分布式事务

第一章:基本理论

1.分布式事务相关

1.1什么是分布式系统

部署在不同结点上的系统通过网络交互来完成协同工作的系统。

比如：充值加积分的业务，用户在充值系统向自己的账户充钱，在积分系统中自己积分相应的增加。充值系统和积
分系统是两个不同的系统，一次充值加积分的业务就需要这两个系统协同工作来完成。

1.2什么是事务？

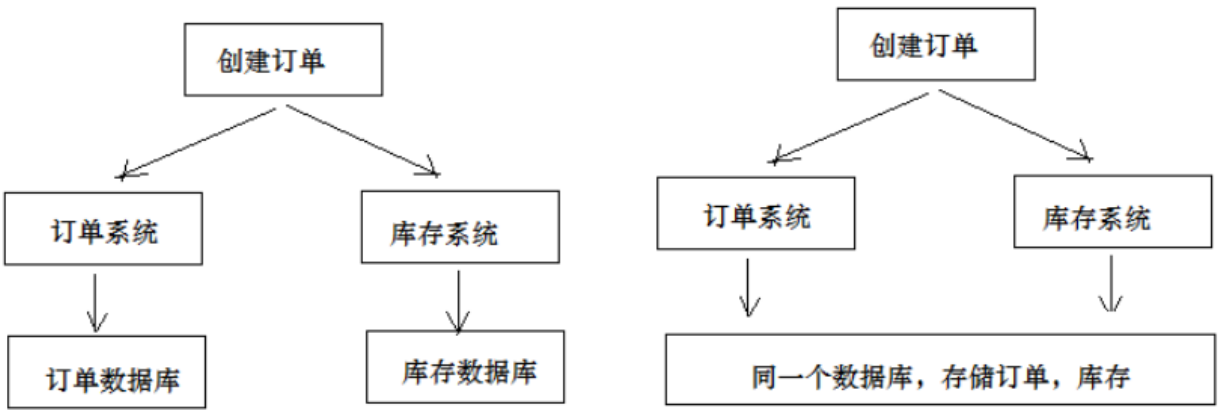
事务是指由一组操作组成的一个工作单元，这个工作单元具有原子性（atomicity）、一致性（consistency）、隔
离性（isolation）和持久性（durability）

1.3什么是本地事务?

本地事务就是用关系数据库来控制事务，关系数据库通常都具有ACID特性，传统的单体应用通常会将数据全部存储
在一个数据库中，会借助关系数据库来完成事务控制。

1.4什么是分布式事务？

在分布式系统中一次操作由多个系统协同完成，这种一次事务操作涉及多个系统通过网络协同完成的过程称为分布
式事务。这里强调的是多个系统通过网络协同完成一个事务的过程，并不强调多个系统访问了不同的数据库，即使多个系统访问的是同一个数据库也是分布式事务，如下图



2.CAP理论

- C:Consistency (一致性)

  在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

- A:Availability（可用性):

