ALIBABA CLOUD

阿里云

全局事务服务 GTS
用户指南

文档版本：20210119

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

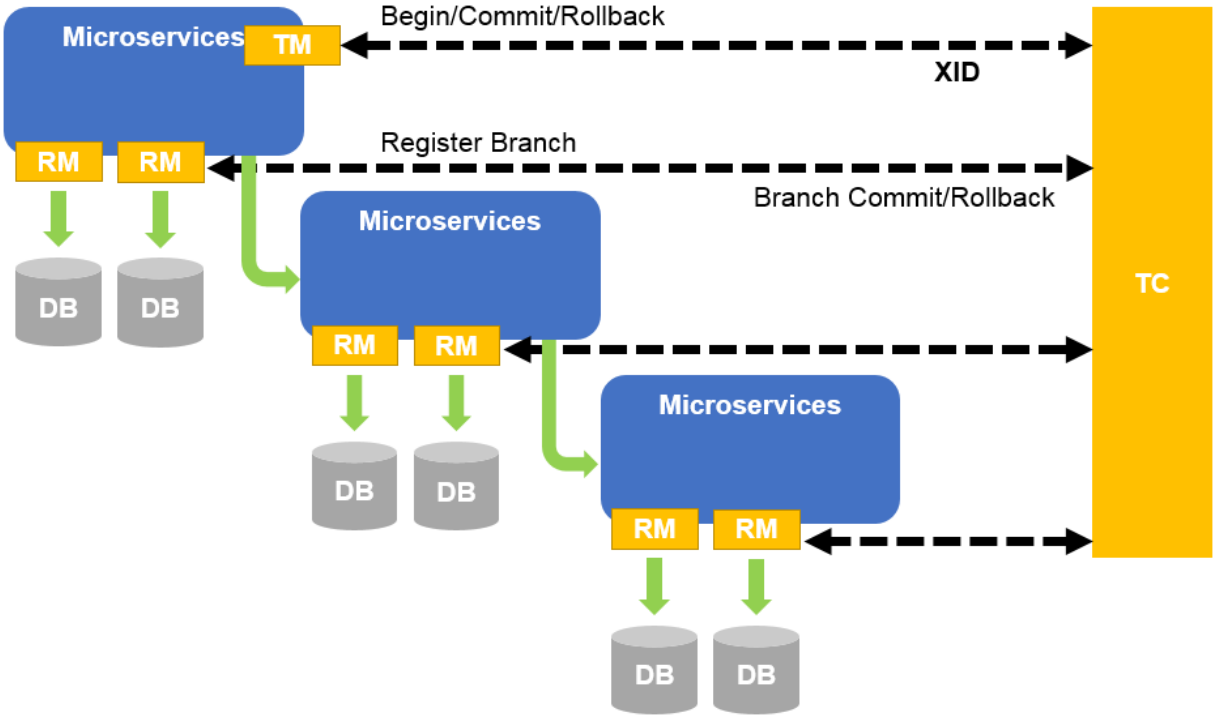
格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.分布式事务框架和事务模式

GTS 定义了一套事务框架以便描述分布式事务，在框架下支持不同事务模式运行。

核心组件定义



分布式事务包含以下 3 个核心组件：

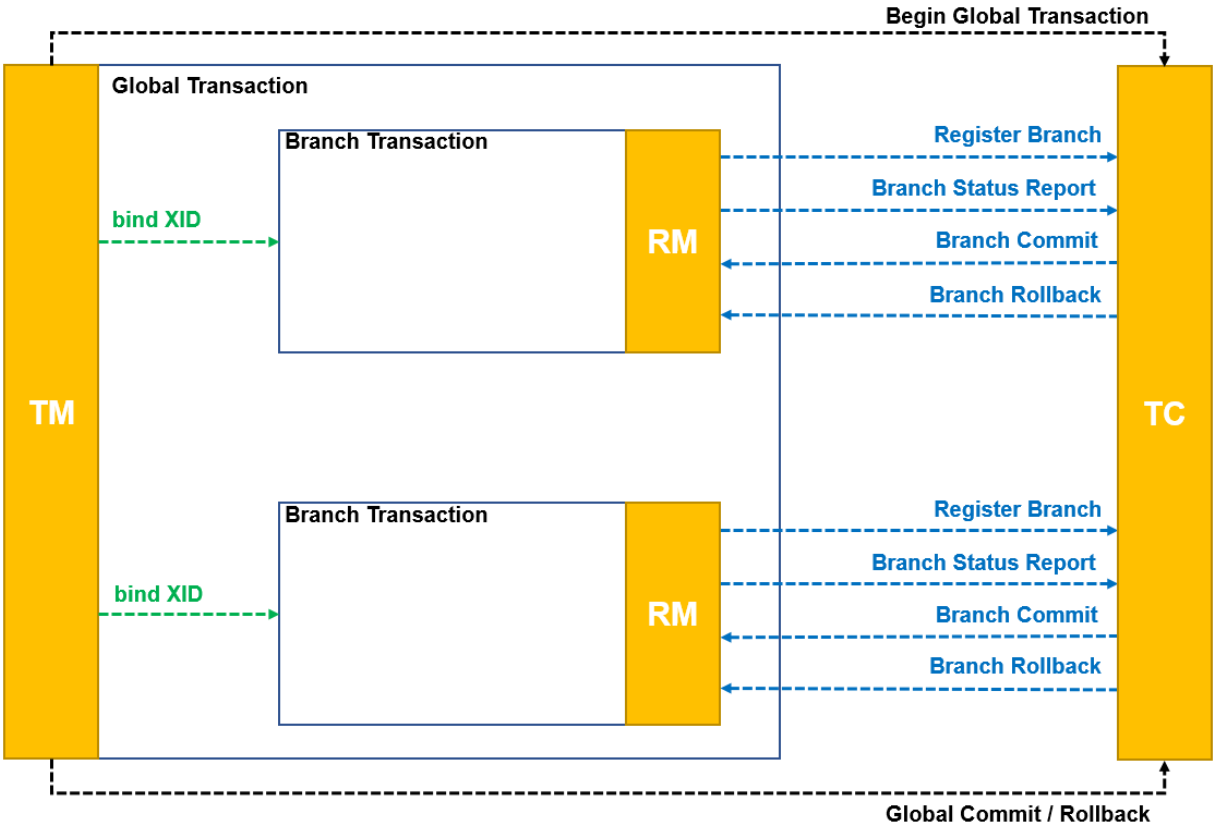
- Transaction Coordinator（TC）：事务协调器，维护全局事务的运行状态，负责协调并驱动全局事务的提交或回滚。
- Transaction Manager（TM）：控制全局事务的边界，负责开启一个全局事务，并最终发起全局提交或全局回滚的决议。
- Resource Manager（RM）：控制分支事务，负责分支注册、状态汇报，并接收事务协调器的指令，驱动分支（本地）事务的提交和回滚。

