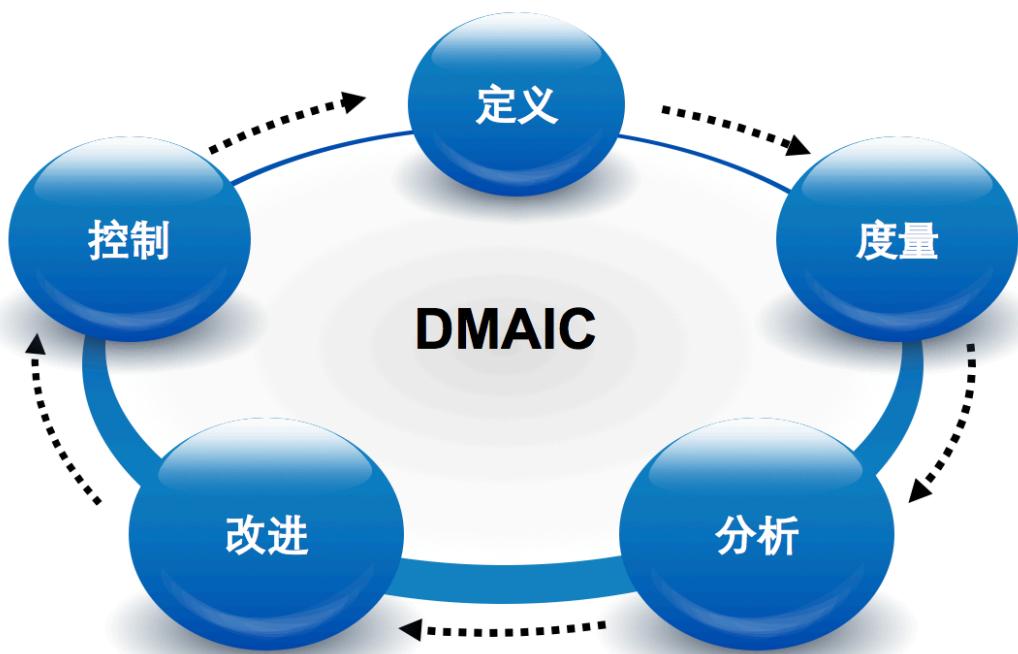
设置健康评分显示的关键指标，关键指标的展示样式和时长。依据应用特性选择合适的关键指标显示到健康评分页面上。

关键交易配置		
关键交易 数据显示		
号卡甩单提交	1小时	
号卡直升提交	1小时	
快递下单提交	1小时	
登录验证码请求	1小时	

设置健康评分显示的关键交易，设置关键交易展示的时长。将应用赖以生存的核心关键交易显示到健康评分页面上。

## 5.产品理念

### 5.1.业务精益管理，持续优化



定义 :确定关键业务的问题是什么 ?识别关键业务并将其定义出来 ,不定义关键业务 ,说明你不重视它 !

度量 :了解关键业务的问题有多严重 ?收集数据并利用数据了解问题发生的情况 ,初步分析问题发生的原因 ,清晰描述业务的问题。

分析 :根本原因是什么 ?分析性能问题的根本原因 ,收集数据并利用数据验证根本原因的准确性。

改进 :如何改善关键业务的的性能问题 ?针对根本原因确定解决方案 ,实施改善措施并收集数据验证改善效果。

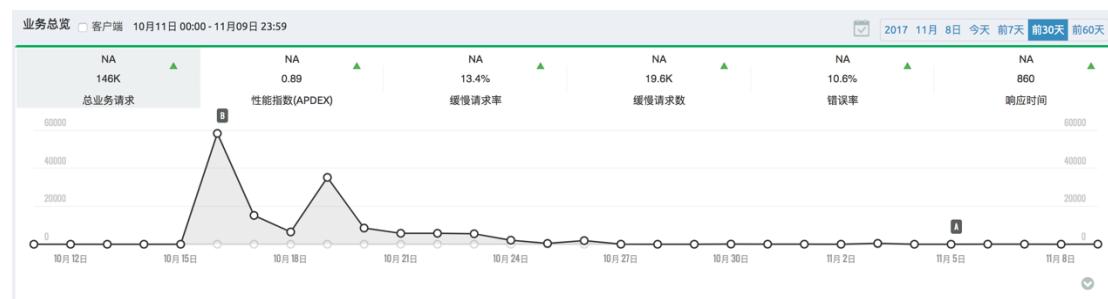
控制：如何维持并持续改善？针对关键交易进行精准告警，将更多的关键交易纳入业务精细管理中，持续监测，定期分析评估关键业务性能指标。

## 5.2. 三层维度覆盖解析应用系统

AppTrace 平台将应用系统分为三个层面宏观层、中观层、微观层，通过宏观层了解应用系统的状态定义应用系统的的阶段，中观层度量应用系统各组件的核心服务，微观层定位应用系统中存在的问题并提供优化修改方案。

以业务请求为例：

宏观层：30 天内应用系统共发生了 146 千次业务系统，性能指数是 0.89，缓慢请求率为 13.4%，缓慢请求数 19.6 千次，错误率为 10.6%，平均响应时间为 860ms，从宏观层了解到应用系统缓慢率与错误率较高，但不知道哪个业务导致缓慢率高和错误率高，或者是所有的业务都比较高。



在业务总览页面对业务列表进行筛选，找出响应时间最高业务。通过表中的数据发现并不是所有业务响应时间都高，只是个别业务拉高了平均响应时间。

业务名称	服务器	路径	方法	次数	最后一次	Apdex	平均(ms)	错误
/callSelf/httpclient4/getGeoCo	shop_37080	/callSelf/httpclient4/getGeoCode.apptrace	GET	4	在 Tue Oct 31 2017 11:32:37	0	128498	3
/httpclient4/getGeoCode.apptrace	shop_37080	/httpclient4/getGeoCode.apptrace	GET	4	在 Tue Oct 31 2017 11:32:37	0	128483	3
/sleep7.apptrace	shop_web02_39080	/sleep7.apptrace	GET	5	1 周前	0	7099	0
/app/order/427ed392d0a8473ea61	shop_ygr1	/app/order/427ed392d0a8473ea616e582dc733c5f	POST	1	6 天前	0.5	4421	0
/httpclient4/listProduct.apptrace	shop_37080	/httpclient4/listProduct.apptrace	GET	39	2 天前	0.9	3349	0
/sleep3.apptrace	shop_web02_39080	/sleep3.apptrace	GET	6	40 分钟前	0.5	3018	0
成功提交订单	shop_ygr1	/app/order/add/	POST	12105	5 小时前	0.58	3016	0
登录	shop_ygr1	/app/user/login-post	POST	39	2 天前	0.06	2614	0