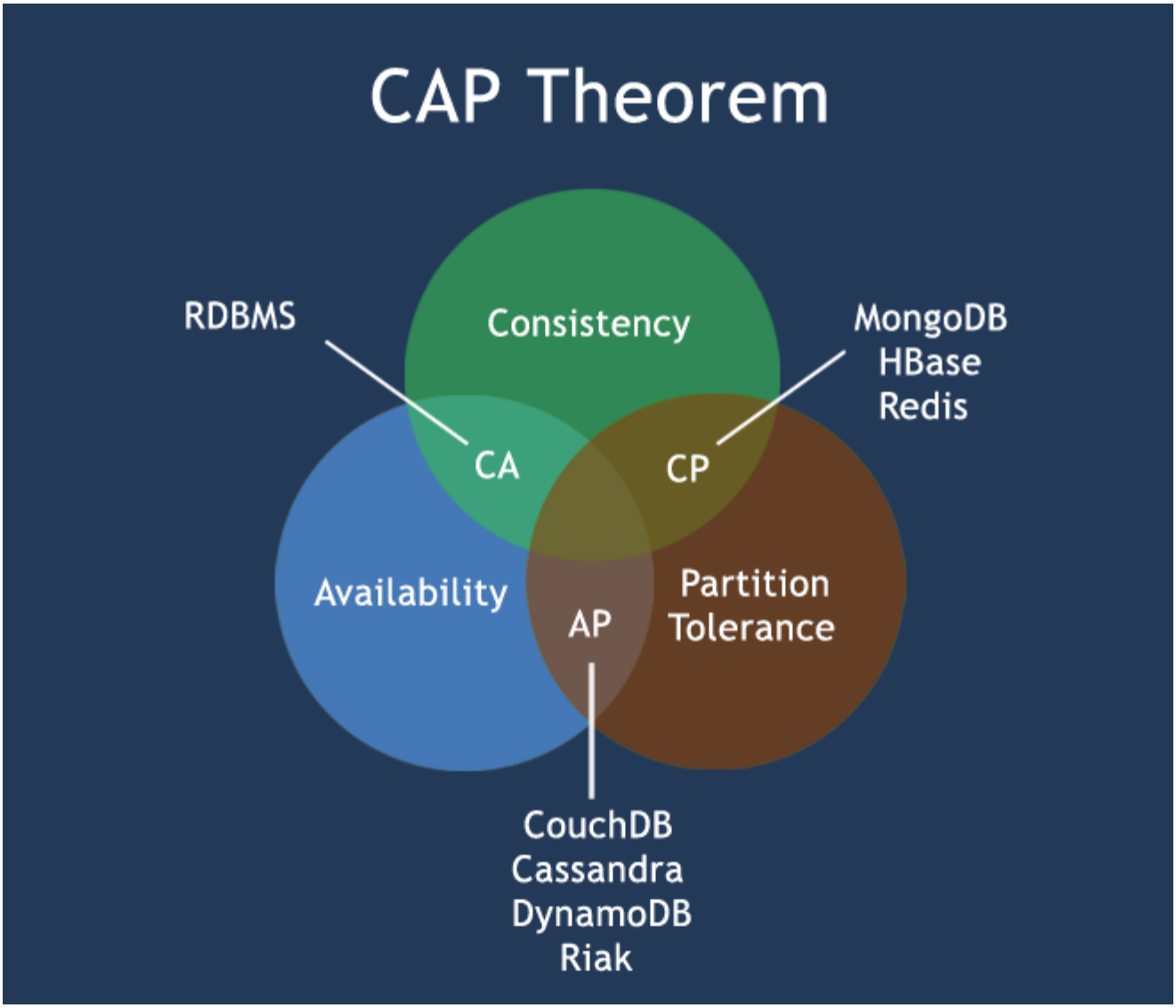
  在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

- P:Partition tolerance（分区容错性）

  以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

  分布式系统不可避免的出现了多个系统通过网络协同工作的场景，结点之间难免会出现网络中断、网延延迟等现
  象，这种现象一旦出现就导致数据被分散在不同的结点上，这就是网络分区。

  CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。因此，根据 CAP 原理将分布式系统分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。CP - 满足一致性，分区容忍必的系统，通常性能不是特别高。AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。而由于当前的网络硬件肯定会出现延迟丢包等问题，所以分区容错性是我们必须需要实现的。所以我们只能在一致性和可用性之间进行权衡.