一个典型的事务过程包括：

1. TM 向 TC 申请开启（Begin）一个全局事务，全局事务创建成功并生成一个全局唯一的 XID。
2. XID 在微服务调用链路的上下文中传播。
3. RM 向 TC 注册分支事务，将其纳入 XID 对应全局事务的管辖。
4. TM 向 TC 发起针对 XID 的全局提交（Commit）或回滚（Rollback）决议。
5. TC 调度 XID 下管辖的全部分支事务完成提交（Commit）或回滚（Rollback）请求。

事务框架

基于架构上定义的 3 个核心组件，分布式事务被抽象成如下事务框架。



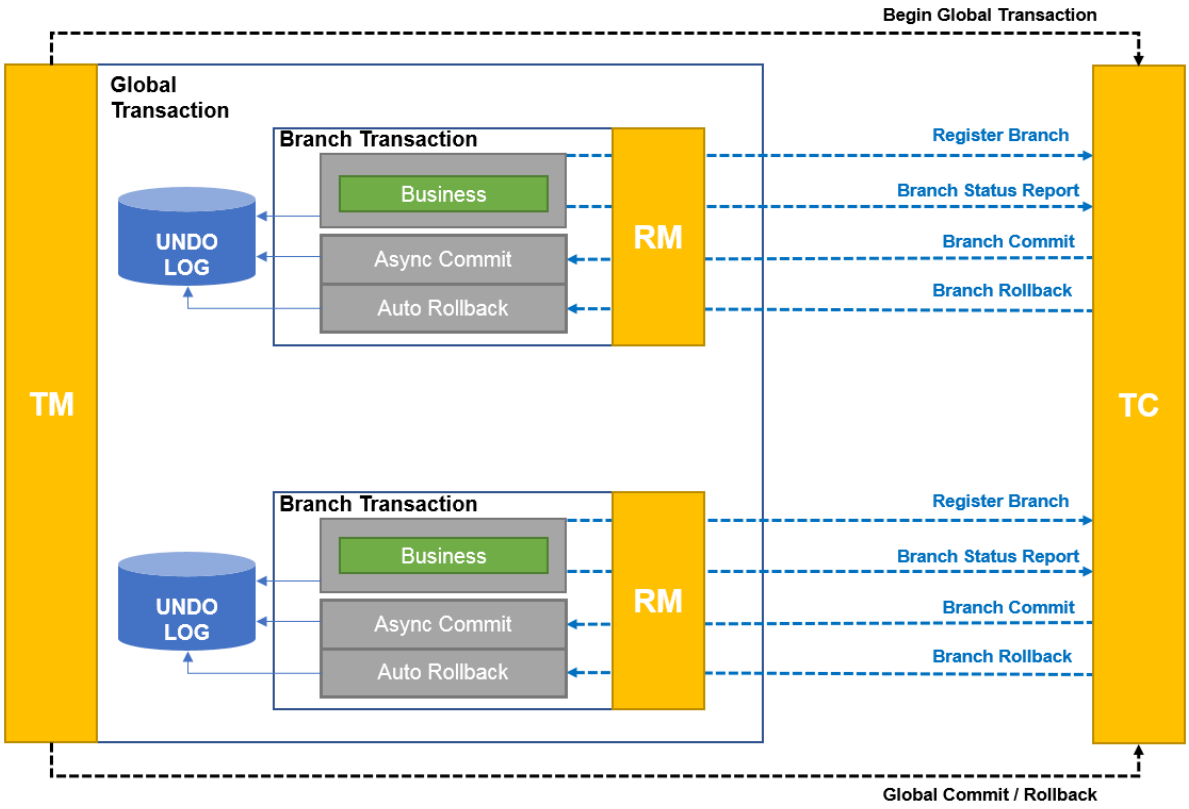
3个核心组件的功能如下：

- TM定义全局事务的边界。
- RM负责定义分支事务的边界和行为。
- TC、TM和RM交互，做全局的协调。交互包括开启（Begin）、提交（Commit）、回滚（Rollback）全局事务；分支注册（Register Branch）、状态上报（Branch Status Report）和分支提交（Branch Commit）、分支回滚（Branch Rollback）。

