5 Sharding-JDBC分布式事务应用

引入Maven依赖

```
<dependency>
    <groupId>io.shardingsphere</groupId>
    <artifactId>sharding-transaction-2pc-xa</artifactId>
    <version>${shardingsphere.version}</version>
</dependency>
```

XA事务管理器将以SPI的方式被Sharding-JDBC所加载。

连接池配置

ShardingSphere支持将普通的数据库连接池，转换为支持XA事务的连接池，对HikariCP, Druid和DBCP2连接池内置支持，无需额外配置。 其它连接池需要用户实现`DataSourceMapConverter`的SPI接口进行扩展，可以参考`org.apache.shardingsphere.transaction.xa.convert.swap.HikariParameterSwapper`的实现。 若ShardingSphere无法找到合适的实现，则会按默认的配置创建XA事务连接池。 默认属性如下：

属性名称	*默认值*
connectionTimeoutMilliseconds	30 * 1000
idleTimeoutMilliseconds	60 * 1000
maintenanceIntervalMilliseconds	30 * 1000
maxLifetimeMilliseconds	0 (无限制)
maxPoolSize	50
minPoolSize	1

数据源配置说明

```
dataSources: # 配置数据源列表,必须是有效的jdbc配置,目前仅支持MySQL与PostgreSQL,另外通过一些未公开(代码中可查,但可能会在未来有变化)的变量,可以配置来兼容其他支持JDBC的数据库,但由于没有足够的测试支持,可能会有严重的兼容性问题,配置时候要求至少有一个
```

```
  master_ds_0: # 数据源名称,可以是合法的字符串,目前的校验规则中,没有强制性要求,只要是合法的yaml字符串即可,但如果要用于分库分表配置,则需要有有意义的标志(在分库分表配置中详述),以下为目前公开的合法配置项目,不包含内部配置参数
    # 以下参数为必备参数
    url: •jdbc:mysql://127.0.0.1:3306/demo_ds_slave_1?serverTimezone=UTC&useSSL=false # 这里的要求合法的jdbc连接串即可,目前尚未兼容MySQL 8.x,需要在maven编译时候,升级MySQL JDBC版本到5.1.46或者47版本(不建议升级到JDBC的8.x系列版本,需要修改源代码,并且无法通过很多测试case)
    username: root # MySQL用户名
    password: password # MySQL用户的明文密码
    # 以下参数为可选参数,给出示例为默认配置,主要用于连接池控制
    connectionTimeoutMilliseconds: 30000 #连接超时控制
    idleTimeoutMilliseconds: 60000 # 连接空闲时间设置
    maxLifetimeMilliseconds: 0 # 连接的最大持有时间,0为无限制
    maxPoolSize: 50 # 连接池中最大维持的连接数量
    minPoolSize: 1 # 连接池的最小连接数量
    maintenanceIntervalMilliseconds: 30000 # 连接维护的时间间隔 atomikos框架需求
  # 以下配置的假设是,3307是3306的从库,3309,3310是3308的从库
  slave_ds_0:
    url: •jdbc:mysql://127.0.0.1:3307/demo_ds_slave_1?serverTimezone=UTC&useSSL=false
    username: root
    password: password
```

事务类型切换

ShardingSphere的事务类型存放在`TransactionTypeHolder`的本地线程变量中，因此在数据库连接创建前修改此值，可以达到自由切换事务类型的效果。

注意：数据库连接创建之后，事务类型将无法更改。

API方式

```
TransactionTypeHolder.set(TransactionType.LOCAL);
```

或

```
TransactionTypeHolder.set(TransactionType.XA);
```

SpringBootStarter使用方式

引入Maven依赖：

```
<dependency>
    <groupId>io.shardingsphere</groupId>
    <artifactId>sharding-transaction-spring-boot-starter</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>
```

SpringBoot使用方式

引入Maven依赖：

```
<dependency>
    <groupId>io.shardingsphere</groupId>
    <artifactId>sharding-transaction-spring</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
    <version>${spring-boot.version}</version>
</dependency>

<spring-boot.version>[1.5.0.RELEASE,2.0.0.M1)</spring-boot.version>
```

AutoConfiguration配置

```
@SpringBootApplication(exclude = JtaAutoConfiguration.class)
@ComponentScan("io.shardingsphere.transaction.aspect")
public class StartMain {
}
```

Spring Namespace使用方式

引入Maven依赖：

```
<dependency>
    <groupId>io.shardingsphere</groupId>
```

```
    <artifactId>sharding-transaction-spring</artifactId>
    <version>${sharding-sphere.version}</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectjweaver.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${springframework.version}</version>
</dependency>

<aspectjweaver.version>1.8.9</aspectjweaver.version>
<springframework.version>[4.3.6.RELEASE,5.0.0.M1)
</springframework.version>
```

加载切面配置信息

```
<import resource="classpath:META-INF/shardingTransaction.xml"/>
```

