微服务架构下分布式事务解决方案——阿里GTS

# 1 微服务的发展

微服务倡导将复杂的单体应用拆分为若干个功能简单、松耦合的服务，这样可以降低开发难度、增强扩展性、便于敏捷开发。当前被越来越多的开发者推崇，很多互联网行业巨头、开源社区等都开始了微服务的讨论和实践。Hailo有160个不同服务构成，NetFlix有大约600个服务。国内方面，阿里巴巴、腾讯、360、京东、58同城等很多互联网公司都进行了微服务化实践。当前微服务的开发框架也非常多，比较著名的有Dubbo、SpringCloud、thrift 、grpc等。

# 2 微服务落地存在的问题

虽然微服务现在如火如荼，但对其实践其实仍处于探索阶段。很多中小型互联网公司，鉴于经验、技术实力等问题，微服务落地比较困难。如著名架构师Chris Richardson所言，目前存在的主要困难有如下几方面：

1）单体应用拆分为分布式系统后，进程间的通讯机制和故障处理措施变的更加复杂。

2）系统微服务化后，一个看似简单的功能，内部可能需要调用多个服务并操作多个数据库实现，服务调用的分布式事务问题变的非常突出。

3）微服务数量众多，其测试、部署、监控等都变的更加困难。

随着RPC框架的成熟，第一个问题已经逐渐得到解决。例如dubbo可以支持多种通讯协议，springcloud可以非常好的支持restful调用。对于第三个问题，随着docker、devops技术的发展以及各公有云paas平台自动化运维工具的推出，微服务的测试、部署与运维会变得越来越容易。

而对于第二个问题，现在还没有通用方案很好的解决微服务产生的事务问题。分布式事务已经成为微服务落地最大的阻碍，也是最具挑战性的一个技术难题。 为此，本文将深入和大家探讨微服务架构下，分布式事务的各种解决方案，并重点为大家解读阿里巴巴提出的分布式事务解决方案----GTS。该方案中提到的**GTS**是全新一代解决微服务问题的分布式事务互联网中间件。

# 3 SOA分布式事务解决方案

## 3.1 基于XA协议的两阶段提交方案

交易中间件与数据库通过 XA 接口规范，使用两阶段提交来完成一个全局事务， XA 规范的基础是两阶段提交协议。
第一阶段是表决阶段，所有参与者都将本事务能否成功的信息反馈发给协调者；第二阶段是执行阶段，协调者根据所有参与者的反馈，通知所有参与者，步调一致地在所有分支上提交或者回滚。

昵称：姜宇-GTS
园龄：10个月
粉丝：10
关注：0
+加关注

< 2019年1月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 30 | 31 | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

GTS(2)
分布式事务(2)
分布式系统(2)
云计算(2)

随笔档案

2018年3月 (1)
2018年2月 (1)

最新评论

1. Re:谈谈分布式事务
对事务分析分很详细，赞一个
--流离岁月
2. Re:微服务架构下分布式事务解决方案——阿里GTS
GTS确实很赞，其核心原理是 补偿。但这个补偿做得很屌，补偿操作由框架自动生成，无需业务干预，框架会记录修改前的记录值到上面的txc_undo_log里，若需要回滚，则拿出undo_log的记录覆盖回.....
--YOYO&#
3. Re:微服务架构下分布式事务解决方案——阿里GTS
很赞
--执笔记忆的空白
4. Re:微服务架构下分布式事务解决方案——阿里GTS
点赞，很nice
--晓晨Master
5. Re:谈谈分布式事务
膜拜大神!!!!!!
--mtsx

