# 事务管理

## 1.1 事务回顾

### 事务简介

一组业务ABCD操作，要么全部成功，要么全部不成功。

### 事务特性：ACID

* 原子性**Atomicity**：整体 【原子性是指事务包含的所有操作要么全部成功，要么全部失败】
* 一致性**Consistency**：数据 【一个事务执行之前和执行之后都必须处于一致性状态】
* 隔离性**Isolation**：并发 【对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。】
* 持久性**Durability**：结果 【持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的】

### 隔离问题

* 脏读：一个事务读到另一个事务未提交的内容**【读取未提交内容】**

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。**本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。**

* 不可重复读：一个事务读到另一个事务已提交的内容（insert）**【读取提交内容】**

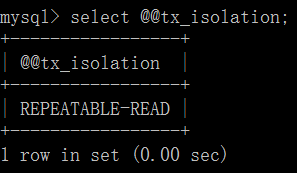
这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。

* 虚读（幻读）：一个事务读到另一个事务已提交的内容（update）

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。

* Serializable（可串行化）

这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。



### 隔离级别--解决问题

* read uncommittd，读未提交。存在3个问题。
* read committed，读已提交。解决：脏读。存在2个问题。
* repeatable read ，可重复读。解决：脏读、不可重复读。存在1个问题。
* serializable，串行化。单事务。没有问题。

|  |
| --- |
|  |

### mysql 事务操作--简单

|  |
| --- |
| ABCD 一个事务  Connection conn = null;  try{  //1 获得连接  conn = ...;  //2 开启事务  conn.setAutoCommit(false);  A  B  C  D  //3 提交事务  conn.commit();  } catche(){  //4 回滚事务  conn.rollback();  } |

### mysql 事务操作--Savepoint

|  |
| --- |
| 需求：AB（必须），CD（可选）  Connection conn = null;  Savepoint savepoint = null; //保存点，记录操作的当前位置，之后可以回滚到指定的位置。（可以回滚一部分）  try{  //1 获得连接  conn = ...;  //2 开启事务  conn.setAutoCommit(false);  A  B  savepoint = conn.setSavepoint();  C  D  //3 提交事务  conn.commit();  } catche(){  if(savepoint != null){ //CD异常  // 回滚到CD之前  conn.rollback(savepoint);  // 提交AB  conn.commit();  } else{ //AB异常  // 回滚AB  conn.rollback();  }  } |

## 1.2 Spring事务管理介绍

### 1.2.1 Spring提供的事务jar包

transaction = tx

|  |
| --- |
|  |

### 1.2.2 Jar中的三个顶级接口

|  |
| --- |
|  |
| **PlatformTransactionManager：**  平台事务管理器，spring要管理事务，必须使用事务管理器,进行事务配置时，必须**配置事务管理器** |
| **TransactionDefinition：**  事务详情（事务定义、事务属性），spring用于确定事务具体详情，  例如：隔离级别、是否只读、超时时间 等  进行事务配置时，**必须配置详情**。spring将配置项封装到该对象实例。 |
| **TransactionStatus：**  事务状态，spring用于记录当前事务运行状态。例如：是否有保存点，事务是否完成。  spring底层根据状态进行相应操作。 |

### 1.2.3 PlatformTransactionManager 事务管理器（顶级）

#### 先导入两个包

|  |
| --- |
|  |

#### 常用的两个事务管理器

|  |
| --- |
|  |

### 1.2.4 TransactionStatus 事务状态

|  |
| --- |
|  |

### 1.2.5 TransactionDefinition

|  |
| --- |
|  |

**传播行为：在两个业务之间如何共享事务**

|  |  |
| --- | --- |
| **PROPAGATION\_REQUIRED**  required , 必须 【默认值】 | 支持当前事务，A如果有事务，B将使用该事务。  如果A没有事务，B将创建一个新的事务。 |
| PROPAGATION\_SUPPORTS  supports ，支持 | 支持当前事务，A如果有事务，B将使用该事务。  如果A没有事务，B将以非事务执行。 |
| PROPAGATION\_MANDATORY  mandatory ，强制 | 支持当前事务，A如果有事务，B将使用该事务。  如果A没有事务，B将抛异常。 |
| **PROPAGATION\_REQUIRES\_NEW**  requires\_new ，必须新的 | 如果A有事务，将A的事务挂起，B创建一个新的事务  如果A没有事务，B创建一个新的事务 |
| PROPAGATION\_NOT\_SUPPORTED  not\_supported ,不支持 | 如果A有事务，将A的事务挂起，B将以非事务执行  如果A没有事务，B将以非事务执行 |
| PROPAGATION\_NEVER  never，从不 | 如果A有事务，B将抛异常  如果A没有事务，B将以非事务执行 |
| **PROPAGATION\_NESTED**  nested ，嵌套 | A和B底层采用保存点机制，形成嵌套事务。 |

掌握：PROPAGATION\_REQUIRED、PROPAGATION\_REQUIRES\_NEW、PROPAGATION\_NESTED

## 1.3 案例：转帐

### 1.3.1 环境搭建

#### 创建数据库表

|  |
| --- |
| create database spring\_day3;  use spring\_day3;  create table account(  id int primary key auto\_increment,  username varchar(50),  money int  );  insert into account(username,money) values('jack','10000');  insert into account(username,money) values('rose','10000'); |