事务模式

事务模式是这个框架下 RM 驱动的分支事务的不同行为模式，即事务（分支）模式。事务模式包括 AT 模式、TCC 模式、Saga 模式和 XA 模式。

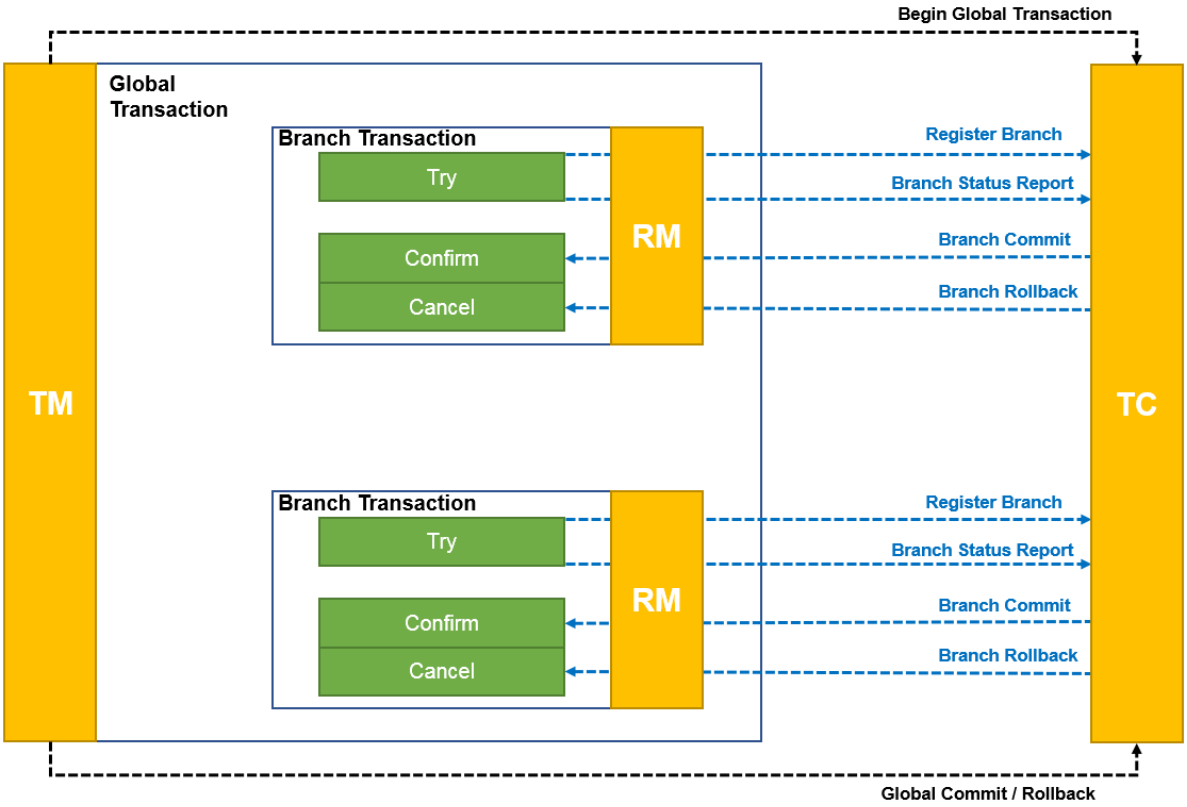
- **AT 模式**



AT 模式 RM 驱动分支事务的行为分为以下两个阶段：

- 执行阶段：
  a. 代理 JDBC 数据源，解析业务 SQL，生成更新前后的镜像数据，形成 UNDO LOG。
  b. 向 TC 注册分支。
  c. 分支注册成功后，把业务数据的更新和 UNDO LOG 放在同一个本地事务中提交。
- 完成阶段：
  - 全局提交，收到 TC 的分支提交请求，异步删除相应分支的 UNDO LOG。
  - 全局回滚，收到 TC 的分支回滚请求，查询分支对应的 UNDO LOG 记录，生成补偿回滚的 SQL 语句，执行分支回滚并返回结果给 TC。

