



log_cd

浏览: 188658 次

性别: ♀

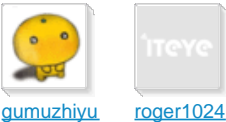
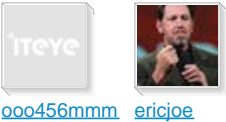
来自: 成都

我现在离线

[详细资料](#) [留言簿](#)

搜索本博客

最近访客 [>>更多访客](#)



博客分类

- 全部博客 (141)
- [Spring/Hibernate/iBatis \(21\)](#)
- [java 基础应用 \(23\)](#)
- [Tapestry/Wicket/JSF2 \(13\)](#)
- [工具类 \(21\)](#)
- [RMI/WebService \(4\)](#)
- [js/css \(14\)](#)
- [应用服务器相关 \(4\)](#)
- [ActiveMQ/JBossRules \(7\)](#)
- [eclipse 应用 \(6\)](#)
- [数据库相关 \(21\)](#)
- [JavaScript 框架 \(7\)](#)
- [Groovy 应用 \(2\)](#)

2010-11-10

[Spring3 Annotation + Hibernate3-jpa2.0 + ...](#) ▶

Spring分布式事务实现

博客分类: [Spring/Hibernate/iBatis](#)

[Spring](#) [SQL](#) [配置管理](#) [XML](#) [Websphere](#)

分布式事务是指操作多个数据库之间的事务，spring的org.springframework.transaction.jta.JtaTransactionManager，提供了分布式事务支持。如果使用WAS的JTA支持，把它的属性改为WebSphere对应的TransactionManager。

在tomcat下，是没有分布式事务的，不过可以借助于第三方软件jotm（Java Open Transaction Manager）和AtomikosTransactionsEssentials实现，在spring中分布式事务是通过jta（jotm，atomikos）来进行实现。

- 1、<http://jotm.objectweb.org/>
- 2、<http://www.atomikos.com/Main/TransactionsEssentials>

一、使用JOTM例子

(1)、Dao及实现

Java代码 ☆

```
1. public interface GenericDao {
2.
3.     public int save(String ds, String sql, Object[] obj) throws Exception;
4.
5.     public int findRowCount(String ds, String sql);
6.
7. }
```

Java代码 ☆

```
1. public class GenericDaoImpl implements GenericDao{
2.
3.     private JdbcTemplate jdbcTemplateA;
4.     private JdbcTemplate jdbcTemplateB;
5.
6.     public void setJdbcTemplateA(JdbcTemplate jdbcTemplate) {
7.         this.jdbcTemplateA = jdbcTemplate;
8.     }
9.
10.    public void setJdbcTemplateB(JdbcTemplate jdbcTemplate) {
11.        this.jdbcTemplateB = jdbcTemplate;
12.    }
13.
14.    public int save(String ds, String sql, Object[] obj) throws Exception{
15.        if(null == ds || "".equals(ds)) return -1;
16.        try{
17.            if(ds.equals("A")){
18.                return this.jdbcTemplateA.update(sql, obj);
19.            }else{
20.                return this.jdbcTemplateB.update(sql, obj);
21.            }
22.        }catch(Exception e){
23.            e.printStackTrace();
24.            throw new Exception("执行" + ds + "数据库时失败! ");
```

我的相册



596881.jpg
[共 2 张](#)

我的留言簿 [>>更多留言](#)

- 你好，我现在也在学
习tapestry，但是很多东西不能
连贯使用，官方的文档虽然全，
...
-- by [loupogames](#)
- 请教问题 请教问题 您好，能帮
我个忙吗，我有几个activemq问
题想请教您，我 ...
-- by [qaz5329223](#)
- 您好，能不能把你研究
的ireport+jasperReports的完成
代码实例发一 ...
-- by [yuandewen](#)

其他分类

- [我的收藏](#) (25)
- [我的代码](#) (0)
- [我的论坛主题帖](#) (5)
- [我的所有论坛帖](#) (5)
- [我的精华良好帖](#) (0)

最近加入群组

- [Jquery](#)
- [Groovy on Grails](#)

存档

- [2010-11](#) (1)
- [2010-08](#) (1)
- [2010-05](#) (3)
- [更多存档...](#)

评论排行榜

- [Spring3 Annotation +](#)
[Hibernate3-jpa2.0 + ...](#)



```
25.         }
26.     }
27.
28.     public int findRowCount(String ds, String sql) {
29.         if(null == ds || "".equals(ds)) return -1;
30.
31.         if(ds.equals("A")){
32.             return this.jdbcTemplateA.queryForInt(sql);
33.         }else{
34.             return this.jdbcTemplateB.queryForInt(sql);
35.         }
36.     }
37.
38. }
```