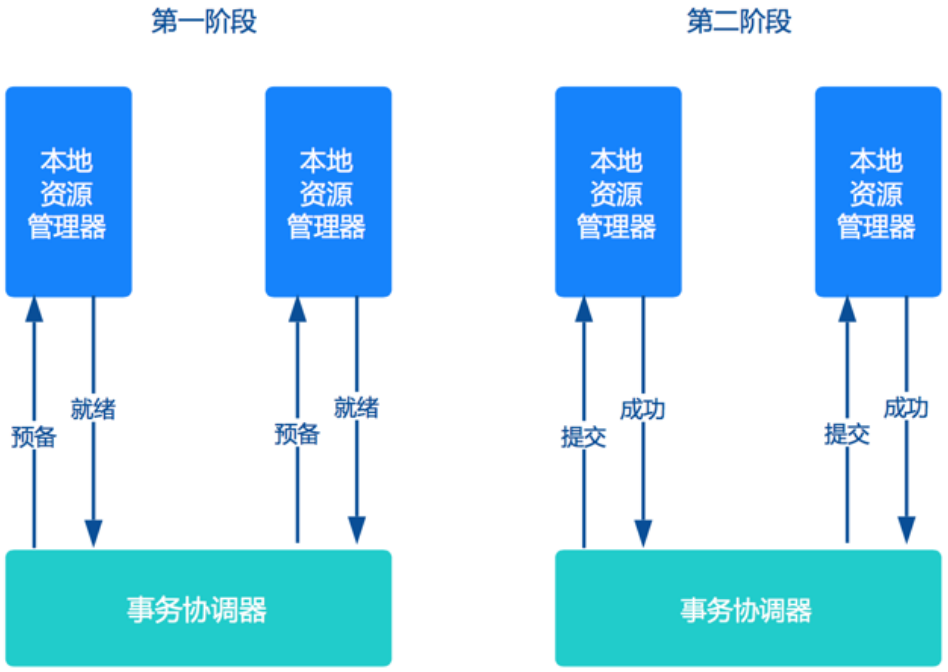
阅读排行榜

1. 微服务架构下分布式事务解决方案——阿里GTS(31683)

2. 谈谈分布式事务(1611)

评论排行榜
1. 微服务架构下分布式事务解决方案——阿里GTS(5)
2. 谈谈分布式事务(2)

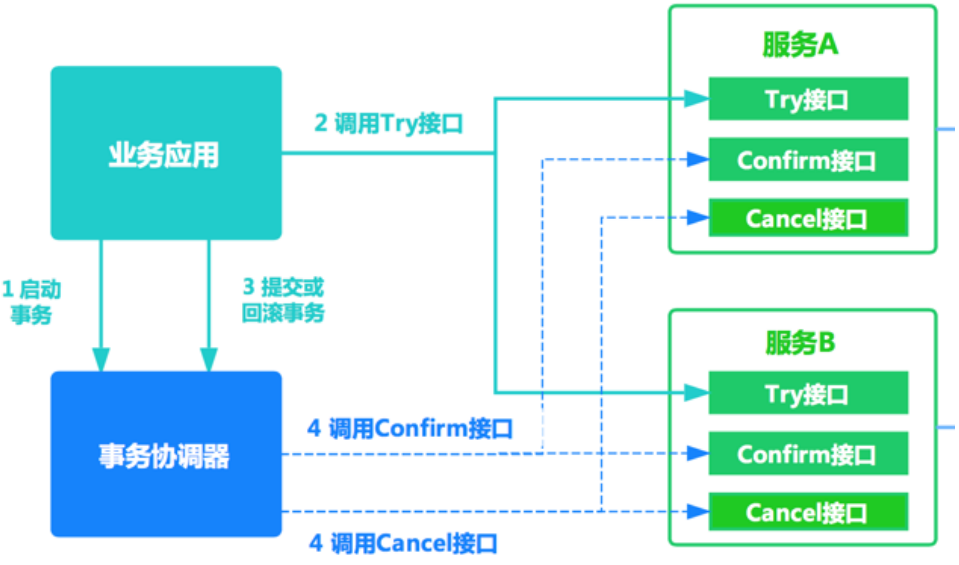
推荐排行榜
1. 微服务架构下分布式事务解决方案——阿里GTS(5)




两阶段提交方案应用非常广泛，几乎所有商业OLTP数据库都支持XA协议。但是两阶段提交方案锁定资源时间长，对性能影响很大，基本不适合解决微服务事务问题。

### 3.2 TCC方案

TCC方案在电商、金融领域落地较多。TCC方案其实是两阶段提交的一种改进。其将整个业务逻辑的每个分支显式的分成了Try、Confirm、Cancel三个操作。Try部分完成业务的准备工作，confirm部分完成业务的提交，cancel部分完成事务的回滚。基本原理如下图所示。




事务开始时，业务应用会向事务协调器注册启动事务。之后业务应用会调用所有服务的try接口，完成一阶段准备。之后事务协调器会根据try接口返回情况，决定调用confirm接口或者cancel接口。如果接口调用失败，会进行重试。