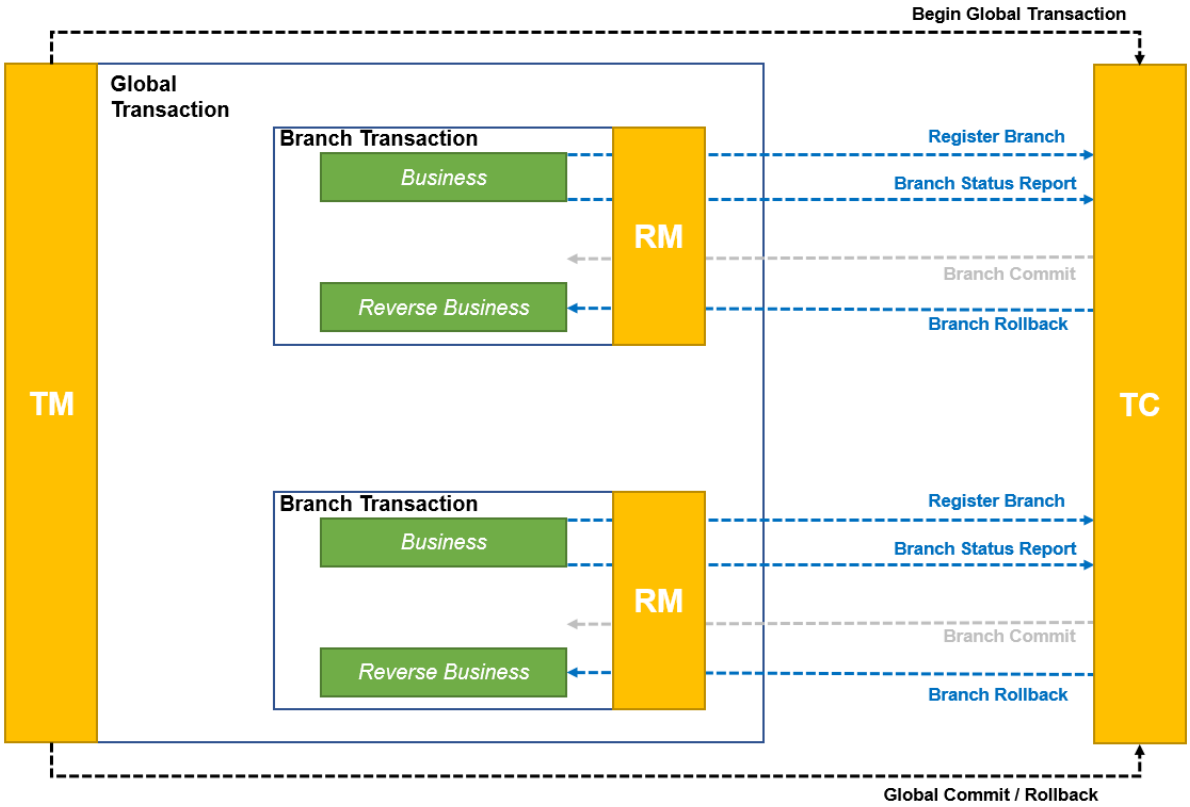
- **TCC 模式**



TCC 模式 RM 驱动分支事务的行为分为以下两个阶段：

- 执行阶段：
  a. 向 TC 注册分支。
  b. 执行业务定义的 Try 方法。
  c. 向 TC 上报 Try 方法执行情况：成功或失败。
- 完成阶段：
  - 全局提交，收到 TC 的分支提交请求，执行业务定义的 Confirm 方法。
  - 全局回滚，收到 TC 的分支回滚请求，执行业务定义的 Cancel 方法。

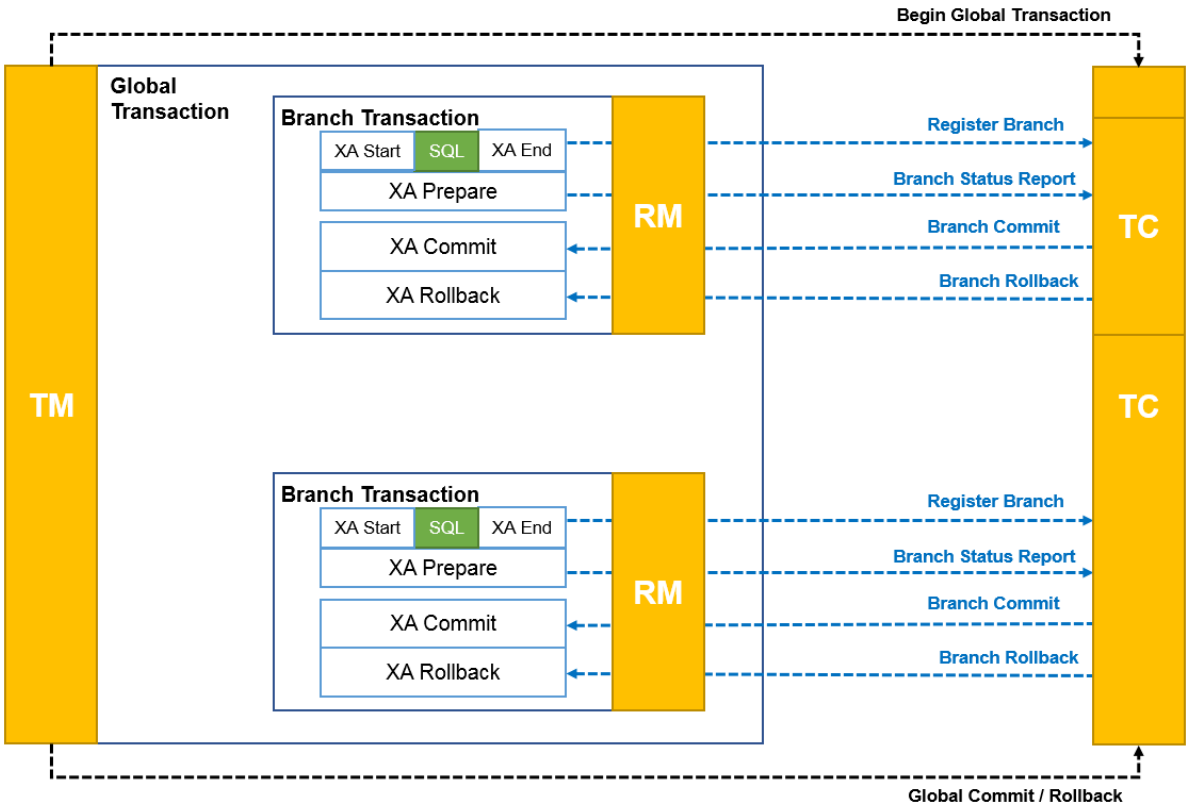
- **Saga 模式**



Saga 模式 RM 驱动分支事务的行为包含以下两个阶段：

- 执行阶段：
  a. 向 TC 注册分支。
  b. 执行业务方法。
  c. 向 TC 上报业务方法执行情况：成功或失败。
- 完成阶段：
  - 全局提交，RM 不需要处理。
  - 全局回滚，收到 TC 的分支回滚请求，执行业务定义的补偿回滚方法。

