Seata 1.3 特性与升级分享

作者简介:陈健斌(funkye) ,Seata Committer ,同盾科技高级开发工程师

github:https://github.com/a364176773

gitee:https://gitee.com/itCjb

1.0 初级入门地址:https://mp.weixin.qq.com/s/BOntA2mcdF-e0t2e4CBjPQ

1.2 教程入门地址: https://mp.weixin.qq.com/s/2KSidJ72YsovpJ94P1aK1g (钉钉各大官方群回放)

视频教程: https://www.bilibili.com/video/BV12Q4y1A7Nt

本期视频直播后将会在Seata各大群及B站上传.可关注

https://space.bilibili.com/9984904

以及Seata各大官方群:

Seata钉钉 3群: 32033786 Seata官方qq群: 254657148

介绍

本篇将介绍,如何升级1.2至1.3版本,**redis会话存储使用**,**服务自动升降级介绍与演示**,**xa模式**介绍与上手

亮点:如何给Seata源码提交PR?如何把自己写的博客提交至Seata官网?

手把手在线演示更改源码及提交PR.

以及常见难题解析:

1.事务分组配置如何正确理解

2.服务熔断后,事务回滚的几种方式介绍与示例

3.如何使用保证捕获异常后还可以回滚事务

4.分布式事务带来的性能降低,优化方式.

社区任务认领列表

正文

第一步:

首先访问: https://seata.io/zh-cn/blog/download.html

下载我们需要使用的seata1.3服务

第二步:

1.在你的参与全局事务的数据库中加入undo_log这张表

```
-- for AT mode you must to init this sql for you business database. the seata
server not need it.
CREATE TABLE IF NOT EXISTS `undo_log`
(
    `branch_id`     BIGINT(20)   NOT NULL COMMENT 'branch transaction id',
```

```
    `xid`           VARCHAR(100) NOT NULL COMMENT 'global transaction id',
    `context`       VARCHAR(128) NOT NULL COMMENT 'undo_log context,such as
serialization',
    `rollback_info` LONGBLOB     NOT NULL COMMENT 'rollback info',
    `log_status`    INT(11)      NOT NULL COMMENT '0:normal status,1:defense
status',
    `log_created`   DATETIME(6)  NOT NULL COMMENT 'create datetime',
    `log_modified`  DATETIME(6)  NOT NULL COMMENT 'modify datetime',
    UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB
  AUTO_INCREMENT = 1
  DEFAULT CHARSET = utf8 COMMENT ='AT transaction mode undo table';
```

2.在你的mysql数据库中创建名为seata的库,并使用以下下sql

```
-- -------------------------------- The script used when storeMode is 'db' -----
--------------------------
-- the table to store GlobalSession data
CREATE TABLE IF NOT EXISTS `global_table`
(
    `xid`                       VARCHAR(128) NOT NULL,
    `transaction_id`            BIGINT,
    `status`                    TINYINT      NOT NULL,
    `application_id`            VARCHAR(32),
    `transaction_service_group` VARCHAR(32),
    `transaction_name`          VARCHAR(128),
    `timeout`                   INT,
    `begin_time`                BIGINT,
    `application_data`          VARCHAR(2000),
    `gmt_create`                DATETIME,
    `gmt_modified`              DATETIME,
    PRIMARY KEY (`xid`),
    KEY `idx_gmt_modified_status` (`gmt_modified`, `status`),
    KEY `idx_transaction_id` (`transaction_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

-- the table to store BranchSession data
CREATE TABLE IF NOT EXISTS `branch_table`
(
    `branch_id`         BIGINT       NOT NULL,
    `xid`               VARCHAR(128) NOT NULL,
    `transaction_id`    BIGINT,
    `resource_group_id` VARCHAR(32),
    `resource_id`       VARCHAR(256),
    `branch_type`       VARCHAR(8),
    `status`            TINYINT,
    `client_id`         VARCHAR(64),
    `application_data`  VARCHAR(2000),
    `gmt_create`        DATETIME(6),
    `gmt_modified`      DATETIME(6),
    PRIMARY KEY (`branch_id`),
    KEY `idx_xid` (`xid`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

-- the table to store lock data
```

```
CREATE TABLE IF NOT EXISTS `lock_table`
(
    `row_key`        VARCHAR(128) NOT NULL,
    `xid`            VARCHAR(96),
    `transaction_id` BIGINT,
    `branch_id`      BIGINT       NOT NULL,
    `resource_id`    VARCHAR(256),
    `table_name`     VARCHAR(32),
    `pk`             VARCHAR(36),
    `gmt_create`     DATETIME,
    `gmt_modified`   DATETIME,
    PRIMARY KEY (`row_key`),
    KEY `idx_branch_id` (`branch_id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
```