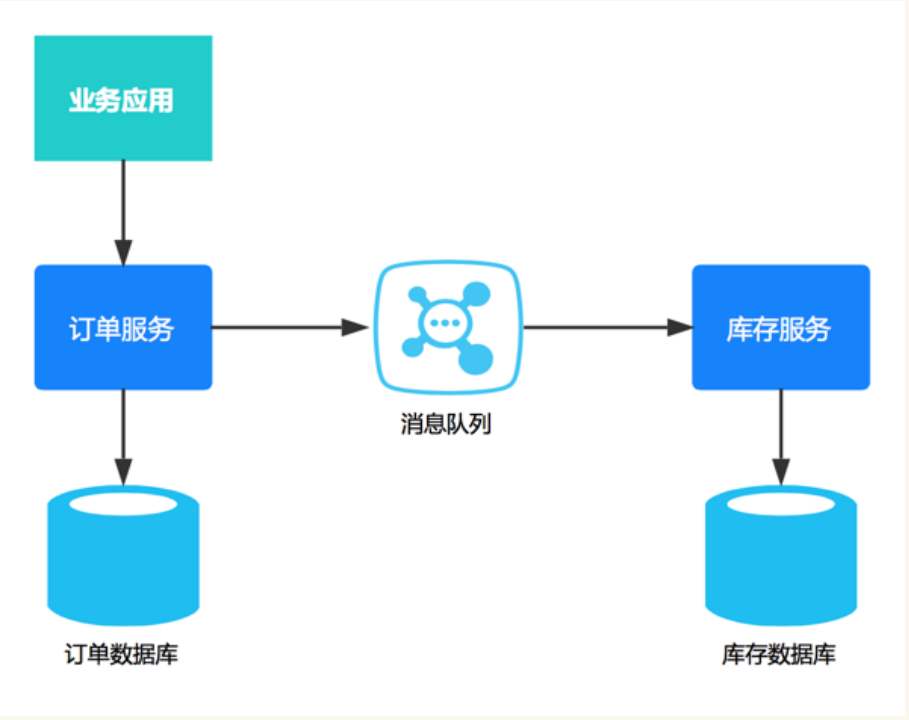
TCC方案让应用自己定义数据库操作的粒度，使得降低锁冲突、提高吞吐量成为可能。 当然TCC方案也有不足之处，集中表现在以下两个方面：

- 对应用的侵入性强。业务逻辑的每个分支都需要实现try、confirm、cancel三个操作，应用侵入性较强，改造成本高。
- 实现难度较大。需要按照网络状态、系统故障等不同的失败原因实现不同的回滚策略。为了满足一致性的要求，confirm和cancel接口必须实现幂等。

上述原因导致TCC方案大多被研发实力较强、有迫切需求的大公司所采用。微服务倡导服务的轻量化、易部署，而TCC方案中很多事务的处理逻辑需要应用自己编码实现，复杂且开发量大。

### 3.3 基于消息的最终一致性方案

消息一致性方案是通过消息中间件保证上、下游应用数据操作的一致性。基本思路是将本地操作和发送消息放在一个事务中，保证本地操作和消息发送要么两者都成功或者都失败。下游应用向消息系统订阅该消息，收到消息后执行相应操作。




消息方案从本质上讲是将分布式事务转换为两个本地事务，然后依靠下游业务的重试机制达到最终一致性。基于消息的最终一致性方案对应用侵入性也很高，应用需要进行大量业务改造，成本较高。

## 4 GTS--分布式事务解决方案

GTS是一款分布式事务中间件，由阿里巴巴中间件部门研发，可以为微服务架构中的分布式事务提供一站式解决方案。

更多GTS资料请访问创始人微博。

### 4.1 GTS的核心优势

- 性能超强

  GTS通过大量创新，解决了事务ACID特性与高性能、高可用、低侵入不可兼得的问题。单事务分支的平均响应时间在2ms左右，3台服务器组成的集群可以支撑3万TPS以上的分布式事务请求。

- 应用侵入性极低

  GTS对业务低侵入，业务代码最少只需要添加一行注解（@TxcTransaction）声明事务即可。业务与事务分离，将微服务从事务中解放出来，微服务关注于业务本身，不再需要考虑反向接口、幂等、回滚策略等复杂问题，极大降低了微服务开发的难度与工作量。

