多线程中的事务回滚，你真的用对了吗？

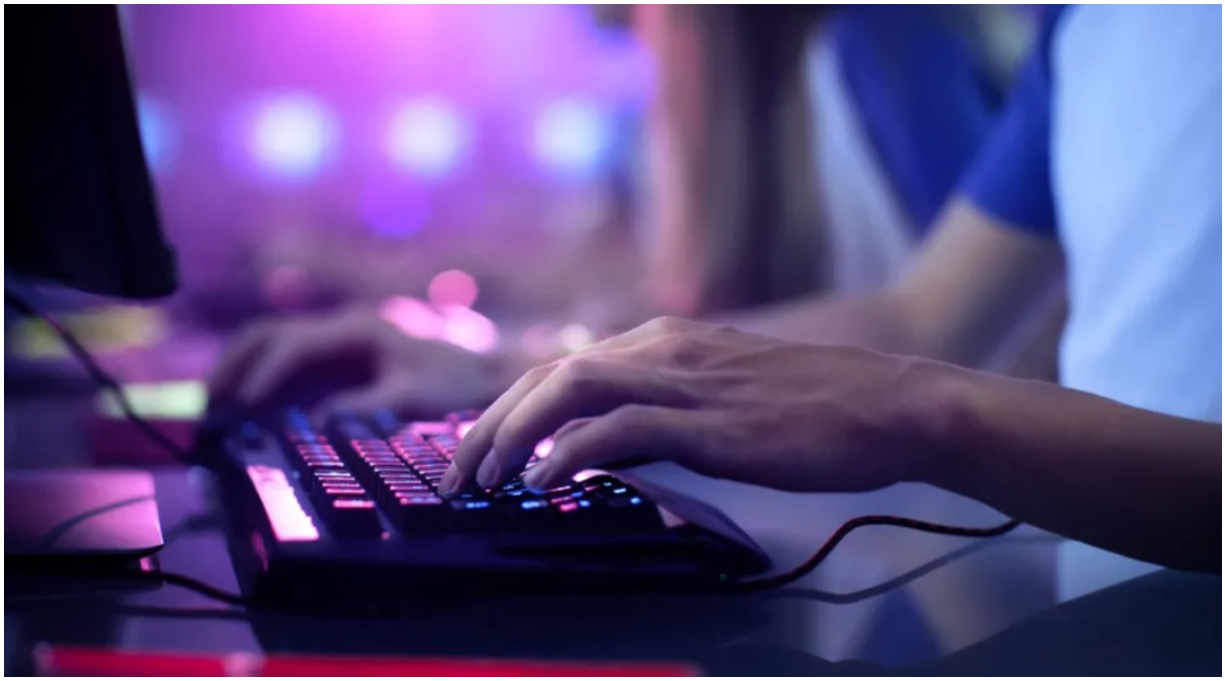
Java架构师宝典 *2022-04-02 11:45*

点击关注公众号，**Java**干货及时送达

Java架构师宝典

专注于 Java 面试题、干货文章分享，不限于算法，数据库，Spring Boot，微服务，高并发，JVM，Docker 容器，ELK相关知识，期待与您一同进步。

Official Account



来源：blog.csdn.net/weixin_43225491/article/details/117705686

背景介绍

1，最近有一个大数据量插入的操作入库的业务场景，需要先做一些其他修改操作，然后在执行插入操作，由于插入数据可能会很多，用到多线程去拆分数据并行处理来提高响应时间，如果有一个线程执行失败，则全部回滚。

2，在spring中可以使用 `@Transactional` 注解去控制事务，使出现异常时会进行回滚，在多线程中，这个注解则不会生效，如果主线程需要先执行一些修改数据库的操作，当子线程在进行处理出现异常时，主线程修改的数据则不会回滚，导致数据错误。