业务代码

在需要事务的方法或类中添加相关注解即可，例如：

```
@ShardingTransactionType(TransactionType.LOCAL)
@Transactional
```

或

```
@ShardingTransactionType(TransactionType.XA)
@Transactional
```

注意：`@ShardingTransactionType`需要同Spring的`@Transactional`配套使用，事务才会生效。

Atomikos参数配置

ShardingSphere默认的XA事务管理器为Atomikos。 可以通过在项目的classpath中添加`jta.properties`来定制化Atomikos配置项。 具体的配置规则请参考Atomikos的官方文档

背景

数据库事务需要满足`ACID`（原子性、一致性、隔离性、持久性）四个特性。

- 原子性（Atomicity）指事务作为整体来执行，要么全部执行，要么全不执行。
- 一致性（Consistency）指事务应确保数据从一个一致的状态转变为另一个一致的状态。
- 隔离性（Isolation）指多个事务并发执行时，一个事务的执行不应影响其他事务的执行。
- 持久性（Durability）指已提交的事务修改数据会被持久保存。

在单一数据节点中，事务仅限于对单一数据库资源的访问控制，称之为本地事务。几乎所有的成熟的关系型数据库都提供了对本地事务的原生支持。 但是在基于微服务的分布式应用环境下，越来越多的应用场景要求对多个服务的访问及其相对应的多个数据库资源能纳入到同一个事务当中，分布式事务应运而生。

关系型数据库虽然对本地事务提供了完美的`ACID`原生支持。 但在分布式的场景下，它却成为系统性能的桎梏。如何让数据库在分布式场景下满足`ACID`的特性或找寻相应的替代方案，是分布式事务的重点工作。

本地事务

在不开启任何分布式事务管理器的前提下，让每个数据节点各自管理自己的事务。 它们之间没有协调以及通信的能力，也并不互相知晓其他数据节点事务的成功与否。 本地事务在性能方面无任何损耗，但在强一致性以及最终一致性方面则力不从心。

两阶段提交

XA协议最早的分布式事务模型是由`X/Open`国际联盟提出的`X/Open Distributed Transaction Processing（DTP）`模型，简称XA协议。

基于XA协议实现的分布式事务对业务侵入很小。 它最大的优势就是对使用方透明，用户可以像使用本地事务一样使用基于XA协议的分布式事务。 XA协议能够严格保障事务`ACID`特性。

严格保障事务`ACID`特性是一把双刃剑。 事务执行在过程中需要将所需资源全部锁定，它更加适用于执行时间确定的短事务。 对于长事务来说，整个事务进行期间对数据的独占，将导致对热点数据依赖的业务系统并发性能衰退明显。 因此，在高并发的性能至上场景中，基于XA协议的分布式事务并不是最佳选择。

柔性事务

如果将实现了`ACID`的事务要素的事务称为刚性事务的话，那么基于`BASE`事务要素的事务则称为柔性事务。 `BASE`是基本可用、柔性状态和最终一致性这三个要素的缩写。

- 基本可用（Basically Available）保证分布式事务参与方不一定同时在线。
- 柔性状态（Soft state）则允许系统状态更新有一定的延时，这个延时对客户来说不一定能够察觉。
- 而最终一致性（Eventually consistent）通常是通过消息传递的方式保证系统的最终一致性。

在`ACID`事务中对隔离性的要求很高，在事务执行过程中，必须将所有的资源锁定。 柔性事务的理念则是通过业务逻辑将互斥锁操作从资源层面上移至业务层面。通过放宽对强一致性要求，来换取系统吞吐量的提升。

基于`ACID`的强一致性事务和基于`BASE`的最终一致性事务都不是银弹，只有在最适合的场景中才能发挥它们的最大长处。 可通过下表详细对比它们之间的区别，以帮助开发者进行技术选型。

	本地事务	***两（三）阶段事务***	***柔性事务***
业务改造	无	无	实现相关接口
一致性	不支持	支持	最终一致
隔离性	不支持	支持	业务方保证
并发性能	无影响	严重衰退	略微衰退
适合场景	业务方处理不一致	短事务 & 低并发	长事务 & 高并发

挑战

由于应用的场景不同，需要开发者能够合理的在性能与功能之间权衡各种分布式事务。

强一致的事务与柔性事务的API和功能并不完全相同，在它们之间并不能做到自由的透明切换。在开发决策阶段，就不得不在强一致的事务和柔性事务之间抉择，使得设计和开发成本被大幅增加。

基于XA的强一致事务使用相对简单，但是无法很好的应对互联网的高并发或复杂系统的长事务场景；柔性事务则需要开发者对应用进行改造，接入成本非常高，并且需要开发者自行实现资源锁定和反向补偿。

目标

整合现有的成熟事务方案，为本地事务、两阶段事务和柔性事务提供统一的分布式事务接口，并弥补当前方案的不足，提供一站式的分布式事务解决方案是ShardingSphere分布式事务模块的主要设计目标。