- 完整解决方案

  GTS支持多种主流的服务框架，包括EDAS，Dubbo，Spring Cloud等。
  有些情况下，应用需要调用第三方系统的接口，而第三方系统没有接入GTS。此时需要用到GTS的MT模式。GTS的MT模式可以等价于TCC模式，用户可以根据自身业务需求自定义每个事务阶段的具体行为。MT模式提供了更多的灵活性，可能性，以达到特殊场景下的自定义优化及特殊功能的实现。

- 容错能力强

  GTS解决了XA事务协调器单点问题，实现真正的高可用，可以保证各种异常情况下的严格数据一致。

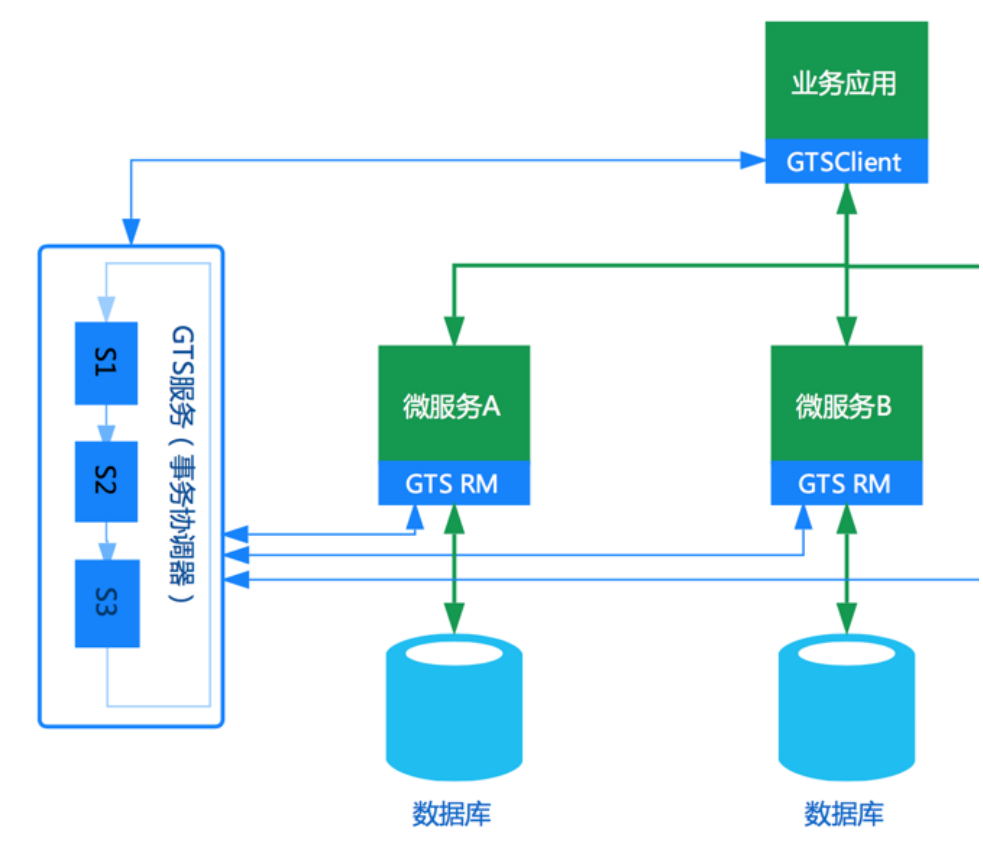
### 4.2 GTS的应用场景

GTS可应用在涉及服务调用的多个领域，包括但不限于金融支付、电信、电子商务、快递物流、广告营销、社交、即时通信、手游、视频、物联网、车联网等，详细介绍可以阅读《GTS--阿里巴巴分布式事务全新解决方案》一文。

### 4.3 GTS与微服务的集成

GTS包括客户端（GTS Client）、资源管理器（GTS RM）和事务协调器（GTS Server）三个部分。GTS Client主要用来界定事务边界，完成事务的发起与结束。GTS RM完成事务分支的创建、提交、回滚等操

作。GTS Server主要负责分布式事务的整体推进，事务生命周期的管理。GTS和微服务集成的结构图如下所示，GTS Client需要和业务应用集成部署，RM与微服务集成部署。