在业务总览页面对业务列表进行筛选，找出错误数最多业务。同样错误数也一样，只有一个业务的错误率比较高，这一个业务拉高了整体的错误率。

业务名称	服务器	路径	方法	次数	最后一次	Apxd	平均(ms)	错误
/httpClient4/getTwitterUrlCount	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET	12098	6 天前	0	0	12098
成功加入购物车	shop_ygr1	/app/cart\$	POST	15365	2 小时前	0.94	1302	36
/httpClient4/getList.apptrace	shop_web02_39080	/httpClient4/getList.apptrace	GET	12726	41 分钟前	0.99	122	9
产品列表	shop_ygr1	/app/product	POST	12432	2 小时前	1	338	7
饮品列表 (get方法)	shop_ygr1	/app/category/list	GET	12662	41 分钟前	1	3	6

中观层：我们在进入该业务中，查看该业务的错误率是多少，是否所有业务请求都发生了错误。



进入该业务的总览页面发现，该业务正在进行的 12.1 千次业务请求全部发生错误，错误率达到了 100%。

状态	入库时间	请求时间	IP地址	设备	状态码	响应时间(ms)	服务器	业务名称	URL	方法	交易结果
HTTP 404	11/03 11:36:12 561	11/03 11:36:12 519	172.16.30.30	java 1.5)	404	4	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET		
HTTP 404	11/03 11:36:12 042	11/03 11:36:12 036	172.16.30.30	java 1.5)	404	78	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET		
HTTP 404	11/03 11:36:12 038	11/03 11:36:12 036	172.16.30.30	java 1.5)	404	78	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET		
HTTP 404	11/03 11:35:52 402	11/03 11:35:52 371	172.16.30.30	java 1.5)	404	5	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET		
HTTP 404	11/03 11:35:52 341	11/03 11:35:52 334	172.16.30.30	java 1.5)	404	6	shop_web02_39080	/httpClient4/getTwitterUrlCount.apptrace	GET		

查看该业务的请求列表，发现所有的错误都是 404 错误。  
微观层：对单次请求进行分析，查看错误时请求所携带的信息和业务请求交易链定位问题错误原因。

状态	入库时间	请求时间	IP地址	设备	状态码	响应时间(ms)	服务器	业务名称	URL	方法	交易结果
http	11/03 11:36:12 561	11/03 11:36:12 519	172.16.30.30	java 1.5)	404	4	shop_web02_39080	/httpclient4/getTwitterUriCount.apprace		GET	
HTTP 头:											
"host": "172.16.30.30:39080", "connection": "Keep-Alive", "user-agent": "Apache-HttpClient/4.3.6 (Java/1.5)", "accept-encoding": "gzip, deflate", "apptrace-spa-receeid": "shop_web02_39080", "apptrace-spnid": "1509415437735", "apptrace-spnid": "453382005690843712", "apptrace-pspnid": "8673804302202889", "apptrace-pflags": "0", "apptrace-pappname": "shop_39080", "apptrace-papptype": "1010", "apptrace-host": "172.16.30.30:39080", "appagentid": "shop_web02_39080"											

点击交易分析查看该次请求的调用链。

The screenshot shows a detailed call chain analysis for a request. At the top, there's a search bar and filter options. Below it, a tree view shows the call stack: Tomcat Servlet Process → REMOTE\_ADDRESS → invoke(Request request, Response response) → http.status.code → http.method. To the right, a timeline table displays the execution details for each step. The first step (invoke) has a duration of 10ms, while the final step (http.method) is a GET request.

序号	方法	类	参数	开始时间	间隔	执行时间(m...)	执行时间(%)	自身>=
1	Tomcat Servlet Process		/httpclient4/getTwitterUriCount.apprace	15:12:38.608		10	2	TOMCAT
2	REMOTE_ADDRESS		172.16.30.30			8	8	TOMCAT_M...
3	invoke(Request request, Response response)	org.apache.catalina.core.StandardH...		15:12:38.610	2			
4	http.status.code		404					
5	http.method		GET					

## 5.3. 数据自动关联

### ● 一、前后端数据自动关联

AppTrace 平台将前端数据与后端数据自动关联，不需要客户进行配合修改，在页面上统一展示，方便客户对前端数据进行分析。

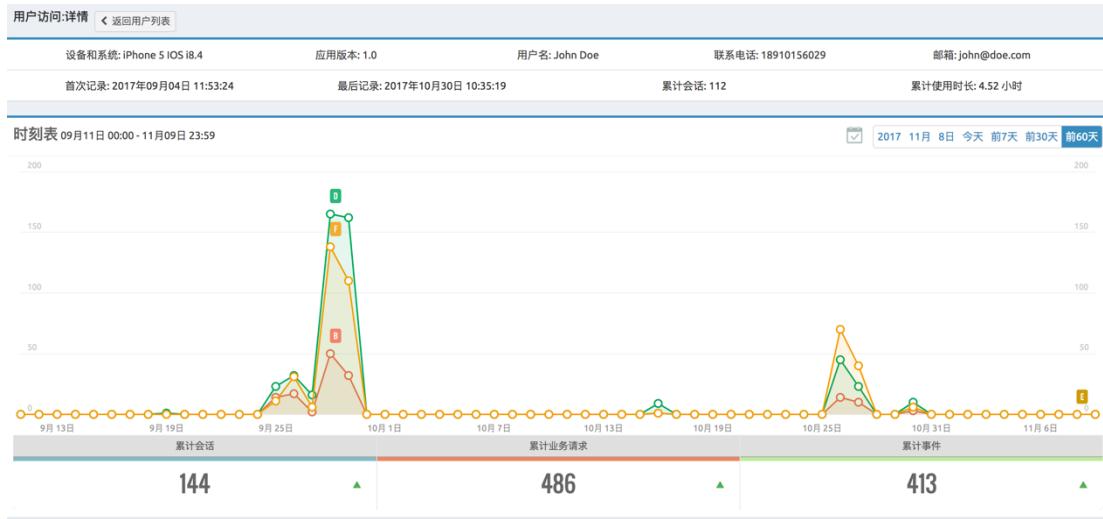
This screenshot demonstrates the automatic data association feature. On the left, the navigation sidebar includes sections like '所有应用' (All Applications), '应用总览' (Application Overview), '交易查询' (Transaction Query), '业务请求' (Business Requests), '用户画像' (User Profile), '告警总览' (Alert Overview), '数据分析' (Data Analysis), '忠诚度' (Loyalty), '事件' (Events), '系统崩溃' (System Crash), '卡锁' (Card Lock), '模型管理' (Model Management), and '应用报表' (Application Reports). The main area shows a transaction query interface with a timeline chart and a detailed call chain table. The timeline chart tracks memory usage (Heap MAX, Heap USED, FGC) over time, showing a significant peak at 14:10. The call chain table shows the flow from a mobile device (192.168.1.166) through a web proxy (shop\_web/gcr) to a business service (iwc\_shop).