第三步:

在你的项目中引入seata依赖

如果你的微服务是dubbo:

```
            <dependency>
                <groupId>io.seata</groupId>
                <artifactId>seata-spring-boot-starter</artifactId>
                <version>1.3.0</version>
            </dependency>
```

如果你是springcloud:

```
            <dependency>
                <groupId>com.alibaba.cloud</groupId>
                <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
                <version>2.2.2.RELEASE</version>
                <exclusions>
                    <exclusion>
                        <groupId>io.seata</groupId>
                        <artifactId>seata-spring-boot-starter</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
            <dependency>
                <groupId>io.seata</groupId>
                <artifactId>seata-spring-boot-starter</artifactId>
                <version>1.3.0</version>
            </dependency>
```

第四步:

从官方github仓库拿到参考配置做修改:https://github.com/seata/seata/tree/develop/script/client

加到你项目的application.yml中.

```
seata:
  enabled: true
  application-id: applicationName
  tx-service-group: my_test_tx_group
```

```yaml
  enable-auto-data-source-proxy: true
  config:
    type: nacos
    nacos:
      namespace:
      serverAddr: 127.0.0.1:8848
      group: SEATA_GROUP
      username: "nacos"
      password: "nacos"
  registry:
    type: nacos
    nacos:
      application: seata-server
      server-addr: 127.0.0.1:8848
      group: SEATA_GROUP
      namespace:
      username: "nacos"
      password: "nacos"
```

第五步:

运行你下载的nacos,并参考https://github.com/seata/seata/tree/develop/script/config-center 的config.txt并修改

```
service.vgroupMapping.my_test_tx_group=default
store.mode=db
store.db.datasource=druid
store.db.dbType=mysql
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://127.0.0.1:3306/seata?useUnicode=true
store.db.user=username
store.db.password=password
store.db.minConn=5
store.db.maxConn=30
store.db.globalTable=global_table
store.db.branchTable=branch_table
store.db.queryLimit=100
store.db.lockTable=lock_table
store.db.maxWait=5000
```

运行仓库中提供的nacos脚本,将以上信息提交到nacos控制台,如果有需要更改,可直接通过控制台更改

第六步:

更改server中的registry.conf

```
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
  type = "nacos"
  nacos {
    application = "seata-server"
    serverAddr = "127.0.0.1:8848"
    group = "SEATA_GROUP"
    namespace = ""
    cluster = "default"
    username = "nacos"
    password = "nacos"
```

```
  }
}

config {
  # file、nacos 、apollo、zk、consul、etcd3
  type = "nacos"

  nacos {
    serverAddr = "127.0.0.1:8848"
    namespace = ""
    group = "SEATA_GROUP"
    username = "nacos"
    password = "nacos"
  }
}
```

第七步:

运行seata-server,成功后,运行自己的服务提供者,服务参与者.

在全局事务调用者(发起全局事务的服务)的接口上加入@GlobalTransactional

自此,整合seata1.2及nacos的配置与注册中心全部整合完毕.