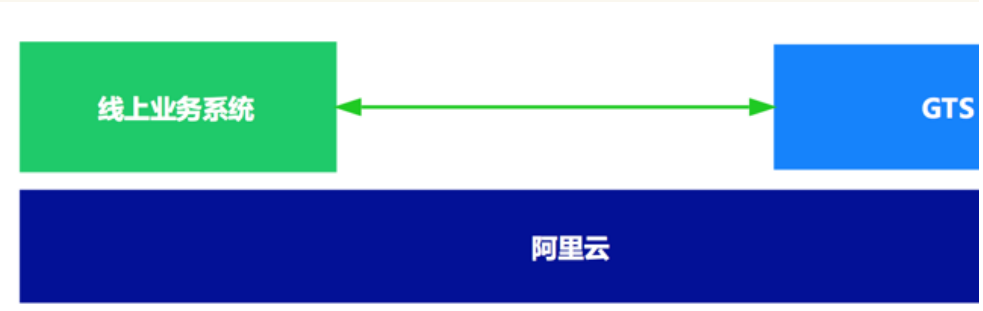



### 4.4 GTS的输出形式

GTS目前有三种输出形式：公有云输出、公网输出、专有云输出。

#### 4.4.1 公有云输出

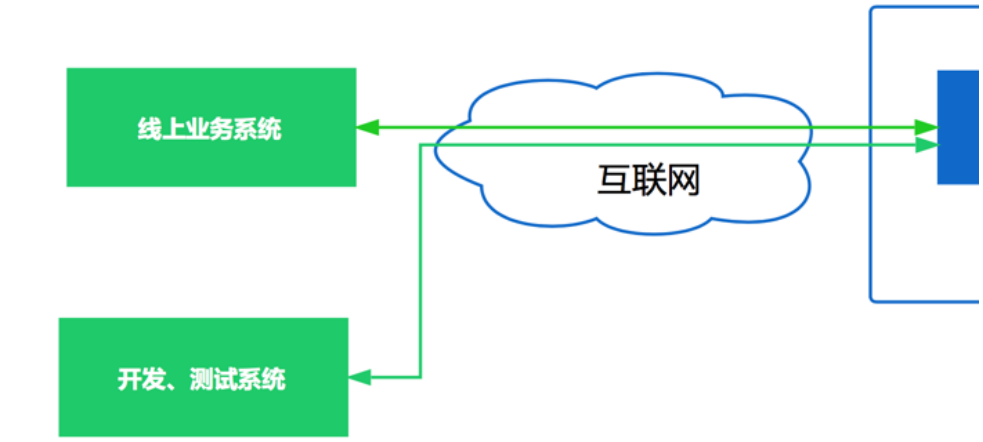
这种输出形式面向阿里云用户。如果用户的业务系统已经部署到阿里云上，可以申请开通公有云GTS。开通后业务应用即可通过GTS保证服务调用的一致性。这种使用场景下，业务系统和GTS间的网络环境比较理想，达到很好性能。




#### 4.4.2 公网输出

这种输出形式面向于非阿里云的用户，使用更加方便、灵活，业务系统只要能连接互联网即可享受GTS提供的云服务（与公有云输出的差别在于客户端部署于用户本地，而不在云上）。

在正常网络环境下，以包含两个本地事务的全局事务为例，事务完成时间在20ms左右，50个并发就可以轻松实现1000TPS以上分布式事务，对绝大多数业务来说性能是足够的。在公网环境，网络闪断很难完全避免，这种情况下GTS仍能保证服务调用的数据一致性。




具体使用样例使用参见4.7节GTS的工程样例。

#### 4.4.3 专有云输出

这种形式主要面向于已建设了自己专有云平台的大用户，GTS可以直接部署到用户的专有云上，为专有云提供分布式事务服务。目前已经有10多个特大型企业的专有云使用GTS解决分布式事务难题，性能与稳定性经过了用户的严格检测。

