Spring源码解析(一) Spring事务控制之Hibernate

原创 2012年07月23日 10:14:05 标签：spring / hibernate / object / aop / session / service

18416

本文将对Spring在整合Hibernate事务方面的源码作一下初步的解析，特别是Spring对线程、事务、Hibernate Session三者的绑定关系。（注：本文基于目前最新的Spring 3.1.2 RELEASE 版本的源码进行分析）
本文原文链接 http://blog.csdn.net/bluishglc/article/details/7774131 转载请注明出处。

众所周知，Spring的事务控制是基于AOP来实现的，而AOP的实现又依赖于JDK的动态代理或者是aspectj，一般情况下，遵循“基于抽象编程”的教义，Service层和DAO层都以接口+实现类的方式提供，此时，Spring会使用动代理来实现AOP拦截，我们的分析也是基于这种拦截方式展开的。那么下面，我们就假设：一个声明了事务的方法（如某个Service的方法）被调用了，在调用时，该方法将被拦截，执行拦截的方法是：

org.springframework.aop.framework.JdkDynamicAopProxy.invoke(Object, Method, Object[])

该方法主要是把所要调用的目标方法和需要施加的操作(即AOP中的Interceptor/Advice)粘合成一个ReflectiveMethodInvocation实例去执行。ReflectiveMethodInvocation最终会调用

org.springframework.transaction.interceptor.TransactionInterceptor.invoke(MethodInvocation)

进行有关事务的操作。作为一个环绕切面，该方法主要负责在目标方法执行前开始一个事务，在方法执行结束后提交事务。

```
91    public Object invoke(final MethodInvocation invocation) throws Throwable {
92        // Work out the target class: may be <code>null</code>.
93        // The TransactionAttributeSource should be passed the target class
94        // as well as the method, which may be from an interface.
95        Class<?> targetClass = (invocation.getThis() != null ? AopUtils.getTargetClass(invocation.getThis()) : null);
96
97        // If the transaction attribute is null, the method is non-transactional.
98        final TransactionAttribute txAttr =
99                getTransactionAttributeSource().getTransactionAttribute(invocation.getMethod(), targetClass);
100       final PlatformTransactionManager tm = determineTransactionManager(txAttr);
101       final String joinpointIdentification = methodIdentification(invocation.getMethod(), targetClass);
102
103       if (txAttr == null || !(tm instanceof CallbackPreferringPlatformTransactionManager)) {
104           // Standard transaction demarcation with getTransaction and commit/rollback calls.
105           TransactionInfo txInfo = createTransactionIfNecessary(tm, txAttr, joinpointIdentification);
106           Object retVal = null;
107           try {
108               // This is an around advice: Invoke the next interceptor in the chain.
109               // This will normally result in a target object being invoked.
110               retVal = invocation.proceed();
111           }
112           catch (Throwable ex) {
113               // target invocation exception
114               completeTransactionAfterThrowing(txInfo, ex);
115               throw ex;
116           }
117           finally {
118               cleanupTransactionInfo(txInfo);
119           }
120           commitTransactionAfterReturning(txInfo);
121           return retVal;
122       }
```

1.目标方法执行前创建事务对象并开始事务

2.执行目标方法

3.目标方法执行结束后提交事务

我们先来深入了解一下事务是如何创建的。从方法createTransactionIfNecessary()上可以看到，创建事务的主要方法是：

org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(TransactionDefinition)

作为抽象类的方法，getTransaction()只处理了一些通用性的检查和设置，实质性的创建事务和开启事务操作都是通过分别调用抽象方法：

org.springframework.transaction.support.AbstractPlatformTransactionManager.doGetTransaction()

和

org.springframework.transaction.support.AbstractPlatformTransactionManager.doBegin(Object,TransactionDefinition)