- **XA 模式**



XA 模式 RM 驱动分支事务的行为包含以下两个阶段：

- 执行阶段：
  a. 向 TC 注册分支。
  b. XA Start，执行业务 SQL，XA End。
  c. XA prepare，并向 TC 上报 XA 分支的执行情况：成功或失败。
- 完成阶段：
  - 收到 TC 的分支提交请求，XA Commit。
  - 收到 TC 的分支回滚请求，XA Rollback。

2.全面兼容和支持 Seata

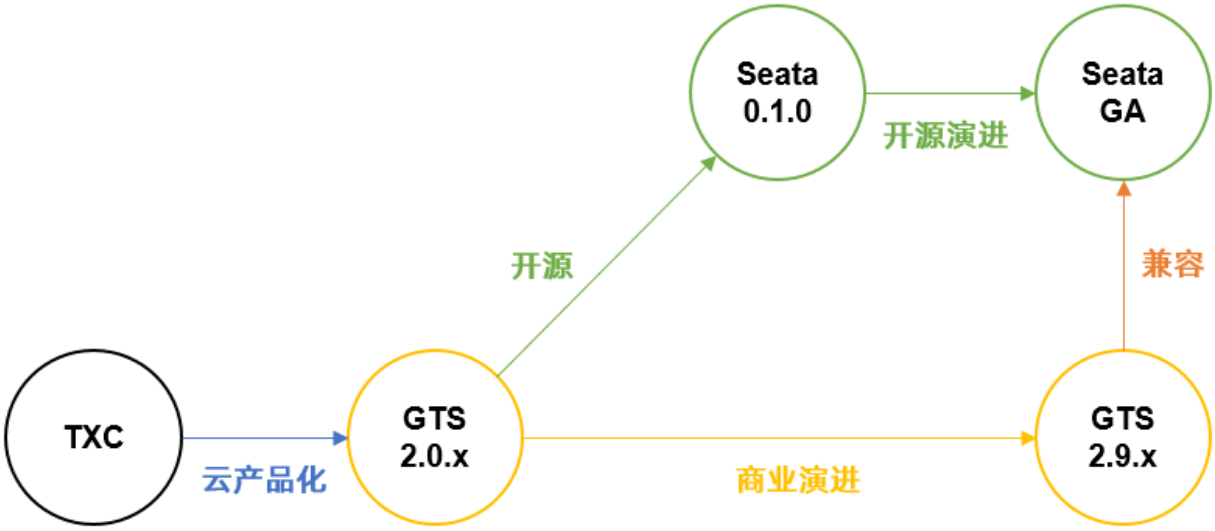
GTS 已经全面兼容和支持开源分布式事务 Seata，实现与 Seata 的协议兼容，支持使用 Seata 的应用无缝迁移到云上，基于 GTS 提供的服务高效运行。

Seata 简介

Simple Extensible Autonomous Transaction Architecture（Seata）是一款开源的分布式事务解决方案，致力于提供高性能和简单易用的分布式事务服务。

2019 年，基于 GTS 的技术积累，阿里巴巴发起了开源项目 Seata。

2020 年 2 月，基于 Seata 项目 GA 后的版本，GTS 实现与 Seata 的协议兼容，支持使用 Seata 的应用无缝迁移到云上，基于 GTS 提供的服务高效运行。