### 4.5 GTS的使用方式

GTS对应用的侵入性非常低，使用也很简单。下面以订单存储应用为例说明。订单业务应用通过调用订单服务和库存服务完成订单业务，服务开发框架为Dubbo。

#### 4.5.1 订单业务应用

在业务函数外围使用@TxcTransaction注解即可开启分布式事务。Dubbo应用通过隐藏参数将GTS的事务xid传播到服务端。

```
 @TxcTransaction(timeout = 1000 * 10)
public void Bussiness(OrderService orderService, StockService stockService, String
userId) {
    //获取事务上下文
    String xid = TxcContext.getCurrentXid();
    //通过RpcContext将xid传到一个服务端
    RpcContext.getContext().setAttachment("xid", xid);

    //执行自己的业务逻辑
    int productId = new Random().nextInt(100);
    int productNum = new Random().nextInt(100);
    OrderDO orderDO = new OrderDO(userId, productId, productNum, new Timestamp(new
Date().getTime()));
    orderService.createOrder(orderDO);

    //通过RpcContext将xid传到另一个服务端
    RpcContext.getContext().setAttachment("xid",xid);
    stockService.updateStock(orderDO);
}
```

#### 4.5.2 服务提供者

更新库存方法

```
public int updateStock(OrderDO orderDO) {

//获取全局事务ID，并绑定到上下文

String xid = RpcContext.getContext().getAttachment("xid");

TxcContext.bind(xid,null);

//执行自己的业务逻辑

int ret = jdbcTemplate.update("update stock set amount = amount - ? where product_id =
?",new Object[]{orderDO.getNumber(), orderDO.getProductId()});

TxcContext.unbind();
```

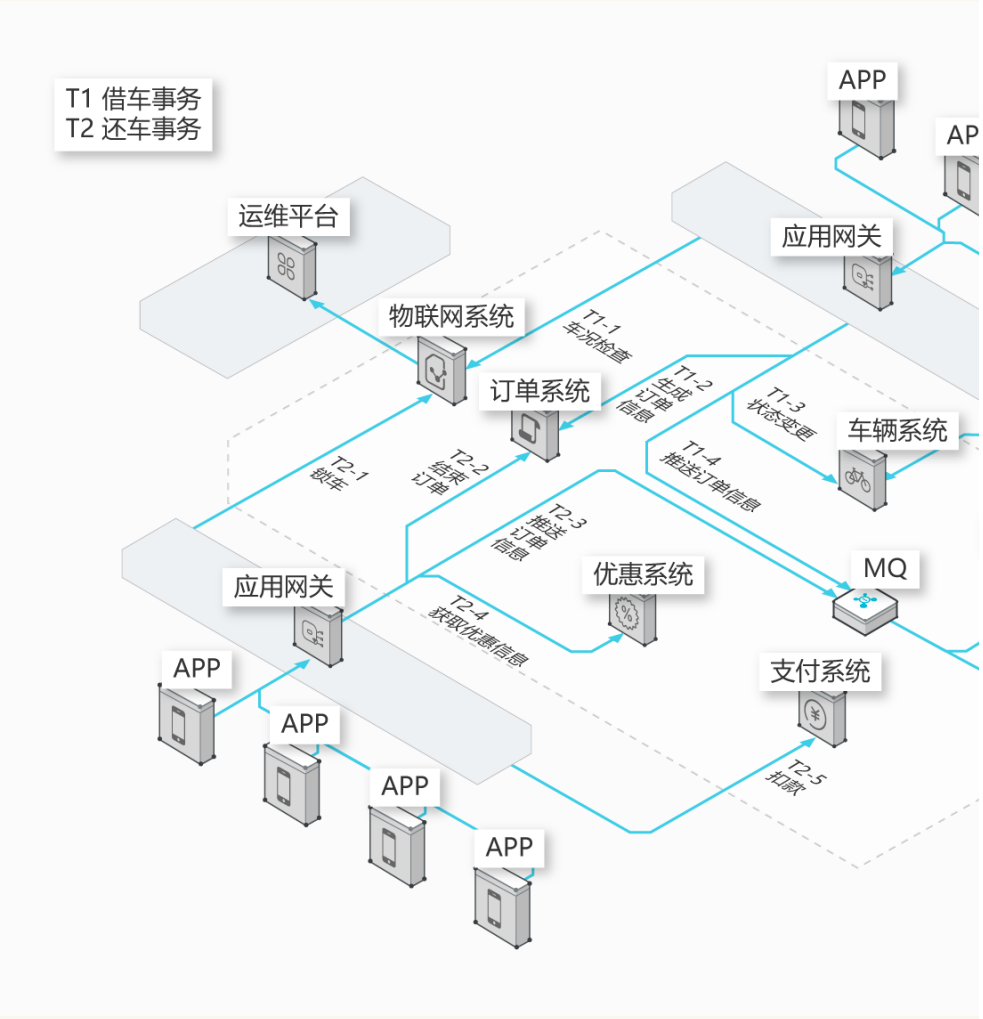
```
return ret;

}
```

### 4.6 GTS的应用情况

GTS目前已经在淘宝、天猫、阿里影业、淘票票、阿里妈妈、1688等阿里各业务系统广泛使用，经受了16年和17年两年双十一海量请求的考验。某线上业务系统最高流量已达十万TPS（每秒钟10万笔事务）。