如果你需要高可用搭建请看第八步

第八步高可用Seata-server搭建

确保你已经完成了以上七步操作后,按照以上的第1,6两步即可把你的seata新节点接入到同一个nacos集群,配置&注册中心中,由于是同一个配置中心,所以db也是采用的共同配置.至此高可用搭建已经顺利完结,如果你想测试,仅需关掉其中一个server节点,验证服务是否可用即可.

Redis作为事务会话存储的教程

运行你下载的nacos,并参考https://github.com/seata/seata/tree/develop/script/config-center 的config.txt并修改

```
store.mode=redis
store.redis.host=127.0.0.1
store.redis.port=6379
store.redis.maxConn=10
store.redis.minConn=1
store.redis.database=0
store.redis.password=null
store.redis.queryLimit=100
```

运行仓库中提供的nacos脚本,将以上信息提交到nacos控制台,如果有需要更改,可直接通过控制台更改

就这么简单,无需像db模式需要建立3张表,直接可用.

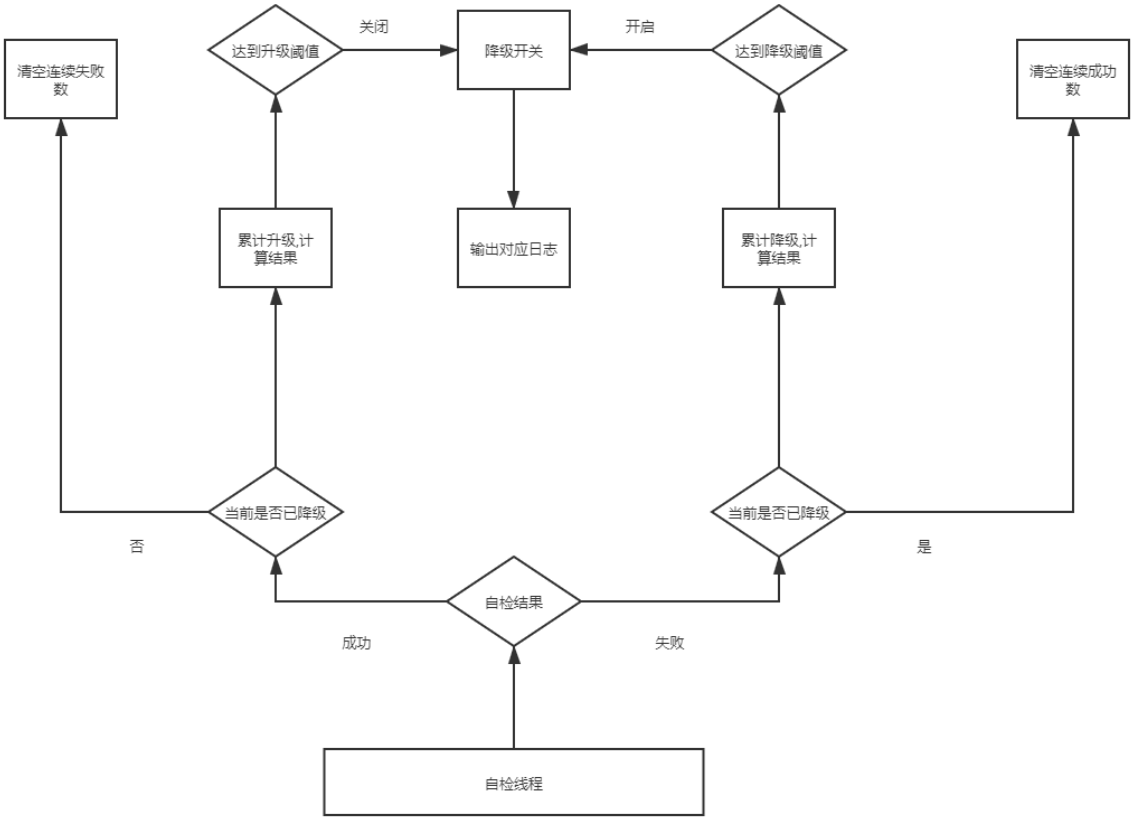
注意事项:

目前多个分支事务unlock的时候会有异常,1.3版本还处于新上阶段,后续还需要改进,目前建议等到1.4发版后再做使用.