分布式事务框架和事务模式

GTS 和 Seata 定义的分布式事务框架是完全一致的，详情请参见分布式事务框架和事务模式。

TM、RM 和应用部署在一起，应用根据业务需求，选择合适的分布式事务模式来解决数据一致性问题。

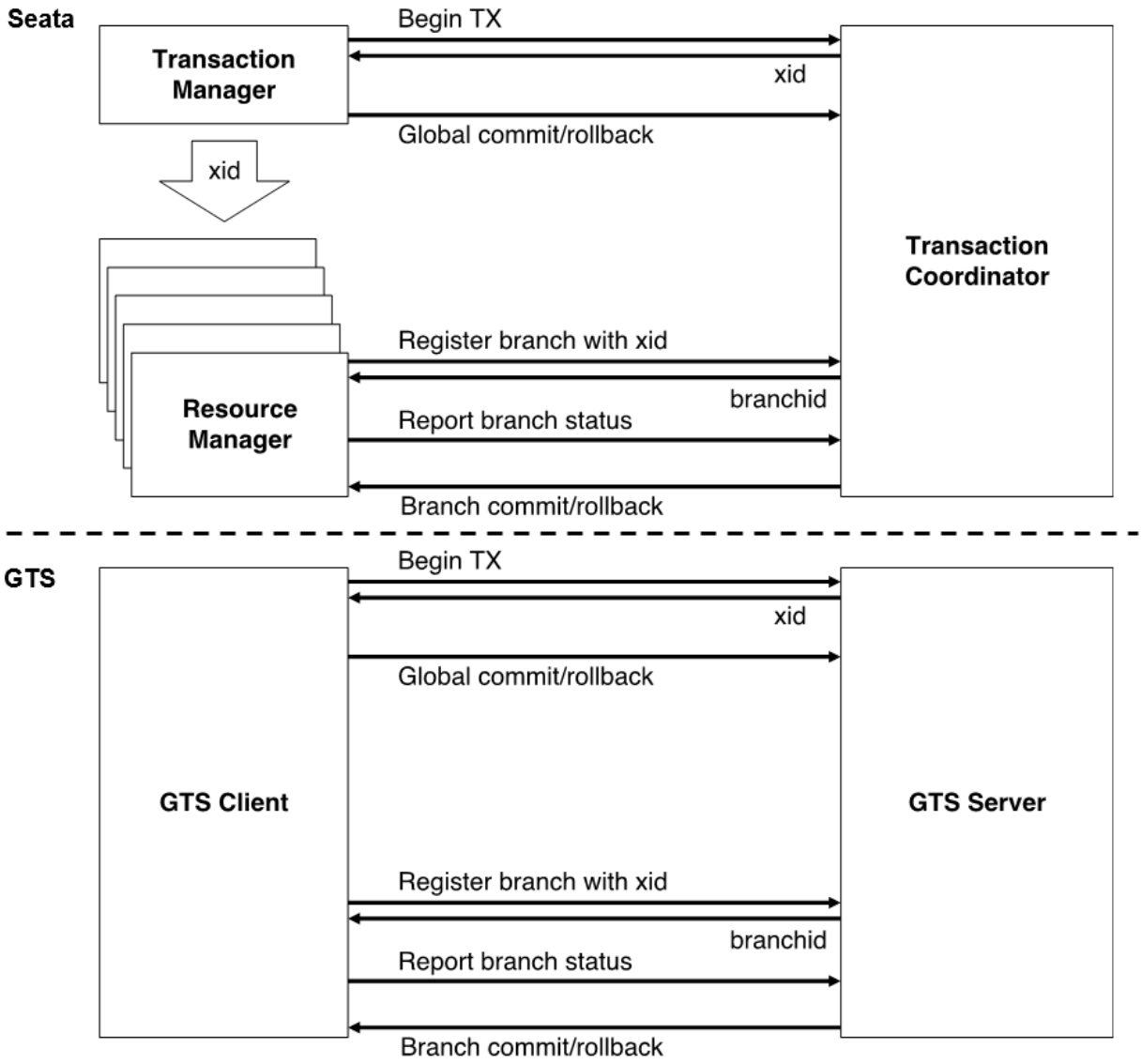
无论使用哪处模式，都离不开一个稳定高效的 TC 提供服务。这些服务包括（但不限于）：

- 记录全局事务状态
- 记录事务分支的注册
- 驱动事务分支进行最终的提交或回滚
- 事务链路监控
- 异常事务的恢复
- 全局事务超时检测
- 全局事务间隔离机制

分布式事务的协调机制被定义为一项标准化的服务，独立部署和运维，给应用的分布式系统提供事务服务。

协议和架构

基于事务框架定义的事务协议如下图所示。



Seata 的分布式事务框架源自 GTS，二者的底层架构和事务协议是完全一致的。

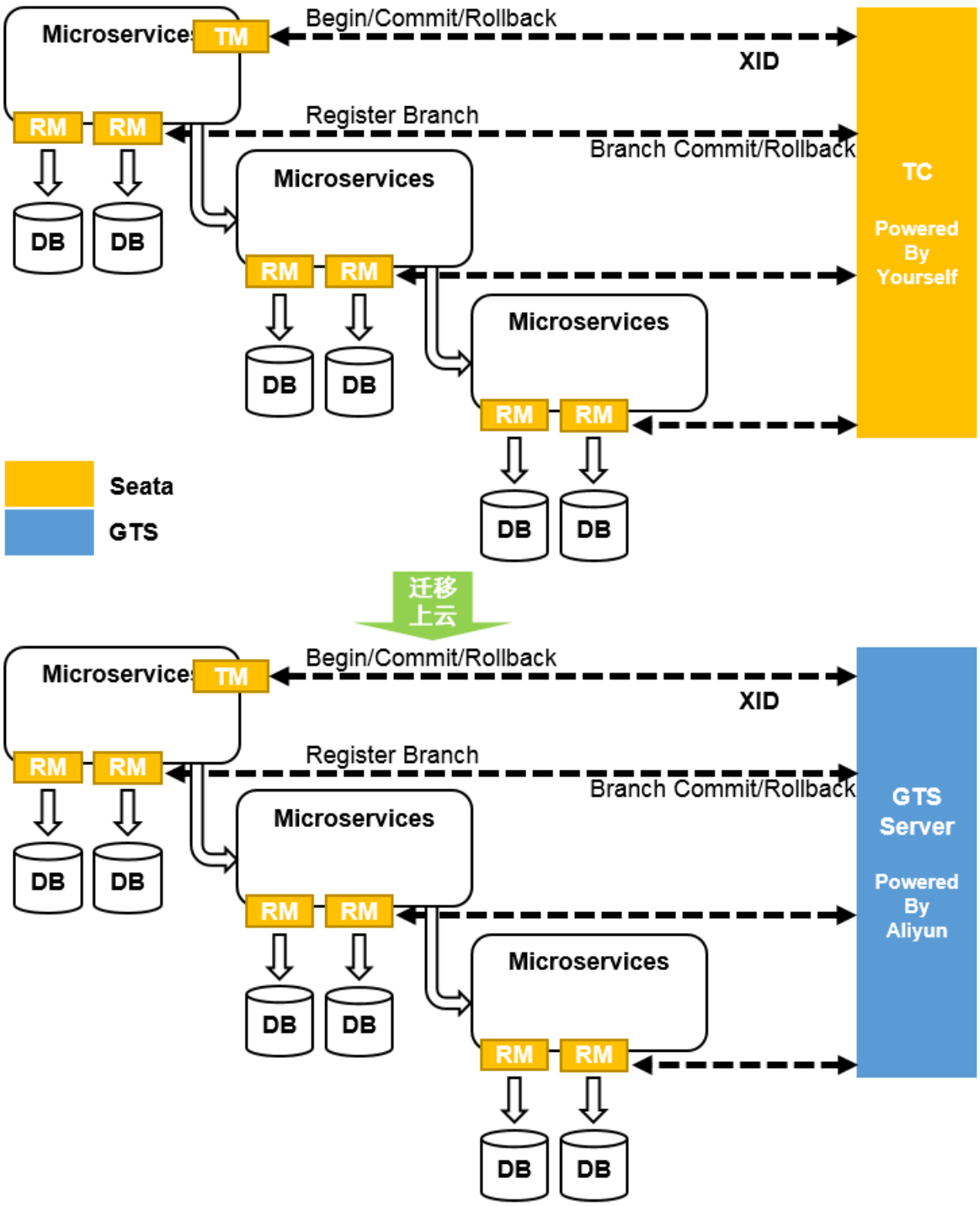
- GTS 把 TM 和 RM 的实现统一打包到 GTS SDK（GTS Client） 中。
- GTS 服务端（GTS Server）就是 TC 的一个高可用实现。