GTS在公有云和专有云输出后，已经有了100多个线上用户，很多用户通过GTS解决SpringCloud、Dubbo、Edas等服务框架的分布式事务问题。业务领域涉及电力、物流、ETC、烟草、金融、零售、电商、共享出行等十几个行业，得到用户的一致认可。




上图是GTS与SpringCloud集成，应用于某共享出行系统。业务共享出行场景下，通过GTS支撑物联网系统、订单系统、支付系统、运维系统、分析系统等系各统应用的数据一致性，保证海量订单和数千万流水的交易。

### 4.7 GTS的工程样例

GTS的公有云样例可参考阿里云网站。在公网环境下提供sample-txc-simple和sample-txc-dubbo两个样例工程。

#### 4.7.1 sample-txc-simple样例

##### 4.7.1.1 样例业务逻辑

该样例是GTS的入门sample，案例的业务逻辑是从A账户转账给B账户，其中A和B分别位于两个MySQL数据库中，使用GTS事务保证A和B账户钱的总数始终不变。

##### 4.7.1.2 样例搭建方法

1) 准备数据库环境

安装MySQL，创建两个数据库db1和db2。在db1和db2中分别创建**txc_undo_log**表（SQL脚本见4.7.3）。在db1库中创建user_money_a表，在db2库中创建user_money_b表。

2) 下载样例

将sample-txc-simple文件下载到本地，样例中已经包含了GTS的SDK。

3) 修改配置

打开sample-txc-simple/src/main/resources目录下的txc-client-context.xml，将数据源的url、username、password修改为实际值。

4) 运行样例

在sample-txc-simple目录下执行build.sh编译本工程。编译完成后执行run.sh。

#### 4.7.2 sample-txc-dubbo 样例

##### 4.7.2.1 样例业务逻辑

本案例模拟了用户下订单、减库存的业务逻辑。客户端（Client）通过调用订单服务（OrderService）创建订单，之后通过调用库存服务（StockService）扣库存。其中订单服务读写订单数据库，库存服务读写库存数据库。由 GTS 保证跨服务事务的一致性。

##### 4.7.2.2 样例搭建方法

1) 准备数据库环境

安装MySQL，创建两个数据库db1和db2。在db1和db2中分别创建txc_undo_log表。在db1库中创建orders表，在db2库中创建stock表。

2) 下载样例

将样例文件sample-txc-dubbo下载到本地机器，样例中已经包含了GTS的SDK。

3) 修改配置

打开sample-txc-dubbo/src/main/resources目录，将dubbo-order-service.xml、dubbo-stock-service.xml两个文件中数据源的url、username、password修改为实际值。

4) 运行样例

a. 编译程序

在工程根目录执行 build.sh 命令，编译工程。编译后会在 sample-txc-dubbo/client/bin 目录下生成 order_run.sh、stock_run.sh、client_run.sh 三个运行脚本对应订单服务、库存服务以及客户端。

b. 运行程序

在根目录执行run.sh，该脚本会依次启动order_run.sh(订单服务)、stock_run.sh(库存服务)和client_run.sh(客户端程序)。

##### 4.7.2.3 其他说明

样例使用Multicast注册中心的声明方式。如果本机使用无线网络，dubbo服务在绑定地址时有可能获取ipv6地址，可以通过jvm启动参数禁用。
方法是配置jvm启动参数 ***-Djava.net.preferIPv4Stack=true***。