序号	方法	类	参数	开始时间	间隔	执行时间(m...)	执行时间(%)	自身>=
1	Tomcat Servlet Process		/app/category/list-with-product	14:15:37.727		86	0	TOMCAT
2	REMOTE_ADDRESS		192.168.1.166			0	86	1 TOMCAT_ME...
3	invoke(Request request, Response response)	org.apache.catalina.core.StandardHostV...		14:15:37.727				
4	http.requestHeader.id		8CC01A155-922B-455D-BBDC-FF0997F11924					
5	http.requestHeader.tid		M2017110914152052088f63e688b3c60114a653160*2					
6	http.status.code		200					
7	http.method		POST					
8	doPost(HttpServletRequest request, HttpServletResponse response)	org.springframework.web.servlet.FrameworkServlet		14:15:37.728	1	85	3	SPRING_MVC
9	setAutoCommit(boolean autoCommitFlag)	com.mysql.jdbc.ConnectionImpl		14:15:37.729	1	1	1	MYSQL(wt...)
10	arg[0]	null						
11	findInstListWithProduct4App	com.iwc.shop.modules.shop.service.ShopCategoryService		14:15:37.730	0	65	0	SPRING_BEAN
12	findByParentId(String parentId)	com.iwc.shop.modules.shop.service.ShopCategoryService		14:15:37.730	0	6	0	SPRING_BEAN
13	selectList(String statement, Object parameter)	org.mybatis.spring.SqlSessionTemplate		14:15:37.730	0	6	4	MYBATIS
14	cache_d_arg[0]	com.iwc.shop.modules.shop.dao.ShopCategoryDao		14:15:37.731	1	0	0	MYBATIS (inv)

### ● 二、用户与业务数据自动关联

AppTrace 对每个用户访问的那些了多少业务，都是访问了业务进行统计，让客户不在对客户一无所知。

用户名为 John Doe 的用户在 60 天内进行 486 次业务请求。



这 486 次业务请求都在列表中展示出来，并将请求的状态进行显示。

业务请求时间表											
状态	入库时间	请求时间	IP地址	设备	状态码	响应时间(m s)	业务名称	URL	方法	事件请求	交易结果
失败	10/30 10:12:38 620	2017年10月30日 10:12:37	192.168.1.190	iPhone 5	200	28	获取用户信息	http://172.16.30.30:38080/app/user/get-user	POST		
失败	10/30 10:09:58 633	2017年10月30日 10:09:57	192.168.1.190	iPhone 5	200	19	获取用户信息	http://172.16.30.30:38080/app/user/get-user	POST		
正常	10/30 10:09:48 648	2017年10月30日 10:09:38	192.168.1.190	iPhone 5	200	368	首页列表	http://172.16.30.30:38080/app/category/list-with-product	POST		
容忍	10/30 10:08:47 795	2017年10月30日 10:08:39	192.168.1.190	iPhone 5	200	1138	成功加入购物车	http://172.16.30.30:38080/app/cart	POST		
失败	10/30 10:08:47 795	2017年10月30日 10:08:41	192.168.1.190	iPhone 5	200	18	获取用户信息	http://172.16.30.30:38080/app/user/get-user	POST		
正常	10/30 10:08:37 828	2017年10月30日 10:08:27	192.168.1.190	iPhone 5	200	460	首页列表	http://172.16.30.30:38080/app/category/list-with-product	POST		
正常	10/30 10:08:37 828	2017年10月30日 10:08:31	192.168.1.190	iPhone 5	200	1085	产品列表	http://172.16.30.30:38080/app/product/list-featured-price	POST		
正常	10/30 10:08:37 828	2017年10月30日 10:08:33	192.168.1.190	iPhone 5	200	139	产品列表	http://172.16.30.30:38080/app/product/list-featured-topic	POST		
容忍	10/30 10:08:37 828	2017年10月30日 10:08:33	192.168.1.190	iPhone 5	200	3597	产品列表	http://172.16.30.30:38080/app/product/list-top-seller	POST		

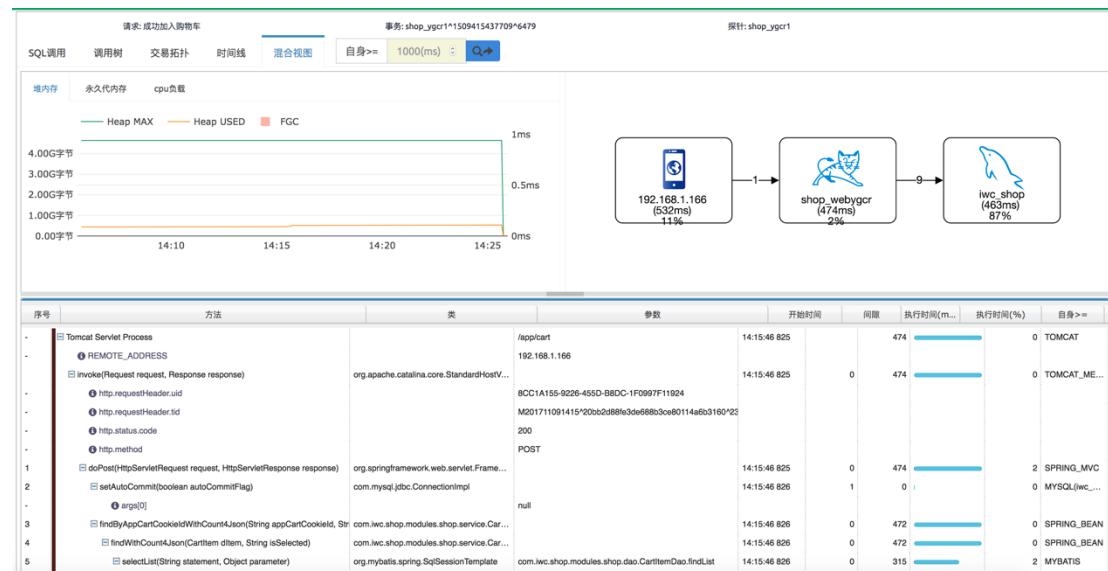
### ● 三、应用性能与业务请求自动关联

AppTrace 平台可以从多个维度关联到业务，包括数据库、虚拟机、程序方法、程序异常等。以 SQL 调用为例；

在 SQL 分析的列表中，每行 SQL 调用都有交易分析按钮，点击交易自动关联到发生该次 SQL 调用的业务请求上。

sql详细信息		sql慢查询						
SQL名称	响应时间	sql信息	请求时间	DB客户端	DB主机	库名	绑定的数据	状态
sql 编辑模型	0	select @@session.tx_read_only	2017年11月09日 17:18:37	shop_ygcr1	172.16.30.66:3306	iwc_shop	com.mysql.jdbc.JDBC4ResultSet@74d30a4c	交易分析
sql	1	select @@session.tx_read_only	2017年11月09日 17:17:37	shop_ygcr1	172.16.30.66:3306	iwc_shop	com.mysql.jdbc.JDBC4ResultSet@73a4cf80	