来完成的，也就是说这些关键性的工作必须由各具体事务管理器来实现，对于hibernate的事务管理器来说，获取事务对象的方法如下：

```
428    @Override
429    protected Object doGetTransaction() {
430        HibernateTransactionObject txObject = new HibernateTransactionObject();
431        txObject.setSavepointAllowed(isNestedTransactionAllowed());
432
433        SessionHolder sessionHolder =
434                (SessionHolder) TransactionSynchronizationManager.getResource(getSessionFactory());
435        if (sessionHolder != null) {
436            if (logger.isDebugEnabled()) {
437                logger.debug("Found thread-bound Session [" +
438                        SessionFactoryUtils.toString(sessionHolder.getSession()) + "] for Hibernate transaction");
439            }
440            txObject.setSessionHolder(sessionHolder);
441        }
442        else if (this.hibernateManagedSession) {
443            try {
444                Session session = getSessionFactory().getCurrentSession();
445                if (logger.isDebugEnabled()) {
446                    logger.debug("Found Hibernate-managed Session [" +
447                            SessionFactoryUtils.toString(session) + "] for Spring-managed transaction");
448                }
449                txObject.setExistingSession(session);
450            }
451            catch (HibernateException ex) {
452                throw new DataAccessResourceFailureException(
453                        "Could not obtain Hibernate-managed Session for Spring-managed transaction", ex);
454            }
455        }
456
457        if (getDataSource() != null) {
458            ConnectionHolder conHolder = (ConnectionHolder)
459                    TransactionSynchronizationManager.getResource(getDataSource());
460            txObject.setConnectionHolder(conHolder);
461        }
462
463        return txObject;
464    }
```

1.创建一个事务对象，hibernate的事务对象主要有两个字段：sessionHolder和connectionHolder

2.从事务同步管理器中查找sessionHolder并设给事务对象.若当前线程此前没有开启过事务，则sessionHolder为null

3.从事务同步管理器中查找connectionHolder并设给事务对象.若当前线程此前没有连接过数据库，则connectionHolder为null

开始事务的方法如下：

```
@Override
protected void doBegin(Object transaction, TransactionDefinition definition) {
    HibernateTransactionObject txObject = (HibernateTransactionObject) transaction;

    if (txObject.hasConnectionHolder() && !txObject.getConnectionHolder().isSynchronizedWithTransaction()) {
        throw new IllegalTransactionStateException(
                "Pre-bound JDBC Connection found! HibernateTransactionManager does not support " +
                "running within DataSourceTransactionManager if told to manage the DataSource itself. " +
                "It is recommended to use a single HibernateTransactionManager for all transactions " +
                "on a single DataSource, no matter whether Hibernate or JDBC access.");
    }

    Session session = null;

    try {
        if (txObject.getSessionHolder() == null || txObject.getSessionHolder().isSynchronizedWithTransaction()) {
            Interceptor entityInterceptor = getEntityInterceptor();
            Session newSession = (entityInterceptor != null ?
                    getSessionFactory().openSession(entityInterceptor) : getSessionFactory().openSession());
            if (logger.isDebugEnabled()) {
                logger.debug("Opened new Session [" + SessionFactoryUtils.toString(newSession) +
                        "] for Hibernate transaction");
            }
            txObject.setSession(newSession);
        }

        session = txObject.getSessionHolder().getSession();
```