也可加入Seata各大官方群,进行下载1.4的bugfix快照版

服务自检,自动升降级

流程介绍:



以上是整个升降级原理流程.

原理解析:

通过服务自检线程+TM的异常,来判断是否当前是由于seata自身出现问题,导致事务异常,如果是的话,连续错误数达到了用户设置的阈值那么当前会直接关闭全局事务,让用户接口可以正常的响应,而不被Seata的异常所影响.

恢复全局事务就是反之了,当前如果是关闭时,自检线程会模拟事务的begin跟commit,如果是正常且连续正常达到阈值,那么就会恢复全局事务的管理,保证事务一致性.

其实这个功能的作用就是尽最大可能保证Seata出现问题的时候,去不影响用户,且服务恢复后,第一时间再继续为用户保驾护航.

总结:

用的放心,用的舒心.

XA模式介绍与上手

注:**XA相关内容大部分来自于Seata发起人之一: 煊穗,GitHub ID：sharajava 特此感谢**

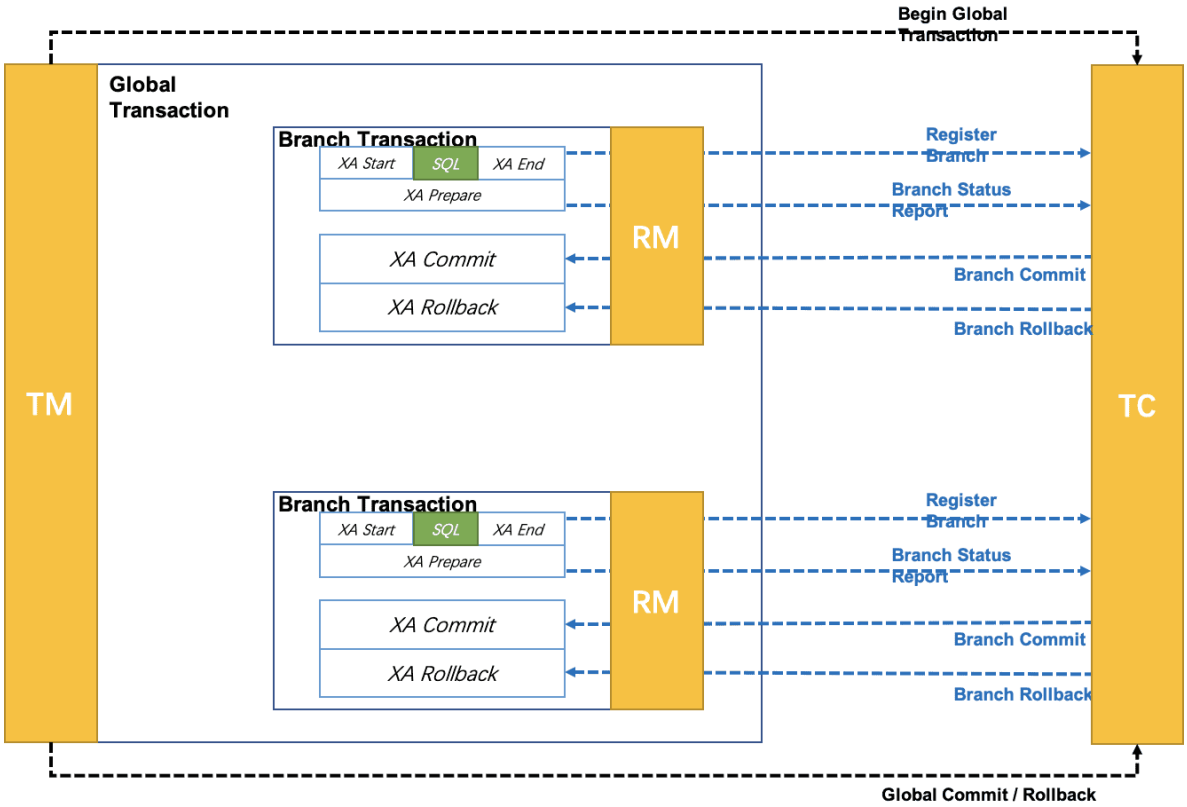
前提:

如何确认我是否可以使用XA模式?