关联到 SQL 调用发生在“成功加入购物车”的业务请求中，这次业务请求共发生了 9 次数据库调用占共响应时间的 87%。



## 5.4. 端到端关联分析

### 5.4.1. 用户统计

现在应用面临的一个难题是无法获知我们的用户到底是谁，都是在应用中操作些什么功能，消费了多少钱，发出了多少个业务请求。对于这个难题 AppTrace 用完整的数据解答了这道难题。

用户编号	IP地址	国家	城市	设备	OS版本	应用版本	累计会话	最近访问时间	累计使用时长
8CC1A155-9226-4550-BBDC-1F099711924	192.168.1.134	Unknown	Unknown	iPhone 6s	iOS	1.0	285	2017年05月11日 11:56:15	4.80 小时
4241D48A-F20F-4681-BE38-77D6321277DB	192.168.1.103	Unknown	Unknown	iPhone 5	iOS	1.0	21	2017年05月09日 16:50:43	1.04 小时
CDA93500-D02 24B0B-84FC-S2 A1E0B0C367	192.168.1.147	Unknown	Unknown	iPhone Simulator	iOS	1.0	10	2017年05月08日 15:49:57	1.02 小时

显示1至3于3项

## ● 一用户到底是谁

无需企业在写 API 和方法，就可以自动获取到用户的手机号，并用手机号和唯一设备标识两层手段来确定唯一的用户。

设备和系统: iPhone 5 iOS i8.4	应用版本: 1.0	用户名: John Doe	联系电话: 18910156029	邮箱: john@doe.com
首次记录: 2017年09月04日 11:53:24	最后记录: 2017年10月30日 10:35:19	累计会话: 112	累计使用时长: 4.52 小时	

## ● 二用户通过操作发出了多少业务请求

根据用户的标识去记录用户进行的每一次会话、每一次请求、每一次事件操作，并可以通过业务请求向后追溯后端的调用路径。

会话历史	业务请求时间表	事件时间表
█		
URL		
http://192.168.1.32-38080/app/category/list-with-product	2017年05月11日 11:56:15	月光茶人
http://192.168.1.32-38080/app/cart	2017年05月11日 09:06:01	购物车
http://192.168.1.32-38080/app/preorder/add	2017年05月11日 09:06:02	去结算(0)
http://192.168.1.32-38080/app/preorder/add	2017年05月11日 09:06:02	去结算(0)
http://192.168.1.32-38080/app/cart	2017年05月11日 09:06:05	购物车
http://192.168.1.32-38080/app/cart/add	2017年05月11日 09:06:07	加入购物车
http://192.168.1.32-38080/app/cart/add	2017年05月11日 09:06:08	加入购物车
http://192.168.1.32-38080/app/product/9ea08446f1964c278b7c1c4h36b7954f	2017年05月11日 09:05:57	加入购物车

## ● 三用户累计使用了多长时间

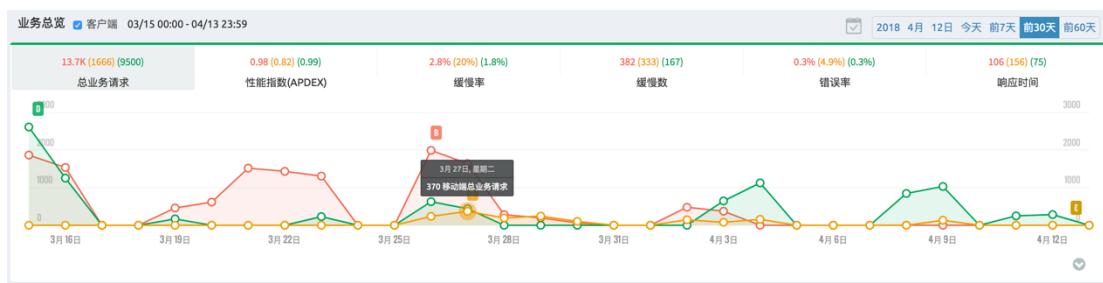
通过帮助用户对业务的梳理，自动学习用户的业务系统，去识别用户访问的业务请求，计算出用户在应用中的及时间。

设备和系统: iPhone 5 iOS i8.4	应用版本: 1.0	用户名: John Doe	联系电话: 18910156029	邮箱: john@doe.com
首次记录: 2017年09月04日 11:53:24	最后记录: 2017年10月30日 10:35:19	累计会话: 112	累计使用时长: 4.52 小时	

## 5.4.2.业务全链路分析

### ● 业务总览前后端对比分析

传统的 APM 将前后端分割开进行监控分析，造成了前后端各自为战的局面，无法全面了解应用业务状态。通过 AppTrace 自动前后端关联后，可以在任何一端进行宏观的业务对比，了解前后端的业务状态。



### ● 业务交易全链路分析

业务异常后如何快速锁定前后端的原因，以便加速问题解决的速度，是现阶段业务交易全链路分析的重中之重，所以 AppTrace 通过业务调用图，展示业务的完整的调用过程，快速确定前后端问题，并进行错误定位。



## 5.5.客户端随时关注应用状态

以前在下班期间和出差期间都是通过短信、微信、电话去得知应用的当前状态，现在通过安装我们 AppTrace 客户端就可实时的了解应用的当前状态，应用告警时客户端发出通知，并可对告警内容进行追溯。