(2)、Service及实现

Java代码 ☆

```
1.     public interface UserService {
2.
3.         public void saveUser() throws Exception;
4.
5.     }
```

Java代码 ☆

```
1.     public class UserServiceImpl implements UserService{
2.
3.         private GenericDao genericDao;
4.
5.         public void setGenericDao(GenericDao genericDao) {
6.             this.genericDao = genericDao;
7.         }
8.
9.         public void saveUser() throws Exception {
10.             String userName = "user_" + Math.round(Math.random()*10000);
11.             System.out.println(userName);
12.
13.             StringBuilder sql = new StringBuilder();
14.             sql.append(" insert into t_user(username, gender) values(?,?); ");
15.             Object[] objs = new Object[]{userName,"1"};
16.
17.             genericDao.save("A", sql.toString(), objs);
18.
19.             sql.delete(0, sql.length());
20.             sql.append(" insert into t_user(name, sex) values(?,?); ");
21.             objs = new Object[]{userName,"男的"}; //值超出范围
22.             genericDao.save("B", sql.toString(), objs);
23.         }
24.
25.     }
```

(3)、 applicationContext-jotm.xml

Xml代码 ☆

```
1.     <?xml version="1.0" encoding="UTF-8"?>
2.
3.     <beans xmlns="http://www.springframework.org/schema/beans"
4.           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.           xmlns:context="http://www.springframework.org/schema/context"
6.           xmlns:aop="http://www.springframework.org/schema/aop"
```

```
7.         xmlns:tx="http://www.springframework.org/schema/tx"
8.         xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframewo
rk.org/schema/beans/spring-beans-2.5.xsd
9.         http://www.springframework.org/schema/context http://www.springframework.org/schema/cont
ext/spring-context-2.5.xsd
10.        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring
-tx-2.5.xsd
11.        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spri
ng-aop-2.5.xsd">
12.
13.        <description>springJTA</description>
14.
15.        <!-- 指定Spring配置中用到的属性文件-->
16.        <bean id="propertyConfig"
17.            class="org.springframework.beans.factory.config.PropertyPlaceholderConf
igurer">
18.            <property name="locations">
19.                <list>
20.                    <value>classpath:jdbc.properties</value>
21.                </list>
22.            </property>
23.        </bean>
24.
25.        <!-- JOTM实例 -->
26.        <bean id="jotm" class="org.springframework.transaction.jta.JotmFactoryBean">
27.            <property name="defaultTimeout" value="500000"/>
28.        </bean>
29.
30.        <!-- JTA事务管理器 -->
31.        <bean id="jtaTransactionManager" class="org.springframework.transaction.jta.JtaTransacti
onManager">
32.            <property name="userTransaction" ref="jotm" />
33.        </bean>
34.
35.        <!-- 数据源A -->
36.        <bean id="dataSourceA" class="org.enhydra.jdbc.pool.StandardXAPoolDataSource" destroy-method
="shutdown">
37.            <property name="dataSource">
38.                <bean class="org.enhydra.jdbc.standard.StandardXADataSource" destroy-method="shutdown
">
39.                    <property name="transactionManager" ref="jotm"/>
40.                    <property name="driverName" value="{jdbc.driver}"/>
41.                    <property name="url" value="{jdbc.url}"/>
42.                </bean>
43.            </property>
44.            <property name="user" value="{jdbc.username}"/>
45.            <property name="password" value="{jdbc.password}"/>
46.        </bean>
47.
48.        <!-- 数据源B -->
49.        <bean id="dataSourceB" class="org.enhydra.jdbc.pool.StandardXAPoolDataSource" destroy-method
="shutdown">
50.            <property name="dataSource">
51.                <bean class="org.enhydra.jdbc.standard.StandardXADataSource" destroy-method="shutdown
">
52.                    <property name="transactionManager" ref="jotm"/>
53.                    <property name="driverName" value="{jdbc2.driver}"/>
54.                    <property name="url" value="{jdbc2.url}"/>
55.                </bean>
56.            </property>
57.            <property name="user" value="{jdbc2.username}"/>
58.            <property name="password" value="{jdbc2.password}"/>
59.        </bean>
60.
```

```
61. <bean id = "jdbcTemplateA"
62.     class = "org.springframework.jdbc.core.JdbcTemplate">
63.     <property name = "dataSource" ref="dataSourceA"/>
64. </bean>
65.
66. <bean id = "jdbcTemplateB"
67.     class = "org.springframework.jdbc.core.JdbcTemplate">
68.     <property name = "dataSource" ref="dataSourceB"/>
69. </bean>
70.
71. <!-- 事务切面配置 -->
72. <aop:config>
73.     <aop:pointcut id="pointCut"
74.         expression="execution(* com.logcd.service..*(..))"/><!-- 包及其子包下的所有方法 -->
75.     <aop:advisor pointcut-ref="pointCut" advice-ref="txAdvice"/>
76.
77.     <aop:advisor pointcut="execution(* *..common.service..*(..))" advice-ref="txAdvice"/>
78.
79. </aop:config>
80.
81. <!-- 通知配置 -->
82. <tx:advice id="txAdvice" transaction-manager="jtaTransactionManager">
83. <tx:attributes>
84.     <tx:method name="delete*" rollback-for="Exception"/>
85.     <tx:method name="save*" rollback-for="Exception"/>
86.     <tx:method name="update*" rollback-for="Exception"/>
87.     <tx:method name="find*" read-only="true" rollback-for="Exception"/>
88. </tx:attributes>
89. </tx:advice>
90.
91. <bean id="genericDao"
92.     class="com.logcd.dao.impl.GenericDaoImpl" autowire="byName">
93. </bean>
94.
95. <bean id="userService"
96.     class="com.logcd.service.impl.UserServiceImpl" autowire="byName">
97. </bean>
98. </beans>
```

