ctrip tech

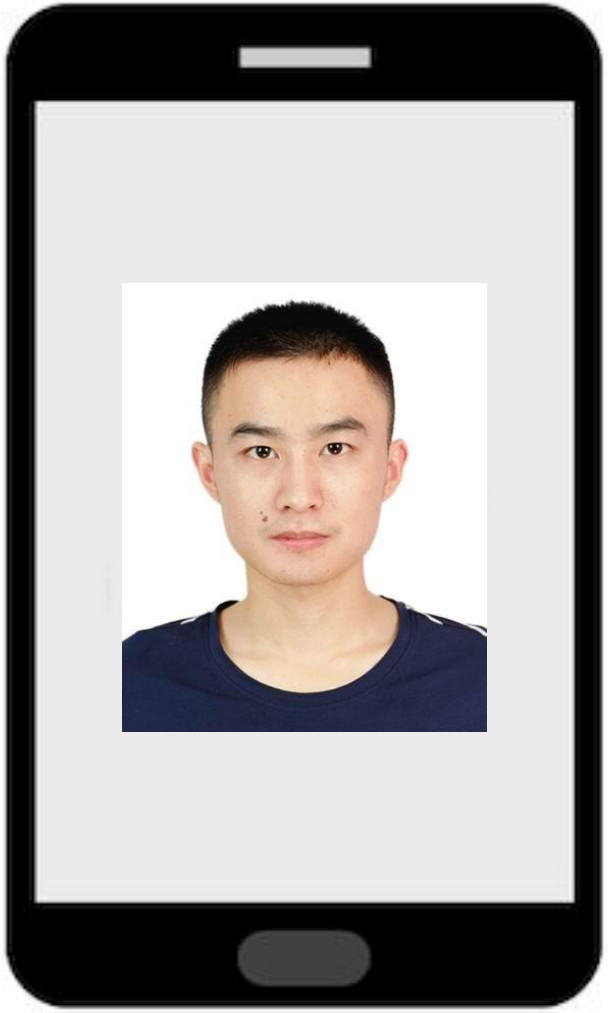
携程技术中心

携程技术沙龙

Qunar全链路跟踪及Debug

分享人：王克礼

讲师介绍



王克礼

- 去哪儿网平台事业部基础架构部门Java开发工程师
- 主要职责是开发和维护去哪儿网内部中间件

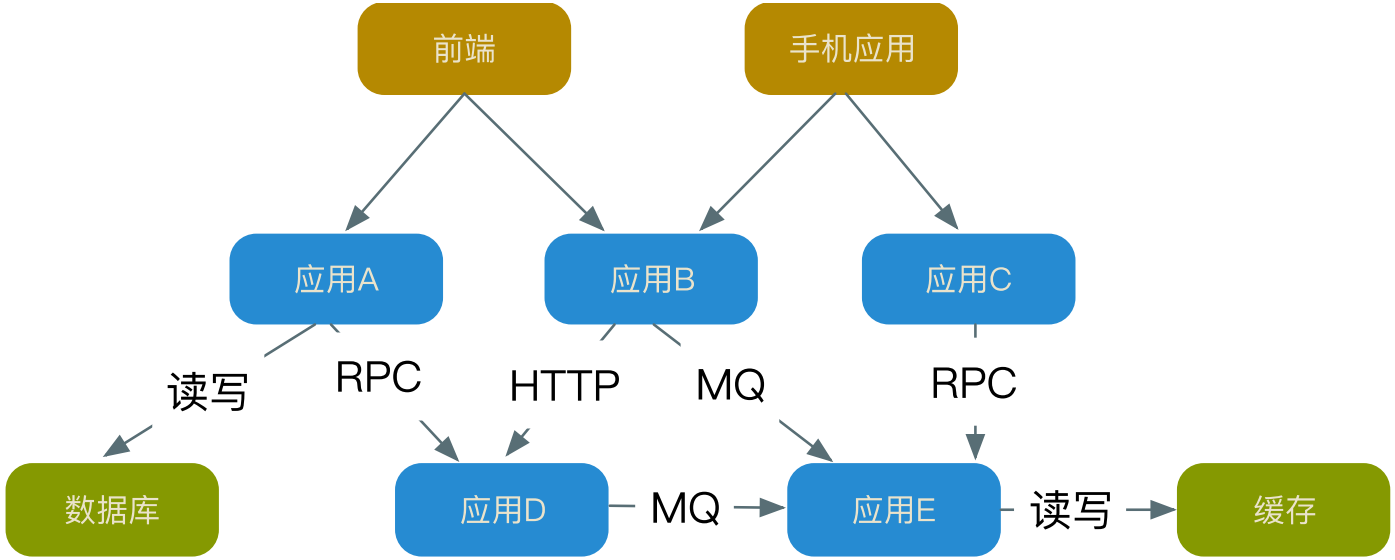
目录

CONTENTS

QTracer 简介

- 背景
  1. 随着业务发展，请求涉及到的分布式系统越来越复杂
  2. RPC 调用、HTTP API 调用和消息队列
  3. 依赖数据库、分布式缓存和其它分布式系统
  4. 服务之间的调用关系越来越复杂
  5. 业务不清楚上下游关系，看不到全局