CA 传统Oracle数据库

AP 大多数网站架构的选择

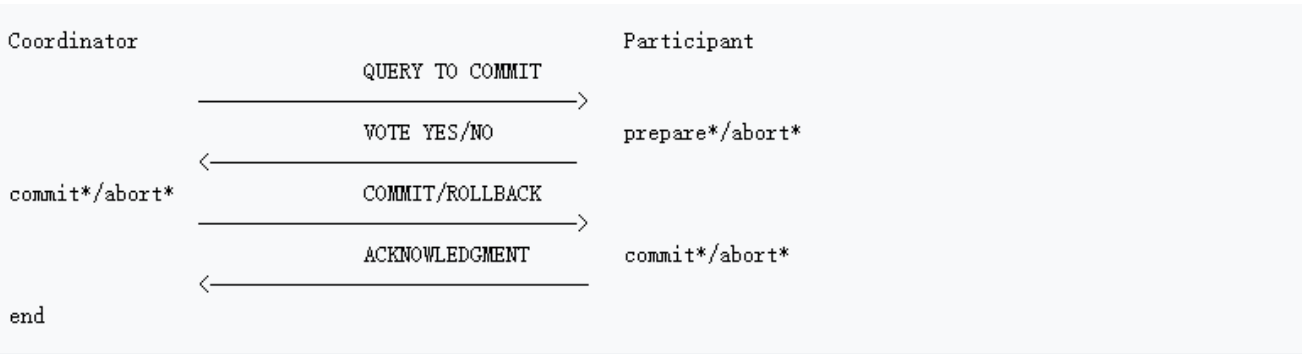
CP Redis、Mongodb

第二章:分布式事务解决方案

1.两阶段提交协议(2PC)

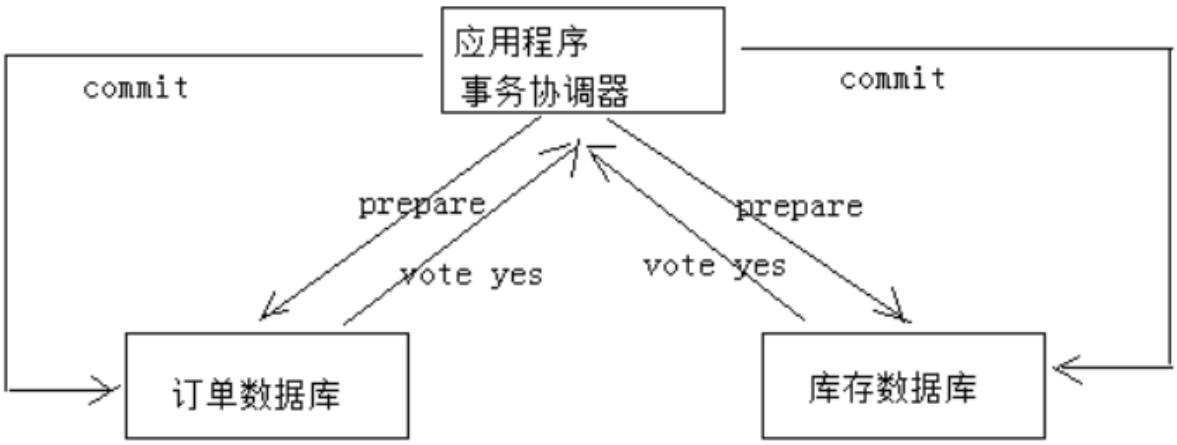
1.1概述

为解决分布式系统的数据一致性问题出现了两阶段提交协议（2 Phase Commitment Protocol），两阶段提交由
协调者和参与者组成，共经过两个阶段和三个操作，部分关系数据库如Oracle、MySQL支持两阶段提交协议，本节讲解关系数据库两阶段提交协议



1）第一阶段：准备阶段（prepare）
协调者通知参与者准备提交订单，参与者开始投票。
协调者完成准备工作向协调者回应Yes。
2）第二阶段：提交(commit)/回滚(rollback)阶段
协调者根据参与者的投票结果发起最终的提交指令。
如果有参与者没有准备好则发起回滚指令。

1.2示例



1、应用程序连接两个数据源。
2、应用程序通过事务协调器向两个库发起prepare，两个数据库收到消息分别执行本地事务（记录日志），但不提交，如果执行成功则回复yes，否则回复no。
3、事务协调器收到回复，只要有一方回复no则分别向参与者发起回滚事务，参与者开始回滚事务。
4、事务协调器收到回复，全部回复yes，此时向参与者发起提交事务。如果参与者有一方提交事务失败则由事务协调器发起回滚事务