依托于阿里云的基础设施，GTS Server 以多副本、高可用集群形态部署，提供高质量的事务协调服务。

将 Seata 应用迁移到 GTS

架构和协议层面一致，使得 GTS 可以很自然地提供对 Seata 应用的事务协调支持。只要实现具体网络通信机制上的兼容适配，就可以完整支撑 Seata 应用运行在 GTS 服务之上。

Seata 应用迁移到云上使用 GTS 服务不需要做任何编程改造，仅仅是把自运维的 Seata 的 TC Server 替换为 GTS 提供的高性能、高可靠、高可用的云服务。



GTS 的 SDK 2.9.0 版本支持基于 Seata 的应用使用 AT 模式，运行在 GTS 服务上。详情请参见 Seata 应用样例。

3.部署 SDK

注解使用方式需要在 GTS 客户端上部署 SDK，才能使用分布式事务。GTS SDK 目前只支持 Java 版本。

操作步骤

1. 下载 GTS SDK 开发包。

   建议选择 GTS SDK 最新版本，也可以根据实际需求选择其它历史版本，详情请参见版本说明。

2. （可选）如果需要 Spring Cloud 原生支持，需要下载 Spring Cloud 原生支持包。

   > **说明** 若之前从未使用过GTS或者Seata建议使用2.8.x版本；若想兼容支持开源 Seata 的功能，请在版本列表选择2.9.x的版本，若想添加Spring Cloud 的支持需要额外 spring-cloud-alibaba-seata。目前Seata 已支持多种数据库，多种 RPC 框架，详情请参见Seata。

3. 将 SDK SDK 开发包上传到 GTS 客户端所在的机器上。
4. 将 SDK 开发包添加到应用的依赖中。

   具体方式没有限制，这里不一一列举。

   典型的情况是：如果使用 Maven 来管理应用工程，可以将 SDK 开发包添加到 *pom.xml* 依赖中。

   将 SDK 开发包添加到 *pom.xml* 依赖中的示例如下：

   ```
   <dependency>
     <groupId>com.taobao.txc</groupId>
     <artifactId>txc-client</artifactId>
     <version>${txc-version}</version>
     <scope>system</scope>
     <systemPath>SDK的存放路径</systemPath>
   </dependency>
   ```

   使用 Spring Cloud 原生支持时，如果应用中自己定义了 WebMvcConfigurationSupport，请添加 TxcInboundHander 的实例，示例代码如下：

   ```
   public class MyWebMvcConfigurationSupport extends WebMvcConfigurationSupport {
     @Override
     protected void addInterceptors(InterceptorRegistry registry) {
       registry.addInterceptor(new TxcInboundHandler()).addPathPatterns(new String[] { "/**" });
     }
   }
   ```

4.注意事项

当您在使用 GTS 时，需要关注相关的注意事项。

GTS 事务分组应与业务系统在同一个地域

GTS 不支持跨地域访问。创建事务分组时，需要选择和业务系统相同的地域，否则使用时会显示 `连不上 GTS Server` 。

GTS 注解方法的调用

建议在当前方法中直接实例化 `@TxcTransaction` 注解的方法类，并直接调用，保证 GTS 事务生效，且代码直观。

GTS 注解方法的调用包含以下三点注意事项：

- 带有 `@TxcTransaction` 注解的方法可以在其他的类中被调用，但需要显式的指定的 Spring bean 实例。
- 如果 `@TxcTransaction` 注解方法在本类的其他方法中被调用，需要显式指定实例。
- 开启事务只能直接调用使用 Spring bean 实例化的类的 `@TxcTransaction` 注解方法。