- 背景



- QTracer 简介
  1. Qunar 内部实现的分布式追踪系统
  2. 每个请求都生成全局唯一的 TraceID
  3. 记录每个系统的操作，然后整合多个系统的数据
  4. 最终还原出一个请求在各个系统上的操作流程

- Trace 链路查询

机房		描述	类型	状态	应用名	时间轴		
▼ cn1		/twell/longwell	HTTP	OK	f_twell_twell	开始时间：23:27:02:458，执行时长：520 ms	过程	更多
cn1		dubbo_consumer:com.qunar.flight.usermark.service.UserMarkService:judgeUserType	QTRACE	OK	f_twell_twell	开始时间：23:27:02:458，执行时长：6 ms	过程	更多
cn1		dubbo_consumer:com.qunar.flight.x.abtest.api.TestGroupService:getTestGroupByUniqueIdAndSource	QTRACE	OK	f_twell_twell	开始时间：23:27:02:465，执行时长：2 ms	过程	更多
▼ cn1		dubbo_consumer:com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo	QTRACE	OK	f_twell_twell	开始时间：23:27:02:468，执行时长：0 ms	过程	更多
cn1		dubbo_provider:com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo	QTRACE	OK	f_farecore_provider	开始时间：23:27:02:468，执行时长：0 ms	过程	更多
cn1		com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo-onreturn	QTRACE	OK	f_twell_twell	开始时间：23:27:02:469，执行时长：0 ms	过程	更多
▼ cn1		dubbo_consumer:com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo	QTRACE	OK	f_twell_twell	开始时间：23:27:02:470，执行时长：1 ms	过程	更多
cn1		dubbo_provider:com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo	QTRACE	OK	f_farecore_provider	开始时间：23:27:02:471，执行时长：0 ms	过程	更多
cn1		com.qunar.flight.farecore.api.FareInfoQueryService:queryFareInfo-onreturn	QTRACE	OK	f_twell_twell	开始时间：23:27:02:474，执行时长：0 ms	过程	更多
▶ cn1		dubbo_consumer:com.qunar.flight.rt.service.TwellRTService:searchOneway	QTRACE	OK	f_twell_twell	开始时间：23:27:02:476，执行时长：318 ms	过程	更多
▶ cn1		activityQueryServiceCaller	QTRACE	OK	f_twell_twell	开始时间：23:27:02:853，执行时长：16 ms	过程	更多
▶ cn1		xInfoAsyncLoader_thread	QTRACE	OK	f_twell_twell	开始时间：23:27:02:864，执行时长：7 ms	过程	更多
▶ cn1		dubbo_consumer:com.qunar.flight.tts.datasupport.api.IFlightInfoService:getFlightAVInfos	QTRACE	OK	f_twell_twell	开始时间：23:27:02:875，执行时长：8 ms	过程	更多
▶ cn1		dubbo_consumer:com.qunar.flight.tts.datasupport.api.IFlightInfoService:getFlightAVInfos	QTRACE	OK	f_twell_twell	开始时间：23:27:02:884，执行时长：9 ms	过程	更多
▶ cn1		dubbo_consumer:com.qunar.flight.tconfig.api.service.ITConfigService:queryAirLineBiddingInfo	QTRACE	OK	f_twell_twell	开始时间：23:27:02:973，执行时长：0 ms	过程	更多
▶ cn1		dubbo_consumer:com.qunar.flight.anticrawl.service.BehaviourService:recordAction4Twell	QTRACE	OK	f_twell_twell	开始时间：23:27:02:973，执行时长：1 ms	过程	更多

- Trace 链路查询的作用
  1. 清晰的展示整个请求链路
  2. 能了解请求经过哪些服务、哪些机器、耗时情况、跨机房调用情况等
  3. 了解各个服务的执行情况，成功还是失败，是否重试，失败是否造成影响
  4. 很容易能看出整个请求的耗时分布，了解请求的瓶颈在哪