1.3总结

优缺点:

- 优点:实现强一致性，部分关系数据库支持（Oracle、MySQL等）。
- 缺点: 整个事务的执行需要由协调者在多个节点之间去协调，增加了事务的执行时间，性能低下。

解决方案:

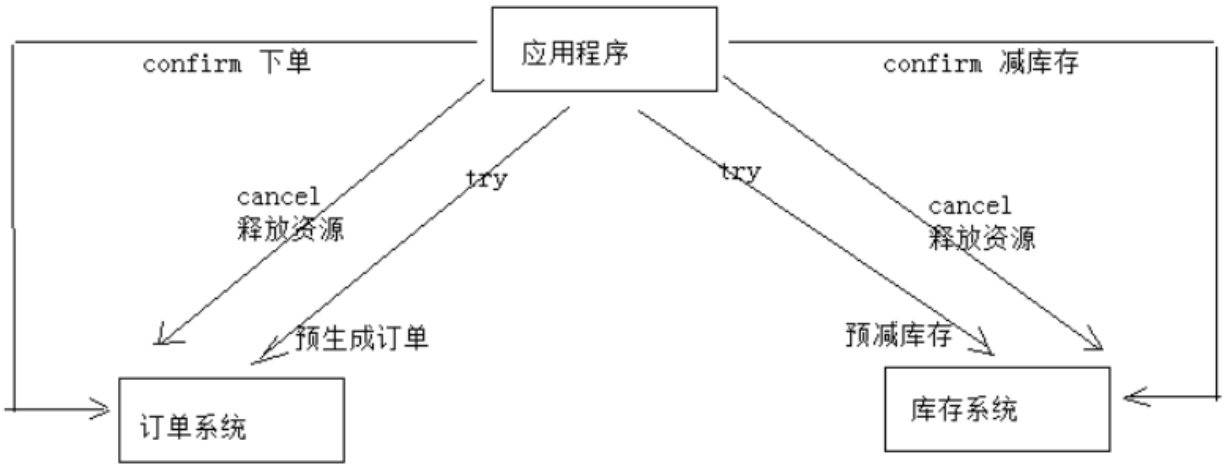
- springboot+Atomikos or Bitronix

2.两阶段提交协议(2PC)

2.1概述

TCC事务补偿是基于2PC实现的业务层事务控制方案，它是Try、Confirm和Cancel三个单词的首字母，含义如下：
1、Try 检查及预留业务资源,完成提交事务前的检查，并预留好资源。
2、Confirm 确定执行业务操作,对try阶段预留的资源正式执行。
3、Cancel 取消执行业务操作,对try阶段预留的资源释放

2.2示例



1、Try
下单业务由订单服务和库存服务协同完成，在try阶段订单服务和库存服务完成检查和预留资源。
订单服务检查当前是否满足提交订单的条件（比如：当前存在未完成订单的不允许提交新订单）。
库存服务检查当前是否有充足的库存，并锁定资源。
2、Confirm
订单服务和库存服务成功完成Try后开始正式执行资源操作。
订单服务向订单写一条订单信息。
库存服务减去库存。
3、Cancel
如果订单服务和库存服务有一方出现失败则全部取消操作。
订单服务需要删除新增的订单信息。
库存服务将减去的库存再还原。

2.3总结

优缺点:

- 优点：最终保证数据的一致性，在业务层实现事务控制，灵活性好。
- 缺点：开发成本高，每个事务操作每个参与者都需要实现try/confirm/cancel三个接口。

注意事项:

- TCC的try/confirm/cancel接口都要实现幂等性，在为在try、confirm、cancel失败后要不断重试。 幂等性是指同一个操作无论请求多少次，其结果都相同。