- 支持XA 事务的数据库。
- Java 应用，通过 JDBC 访问数据库。

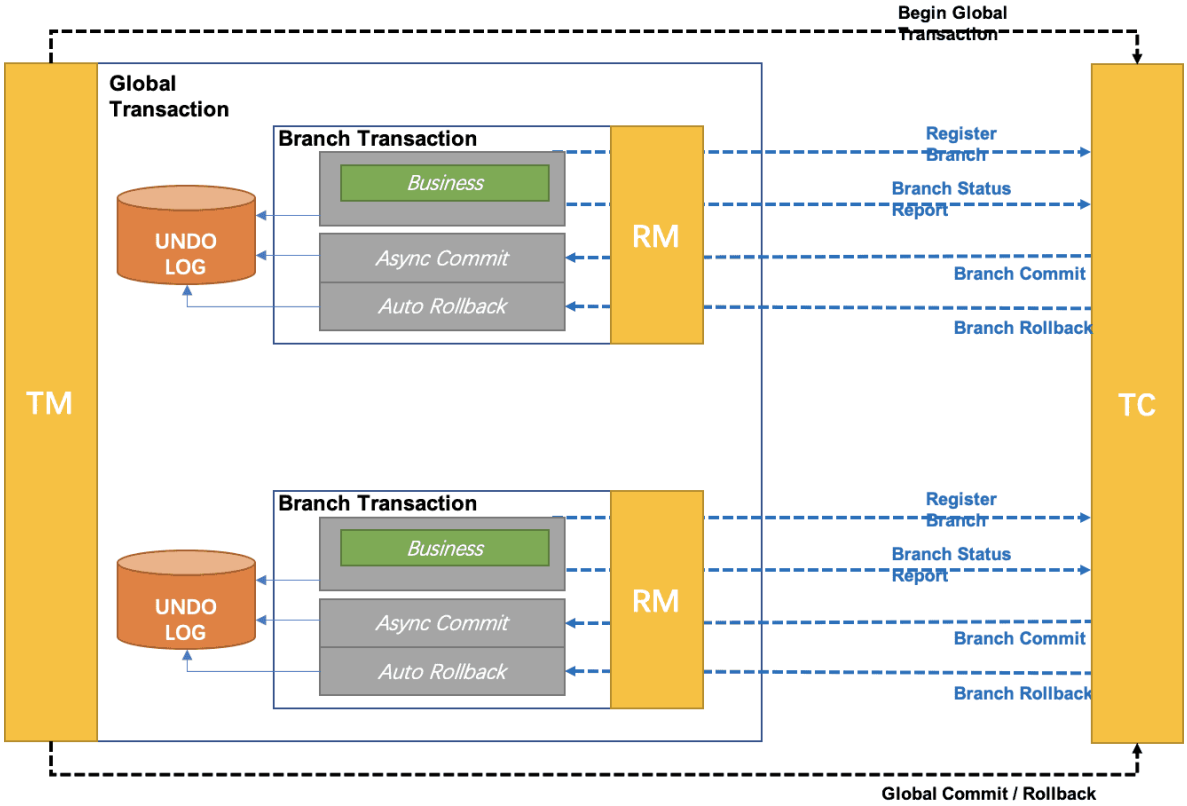
整体机制

在 Seata 定义的分布式事务框架内，利用事务资源（数据库、消息服务等）对 XA 协议的支持，以 XA 协议的机制来管理分支事务的一种 事务模式。



- 执行阶段：
- - 可回滚：业务 SQL 操作放在 XA 分支中进行，由资源对 XA 协议的支持来保证 *可回滚*
  - 持久化：XA 分支完成后，执行 XA prepare，同样，由资源对 XA 协议的支持来保证 *持久化*（即，之后任何意外都不会造成无法回滚的情况）
- 完成阶段：
- - 分支提交：执行 XA 分支的 commit
  - 分支回滚：执行 XA 分支的 rollback

