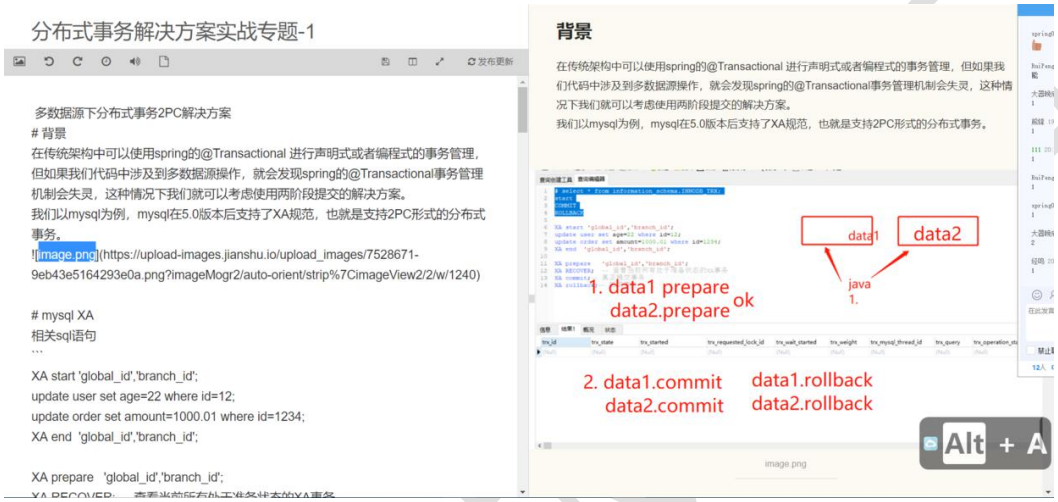
多数据源下分布式事务 2PC 解决方案

原文地址 https://www.jianshu.com/p/d5b5c84c98a2

背景

在传统架构中可以使用 spring 的@Transactional 进行声明式或者编程式的事务管理，但如果我们代码中涉及到多数据源操作，就会发现 spring 的@Transactional 事务管理机制会失灵，这种情况下我们就可以考虑使用两阶段提交的解决方案。我们以 mysql 为例，mysql 在 5.0 版本后支持了 XA 规范，也就是支持 2PC 形式的分布式事务。



mysql XA

相关 sql 语句

```
XA start 'global_id','branch_id';

update user set age=22 where id=12;

update order set amount=1000.01 where id=1234;

XA end  'global_id','branch_id';



XA prepare   'global_id','branch_id';

XA RECOVER;  -- 查看当前所有处于准备状态的 XA 事务

XA commit;-- 真正提交事务

XA rollback;-- 回滚事务
```