### ● 数据详情查看

在了解了应用整体状态后，点击不同模块进入详情进行分析，及时的对应用当前的状态作出判断，是否存在错误和异常。



失败业务

绑定账户信息

出现次数: 2279       上次出现: 刚刚

首次登录校验

出现次数: 2623       上次出现: 刚刚

查询红包

出现次数: 3742       上次出现: 刚刚

## ● 网络错误追踪

若在应用的状态中发现了大量网络错误的发生 ,可以进行对网络错误的追踪和分析 ,客户端做到和 PC 端同样的功能和操作。



## ● 实时接收告警通知

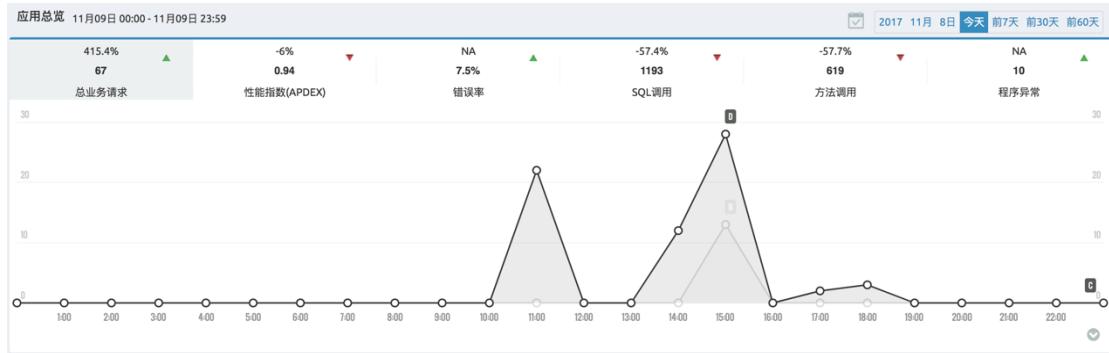
当监控应用发生告警时，客户端会实时的向客户推送通知，方便客户了解告警内容和级别，作出及时的判断和处理方案。



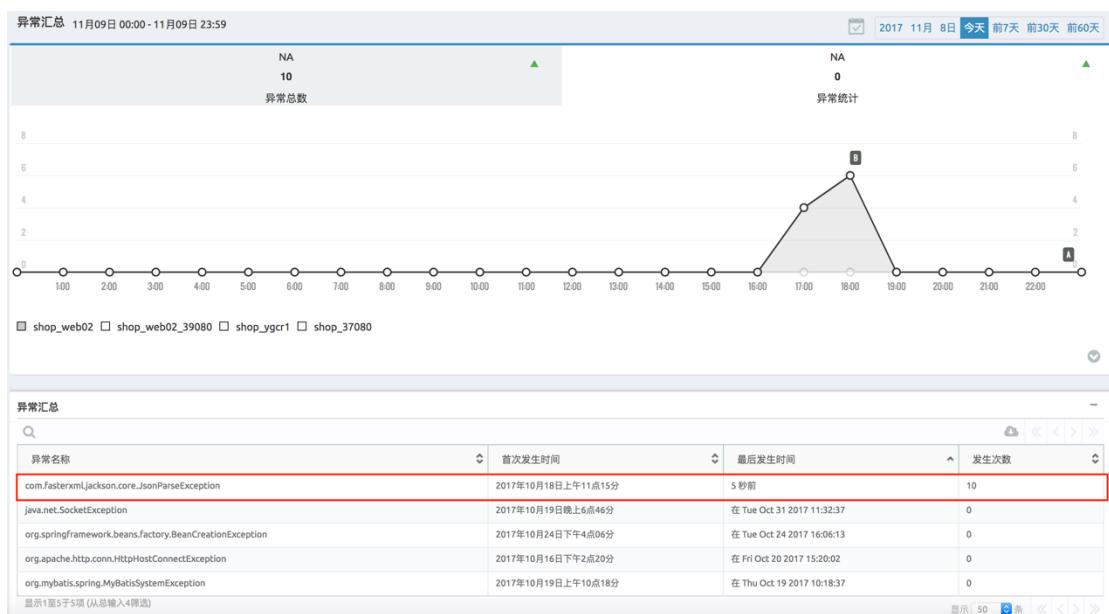
## 6.应用场景

### 6.1.30 秒定位问题根源

连续发生程序异常后，向客户进行告警，在接收到告警信息后，登录平台服务器应用发生了 10 次程序异常。



在左侧菜单栏中点击程序方法下的“程序异常”，进入程序异常页面。发现在 5 秒钟发生了程序异常 com.fasterxml.jackson.core.JsonParseException。

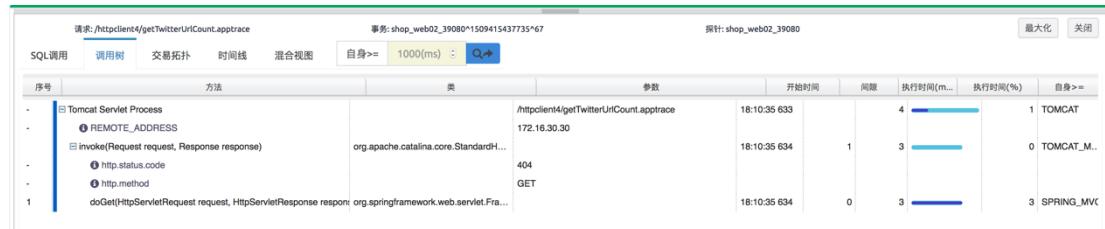


点击 com.fasterxml.jackson.core.JsonParseException 程序异常，进入异常详情页面，查看异常列表，分析错误信息 Unexpected character ('<' (code 60)): expected a valid value (number, String, array, object, 'true', 'false' or 'null') at [Source: org.apache.http.conn.EofSensorInputStream@33481307; line: 1, column: 2]

异常名称	发生时间	错误信息	方法	类	程序接口	异常url	agent	组件
com.fasterxml.jackson.core.JsonParseException 交易分析	2017年11月9日上午6点10分	Unexpected character ('<' (code 60)): expected a valid value (number, String, array, object, 'true', 'false' or 'null') at [Source: org.apache.http.conn.EofSensorInputStream@33481307; line: 1, column: 2]	readValue	com.fasterxml.jackson.databind.ObjectMapper	com.fasterxml.jackson.databind.ObjectMapper...	/callSelf/httpclient4/getTwitterUrlCount/app trace	shop_web02_390 80	shop_39080