创建一个hibernate Session（非线程绑定）

setSession的同时就创建出了事务对象的一个sessionHolder实例

```
if (this.prepareConnection && isSameConnectionForEntireSession(session)) {
    // We're allowed to change the transaction settings of the JDBC Connection.
    if (logger.isDebugEnabled()) {
        logger.debug(
                "Preparing JDBC Connection of Hibernate Session [" + SessionFactoryUtils.toString(session) + "]");
    }
    Connection con = session.connection();
    Integer previousIsolationLevel = DataSourceUtils.prepareConnectionForTransaction(con, definition);
    txObject.setPreviousIsolationLevel(previousIsolationLevel);
}
else {
    // Not allowed to change the transaction settings of the JDBC Connection.
    if (definition.getIsolationLevel() != TransactionDefinition.ISOLATION_DEFAULT) {
        // We should set a specific isolation level but are not allowed to...
        throw new InvalidIsolationLevelException(
                "HibernateTransactionManager is not allowed to support custom isolation levels: " +
                "make sure that its 'prepareConnection' flag is on (the default) and that the " +
                "Hibernate connection release mode is set to 'on_close' (SpringTransactionFactory's default). " +
                "Make sure that your LocalSessionFactoryBean actually uses SpringTransactionFactory: Your " +
                "Hibernate properties should *not* include a 'hibernate.transaction.factory_class' property!");
    }
    if (logger.isDebugEnabled()) {
        logger.debug(
                "Not preparing JDBC Connection of Hibernate Session [" + SessionFactoryUtils.toString(session) + "]");
    }
}

if (definition.isReadOnly() && txObject.isNewSession()) {
    // Just set to NEVER in case of a new Session for this transaction.
    session.setFlushMode(FlushMode.MANUAL);
}

if (!definition.isReadOnly() && !txObject.isNewSession()) {
    // We need AUTO or COMMIT for a non-read-only transaction.
    FlushMode flushMode = session.getFlushMode();
    if (flushMode.lessThan(FlushMode.COMMIT)) {
        session.setFlushMode(FlushMode.AUTO);
        txObject.getSessionHolder().setPreviousFlushMode(flushMode);
    }
}

Transaction hibTx;

// Register transaction timeout.
int timeout = determineTimeout(definition);
if (timeout != TransactionDefinition.TIMEOUT_DEFAULT) {
    // Use Hibernate's own transaction timeout mechanism on Hibernate 3.1+
    // Applies to all statements, also to inserts, updates and deletes!
    hibTx = session.getTransaction();
    hibTx.setTimeout(timeout);
    hibTx.begin();
}
else {
    // Open a plain Hibernate transaction without specified timeout.
    hibTx = session.beginTransaction();
}

// Add the Hibernate transaction to the session holder.
txObject.getSessionHolder().setTransaction(hibTx);

// Register the Hibernate Session's JDBC Connection for the DataSource, if set.
if (getDataSource() != null) {
    Connection con = session.connection();
    ConnectionHolder conHolder = new ConnectionHolder(con);
    if (timeout != TransactionDefinition.TIMEOUT_DEFAULT) {
        conHolder.setTimeoutInSeconds(timeout);
    }
    if (logger.isDebugEnabled()) {
        logger.debug("Exposing Hibernate transaction as JDBC transaction [" + con +
    }
    TransactionSynchronizationManager.bindResource(getDataSource(), conHolder);
    txObject.setConnectionHolder(conHolder);
}

// Bind the session holder to the thread.
if (txObject.isNewSessionHolder()) {
    TransactionSynchronizationManager.bindResource(getSessionFactory(), txObject.getSessionHolder());
}
txObject.getSessionHolder().setSynchronizedWithTransaction(true);
}

catch (Exception ex) {
    if (txObject.isNewSession()) {
        try {
            if (session.getTransaction().isActive()) {
                session.getTransaction().rollback();
            }
        }
        catch (Throwable ex2) {
            logger.debug("Could not rollback Session after failed transaction begin", ex);
        }
        finally {
            SessionFactoryUtils.closeSession(session);
        }
```

开始实质上的hibernate事务

到此，SessionHolder拥有了一个session与一个transaction

把DataSource和当前Connection又做了绑定（即以键值对形式存入当前线程变量）

至此，对像txObject拥有了一个sessionHolder和一个connectionHolder

把SessionHolder和SessionFactory也做了绑定（即以键值对的形式存入当前线程变量）

```
            }
            throw new CannotCreateTransactionException("Could not open Hibernate Session for transaction", ex);
        }
    }
```


以上是关于事务开始部分的代码，下面我们来看一下事务提交时的代码：
同样的，从方法commitTransactionAfterReturning()我们可以看出执行事务提交的方法主要通过回调
org.springframework.orm.hibernate3.HibernateTransactionManager.doCommit(DefaultTransactionStatus)
来实现的。

```
648    @Override
649    protected void doCommit(DefaultTransactionStatus status) {
650        HibernateTransactionObject txObject = (HibernateTransactionObject) status.getTransaction();
651        if (status.isDebug()) {
652            logger.debug("Committing Hibernate transaction on Session [" +
653                    SessionFactoryUtils.toString(txObject.getSessionHolder().getSession()) + "]");
654        }
655        try {
656            txObject.getSessionHolder().getTransaction().commit();
657        }
658        catch (org.hibernate.TransactionException ex) {
659            // assumably from commit call to the underlying JDBC connection
660            throw new TransactionSystemException("Could not commit Hibernate transaction", ex);
661        }
662        catch (HibernateException ex) {
663            // assumably failed to flush changes to database
664            throw convertHibernateAccessException(ex);
665        }
666    }
```

完成实质上的
hibernate事务提交

补充：

关于方法
org.springframework.transaction.support.TransactionSynchronizationManager.getResource(Object key)
如该方法的注释所说，它主要是通过给定的key找到对应的资源，特别之处是这些资源实例是绑定在线程上的，也就是spring保证一个线程上一个key对应一个资源实例，不同的线程上绑定的是不同的资源实例。对应到Hibernate上来说，key是sessionFactory,资源是sessionHolder！