Java 代码

使用 druid 管理连接池,其支持 XA import com.alibaba.druid.pool.xa.DruidXADataSource; import com.mysql.jdbc.jdbc2.optional.MysqlXADataSource;

```
import com.alibaba.druid.pool.xa.DruidXADataSource;

import com.mysql.jdbc.jdbc2.optional.MysqlXid;



import javax.sql.XAConnection;

import javax.transaction.xa.XAResource;

import javax.transaction.xa.Xid;

import java.sql.Connection;

import java.sql.Statement;

import java.util.Properties;

/**

 * @author Jam Fang  https://www.jianshu.com/u/0977ede560d4

 * @version 创建时间：2019/4/14 13:58

 */

public class TwoPhaseCommitApplication {

    public void multiDataSourceTest() throws Exception {

        String propertyfile = "/app.properties";

        Properties props = new Properties();

        props.load(getClass().getResourceAsStream(propertyfile));

        //初始化数据源

        DruidXADataSource xaDataSource_1 = initXADataSource(props, "db1.");

        //初始化 XA 连接

        XAConnection xaConnection_1 = xaDataSource_1.getXAConnection();

        //初始化 XA 资源

        XAResource xaResource_1 = xaConnection_1.getXAResource();

        //获得数据库连接

        Connection connection_1 = xaConnection_1.getConnection();
```

```
        connection_1.setAutoCommit(false);

        //创建 XID

        Xid xid_1 = new MysqlXid("globalid".getBytes(), "branch-1".getBytes(), 0);

        //关联事务 start end

        xaResource_1.start(xid_1, XAResource.TMNOFLAGS);

        Statement stmt = connection_1.createStatement();

        String sql_1 = "INSERT INTO `order`(orderid,amount,product) values('00001','3000.00','
苹果笔记本');";//"delete from test3 where pk_t=3;";

        stmt.executeUpdate(sql_1);

        xaResource_1.end(xid_1, XAResource.TMSUCCESS);

        //事务准备

        int result_1 = xaResource_1.prepare(xid_1);



        DruidXADataSource xaDataSource_2 = initXADataSource(props, "db2.");

        XAConnection xaConnection_2 = xaDataSource_2.getXAConnection();

        XAResource xaResource_2 = xaConnection_2.getXAResource();

        Connection connection_2 = xaConnection_2.getConnection();

        connection_2.setAutoCommit(false);

        Xid xid_2 = new MysqlXid("globalid".getBytes(), "branch-2".getBytes(), 0);

        xaResource_2.start(xid_2, XAResource.TMNOFLAGS);

        Statement stmt2 = connection_2.createStatement();

        String sql_2 = "update shipping set address='北京黄浦江畔' where id=1;";

        stmt2.executeUpdate(sql_2);

        xaResource_2.end(xid_2, XAResource.TMSUCCESS);

        int result_2 = xaResource_2.prepare(xid_2);

        //XA 事务 准备阶段

        if (result_1 == XAResource.XA_OK &&

                result_2 == XAResource.XA_OK) {
```

```
            //都返回 OK 的话，进行提交阶段
            xaResource_1.commit(xid_1, false);
            xaResource_2.commit(xid_2, false);
        } else {
            //回滚事务
            xaResource_1.rollback(xid_1);
            xaResource_2.rollback(xid_2);
        }
    }

    DruidXADataSource initXADataSource(Properties props, String prefix) {
        DruidXADataSource xaDataSource = new DruidXADataSource();
        xaDataSource.setDbType(props.getProperty(prefix + "dbtype"));
        xaDataSource.setUrl(props.getProperty(prefix + "url"));
        xaDataSource.setUsername(props.getProperty(prefix + "username"));
        xaDataSource.setPassword(props.getProperty(prefix + "password"));
        return xaDataSource;
    }

    public static void main(String args[]) {
        try {
            new TwoPhaseCommitApplication().multiDataSourceTest();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

app.properties 文件

```
db1.dbtype=mysql

db1.url=jdbc:mysql://127.0.0.1:3306/archdemo1?useUnicode=true&characterEncoding=utf8&useSSL=fa
lse&serverTimezone=Asia/Shanghai

db1.username=root

db1.password=123456

db2.dbtype=mysql



db2.url=jdbc:mysql://127.0.0.1:3306/archdemo2?useUnicode=true&characterEncoding=utf8&useSSL=fa
lse&serverTimezone=Asia/Shanghai

db2.username=root

db2.password=123456
```

分析

在这种方案下,我们的代码充当了 TM 也就是事务资源协调者,而两个不同的数据源 mysql 充当了 RM 资源管理着角色,在我们代码中对每个事务的准备情况进行判断,如果都 OK 则提交事务,如果有没有准备好的则 rollback 事务.

修改为一个不能正常执行的 sql,来查看他的执行过程

通过断点分析,我们发现程序在执行到这句的时候就会出现异常,也就是在 prepare 之前 sql 语句已经在执行了,只不过我们设置了事务不自动提交,所以在数据库中看不到 sql_1 的执行结果.

修改正常 sql,在 prepare 阶段加断点