点击交易分析，进入到发生该程序异常的业务请求中，查看调用链情况。最终锁

定为“字符串不符合规则”导致的错误。

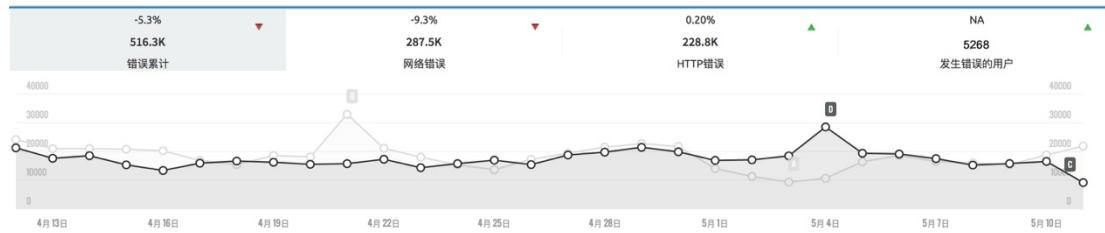


## 6.2. 业务端到端关联分析

现阶段应用中发生问题后，由于前后端关联度高，很难快速确定是前端或后端的问题，所以经常会出现主要负责人互相推脱责任的现象，而且等到问题最终排查出来也许也造成了大量的业务损失。

我们通过前后端的关联，问题发生到问题的定位仅仅需要一分钟的时间，让责任一分钟定位，同时提供错误分析帮助企业将业务损失降到最小。

下面是应用在 30 天内发生的网络错误数据，单从数据上是无法判断错误的位置是在前端还是后端。更无法处理这些问题。



通过对网络错误的详情分析，虽然可以对请求报文内容和错误信息进行分析，但是还是难得到准确的错误位置和解决方法。

请求错误	URL	方法	状态码	设备	用户编号	应用版本
2017年05月12日 16:40:28	192.168.1.32:39080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:40:28	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:38:28	192.168.1.32:39080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:32:28	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:28:28 交易分析	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:26:28	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:24:28	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:18:28	192.168.1.32:39080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:16:28	192.168.1.32:39080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0
2017年05月12日 16:08:28	192.168.1.32:37080/httpclient4/getTwitterUrlCount.apprace	GET	404	unknown unknown	4	1.0

通过对业务请求的内部调用路径的分析，不断是前端问题和后端问题显露无疑，即可就可对问题的责任进行定责。从单次调用的拓扑中可以看出错误发生在前端中。



对责任定责后就应该去，定位问题并快速的解决该问题，让对业务的影响降到最低，从调用路径中可以清晰看到错误位置和原因。

方法	参数	开始时间	间隙	执行时间(m...)	执行时间(%)	自身>=	类	应用程序接口	代理
API-METADATA-NOT-FOUND	/callSelf/httpclient4/getTwitterUrlCount.apprace	16:40:28 132	0	203	100%	0	TOMCAT	58e5a402e951920cdc... test3	
REMOTE_ADDRESS	192.168.1.21	16:40:28 132	0	203	100%	2	TOMCAT_ME...	58e5a402e951920cdc... test3	
API-METADATA-NOT-FOUND	http.status.code http.method http.requestHeader	500 GET content-type, content-length,0,host,192.168.1.32:39080,conn	16:40:28 133	1	201	8	SPRING	58e5a402e951920cdc... test3	
JsonParseException	Unexpected character (< (code 60)): expected a valid value	16:40:28 140	7	192	100%	37	CloseableHttpClient	58e5a402e951920cdc... test3	
execute(HttpRequest request)	connect(HttpURLConnection managedConn, HttpRoute route, in 192.168.1.32:39080	16:40:28 175	35	29	15%	29	PoolingHttpClientCo...	58e5a402e951920cdc... test3	
execute(HttpRequest request, HttpClientConnection conn, HttpC... /httpclient4/getTwitterUrlCount.apprace	404 writeTo, read=122 HTTP_CLIENT	16:40:28 205	1	126	100%	126	HttpRequestExecutor	58e5a402e951920cdc... test3	
API-METADATA-NOT-FOUND	/httpclient4/getTwitterUrlCount.apprace	16:40:28 307	102	5	100%	1	TOMCAT	58e5a402e951920cdc... test3	

## 6.3.持续业务优化

随着 IT 的不断发展 APM 在解决突发问题的需求正在逐渐弱化，对应用的持续优化越来越重要，APM 正在从应用性能管理向应用业务优化及性能管理不

断转变。

现在应用的可用性基本可以得到保证，但是在满足可用性的要求后，企业和用户对应用有了更高的要求，要实现应用系统与业务结构的最优化，如何能加快页面之间的跳转，如何避免请求中的错误，如何提升服务器的处理速度等问题成为了企业与用户关注的重点。

## ● 应用性能的持续优化

通过总览页面发现，近七天的数据中缓慢率达到了 31.6% 接近总请求的三分之一，这样的缓慢率持续下去将会影响大量业务请求的正常使用，最终会导致业务下降和用户流失。



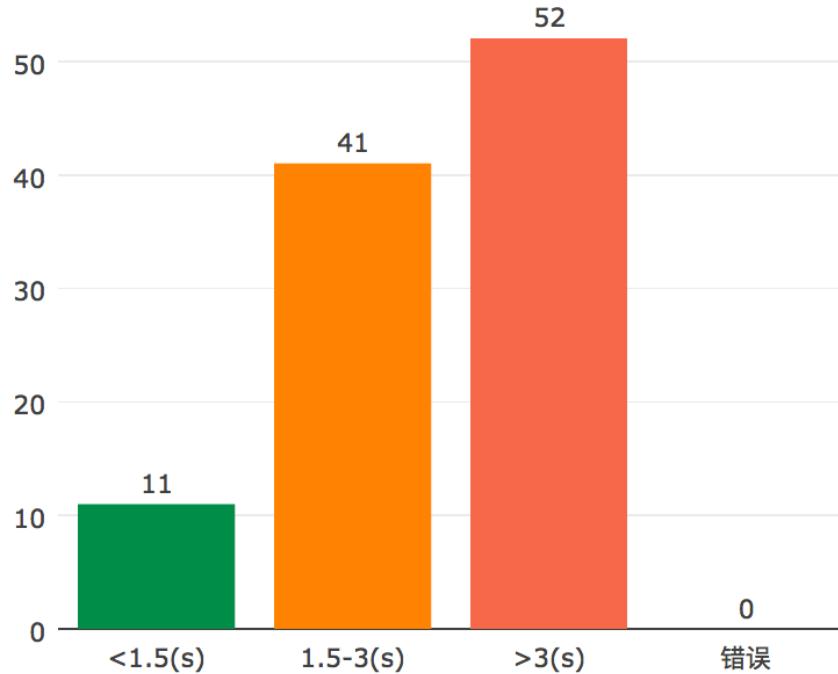
通过列表筛选出平均响应时间超过 2s 的业务请求，排查导致缓慢率升高的业务请求。将发生次数最多的业务进行重点排查，因为次数越多在总的请求中占比越重，是最有可能提升缓慢率的因素。发生最多的是“加入购物车”业务，对“加入购物车”业务进行重点排查。