- 关联日志查询


全部折叠
跳转至
全部展开
[h_qta_openapi][/home/q/www/qta.openprice.provider/logs/qta.log]
[h_qta_orderbooking][/home/q/www/qta.order.qtaman/logs/catalina.out]
[h_qta_orderpackage_p][/home/q/www/qta.order.package/logs/catalina.out]
[h_qta_orderpackage_p][/home/q/www/qta.order.package/logs/dubbo-access-provider.2017-07-09-17.log]
[h_qta_orderpackage_p][/home/q/www/qta.order.package/logs/catalina.out]
[h_qta_orderpackage_p/home/q/www/qta.order.package/logs/catalina.out]

- 按条件搜索
  1. 关键数据记录及搜索，例如记录下订单号，可以根据订单号搜索

- 服务上下游关系统计

应用或服务	执行次数	平均执行时间
▼ [应用] h_qta_orderbooking		
[数据库] qta_order_booking	46435次	0.44ms
▼ [应用] cc_online_complain		
[服务] /complain/batchAppGet.do	19980次	3.59ms
[服务] /orderfaq/orderActionList.*	1223次	45.31ms
[服务] /onlinefaq/api/orderfaqlist	128次	37.68ms
▶ [应用] complain.order.qunar.com		
▼ [应用] qta_order_core_p		
[服务] RefundPriceRemote:queryCancelRefundProcess	3568次	13.58ms
[服务] PayFlowService:goPayProductDetailForMobile	619次	17.70ms
[服务] OrderFlowService:queryOrderRemark	282次	3.70ms
[服务] PromoBalanceService:queryMemberHongBaoBalance	93次	15.24ms

- 数据库操作统计

1. 按照库、表、语句三个维度，分析QPS、耗时情况

other	共有23次操作
file_public_status	共有51次操作
config_op_log	共有59次操作
config_candidate	共有62次操作
file_description	共有93次操作
config_snapshot	共有94次操作
file_permission	共有147次操作

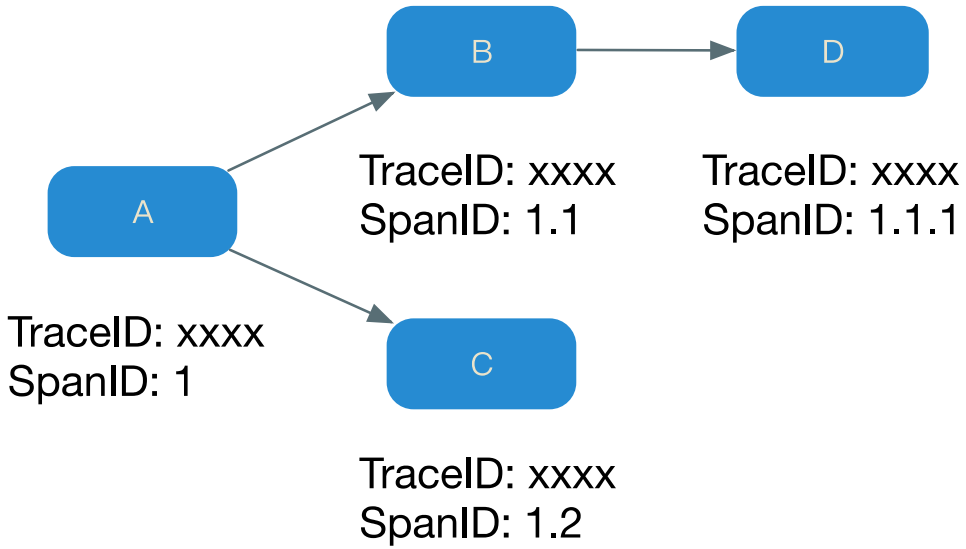
SELECT version FROM ...ET_ATON(?)	5315次
INSERT INTO config_l..., ?, ?, ?)	948次
SELECT based_version...ersion = ?	935次
INSERT INTO `config_...me = now()	560次
DELETE FROM `config_...D port = ?	508次
SELECT operator FROM...ersion = ?	362次
SELECT group_id, dat...tatus != ?	175次
SELECT ref_group_id,...tatus != ?	141次

- 透明数据传递
  1. Trace 链路需要贯穿多个系统，是否可以作为一个旁路来传输数据？
  2. 透明数据传递就是利用 Trace 链条让上游可以将一些数据一直向下传递
  3. 主要用于传递一些开关、标记，也可用来传递整个链路都需要的数据
  4. ABTest 时可以传输分支标记，控制流程走向
  5. 单元化里利用这个传递单元 ID
  6. 业务里可能输出日志太多，平时不记录一些日志，利用这个在最开头的节点添加记录详细日志标识，后续系统都可以根据开关记录详细日志

- 其它
  1. 根据日志分析出的异常自动关联 Trace
  2. 服务调用 QPS、耗时情况统计
  3. 最近最慢的服务调用统计
  4. 最近最慢的 SQL 语句统计
  5. 服务调用失败统计