```
stmt2.executeUpdate(sql_2);   stmt2: "com.mysql.jdbc.StatementImpl@7fe8ea47
xaResource_2.end(xid_2, XAResource.TMSUCCESS);
int result_2 = xaResource_2.prepare(xid_2);   result_2: 0   xaResource_2: My
//XA事务 准备阶段
if (result_1 == XAResource.XA_OK &&   result_1: 0
        result_2 == XAResource.XA_OK) {
    //都返回OK的话，进行提交阶段
    xaResource_1.commit(xid_1, b: false);
    xaResource_2.commit(xid_2, b: false);
```

TwoPhaseCommitApplication › multiDataSourceTest()

我们在数据库中查看事务情况,因为我是在一个数据库服务器上做的的跨库数据源,所以我们能看到两条 xa 记录

```
-- insert into archdemo1.order(amount,product) values(6000,'iphone');
-- INSERT INTO `order`(orderid,amount,product) values('00001',3000.00,'苹果笔记本');
use archdemo1;
XA RECOVER;
-- XA ROLLBACK 'MyGlobal', 'order';
-- XA ROLLBACK 'aaa', 'bbb';
-- REPEATABLE-READ
-- set global tx_isolation='REPEATABLE-READ',session tx_isolation='REPEATABLE-READ';

use archdemo2;
xa RECOVER;
```

信息 | 结果1 | 结果2 | 概况 | 状态

formatID	gtrid_length	bqual_length	data
0	8	8	globalidbranch-2
0	8	8	globalidbranch-1

我们继续执行到 commit 语句,放多第一条 commit

```
        result_2 == XAResource.XA_OK) {   result_2: 0
    //都返回OK的话，进行提交阶段
    xaResource_1.commit(xid_1, b: false);   xaResource_1: MysqlXAConnection@1435   xid_1: MysqlXid
    xaResource_2.commit(xid_2, b: false);   xaResource_2: MysqlXAConnection@1449   xid_2: MysqlXid
} else {
    //回滚事务
    xaResource
```

method void multiDataSourceTest()

TwoPhaseCommitApplication › multiDataSourceTest()

这时候可以发现已经第一个事务的 xa 信息已经没有了,也就是第一个事务分支已经提交成功了.

```
mysql> xa recover;
+----------+--------------+--------------+------------------+
| formatID | gtrid_length | bqual_length | data             |
+----------+--------------+--------------+------------------+
|        0 |            8 |            8 | globalidbranch-2 |
+----------+--------------+--------------+------------------+
1 row in set (0.00 sec)

mysql>
```

数据库中可以看到新插入成功了一条数据

30	00001	3000	21:03:38 CST 2019
32	00001	3000	苹果笔记本

这是我们尝试修改结构或者插入一条语句,都会发现数据库处于锁定状态

```
|        0 |            8 |            8 | globalidbranch-2 |
+----------+--------------+--------------+------------------+
1 row in set (0.00 sec)

mysql> update shipping set address='北京黄浦江畔' where id=1;
```

等我们把断点放行之后,才可以看到其他语句正常执行,也就是说 xa 在提交阶段会对数据库进行加锁处理,经过进一步的分析我们发现 xa 在进入 xa end 后就对整个表进行加锁操作,因为该 sql 是 update 语句,所以在 xa end 一直到事务提交或者回滚之前,整个表都处于锁定状态.

```
mysql>
mysql> update shipping set address='北京黄浦江畔' where id=1;
Query OK, 0 rows affected (4 min 9.33 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql>
```

延伸

我们很容易将这种 XA 机制扩展到到微服务情况,需要各个微服务提供相应的机制,各个微服务提供对应的 prepare 接口、commit 接口、rollback 接口。

缺点

xA 的性能问题

XA 的性能很低。一个数据库的事务和多个数据库间的 XA 事务性能对比可发现，性能差 10 倍左右。因此要尽量避免 XA 事务，例如可以将数据写入本地，用高性能的消息系统分发数据。或使用数据库复制等技术。只有在这些都无法实现，且性能不是瓶颈时才应该使用 XA

这种机制假定 prepare ok 的事务都可以正常 commit

也就是进入 prepare 返回 ok 后，在执行 commit 阶段两个事务就有可能出现一些异常情况，比如第一个正常提交了，但第二个却出现了某种异常失败了。

博学谷版权所有