实现幂等性解决方案

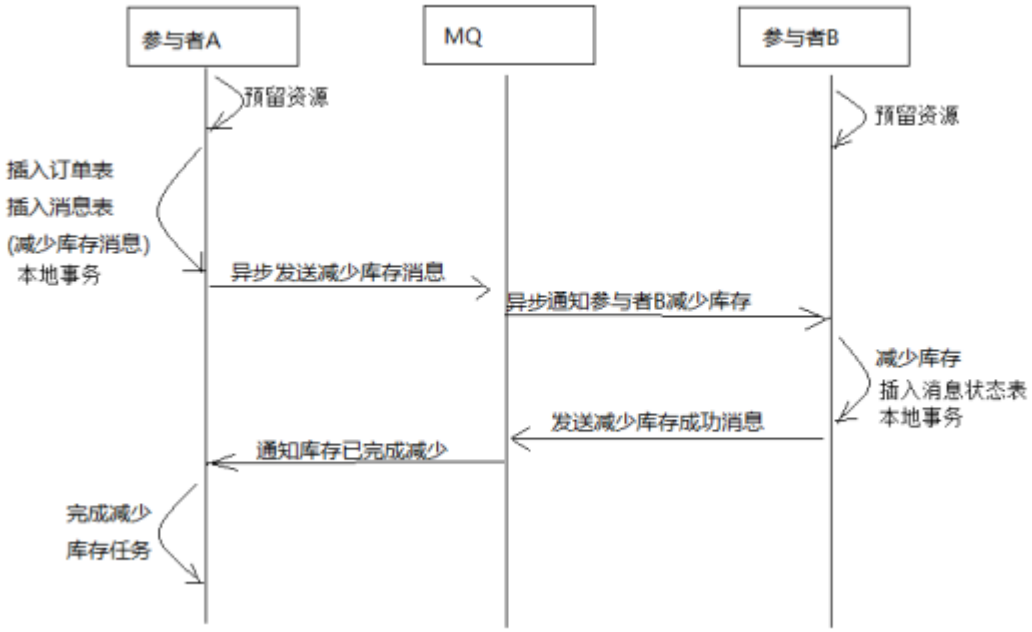
- 操作之前在业务方法进行判断如果执行过了就不再执行。
- 缓存所有请求和处理的结果，已经处理的请求则直接返回结果。
- 在数据库表中加一个状态字段（未处理，已处理），数据操作时判断未处理时再处理。

3.消息队列实现最终一致

3.1概述

将分布式事务拆分成多个本地事务来完成，并且由消息队列异步协调完成

3.2示例



1、订单服务和库存服务完成检查和预留资源。
2、订单服务在本地事务中完成添加订单表记录和添加“减少库存任务消息”。
3、由定时任务根据消息表的记录发送给MQ通知库存服务执行减库存操作。
4、库存服务执行减少库存，并且记录执行消息状态（为避免重复执行消息，在执行减库存之前查询是否执行过此消息）。
5、库存服务向MQ发送完成减少库存的消息。
6、订单服务接收到完成库存减少的消息后删除原来添加的“减少库存任务消息”。
实现最终事务一致要求：预留资源成功理论上要求正式执行成功，如果执行失败会进行重试，要求业务执行方法实现幂等

3.3总结

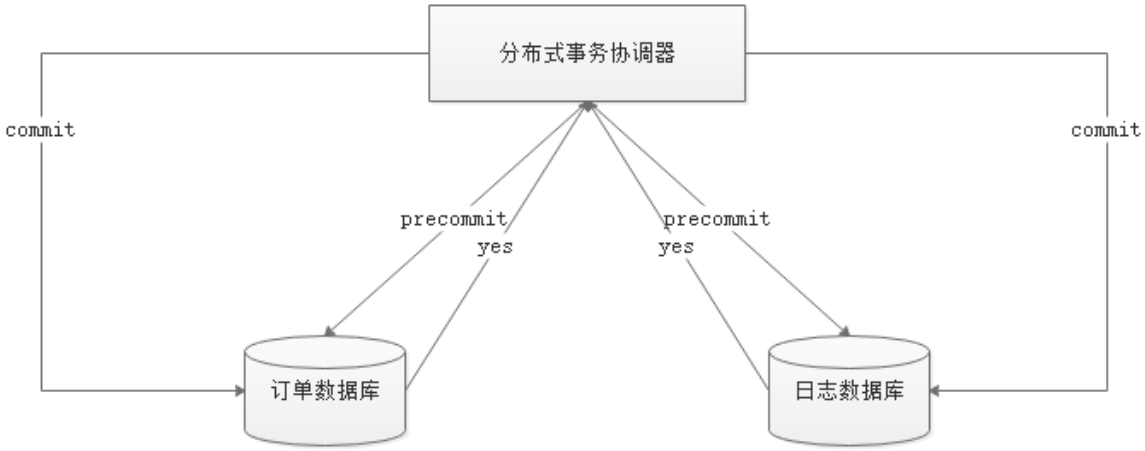
优缺点:

- 优点:由MQ按异步的方式协调完成事务，性能较高。不用实现try/confirm/cancel接口，开发成本比TCC低。
- 缺点: 此方式基于关系数据库本地事务来实现，会出现频繁读写数据库记录，浪费数据库资源，另外对于高并发操作不是最佳方案

第三章-具体实现

1.Atomikos实现分布式事务(2PC)

1.1需求



我们这里准备2个数据库，分别是订单数据库和日志数据库，订单数据库用于接收用户订单，日志数据库用于记录用户的订单创建操作。

1.2Atomikos介绍

Atomikos TransactionsEssentials 是一个为Java平台提供增值服务的并且开源类事务管理器，以下是包括在这个开源版本中的一些功能：

- 全面崩溃 / 重启恢复
- 兼容标准的sun公司jta api
- 嵌套事务
- 为xa和非xa提供内置的jdbc适配器

1.3准备工作

1.3.1数据库的准备和pojo

- order数据库

```
CREATE TABLE `order_info` (
  `id` int(11) NOT NULL,
  `money` double NOT NULL,
  `userid` varchar(20) DEFAULT NULL,
  `address` varchar(200) DEFAULT NULL,
  `createTime` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- log数据库

```
CREATE TABLE `log_info` (
  `id` int(11) NOT NULL,
  `createTime` datetime DEFAULT NULL,
  `content` longtext,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- OrderInfo

```
public class OrderInfo implements Serializable{
    private Integer id;
    private Double money;
    private String userid;
    private String address;
    private Date createTime;
}
```

- LogInfo

```
public class LogInfo implements Serializable {

    private Integer id;
    private Date createTime;
    private String content;
 }
```

1.3.2创建工程atomikos-transaction

- pom.xml 添加坐标

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.itheima</groupId>
    <artifactId>atomikos-transaction</artifactId>
    <version>1.0-SNAPSHOT</version>

    <packaging>jar</packaging>

    <!--常量和版本号-->
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <aspectj.version>1.8.6</aspectj.version>
        <aspectj.weaver>1.8.6</aspectj.weaver>
        <mybatis.spring.version>1.3.0</mybatis.spring.version>
        <mybatis.version>3.4.5</mybatis.version>
        <mysql.version>5.1.32</mysql.version>
        <junit.version>4.12</junit.version>
        <spring.version>4.3.10.RELEASE</spring.version>
        <jta.version>1.1</jta.version>
        <atomikos.version>4.0.6</atomikos.version>
        <cglib.nodep.version>3.2.5</cglib.nodep.version>
        <druid.version>1.0.13</druid.version>
    </properties>

    <dependencies>
        <!--JTA atomikos-->
        <dependency>
            <groupId>javax.transaction</groupId>
            <artifactId>jta</artifactId>
            <version>${jta.version}</version>
        </dependency>
        <dependency>
            <groupId>com.atomikos</groupId>
            <artifactId>atomikos-util</artifactId>
            <version>${atomikos.version}</version>
        </dependency>
        <dependency>
            <groupId>com.atomikos</groupId>
            <artifactId>transactions</artifactId>
            <version>${atomikos.version}</version>
        </dependency>
        <dependency>
            <groupId>com.atomikos</groupId>
            <artifactId>transactions-jta</artifactId>
            <version>${atomikos.version}</version>
        </dependency>