3，下面用一个简单示例演示多线程事务。

公用的类和方法

```java
/**
 * 平均拆分list方法.
 * @param source
 * @param n
 * @param <T>
 * @return
 */
public static <T> List<List<T>> averageAssign(List<T> source,int n){
    List<List<T>> result=new ArrayList<List<T>>();
    int remaider=source.size()%n;
    int number=source.size()/n;
    int offset=0;//偏移量
    for(int i=0;i<n;i++){
        List<T> value=null;
        if(remaider>0){
            value=source.subList(i*number+offset, (i+1)*number+offset+1);
            remaider--;
            offset++;
        }else{
            value=source.subList(i*number+offset, (i+1)*number+offset);
        }
        result.add(value);
    }
    return result;
}
```

```java
/**  线程池配置
 * @version V1.0
 */
public class ExecutorConfig {
    private static int maxPoolSize = Runtime.getRuntime().availableProcessors();
    private volatile static ExecutorService executorService;
    public static ExecutorService getThreadPool() {
        if (executorService == null){
            synchronized (ExecutorConfig.class){
                if (executorService == null){
                    executorService =  newThreadPool();
                }
            }
        }
        return executorService;
    }

    private static  ExecutorService newThreadPool(){
        int queueSize = 500;
        int corePool = Math.min(5, maxPoolSize);
        return new ThreadPoolExecutor(corePool, maxPoolSize, 10000L, TimeUnit.MILLISECONDS,
            new LinkedBlockingQueue<>(queueSize),new ThreadPoolExecutor.AbortPolicy());
    }
    private ExecutorConfig(){}
}
```

```java
/** 获取sqlSession
 * @author 86182
 * @version V1.0
 */
@Component
public class SqlContext {
    @Resource
    private SqlSessionTemplate sqlSessionTemplate;

    public SqlSession getSqlSession(){
        SqlSessionFactory sqlSessionFactory = sqlSessionTemplate.getSqlSessionFactory();
        return sqlSessionFactory.openSession();
    }
}
```

示例事务不成功操作

```java
 /**
 * 测试多线程事务.
 * @param employeeDOList
 */
@Override
@Transactional
public void saveThread(List<EmployeeDO> employeeDOList) {
    try {
        //先做删除操作,如果子线程出现异常,此操作不会回滚
        this.getBaseMapper().delete(null);
        //获取线程池
        ExecutorService service = ExecutorConfig.getThreadPool();
        //拆分数据,拆分5份
        List<List<EmployeeDO>> lists=averageAssign(employeeDOList, 5);
        //执行的线程
        Thread []threadArray = new Thread[lists.size()];
        //监控子线程执行完毕,再执行主线程,要不然会导致主线程关闭,子线程也会随着关闭
        CountDownLatch countDownLatch = new CountDownLatch(lists.size());
        AtomicBoolean atomicBoolean = new AtomicBoolean(true);
        for (int i =0;i<lists.size();i++){
            if (i==lists.size()-1){
                atomicBoolean.set(false);
            }
            List<EmployeeDO> list  = lists.get(i);
            threadArray[i] =  new Thread(() -> {
                try {
                 //最后一个线程抛出异常
                    if (!atomicBoolean.get()){
                        throw new ServiceException("001","出现异常");
                    }
                    //批量添加,mybatisPlus中自带的batch方法
                    this.saveBatch(list);
                }finally {
                    countDownLatch.countDown();
                }

            });
        }
        for (int i = 0; i <lists.size(); i++){
            service.execute(threadArray[i]);
        }
        //当子线程执行完毕时,主线程再往下执行
        countDownLatch.await();
        System.out.println("添加完毕");
    }catch (Exception e){
        log.info("error",e);
        throw new ServiceException("002","出现异常");
    }finally {
         connection.close();
     }
}
```