QTracer 客户端核心设计

- 数据模型
  1. Trace，对应右图中的整个链路
     a. 由 Span 聚合而成的树形结构
     b. 代表了一次完整的请求流程
  2. Span，对应右图中的每个节点
     a. 记录系统中的一次操作
     b. 存储和展示的基本单元
     c. 会保存服务名、起始时间、结果、类型等信息



- 基本概念 —— TraceID
  1. 全局唯一 ID，用于关联一次请求链路上的所有系统调用
  2. 需要在各个系统间传递
  3. QTracer 的 TraceID 设计
     a. 举例：pf_pitcher_170709.192436.10.90.14.241.2509.1802772394_1
     b. 起始应用名，方便了解链路起始应用，便于按照前缀搜索
     c. 生成时间、IP 地址，提供更多信息，方便查问题
     d. 递增计数，保证 ID 唯一性
     e. 采样标识，能方便的区分一条 Trace 是否采样
  4. TraceID 只要保证全局唯一即可，规则可以自行按需定制

- 基本概念 —— SpanID
  1. 标识执行顺序和调用关系
  2. 需要在各个系统间传递
  3. QTracer 的 SpanID 设计
     a. 取 SpanID 为 1 的 Span 作为 Root Span，表示请求的起始点
     b. 如果是同一级的调用，则增长本级 ID ，例如：1.1, 1.2, 1.3
     c. 如果有调用关系，那么需要增长 SpanID 的级别，例如：1, 1.1, 1.1.1

- 基本概念 —— TimelineAnnotation
  1. 记录一个 Span 内部的时序性信息
  2. 限制在单个 Span 内部，不会传递
  3. 举例如下：
     a. HTTP 客户端一次请求记录一个 Span
     b. 请求会有多个步骤，比如建立连接、写入请求完成、接收回复完成等
     c. 这种可以记录到 TimelineAnnotation，能方便的展示内部时序流程

- 基本概念 —— KVAnnotation
  1. 用于记录业务自定义数据，比如订单号、UID等
  2. 限制在单个 Span 内部，不会传递
  3. 收集完成后，能根据 KVAnnotation 的值搜索相关的 Trace

- 基本概念 —— TraceContext
  1. 保存业务自定义数据，但不会被收集
  2. 能够在整个 Trace 链路中一直向下传递
  3. 比如 AB Test，可以利用 TraceContext 传递标志，走不同的流程

- 核心 API
  1. startTrace 开启一个新的 Span，可以在 Span 中添加各种数据
  2. 用户基本不需要使用，提供的各个组件都添加了默认埋点

```
final QTraceScope scope = client.startTrace( desc: "service call");
try {
    scope.addAnnotation("uid", "129xkk31");

    scope.addTimeAnnotation( msg: "start process");
    // do step 1
    scope.addTimeAnnotation( msg: "step 1 done");
    // do step 2
    scope.addTimeAnnotation( msg: "step 2 done");

    scope.addTraceContext("branchSwitch", "1");
} finally {
    scope.close();
}
```

- 基本实现原理
  1. 在单个系统内部
     a. 同步调用，利用 ThreadLocal 保存上下文关系
     b. 异步调用及跨线程调用，需要显式调用延续上下文关系
  2. 跨系统情况下
     a. 需要在网络传输的同时传递上下文关系数据，保证 Trace 的延续，支持 Dubbo、HTTP、MQ 等

- 无侵入埋点
  1. 直接使用 API 记录对用户不够友好，为此我们对各个组件都添加了无侵入的埋点，用户直接开启采样就能得到足够的信息
  2. 主要分为两类：
     a. 对于公司内部维护的组件，可以直接添加埋点代码，这样能够更精确的控制功能，记录更多信息。比如Dubbo、消息队列。
     b. 对于不是公司内部维护的组件，由于无法修改源码添加埋点功能，所以采用了字节码修改的方式，能够在运行时为指定的类添加埋点功能。比如 MySQL 和 PG 的 driver。

- 字节码插桩
  1. 运行时动态修改字节码的技术，能调整代码的行为
  2. 通过为 JVM 添加代理（agent）来启用字节码插桩功能
  3. QTracer 实现了一套利用配置指定对某些类的方法进行插桩的功能。
  4. 针对 MySQL 和 PG 的操作的插桩就是通过在配置文件中指定驱动中的方法、字段等实现的。
  5. 有新的客户端需要插桩时，通过添加新的插桩配置即可。