(4)、测试

Java代码




```
1. public class TestUserService{
2.
3.     private static UserService userService;
4.
5.     @BeforeClass
6.     public static void init(){
7.         ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext
-jotm.xml");
8.         userService = (UserService)app.getBean("userService");
9.     }
10.
11.     @Test
12.     public void save(){
13.         System.out.println("begin...");
14.         try{
15.             userService.saveUser();
16.         }catch(Exception e){
17.             System.out.println(e.getMessage());
18.         }
19.         System.out.println("finish...");
20.     }
21. }
```

```
20.         }
21.
22.     }
```


二、关于使用atomikos实现

(1)、数据源配置

Xml代码 

```
1.  <bean id="dataSourceA" class="com.atomikos.jdbc.SimpleDataSourceBean" init-method="init" destroy
   -method="close">
2.      <property name="uniqueResourceName">
3.          <value>${datasource.uniqueResourceName}</value>
4.      </property>
5.      <property name="xaDataSourceClassName">
6.          <value>${database.driver_class}</value>
7.      </property>
8.      <property name="xaDataSourceProperties">
9.          <value>URL=${database.url};user=${database.username};password=${database.passwo
rd}</value>
10.     </property>
11.     <property name="exclusiveConnectionMode">
12.         <value>${connection.exclusive.mode}</value>
13.     </property>
14.     <property name="connectionPoolSize">
15.         <value>${connection.pool.size}</value>
16.     </property>
17.     <property name="connectionTimeout">
18.         <value>${connection.timeout}</value>
19.     </property>
20.     <property name="validatingQuery">
21.         <value>SELECT 1</value>
22.     </property>
23. </bean>
```

(2)、事务配置

Xml代码 

```
1.  <bean id="atomikosTransactionManager" class="com.atomikos.icatch.jta.UserTransactionManager"
2.      init-method="init" destroy-method="close">
3.      <property name="forceShutdown" value="true"/>
4.  </bean>
5.
6.  <bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.UserTransactionImp">
7.      <property name="transactionTimeout" value="${transaction.timeout}"/>
8.  </bean>
9.
10. <!-- JTA事务管理器 -->
11. <bean id="springTransactionManager" class="org.springframework.transaction.jta.JtaTransactionMan
   ager">
12.     <property name="transactionManager" ref="atomikosTransactionManager"/>
13.     <property name="userTransaction" ref="atomikosUserTransaction"/>
14. </bean>
15.
16. <!-- 事务切面配置 -->
17. <aop:config>
18.     <aop:pointcut id="serviceOperation" expression="execution(* *..service*..(..))"/>
19.     <aop:advisor pointcut-ref="serviceOperation" advice-ref="txAdvice"/>
20. </aop:config>
21.
22. <!-- 通知配置 -->
23. <tx:advice id="txAdvice" transaction-manager="springTransactionManager">
24.     <tx:attributes>
```

```
25.         <tx:method name="*" rollback-for="Exception"/>
26.     </tx:attributes>
27. </tx:advice>
```

[lib.rar](#) (5.2 MB)
描述: JOTM使用包
下载次数: 124

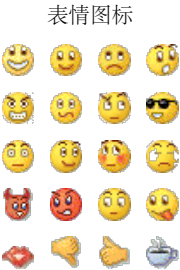


分享到: [Spring3 Annotation + Hibernate3-jpa2.0 + ...](#)

14:28 | [评论 / 浏览 \(0 / 724\)](#) | 分类: [企业架构](#) | [相关推荐](#) [▶ MORE](#)

评论

发表评论



字体颜色: 字体大小: 对齐:

提示: 选择您需要装饰的文字, 按上列按钮即可添加上相应的标签

您还没有登录, 请[登录](#)后发表评论(快捷键 Alt+S / Ctrl+Enter)