数据库中存在一条数据：

employee_id	age	employee_name	birth_date	gender	id_number	creat_time	update_time	status
222	12	测试232	2021-06-08 15:57:33	1	1111	2021-06-08 15:57:40	2021-06-08 15:57:43	1

```
//测试用例
@RunWith(SpringRunner.class)
@SpringBootTest(classes = { ThreadTest01.class, MainApplication.class})
public class ThreadTest01 {

    @Resource
    private EmployeeBO employeeBO;

    /**
     *  测试多线程事务.
     * @throws InterruptedException
     */
    @Test
    public  void MoreThreadTest2() throws InterruptedException {
        int size = 10;
        List<EmployeeDO> employeeDOList = new ArrayList<>(size);
        for (int i = 0; i<size;i++){
            EmployeeDO employeeDO = new EmployeeDO();
            employeeDO.setEmployeeName("lol"+i);
            employeeDO.setAge(18);
            employeeDO.setGender(1);
            employeeDO.setIdNumber(i+"XX");
            employeeDO.setCreatTime(Calendar.getInstance().getTime());
            employeeDOList.add(employeeDO);
        }
        try {
            employeeBO.saveThread(employeeDOList);
            System.out.println("添加成功");
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

测试结果：

```
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.lambda$saveThread$0(EmployeeBOImpl.java:152)
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$Lambda$616/394840929.run(Unknown Source)
    at java.lang.Thread.run(Thread.java:745) <2 internal calls>
    at java.lang.Thread.run(Thread.java:745)
ServiceException{code='001'message='出现异常'}
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.lambda$saveThread$0(EmployeeBOImpl.java:152)
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$Lambda$616/394840929.run(Unknown Source)
    at java.lang.Thread.run(Thread.java:745) <1 internal call>
添加完毕 <1 internal call>
    at java.lang.Thread.run(Thread.java:745)
ServiceException{code='001'message='出现异常'}
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.lambda$saveThread$0(EmployeeBOImpl.java:152)
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$Lambda$616/394840929.run(Unknown Source)
    at java.lang.Thread.run(Thread.java:745) <2 internal calls>
    at java.lang.Thread.run(Thread.java:745)
Exception in thread "pool-4-thread-4" ServiceException{code='001'message='出现异常'}
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.lambda$saveThread$0(EmployeeBOImpl.java:152)
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$Lambda$616/394840929.run(Unknown Source)
    at java.lang.Thread.run(Thread.java:745) <2 internal calls>
    at java.lang.Thread.run(Thread.java:745)
Exception in thread "pool-4-thread-3" ServiceException{code='001'message='出现异常'}
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.lambda$saveThread$0(EmployeeBOImpl.java:152)
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$Lambda$616/394840929.run(Unknown Source)
```


employee_id	age	employee_name	birth_date	gender	id_number	creat_time	update_time	status
(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	(Null)

可以发现子线程组执行时，有一个线程执行失败，其他线程也会抛出异常，但是主线程中执行的删除操作，没有回滚，`@Transactional` 注解没有生效。

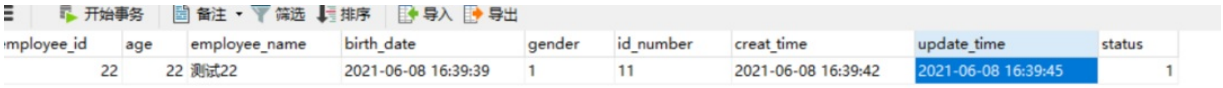
使用 `sqlSession` 控制手动提交事务

```
@Resource
 SqlContext sqlContext;
/**
* 测试多线程事务.
* @param employeeDOList
*/
@Override
public void saveThread(List<EmployeeDO> employeeDOList) throws SQLException {
    // 获取数据库连接,获取会话(内部自有事务)
    SqlSession sqlSession = sqlContext.getSqlSession();
    Connection connection = sqlSession.getConnection();
    try {
        // 设置手动提交
        connection.setAutoCommit(false);
        //获取mapper
        EmployeeMapper employeeMapper = sqlSession.getMapper(EmployeeMapper.class);
        //先做删除操作
        employeeMapper.delete(null);
```