        <dependency>
```

```
            <groupId>com.atomikos</groupId>
            <artifactId>transactions-jdbc</artifactId>
            <version>${atomikos.version}</version>
        </dependency>
        <dependency>
            <groupId>com.atomikos</groupId>
            <artifactId>transactions-api</artifactId>
            <version>${atomikos.version}</version>
        </dependency>
        <dependency>
            <groupId>cglib</groupId>
            <artifactId>cglib-nodep</artifactId>
            <version>${cglib.nodep.version}</version>
        </dependency>


        <!--数据连接池-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>${druid.version}</version>
        </dependency>


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

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

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

        <!-- AspectJ Runtime -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>${aspectj.version}</version>
        </dependency>

        <!-- AspectJ Weaver -->

        <dependency>
```

```
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>${aspectj.weaver}</version>
        </dependency>

        <!-- Spring Jdbc 的支持 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- mybatis-spring 整合 -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>${mybatis.spring.version}</version>
        </dependency>

        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>

        <!-- MySql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- Test dependencies -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

1.4代码实现

1.4.1数据源信息配置

- 创建jdbc.properties，分别配置2个数据源

```
#订单数据库
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://127.0.0.1:3306/order?
useUnicode=true&characterEncoding=utf8&autoReconnect=true
jdbc.username=root
jdbc.pwd=123456
#日志数据库
jdbc.log.driver=com.mysql.jdbc.Driver
jdbc.log.url=jdbc:mysql://127.0.0.1:3306/log?
useUnicode=true&characterEncoding=utf8&autoReconnect=true
jdbc.log.username=root
jdbc.log.pwd=123456
```

1.4.2spring.xml配置文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!--加载配置文件-->
    <context:property-placeholder location="jdbc.properties" />

    <!--包扫描-->
    <context:component-scan base-package="com.itheima" />

    <!-- the transactional advice (what 'happens'; see the <aop:advisor/> bean
        below) 事务传播特性配置 -->
    <tx:advice id="txAdvice" transaction-manager="springTransactionManager">
        <!-- the transactional semantics... -->
        <tx:attributes>
            <tx:method name="add*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <tx:method name="save*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <tx:method name="insert*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <tx:method name="update*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <tx:method name="modify*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <tx:method name="delete*" propagation="REQUIRED" isolation="DEFAULT"
                       rollback-for="java.lang.Exception" />
            <!-- 查询方法 -->
            <tx:method name="query*" read-only="true" />
            <tx:method name="select*" read-only="true" />
            <tx:method name="find*" read-only="true" />
        </tx:attributes>
    </tx:advice>


    <!-- 配置事务管理器 -->
    <bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.UserTransactionImp">
        <property name="transactionTimeout" value="300000"/>
    </bean>

    <!--JTA事务管理器-->
```

```
    <bean id="springTransactionManager"
class="org.springframework.transaction.jta.JtaTransactionManager">
        <property name="userTransaction">
            <ref bean="atomikosUserTransaction"/>
        </property>
        <property name="allowCustomIsolationLevels" value="true"/>
    </bean>


    <!--数据源基础配置-->
    <bean id="abstractXADataSource" class="com.atomikos.jdbc.AtomikosDataSourceBean" init-
method="init" destroy-method="close" abstract="true">
        <property name="xaDataSourceClassName"
value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
        <property name="poolSize" value="10"/>
        <property name="minPoolSize" value="10"/>
        <property name="maxPoolSize" value="30"/>
        <property name="borrowConnectionTimeout" value="60"/>
        <property name="reapTimeout" value="20"/>
        <property name="maxIdleTime" value="60"/>
        <property name="maintenanceInterval" value="60"/>
        <property name="testQuery">
            <value>SELECT 1</value>
        </property>
    </bean>