#### 4.7.3 SQL

##### 4.7.3.1 建表 txc_undo_log

CREATE TABLE `txc_undo_log` (

`id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',

`gmt_create` datetime NOT NULL COMMENT '创建时间',

`gmt_modified` datetime NOT NULL COMMENT '修改时间',

`xid` varchar(100) NOT NULL COMMENT '全局事务ID',

`branch_id` bigint(20) NOT NULL COMMENT '分支事务ID',

`rollback_info` longblob NOT NULL COMMENT 'LOG',

`status` int(11) NOT NULL COMMENT '状态',

`server` varchar(32) NOT NULL COMMENT '分支所在DB IP',

PRIMARY KEY ( `id` ),

KEY `unionkey` ( `xid` , `branch_id` )

) ENGINE=InnoDB AUTO_INCREMENT=211225994 DEFAULT CHARSET=utf8 COMMENT='事务日志表';

##### 4.7.3.2 建表 user_money_a

```
CREATE TABLE `user_money_a` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `money` int(11) DEFAULT NULL,
PRIMARY KEY ( `id` )
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

**4.7.3.3 建表 user_money_b**

```
CREATE TABLE `user_money_b` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `money` int(11) DEFAULT NULL,
PRIMARY KEY ( `id` )
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

**4.7.3.4 建表 orders**

```
CREATE TABLE `orders` (
 `id` bigint(20) NOT NULL AUTO_INCREMENT,
 `user_id` varchar(255) NOT NULL,
 `product_id` int(11) NOT NULL,
 `number` int(11) NOT NULL,
 `gmt_create` timestamp NOT NULL,
PRIMARY KEY ( `id` )
) ENGINE=MyISAM AUTO_INCREMENT=351 DEFAULT CHARSET=utf8
```

**4.7.3.5 建表 stock**

```
CREATE TABLE `stock` (
 `product_id` int(11) NOT NULL,
 `price` float NOT NULL,
 `amount` int(11) NOT NULL,
PRIMARY KEY ( `product_id` )
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

标签: 分布式事务, GTS, 分布式系统, 云计算



#1楼[楼主] 2018-03-08 14:18 | 姜宇-GTS

> 跑个例子（10分钟搞定）就知道是否靠谱了。

支持(0) 反对

#2楼 2018-04-25 11:32 | cleverlzc

> nice

支持(0) 反对

#3楼 2018-08-15 10:09 | 晓晨Master

> 点赞，很nice

支持(0) 反对

#4楼 2018-08-15 10:40 | 执笔记忆的空白

> 很赞

支持(0) 反对

#5楼 2018-08-28 09:29 | YOYO&#

> GTS确实很赞，其核心原理是 补偿。
>
> 但这个补偿做得很屌，补偿操作由框架自动生成，无需业务干预，框架会记录修改前的记录值到上面的txc_undo_log里，若需要回滚，则拿出undo_log的记录覆盖回原有记录
>
> 同时这里存在一个事务隔离级别的问题，GTS的做法是默认脏读，那么就可以直接拿数据库记录展示（但个人觉得应可以不做脏读，直接拿undo_log里的记录做mvcc,只要undo_log记录不大，都可以加载到内存里）。
>
> 还有另外一个问题是如何禁止其他事务对进行中的全局事务记录的更新，GTS的做法是需要接管APP中的数据源，这就可以解析控制业务要执行的SQL，对于update操作（或者select for update），予以禁止或等待。
>
> 不过整体的做法相当于魔改数据库，将数据库的部分功能拉到了业务APP里进行，并修改了默认隔离级别（脏读，如业务有用数据库记录乐观锁来控制并发的话，将会失效），还有就是，不通过GTS的定制数据源访问会访问修改到未交数据
>
> 如果作者能自行介绍下GTS的优缺点会更方便，更权威，毕竟大家做选型肯定要了解原理才敢用
>
> 还有这里也有个我自己写的分布式事务框架，欢迎大家喵喵 https://github.com/QNJR-GROUP/EasyTransaction

支持(1) 反对

刷新评论 刷新页面 返回顶

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库！



相关博文：
· 微服务架构的分布式事务解决方案
· 微服务架构下分布式事务方案
· 设计----【分布式事务】分布式事务和解决方案
· 分布式事务解决方案
· 微服务分布式事务



最新新闻：
· 魏武挥：我们该如何获取信息？
· “知识商人”吴晓波、罗振宇，且行且珍惜
· 75英寸4K屏幕！三星Micro LED模块化产品亮相CES
· 恒大、FF分手之后：许家印与贾跃亭的当务之急
· 蚂蚁金服旗下2公司变更法人 阿里最小合伙人胡喜接任
» 更多新闻...