```
        //获取执行器
        ExecutorService service = ExecutorConfig.getThreadPool();
        List<Callable<Integer>> callableList  = new ArrayList<>();
        //拆分list
        List<List<EmployeeDO>> lists=averageAssign(employeeDOList, 5);
        AtomicBoolean atomicBoolean = new AtomicBoolean(true);
        for (int i =0;i<lists.size();i++){
            if (i==lists.size()-1){
                atomicBoolean.set(false);
            }
            List<EmployeeDO> list  = lists.get(i);
            //使用返回结果的callable去执行,
            Callable<Integer> callable = () -> {
                //让最后一个线程抛出异常
                if (!atomicBoolean.get()){
                    throw new ServiceException("001","出现异常");
                }
             return employeeMapper.saveBatch(list);
            };
            callableList.add(callable);
        }
        //执行子线程
       List<Future<Integer>> futures = service.invokeAll(callableList);
        for (Future<Integer> future:futures) {
        //如果有一个执行不成功,则全部回滚
            if (future.get()<=0){
                connection.rollback();
                 return;
            }
        }
        connection.commit();
        System.out.println("添加完毕");
    }catch (Exception e){
        connection.rollback();
        log.info("error",e);
        throw new ServiceException("002","出现异常");
    }finally {
         connection.close();
     }
}
```

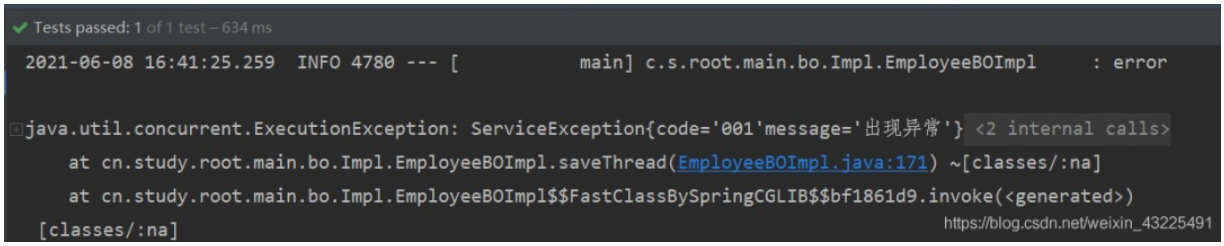
```
// sql
<insert id="saveBatch" parameterType="List">
 INSERT INTO
 employee (employee_id,age,employee_name,birth_date,gender,id_number,creat_time,update_time,status)
 values
    <foreach collection="list" item="item" index="index" separator=",">
    (
    #{item.employeeId},
    #{item.age},
    #{item.employeeName},
    #{item.birthDate},
    #{item.gender},
    #{item.idNumber},
    #{item.creatTime},
    #{item.updateTime},
    #{item.status}
      )
    </foreach>
 </insert>
```

数据库中一条数据：



employee_id	age	employee_name	birth_date	gender	id_number	creat_time	update_time	status
22	22	测试22	2021-06-08 16:39:39	1	11	2021-06-08 16:39:42	2021-06-08 16:39:45	1

测试结果：抛出异常，



```
Tests passed: 1 of 1 test – 634 ms
2021-06-08 16:41:25.259  INFO 4780 --- [           main] c.s.root.main.bo.Impl.EmployeeBOImpl     : error

java.util.concurrent.ExecutionException: ServiceException{code='001'message='出现异常'} <2 internal calls>
    at cn.study.root.main.bo.Impl.EmployeeBOImpl.saveThread(EmployeeBOImpl.java:171) ~[classes/:na]
    at cn.study.root.main.bo.Impl.EmployeeBOImpl$$FastClassBySpringCGLIB$$bf1861d9.invoke(<generated>)
[classes/:na]
```

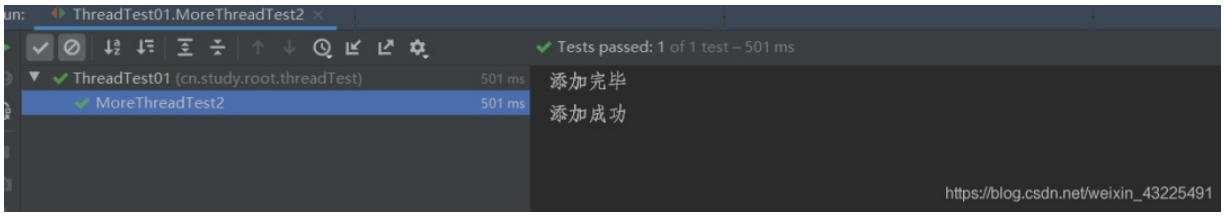

删除操作的数据回滚了，数据库中的数据依旧存在，说明事务成功了。

employee_id	age	employee_name	birth_date	gender	id_number	creat_time	update_time	status
22	22	测试22	2021-06-08 16:39:39	1	11	2021-06-08 16:39:42	2021-06-08 16:39:45	1

成功操作示例：