- 本地方法快速插桩
  1. 除了中间件、数据库等预先埋点的组件，有些业务系统还需要跟踪本地方法
  2. 直接使用 API ？对业务来说比较麻烦，还需要熟悉 API 的使用，加入业务无关代码。
  3. 解决方法：注解。利用注解标记需要跟踪的方法和需要记录的数据。编译时自动生成插桩配置，启动时 QTracer 载入配置即可自动插桩。
  4. @QTrace 注解标记要跟踪的方法，@QP 标记要记录的参数，@QF 标记要记录的成员变量

- 本地方法快速插桩

```
@QF
private String endpoint;

@QTrace
public byte[] request(@QP final String path, final String token) {
    // do request
    return new byte[]{};
}
```

- 日志关联
  1. 利用 MDC(Mapped Diagnostic Context) 保存 TraceID 和 SpanID
  2. Span 生成时保存数据到 MDC，关闭时清理掉 MDC 中的数据
  3. 使用方配置应用的 log pattern ，打印 QTracer 的 MDC 到日志中
  4. 日志收集收集日志之后，会有实时任务分析并关联日志和 Trace

- 遇到的问题
  1. 在跨线程或者使用了线程池的情况下，ThreadLocal 中的上下文无法自动传递。我们提供了针对线程和线程池的 wrapper，在 wrapper 中暂存上下文并跨线程传递。

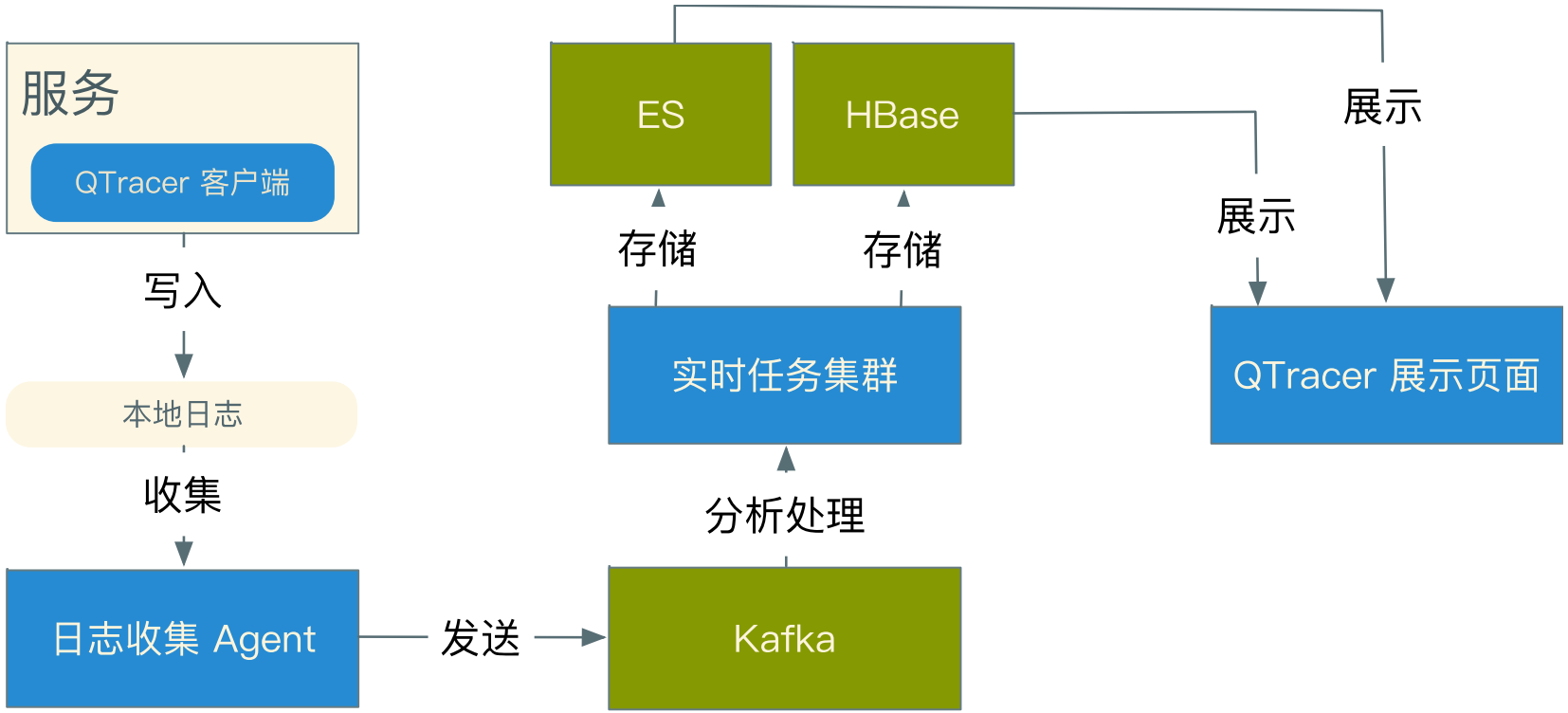
```
QTracer.wrap(Executors.newFixedThreadPool( nThreads: 4));

QTracer.wrap(new Runnable() {...});
```