近期其他博文：

数据库分库分表(sharding)系列(三) 关于使用框架还是自主开发以及sharding实现层面的考量

数据库分库分表(sharding)系列(二) 全局主键生成策略

数据库分库分表(sharding)系列(一) 拆分实施策略和示例演示

boluoy 2018-01-15 15:41 回复 9楼

我发现新servlet和filter的init函数里调用service的方法，dao层如果用getcurrentsession()都获取不到session，但是在doget(),dofilter()函数里就可以。而且如果在doget()，dofiler()函数里再开一个线程也获取不到session。难道通过spring管理的session是和和带请求的线程绑定的，一般线程无法创建session？有什么方法可以解这个问题吗

a417074865 2017-06-29 18:10 回复 8楼

6666厉害了楼主

DIrtyG 2015-11-02 14:10 回复 7楼

太厉害

查看 10 条热评

Spring 中配置的问题 org.springframework.transaction.interceptor.Transacti...

问题描述： org.springframework.beans.factory.BeanNotOfRequiredTypeException: Bean named 'org.springframe...

NBA_2011 2011年08月22日 15:49 15863

三大框架开发时，spring配置文件出现org.springframework.transaction.interc...

在最近利用三大框架进行项目开发时，spring配置文件里出现了一个橘黄色的双向箭头，鼠标放上去，会提示你advised by org.springframework.transaction.int...

qq_30408111 2016年03月10日 13:17 4785

Android之beginTransaction - CSDN博客

beginTransaction()方法只能commit()提交一次,要关闭两个Fragment,需要定义两个...Spring 10篇 hibernate 4篇 其他 3篇 展开 文章存档 2017年8月 1篇 ...

2018-2-5

spring transaction--事务架构 - CSDN博客

事务通常是以begin transaction开始,以commit或rollback结束。Commint表示提交,即...3. spring事务实现源码分析 3.1 dao模块 dao模块定义了数据库层的各种异常,其中...

2018-2-2

Spring事务传播机制小记

sgls652709 2015年10月30日 17:08 7643

之前对spring的事务传播机制没有概念，花点时间去看了事务的源码，以及这些事务传播机制使用的文档，在此做一下简单的笔记...

Spring Transaction 源码 - CSDN博客

package org.springframework.transaction.interceptor; import java.io.IOException;...// 抽象方法 doBegin(transaction, definition); prepareSynchronization(status, ...

2018-1-27

Transaction-事务 - CSDN博客

2.事务的语句 开始事物:BEGIN TRANSACTION 提交事物:COMMIT TRANSACTION 回滚事务:...spring transaction源码分析--事务架构 1. 引言 事务特性事务是并发控制的单元...

2018-1-30

Spring事务异常rollback-only

sgls652709 2015年10月28日 18:09 21331

在利用单元测试验证spring事务传播机制的时候出现了下面的异常： Transaction rolled back because it has been marked as rollback-on...

源码解析：Hibernate 事务 设计 （ 版本 3.1.3 ）

内容举报

返回顶部

Hibernate是一个ORM工具，即实现对象到关系型数据库的映射，使用者不需要关系SQ，只要通过曹组对象就可以实现数据的增删改查，简单了解Hibernate的使用： 要使用Hibernate，一个非...


conquer0715 2016年03月24日 19:34 3406

begin transaction/rollback transaction - CSDN博客

BEGIN TRANSACTION represents a point at which the data referenced by a connection...Spring Framework 14篇 Report & Jasper Report 1篇 BA Writing Skills ...

2018-2-2

Spring Transaction - CSDN博客

用过 Hibernate 的人都知道,我们需要在代码中显式调用beginTransaction()、commit()、rollback()等事务管理相关的方法,这就是编程式事务管理。通过 Spring 提供的...

2018-2-5

Spring源码解析(一) Spring事务控制之Hibernate

本文将对Spring在整合Hibernate事务方面的源码作一下初步的解析，特别是Spring对线程、事务、Hibernate Session三者的绑定关系。（注：本文基于目前最新的Spring 3....


kungfu_star 2014年03月26日 08:40 368

begin transaction/rollback transaction - CSDN博客

BEGIN TRANSACTION represents a point at which the data referenced by a connection...Spring Framework 14篇 Report & Jasper Report 1篇 BA Writing Skills ...

2018-2-2

Transaction 浅析 - CSDN博客

(); Session ss =sf.openSession(); ss.beginTransaction(); //执行增删改...(''] for Spring-managed transaction").toString()); txObject.setExisting...