业务名称	服务器	路径	方法	次数	最后一次	最大(ms)	平均(ms)	错误
订单成功提交	192.168.1.32	/app/order	POST	152	2 天前	20038	1407.10	0
加入预购订单	192.168.1.32	/app/preorder/add	POST	147	1 天前	19789	1973.56	0
产品列表	192.168.1.32	/app/product	POST	251	1小时前	17386	805.62	1
选择支付方式	192.168.1.32	/app/preorder/select-pay-type	POST	75	2 天前	16337	506.99	0
获取用户信息	192.168.1.32	/app/user/get-user	POST	152	1小时前	14203	453.26	0
/app/category/list-with-product	192.168.1.32	/app/category/list-with-product	POST	112	1小时前	13058	1159.28	6
加入购物车	192.168.1.32	/app/cart/add	POST	348	1 天前	6696	800.68	15
预购订单详情	192.168.1.32	/app/preorder	POST	85	1小时前	4611	532.24	0
/app/user/logout	192.168.1.32	/app/user/logout	POST	12	3 天前	4532	847.67	0
/app/address/list	192.168.1.32	/app/address/list	POST	9	3 天前	4024	1346.67	0
成功加入购物车	192.168.1.32	/app/cart	POST	156	1小时前	3734	415.99	0
登录	192.168.1.32	/app/user/login/post	POST	25	2 天前	2960	1224.88	0

七天中“加入购物车”业务共发生了 104 次，却有 93 次大于 1.5S 的请求，七天内所有的业务请求中出现 121 次缓慢，“加入购物车”业务就占了 93 次，

占了缓慢业务的 76%。

## 访问统计

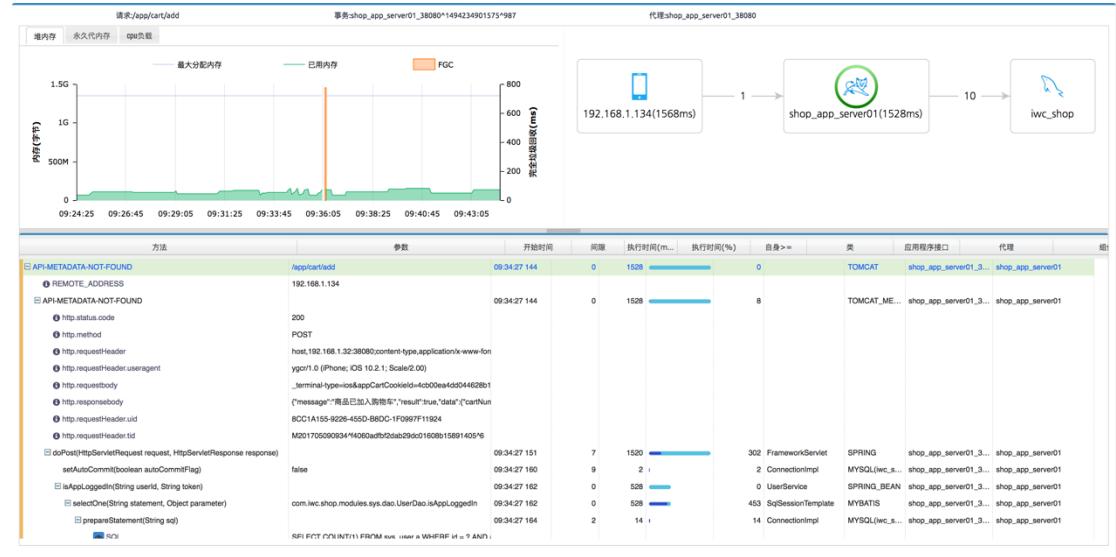


选取最近发生的前十条访问记录，最低的响应时间也在 297ms，其中 7 条记录的响应时间在 500ms-1100ms 之间，且都是由同一个设备，同样的网络连接方式访问的，初步判断为内部调用结构导致的访问缓慢。

入库时间	请求时间	IP地址	设备	状态码	网络连接	响应时间(ms)	服务器端响应时间	业务名称	URL	方法	事件请求	交易结果
2017年05月12日 09:34:41	2017年05月12日 09:34:30	192.168.1.34	iPhone 6s	200	WiFi	672		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 09:36:26	2017年05月12日 09:36:08	192.168.1.34	iPhone 6s	200	WiFi	1098		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 09:46:26	2017年05月12日 09:46:09	192.168.1.34	iPhone 6s	200	WiFi	882		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:01	2017年05月12日 11:28:47	192.168.1.34	iPhone 6s	200	WiFi	297		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:01	2017年05月12日 11:28:49	192.168.1.34	iPhone 6s	200	WiFi	369		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:01	2017年05月12日 11:28:50	192.168.1.34	iPhone 6s	200	WiFi	317		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:59	2017年05月12日 11:25:40	192.168.1.34	iPhone 6s	200	WiFi	785		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:59	2017年05月12日 11:25:41	192.168.1.34	iPhone 6s	200	WiFi	756		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:59	2017年05月12日 11:25:41	192.168.1.34	iPhone 6s	200	WiFi	733		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	
2017年05月12日 11:29:59	2017年05月12日 11:25:41	192.168.1.34	iPhone 6s	200	WiFi	706		加入购物车	http://192.168.1.32:38080/app/cart/add	POST	加入购物车	

通过对后台调用流程的查看，验证我们之前的判断，调用数据库越多响应时间越长，但是对于数据库在查询与写入和修改中并不耗费时间，时间全部耗费在

了调取数据库的方法中。通过优化数据库的调用方法，极大的缩减了请求的响应时间。



## ● 应用业务流程的持续优化

对于应用而言，不管运营如何肯定存在问题是存在的，但是用有限的资源去盯住关键问题呢，首先我们来定义关键问题，应用的是业务承载体，最终的目的是为了实现业务和驱动业务，所以我们要盯住实现业务这一点。