- 参考对象
  1. Dapper，Google 的分布式追踪系统，先驱者
  2. Zipkin，Google Dapper 的开源实现
  3. EagleEye，淘宝的分布式追踪系统，提供更丰富的功能

QTracer 系统架构

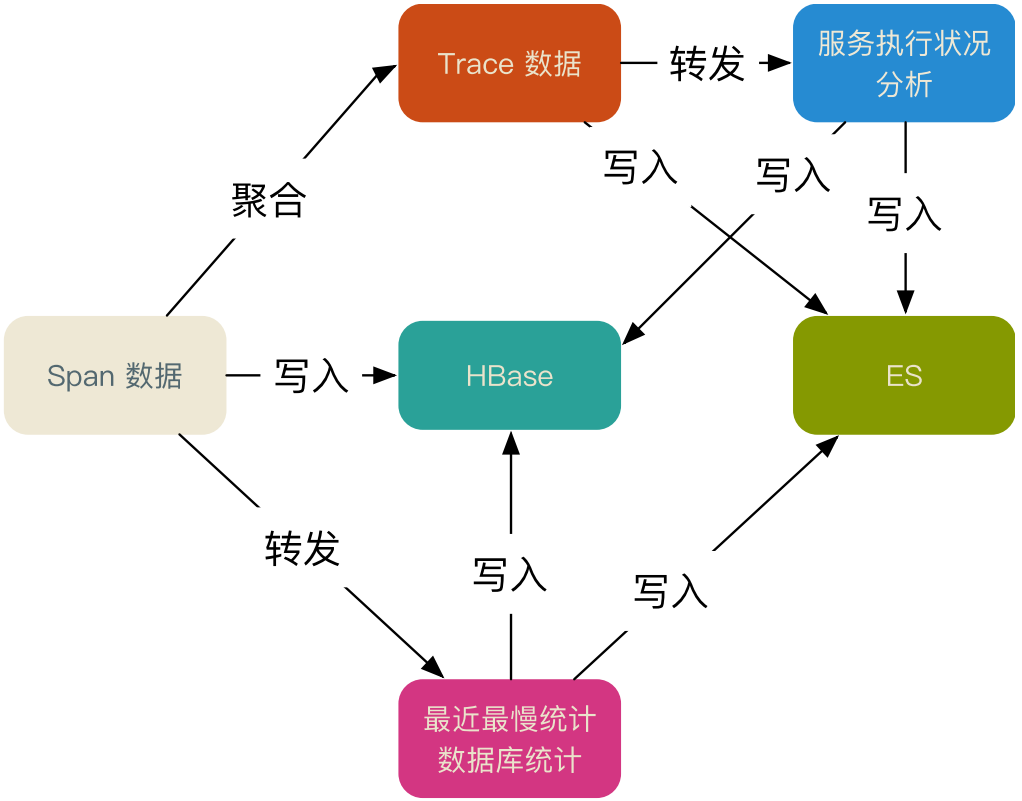
- 系统架构


服务
QTracer 客户端
写入
本地日志
收集
日志收集 Agent
发送
Kafka
分析处理
实时任务集群
存储
存储
ES
HBase
展示
展示
QTracer 展示页面

- Trace 数据记录和收集
  1. 写本地的日志来暂存 Trace 数据
  2. 要尽量降低资源消耗，控制磁盘空间占用
  3. 异步批量写日志，按大小轮转，控制日志文件数量，极端情况下丢弃数据
  4. 利用日志收集 Agent 收集 Trace 数据
  5. Trace 数据全部发送到 Kafka 暂存，等待处理

- Trace 数据处理与分析
  1. 实时任务方面选择了 Samza 实时任务框架
     a. 和 Kafka 配合良好
     b. 使用简单，功能足够
     c. 提供了方便缓存状态的本地cache，基于rocksdb，提供异常恢复功能
  2. 利用 Kafka 暂存中间数据，方便下游直接接收继续分析
     a. 一开始 Trace 数据可能直接保存即可
     b. 先暂存到 Kafka ，后续需要的时候可以方便的接入继续分析

- Trace 数据处理与分析



- Trace 数据处理与分析
  1. 主要分为两条路线，针对 Span 的处理与针对聚合出的 Trace 的处理
  2. 针对 Span 数据：
     a. 直接保存到 HBase 提供实时查询
     b. 聚合形成 Trace 链路数据
     c. 最近最慢服务请求、数据库操作等的统计
     d. 按照类别统计 QPS 等数据，比如数据库操作类型、语句。