```
 @Resource
SqlContext sqlContext;
/**
 * 测试多线程事务.
 * @param employeeDOList
 */
@Override
public void saveThread(List<EmployeeDO> employeeDOList) throws SQLException {
    // 获取数据库连接,获取会话(内部自有事务)
    SqlSession sqlSession = sqlContext.getSqlSession();
    Connection connection = sqlSession.getConnection();
    try {
        // 设置手动提交
        connection.setAutoCommit(false);
        EmployeeMapper employeeMapper = sqlSession.getMapper(EmployeeMapper.class);
        //先做删除操作
        employeeMapper.delete(null);
        ExecutorService service = ExecutorConfig.getThreadPool();
        List<Callable<Integer>> callableList  = new ArrayList<>();
        List<List<EmployeeDO>> lists=averageAssign(employeeDOList, 5);
        for (int i =0;i<lists.size();i++){
            List<EmployeeDO> list  = lists.get(i);
            Callable<Integer> callable = () -> employeeMapper.saveBatch(list);
            callableList.add(callable);
        }
        //执行子线程
       List<Future<Integer>> futures = service.invokeAll(callableList);
        for (Future<Integer> future:futures) {
            if (future.get()<=0){
                connection.rollback();
                 return;
            }
        }
        connection.commit();
        System.out.println("添加完毕");
    }catch (Exception e){
        connection.rollback();
        log.info("error",e);
        throw new ServiceException("002","出现异常");
       // throw new ServiceException(ExceptionCodeEnum.EMPLOYEE_SAVE_OR_UPDATE_ERROR);
    }
}
```

测试结果：



数据库中数据：

删除的删除了，添加的添加成功了，测试成功。

对象 employee @study-root (loc...

开始事务 备注 筛选 排序 导入 导出

employee_id	age	employee_name	birth_date	gender	id_number	creat_time	update_time	status
223	18	lol8	(Null)	1	8XX	2021-06-08 16:58:10	(Null)	(Null)
224	18	lol9	(Null)	1	9XX	2021-06-08 16:58:10	(Null)	(Null)
225	18	lol4	(Null)	1	4XX	2021-06-08 16:58:10	(Null)	(Null)
226	18	lol5	(Null)	1	5XX	2021-06-08 16:58:10	(Null)	(Null)
227	18	lol2	(Null)	1	2XX	2021-06-08 16:58:10	(Null)	(Null)
228	18	lol3	(Null)	1	3XX	2021-06-08 16:58:10	(Null)	(Null)
229	18	lol0	(Null)	1	0XX	2021-06-08 16:58:10	(Null)	(Null)
230	18	lol1	(Null)	1	1XX	2021-06-08 16:58:10	(Null)	(Null)
231	18	lol6	(Null)	1	6XX	2021-06-08 16:58:10		(Null)
232	18	lol7	(Null)	1	7XX	2021-06-08 16:58:10	(Null)	(Null)

推荐阅读

IDEA 中文汉化，**So Easy** ！
Git 不能只会 **pull** 和 **push**，试试这**5**条提高效率的命令吧！
Java 8 新特性：**Comparator.naturalOrder | 自然排序**
JDK 18 / Java 18 GA 发布
如何设计一个支撑数亿用户的系统

最近面试BAT，整理一份面试资料**《Java面试BATJ通关手册》**，覆盖了Java核心技术、JVM、Java并发、SSM、微服务、数据库、数据结构等等。

获取方式：点“**在看**”，关注公众号并回复 **Java** 领取，更多内容陆续奉上。

PS：因公众号平台更改了推送规则，如果不想错过内容，记得读完点一下“在看”，加个“星标”，这样每次新文章推送才会第一时间出现在你的订阅列表里。**点“在看”支持我呀，谢谢啦！**😀