下列购物流程 30 天内的数据：

加入购物车：用户数 18，用户留存率 100%，业务请求 349，业务请求留存率 100%

加入购物订单：用户数 13，用户留存率 72.2%，业务请求 142，业务请求留存率 40.7%

选择支付方式：用户数 10，用户留存率 55.6%，业务请求 72，业务请求留存率 20.6%

获取用户信息：用户数 11，用户留存率 61.1%，业务请求 148，业务请求留存率 43%



其中加入购物订单环节流失用户和业务请求减少最多，查看发现加入购物车的环节缓慢率也达到了 21%，损失了大多半的业务请求。当然这里面也包含自然流失。

	用户数	业务请求数	APDEX	失败率	缓慢率	错误率	平均响应时间
1.加入购物车	18(100%)	349(100%)	0.21	0%	21%	55.5%	803.5
2.加入预购订单	13(72.2%)	142(40.7%)	0.86	0%	83%	0%	1871.3
3.选择支付方式	10(55.6%)	72(20.6%)	0.99	0%	16.7%	0%	516.5
4.获取用户信息	11(61.1%)	148(42.4%)	0.96	17%	78.3%	0%	462.4

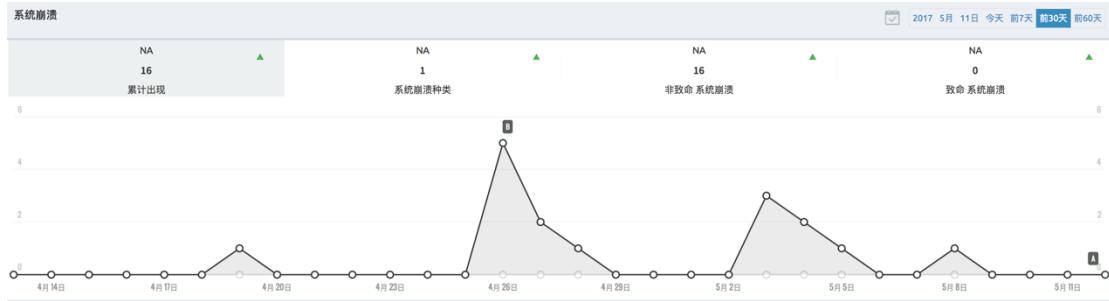
加入购物车订单环节业务请求也减少了接近一般，对于用户而言进入订单页面很少在会离开页面方式支付，而在这里却出现这么多的请求减少，必定和 83% 的缓慢率相关。

如果中间环节缓慢率极高，又不能及时解决，不如避免出现这种情况发生跳过中间环节，直接进行支付方式的选择。

## 6.4.突发问题处理

对于应用中突出问题的处理，我们坚持多方式告警通知，问题发生环境记录，收集问题发生的原因，帮助企业快速定位问题位置。

下面是近 30 天的应用崩溃数据，累计发生了 16 次，崩溃种类 1 种，属于非致命崩溃，崩溃在 4 月 26 日发现最多。



通过崩溃记录可以看出，发生崩溃全都是 iPhone6S，版本都是 1.0 版本，且设备的其他信息一样，初步断定是 1.0 版本在做 iPhone6S 适配中某个环节发现了问题。

系统崩溃	OS版本	设备	应用版本
4天前	iOS 10.21	iPhone Es	1.0
应用版本:	设备:	设备状态: 内存: 450/2005 Mb 剩余: 262/2653 Mb 之后 7 天会话 在线系统崩溃 yes 后台系统崩溃 no 静音状态的系统崩溃 no	
1周前	iOS 10.21	iPhone Es	1.0
1周前	iOS 10.21	iPhone Es	1.0
1周前	iOS 10.21	iPhone Es	1.0
1周前	iOS 10.21	iPhone Es	1.0
在 Wed May 03 2017 10:40:05 在 Wed May 03 2017 10:23:39 在 Fri Apr 28 2017 10:31:34 在 Thu Apr 27 2017 16:43:09 在 Thu Apr 27 2017 10:09:27 在 Wed Apr 26 2017 17:51:46 在 Wed Apr 26 2017 15:32:30 在 Wed Apr 26 2017 13:21:45 在 Wed Apr 26 2017 13:24:46 在 Wed Apr 26 2017 11:05:19	iOS 10.21	iPhone Es	1.0

在崩溃发生时，将崩溃日志进行采集上传，帮助企业通过崩溃日志定位代码位置和崩溃原因。在结合之前的数据判断最终定位问题。

```
*** -[__NSArrayM objectAtIndexAtIndex]: index 5 beyond bounds [0 .. 2] 分解于 1.0

错误    注释

1 | CoreFoundation          0x00000001816f51d0 <redacted> + 148
2 | libobjc.A.dylib          0x000000018121c5d0 objc_exception_throw + 56
3 | CoreFoundation          0x00000001815d07f4 CFRunLoopRemoveTimer + 0
4 | ycr                     0x00000001000e50b4 -[TabbyMyController tableView:didSelectRowAtIndexPath:] + 2508
5 | UIKit                   0x00000001816a0124
6 | UIKit                   0x00000001817788134 <redacted> + 136
7 | UIKit                   0x000000018783bf1dc <redacted> + 292
8 | UIKit                   0x000000018782dd58 <redacted> + 560
9 | UIKit                   0x000000018759d004 <redacted> + 168
10 | CoreFoundation          0x00000001816a22c0 <redacted> + 32
11 | CoreFoundation          0x000000018169fcf0 <redacted> + 372
12 | CoreFoundation          0x00000001816a0108 <redacted> + 1024
13 | CoreFoundation          0x00000001815c208 CFRunLoopRunSpecific + 444
14 | GraphicsServices        0x000000018308210c GSEventRunModal + 180
15 | UIKit                   0x0000000187617fc <redacted> + 684
16 | UIKit                   0x0000000187610534 UIApplicationMain + 208
17 | ycr                     0x0000000100155a0c main + 124
18 | libdyld.dylib           0x0000000100351508 <redacted> + 4
```

## 7.技术指标

### 7.1.运行环境

具体配置应根据应用的月活用户和月请求量具体配置

一、数据分析配置要求	
软件配置要求	1.两台操作系统 : CentOS 6.5 64 位及以上版本；
硬件配置要求	1. CPU 处理器 : 4 核 , 主频 3GHz 以上 ; 2. 内存 : 16GB 以上内存 ; 3. 硬盘 : 至少 500GB 的剩余硬盘空间 ;
二、数据库配置要求 :	
软件配置要求	1. 操作系统 : CentOS 6.5 64 位及以上版本 ;
硬件配置要求 :	1. CPU 处理器 : 8 核 , 主频 3GHz 以上 ; 2. 内存 : 48GB 以上内存 ; 3. 硬盘 : 至少 2TB 的剩余硬盘空间 ;

### 7.2.数据精度

#### 数据精度

NO.	各类数据	精度 ( 粒度 )

1	统计类数据	1 小时
2	告警类数据	1 分钟精度
3	原始数据	原始数据精度

### 数据保存时间

NO.	各类数据	保存时间
1	统计类数据	无限长
2	原始类数据	30 个自然日