- Trace 数据处理与分析

3. 针对 Trace 数据：
    a. 整合精简后保存到 ES 提供按条件搜索功能
    b. 分析统计链路的上下游调用关系
    c. 能分析出服务调用的 QPS、平均耗时情况
    d. 能分析出服务之间的依赖关系

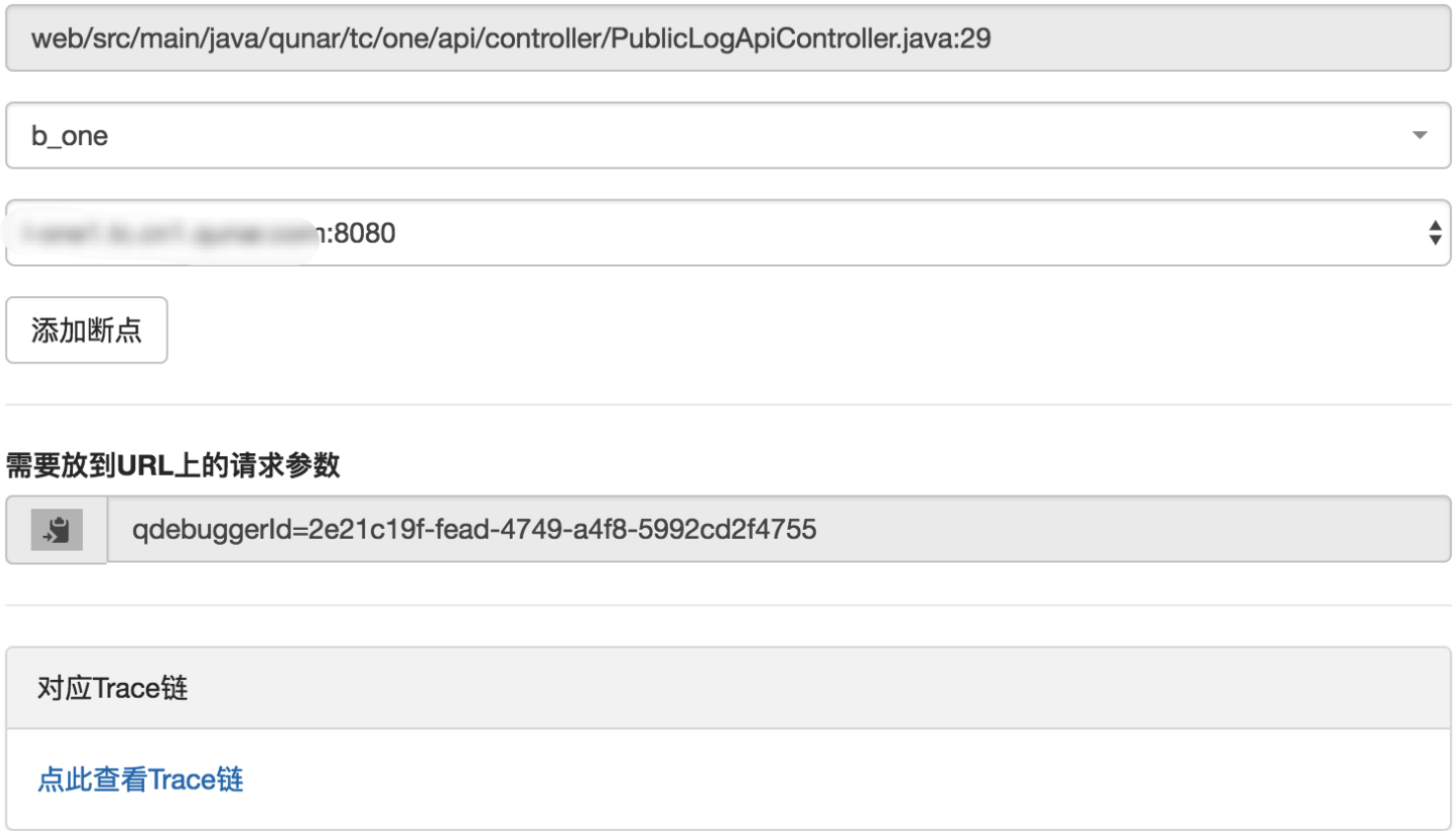
- Trace 数据存储 —— HBase
  1. 每行保存一条 Trace 链路数据，提供实时查询 Trace 链路的功能
  2. TraceID 的变形作为 RowKey，SpanID 作为列名
  3. 为什么对 TraceID 做变形？为了使数据在 Hbase 中尽量均匀分布