2018-2-5

[转]ibatis +spring ,hibernate 批量操作心得

xijinlan 2008年01月08日 18:06 1651

ibatis +spring批量操作心得 http://www.blogjava.net/zp0127/default.html?page=2 程序功能：使用ibatis+spring将oracle数...

使用SPRING+HIBERNATE 控制事务

a6697238 2016年01月12日 19:57 1850

使用Spring集成Hinernate按照从下向上方法建立 1.Hibernate对应的Dao层集成HibernateDaoSupport这个类，这个类是Spring对Hibernate的集...

Transaction 事务 - CSDN博客

并删除 session.beginTransaction().commit(); //在获得当前线程绑定的session时...那些年spring声明式事务@Transaction的坑 作为开发人员,我相信同学们都会遇到这样...

2018-1-27

spring-framework-2.0.8-with-dependencies-CSDN下载

This product includes software developed by Clinton Begin (http://www.ibatis...transaction infrastructure - Dependencies: spring-core, (spring-aop, spring-...

2018-2-5

《Spring技术内幕——深入解析Spring架构与设计原理》连载3

Spring技术内幕——深入解析Spring架构与设计原理》 书名：Spring技术内幕——深入解析Spring架构与设计原理作者：计文柯ISBN：9787111288060丛书名：揭秘系列丛书出版社...

jiwenke 2010年03月25日 15:38 1675

Struts,Spring,Hibernate三大框架的面试与笔试题

1.Hibernate工作原理及为什么要用？ 原理： 1.读取并解析配置文件 2.读取并解析映射信息，创建SessionFactory 3.打开Session 4.创建事务Transation 5....

qiaohuang6026 2017年06月02日 11:04 163

spring、hibernate事务管理的区别及整合方式

原文地址：点击打开链接 在谈Spring事务管理之前我们想一下在我们不用Spring的时候，在Hibernate中我们是怎么进行数据操作的。在Hibernate中我们每次进行一个操作的的时候我们...

u010921701 2017年02月20日 17:28 403

Hibernate中的接口

zhaodandan19910306 2012年02月25日 21:14 925

Hibernate的概念前面我们已经有所了解，现在我们就从它的接口开始对它进行更深一步的认识。 一、Hibernate的接口分布图： 我先对该接口分布图进行一下讲解： ...

spring整合hibernate事务管理的四种方式，以及事务的传播行为和隔离级别介绍

转自:http://blog.csdn.net/jianxin1009/article/details/9202907 为了保证数据的一致性，在编程的时候往往需要引入事务这个概念。事务有...

zhangqindabendan 2017年04月03日 16:37 2015

Hibernate学习之---事务控制

StrongHYQ 2017年08月07日 23:38 95

事务是一步或几步基本操作组成的逻辑执行单元，这些基本操作作为一个整体执行单元，它们要么全部执行，要么全部取消执行，绝不能仅仅执行部分。 事务具备4个特性:原子性、一致性、隔离性和持续性。1.Sess...

Spring事务之事务控制方式

oTengYue 2016年04月13日 19:32 2983

编程式事务控制和XML配置（AOP）事务控制

spring springmvc hibernate(ssh)项目整合开发---总体架构搭建

ssm项目的整合-先进行整体架构的搭建

u013871100 2016年10月02日 23:17 6845

spring与mybatis整合及事务控制


zdp072 2014年09月01日 09:43 5315

一. 简介 本文将会使用spring整合mybatis, 并添加事务管理, 以此为记, 方便以后查阅。...

Hibernate与Jpa的关系, Spring JpaTransactionManager事务管理

Jpa是一种规范，而Hibernate是它的一种实现 jpa中有Entity, Table，hibernate中也有，但是内容不同jpa中有Column,OneToMany等，Hibernate...

weikzhao0521 2017年04月12日 21:38 2118

Spring集成ORM框架之Hibernate的使用

polo_longsan 2015年06月10日 21:43 792

最近在学习Spring整合ORM框架。在使用Hibernate的时候遇到很多问题，花费了几天时间总算解决了。Hibernate的主要接口是org.hibernate.session，session提供...

Spring整合hibernate关于控制事务的问题

qq_27739989 2017年07月25日 14:31 124

今天在使用spring整合hibernate的框架中控制事务遇到数据更新未取到最新的数据，代码结构如下: 注:此处调用查询方法定义为query(), 调用更新方法定义为update()， 调用执...