接下来我们AT 模式



- 执行阶段：
- - 可回滚：根据 SQL 解析结果，记录回滚日志
  - 持久化：回滚日志和业务 SQL 在同一个本地事务中提交到数据库
- 完成阶段：
- - 分支提交：异步删除回滚日志记录
  - 分支回滚：依据回滚日志进行反向补偿更新

对比:

通过以上的区别,其实我们可以发现,其实XA流程相对AT模式,是简单许多.

AT: 解析sql生成undolog+竞争全局锁(rpc)

XA: 支持xa的数据库即可

无侵入式 XA>AT　性能,事务链路不出现异常情况下,由于AT一阶段预先提交,以及锁由TC一侧统一管理,理论上AT的性能要高于XA.

但如果涉及到rollback时,由于AT还需要去读取undolog,镜像校验,回滚数据等步骤.个人认为在rollback时.XA是要高于AT

注:不支持AT+XA的一个事务.无法保证隔离性

如果你用的是Redis+XA这样的配置,那么你等于主需要在发起事务的方法或接口上加入全局事务注解.

常见问题解析

1.事务分组配置如何正确理解

首先我们要明确的是事务分组配置存在于client端

示例

```
seata:
  tx-service-group: test
```

那么以上信息仅代表我的事务分组为test

而client端与server的服务发现基于通过事务分组所对应的server集群名,来做服务发现.

所以我们必须把我们的事务分组对应的那个server集群给提交到nacos,以便于我们可以及时发现,和及时切换集群

```
service.vgroupMapping.test=default
```

上面的示例解读:我的事务分组名为test所对应的server集群名叫default

所以我们的client端会去通过这个default集群去找到对应可用的server服务

2.服务熔断或者参与方做了全局异常捕获后,事务回滚的几种方式介绍与示例

方式一:

通过result code 来做手动api回滚

方式二:

通过result code 抛出异常触发回滚

方式三:

服务熔断实现内直接抛出异常

方式四:

服务熔断内api触发回滚

欢迎更多方式补充.....

3.如何使用保证捕获异常后还可以回滚事务

方式一:

使用api方式回滚事务

方式二:

全局异常捕获器位于全局事务的外层

4.分布式事务带来的性能降低,优化方式.

异步化当前全局事务整体调用链,缓存对应的事务状态

前端通过轮询来得到结果

请稍后...

10s 倒计时

已完成

欢迎更多补充.....

如何参与到Seata的源码贡献呢?

1.首先fork Seata的仓库

2.拉取官方的develop作为基础

3.创建自己的分支,进行代码编写并提交到这个分支

5.提交这个分支到官方仓库的develop分支

6.等待review,及时响应,及时根据建议修改自己的代码逻辑

7.恭喜你成为Seata的Contributor

更加详细的参与到Seata贡献的指导请访问: https://github.com/seata/seata/blob/develop/CONTRIBUTING.md

待认领的任务列表

1.Redis的事务会话模式结构优化

2.Redis作为undolog 二级缓存

3.更多数据库的支持

4.AT模式oracle的多语句支持

5.支持Raft的高可用Seata-Servier集群

6.XA模式mysql 8.0.11以上驱动支持

7.消息队列分布式事务

8.Seata-Server 改造为Spring Boot

9.Redis String结构换为Hash结构

还有更多可以从issues,官方交流群等区域进行沟通交流时发现.