    <!-- 数据库基本信息配置 -->
    <bean id="dataSourceOne" parent="abstractXADataSource">
        <property name="uniqueResourceName">
            <value>dataSourceOne</value>
        </property>
        <property name="xaDataSourceClassName"
value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
        <property name="xaProperties">
            <props>
                <prop key="URL">${jdbc.url}</prop>
                <prop key="user">${jdbc.username}</prop>
                <prop key="password">${jdbc.pwd}</prop>
            </props>
        </property>
    </bean>

    <!--日志数据源-->
    <bean id="dataSourceLog" parent="abstractXADataSource">
        <property name="uniqueResourceName">
            <value>dataSourceLog</value>
        </property>
        <property name="xaDataSourceClassName"
value="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource"/>
        <property name="xaProperties">
            <props>

                <prop key="URL">${jdbc.log.url}</prop>
```

```
                <prop key="user">${jdbc.log.username}</prop>
                <prop key="password">${jdbc.log.pwd}</prop>
            </props>
        </property>
    </bean>


    <!-- 声明式事务AOP配置 -->
    <aop:config>
        <aop:pointcut expression="execution(* com.itheima.service.impl.*.*(..))"
id="tranpointcut" />

        <!--生命式事务通知-->
        <aop:advisor advice-ref="txAdvice" pointcut-ref="tranpointcut" />
    </aop:config>


    <!--SqlSessionFactoryBean的配置-->
    <bean id="sqlSessionFactoryBeanOne" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="com.itheima.domain" />
        <property name="mapperLocations">
            <array>
                <value>classpath:com/itheima/mapper/*Mapper.xml</value>
            </array>
        </property>
        <property name="dataSource" ref="dataSourceOne"/>
    </bean>

    <bean id="sqlSessionFactoryBeanLog" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="com.itheima.domain" />
        <property name="mapperLocations">
            <array>
                <value>classpath:com/itheima/logmapper/*Mapper.xml</value>
            </array>
        </property>
        <property name="dataSource" ref="dataSourceLog"/>
    </bean>


    <!--包扫描-->
    <bean id="mapperScannerConfigurerOne"
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.itheima.mapper" />
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactoryBeanOne" />
    </bean>
    <bean id="mapperScannerConfigurerLog"
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.itheima.logmapper" />
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactoryBeanLog" />
    </bean>
</beans>
```

1.4.3业务层

- OrderInfoService

```
public interface OrderInfoService {

    /**
     * 增加订单测试事务
     * @param orderInfo
     */
    void save(OrderInfo orderInfo);

}
```

- OrderInfoServiceImpl

```
@Service
public class OrderInfoServiceImpl implements OrderInfoService {

    @Autowired
    private LogInfoMapper logInfoMapper;

    @Autowired
    private OrderInfoMapper orderInfoMapper;

    /**
     * 增加订单测试事务
     * @param orderInfo
     */
    @Override
    public void save(OrderInfo orderInfo) {
        //增加订单
        int acount = orderInfoMapper.add(orderInfo);

        System.out.println("增加订单，受影响行数="+acount);

        //增加日志记录
        int lcount = logInfoMapper.add(new LogInfo((int)(Math.random()*10000),new Date(),"测试事
务。。。。。"));
        System.out.println("增加日志，受影响行数="+lcount);

        //制造异常
        //int q=10/0;
    }
}
```

1.4.4Mapper

- OrderInfoMapper

```
public interface OrderInfoMapper {
    int add(OrderInfo orderInfo);
}
```

- LogInfoMapper

```
public interface LogInfoMapper {
    int add(LogInfo logInfo);
}
```

1.4.5测试

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:spring.xml")
public class AtomikosTest {

    @Autowired
    private OrderInfoService orderInfoService;


    /***
     * 事务测试
     */
    @Test
    public void testTransaction(){
        OrderInfo orderInfo = new OrderInfo();
        orderInfo.setAddress("深圳市");
        orderInfo.setCreateTime(new Date());
        orderInfo.setId(5);
        orderInfo.setMoney(99d);
        orderInfo.setUserid("zhangsan");

        orderInfoService.save(orderInfo);
    }

}
```