示例如下。

```
class SampleClient { /* 此类已经被声明为Spring bean */
  //此方法被声明为GTS事务
  @TxcTransaction(timeout = 60000)
  void dataUpdate(Connection con1, Connection con2) {
    // 操作数据源con
    update1(con1);
    update2(con2);
  }
  public void callUpdate1(Connection con1, Connection con2){
    dataUpdate(con1,con2);
  }
  public void callUpdate2(Connection con1, Connection con2, SampleClient clienttest){
    clienttest.dataUpdate(con1,con2);
  }
  public static void main(String[] args){
    SampleClient clienttest1 = (SampleClient)context.getBean("clientTest");
    Connection con1=getCon1();
    Connection con2=getCon2();
    clienttest1.dataUpdate(con1,con2);// 这个是正确的调用，最佳实践
    SampleClient clienttest2 = new SampleClient();
    clienttest2.dataUpdate(con1,con2);// 这个是错误的调用，不能开启GTS事务
    clienttest1.callUpdate1(con1,con2); // 这个是错误的调用，不能开启GTS事务
    clienttest1.callUpdate2(con1,con2, clienttest1);// 这个是正确的调用，但是不推荐使用
  }
}
class UseSampleClient {
  void updateSampleClient(){
    SampleClient clienttest1 = (SampleClient)context.getBean("clientTest");
    clienttest1.dataUpdate(con1,con2);// 这个是正确的调用，最佳实践
    SampleClient clienttest2 = new SampleClient();
    clienttest2.dataUpdate(con1,con2);// 这个是错误的调用，不能开启GTS事务
  }
}
```

AT 模式注意事项

AT 模式注意事项请参见AT 模式接入注意事项。

TCC 模式注意事项

TCC 模式注意事项请参见TCC 模式接入注意事项。

SAE 环境注意事项

非公网测试环境（线上正式环境）需要配置 URL 参数 https://cs2.gts.aliyuncs.com。

- 使用 txc-client.jar

```
<bean class="com.taobao.txc.client.aop.TxcTransactionScaner">
    <constructor-arg value="myapp"/><!-- 应用自定义的标识 -->
    <constructor-arg value="mygroup.xxxx.xx"/><!-- 事务分组(全名) -->
    <constructor-arg type="int" value="1"/>
    <constructor-arg value="https://cs2.gts.aliyuncs.com"/>
    <property name="accessKey" value="xxx"/>
    <property name="secretKey" value="xxx"/>
</bean>
```

- 使用 txc-client-springcloud.jar

  在 properties 配置文件添加 spring.cloud.txc.url=https://cs2.gts.aliyuncs.com。

5.TCC 模式

5.1. TCC 模式接入

如果您使用了 TCC 事务模式，本文将帮助您了解 TCC 事务模式如何接入 GTS。

前提条件

在将 TCC 事务模式接入 GTS 前，准备两个 RDS 实例和一个 ECS 实例。

背景信息

TCC 事务模式的详细介绍，请参见分布式事务框架和事务模式。

TCC 事务模式接入流程

1. 开通 GTS（创建事务分组）。
2. 在 RDS 中建表。
3. 将样例工程上传到 ECS 上，并修改数据源、GTS 分组、以及 Access Key ID 和 Access Key Secret。
4. 编译样例工程，查看运行结果。
5. 查看事务分组运行状态。

TCC 事务模式样例

- sample-txc-mt-reserve-simple 样例
- sample-txc-mt-compensate-simple 样例

5.2. TCC 模式接入注意事项

本文介绍使用 TCC 模式接入 GTS 时的注意事项。

开启 TCC 模式

单独使用 TCC 模式时，需要开启 TCC 模式。

```
<bean class="com.taobao.txc.client.aop.TxcTransactionScaner">
   <constructor-arg value="gtstest.123213123.HZ"/>
   <!-- 1:AT 2:MT 3:AT&MT -->
   <constructor-arg value="2"/>
</bean>
```

GTS 对 MQ 事务消息纳入全局事务管理的支持是通过 TCC 模式实现的。 所以，使用 AT 模式的同时，又需要把 MQ 事务消息纳入全局事务管理时，就需要同时开启 AT 和 TCC 模式的配置 。

```
<bean class="com.taobao.txc.client.aop.TxcTransactionScaner">
   <constructor-arg value="gtstest.123213123.HZ"/>
   <!-- 1:AT 2:MT 3:AT&MT -->
   <constructor-arg value="3"/>
</bean>
```

在 GTS 中使用 MQ 时，MTRelationShipManager 一定要先于 TxcTransactionScaner 进行声明

> **说明** 非 XML 配置方式的应用，可以忽略该注意事项。

```
<!-- 先实例化MTRelationShipManager -->
<bean class="com.taobao.txc.client.aop.MTRelationShipManager">
   <property name="beanNames" ref="mtServicesClassList" />
   <property name="interceptorNames">
     <list>
       <value>mtBranchInterceptor</value>
     </list>
   </property>
   <property name="order" value="1"></property>
   <property name="proxyTargetClass" value="false">
   </property>
</bean>
<!-- 再实例化TxcTransactionScaner -->
<bean class="com.taobao.txc.client.aop.TxcTransactionScaner">
   <constructor-arg value="gtstest.123213123.HZ"/>
   <!-- 1:AT 2:MT 3:AT&MT -->
   <constructor-arg value="3"/>
</bean>
```