- Trace 数据存储 —— ES
  1. 保存精简后的 Trace 链路数据，提供按照条件搜索的功能，有延时
  2. 保存根据 Trace 链路分析出的上下游依赖关系

QTracer Debug

- 简介
  1. 除了 QTracer 里提供的预先埋点，有时也想要获取代码运行到某位置时的状态
  2. 类似于 IDE 中提供的 debug 功能，通过在源码中设置断点，可以收集断点处所能访问的所有变量的值及调用栈，但是不会暂停应用
  3. 使用流程
     a. 在 前端页面上选择项目，浏览代码，在代码行上标记断点
     b. 选择应用的机器，启用断点
     c. 访问能经过断点的 URL，带上指定的参数
     d. 等待数据收集完成即可自动展示

- 简介


web/src/main/java/qunar/tc/one/api/controller/PublicLogApiController.java:29
b_one
:8080
添加断点
需要放到URL上的请求参数
qdebuggerId=2e21c19f-fead-4749-a4f8-5992cd2f4755
对应Trace链
点此查看Trace链

- 简介

成员变量

logService	`qunar.tc.one.trace.service.impl.LogServiceImpl@1b86eb64`

参数及局部变量

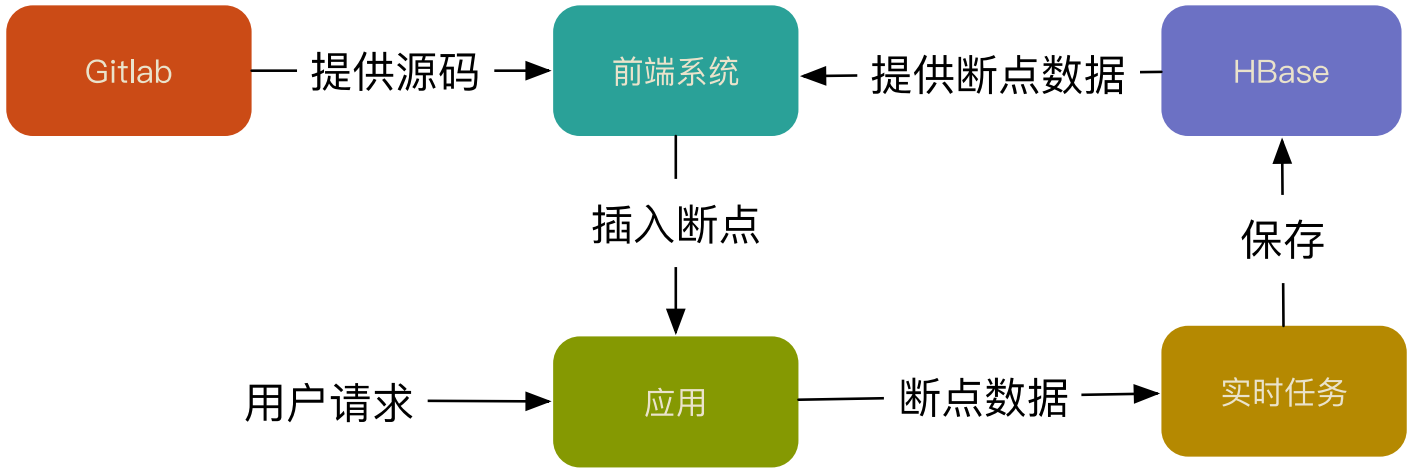
traceId	`"xxxxx"`
this	`qunar.tc.one.api.controller.PublicLogApiController@472fcf1c`

调用栈

qunar.tc.one.api.controller.PublicLogApiController.byTrace

sun.reflect.GeneratedMethodAccessor300.invoke

- 实现概要



- 实现概要
  1. 展示层面借助 Gitlab 的 API 实现代码浏览和按行设置断点功能
  2. 借助 QTracer 的字节码插桩加入断点代码，执行时记录状态
  3. 利用 QTracer 进行数据的记录和收集
  4. 实时任务筛选出含有 debug 数据的 Trace 链条，提取断点数据。
  5. 在前端展示断点数据

- 断点的添加
  1. 分析应用所有的类，建立源文件+行号到类的映射关系。
  2. 根据断点位置找到需要添加断点的类
  3. 分析类，收集类中所有变量的作用域信息
  4. 修改类的字节码，在指定位置插入收集所有作用域内变量等数据的代码

- 数据记录及收集
  1. 数据记录依赖 QTracer 自身的 KVAnnotation 功能，直接存放到 QTracer 的 Span 中。
  2. 数据收集直接通过 QTracer 的写本地日志+日志收集方式。
  3. 实时任务对 QTracer 数据进行筛选，保存到 HBase 中。

ctrip tech

携程技术中心

THANK YOU!

Q&A