#### 导入jar包

* 核心：4+1
* aop ： 4 (aop联盟、spring aop、aspectj规范、spring aspect)
* 数据库：2 （jdbc/tx）
* 驱动：mysql
* 连接池：c3p0

|  |
| --- |
|  |

#### Dao层

|  |
| --- |
|  |

#### Service层

|  |
| --- |
|  |

#### Spring的配置

配置c3p0数据源->dao -> service

|  |
| --- |
|  |

#### 测试转帐

|  |
| --- |
|  |

### 1.3.2 手动管理事务【了解】

**spring底层使用 TransactionTemplate 事务模板进行操作。**

**操作**

1.service 需要获得 TransactionTemplate

2.spring 配置模板，并注入给service

3.模板需要注入事务管理器

4.配置事务管理器：DataSourceTransactionManager ，需要注入DataSource

**了解底层即可，因为以后都是通过aop来配置事务**

#### 修改Service

|  |
| --- |
|  |

#### 修改spring的配置文件

|  |
| --- |
|  |

### 1.3.3 工厂bean生成代理：半自动

Spring提供 管理事务的代理工厂bean **TransactionProxyFactoryBean**

#### 修改spring配置文件

|  |
| --- |
| transactionAttributes:事务详情  prop.key ：确定哪些方法使用当前事务配置  prop.text:用于配置事务详情  格式：PROPAGATION,ISOLATION,readOnly,-Exception,+Exception  传播行为 隔离级别 是否只读 异常回滚 异常提交 |

|  |
| --- |
|  |

代理 ： 需要提供代理类接口，目标代理对象，事务管理器，事务相关配置（transer 事务的传播行为，隔离级别）

#### 测试：

|  |
| --- |
|  |

### 1.3.4 基本AOP的事务配置【掌握】

#### Spring的配置文件

|  |
| --- |
|  |
|  |

#### 测试同上

### 1.3.5 基本于注解的事务

#### spring配置

|  |
| --- |
|  |

|  |
| --- |
|  |

### 1.3.5 整合Junit

目的：少写一些代码

导入一个spring-test包

|  |
| --- |
|  |

# SSH整合

## 2.1 web 整合spring

配置tomcat加载spring的配置文件

第一步：需要添加

第二步：在web.xml配置spring的监听

|  |
| --- |
|  |
| 出现下面的错误是配置文件加载位置不对，在web.xml改成classpath目录下 |
|  |
|  |

第三步：创建Servlet获取Spring的应用上下文件ApplicationContext

|  |
| --- |
|  |

## 2.2 web整合struts+hibernate+spring

### 整合版本

**struts-2.3.33-all**

**spring-framework-3.0.2.RELEASE**

**hibernate-distribution-3.6.10.Final-dist**

### 第一步：jar包整合

#### Struts的jar包

|  |
| --- |
|  |

#### Spring的jar包

|  |
| --- |
| 基础：4+1 ， beans、core、context、expression ， commons-logging (struts已经导入)  AOP：aop联盟(aopalliance)、spring aop 、aspect规范（aspect.weaver）、spring aspect  db：jdbc、tx  测试：test  web开发：spring web  驱动：mysql  连接池：c3p0  整合hibernate：spring orm |

#### Hibernate的jar包

|  |
| --- |
| 核心包 |
|  |
| required包下的介绍    **jpa用于注解开发@Entity @Id** |

#### 整合log4j

导入 log4j...jar (struts已经导入)



整合（过渡）：slf4j-log4j12-1.7.2.jar

|  |
| --- |
|  |

#### 二级缓存

Commons-loggin.jar已经存在

|  |
| --- |
|  |

#### 整合包

spring整合hibernate： spring orm

struts 整合spring：struts2-spring-plugin-2.3.15.3.jar

#### 删除重复jar包



### 第二步：spring整合hibernate的单元测试

#### 创建表

|  |
| --- |
| create table t\_user(  id int primary key auto\_increment,  username varchar(50),  password varchar(32),  age int  ); |

#### po类和映射文件

|  |
| --- |
| **public** **class** User {  **private** Integer id;  **private** String username;  **private** String password;  **private** Integer age; |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"com.gyf.borrowsys.domain.User"* table=*"t\_user"*>  <id name=*"id"*>  <generator class=*"native"*></generator>  </id>    <property name=*"username"*></property>  <property name=*"password"*></property>  <property name=*"age"*></property>  </class>  </hibernate-mapping> |

#### dao

|  |
| --- |
|  |

#### Service

|  |
| --- |
|  |

#### hibernate.cfg.xml

|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- 1、配置数据库连接的4个参数 -->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/web\_ssh</property>  <property name=*"hibernate.connection.username"*>root</property>  <property name=*"hibernate.connection.password"*>123456</property>    <!-- 2、是否显示sql语句 -->  <property name=*"show\_sql"*>true</property>    <!-- 3、是否格式化sql语句 -->  <property name=*"format\_sql"*>true</property>    <!-- 4、Hiberante映射与DDl语句的策略 update【常用】: 如果数据库没有表，会创建表  hibernate.-->  <property name=*"hibernate.hbm2ddl.auto"*>update</property>    <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property>    <!-- 配置JavaBean与表的映射文件 -->  <mapping resource=*"com/gyf/borrowsys/domain/User.hbm.xml"*/>    </session-factory>  </hibernate-configuration> |

#### applicationContext.xml

|  |
| --- |
|  |

#### 单元测试

|  |
| --- |
|  |

#### 配置Hibrenate的事务

|  |
| --- |
|  |

#### 简化：去除hibernate.cfg.xml文件

|  |
| --- |
| <!-- dataSource -->  <bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"jdbcUrl"* value=*"jdbc:mysql:///web\_ssh"*></property>  <property name=*"user"* value=*"root"*></property>  <property name=*"password"* value=*"123456"*></property>  </bean>    <!-- sessionFactory -->  <bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>  <!--configLocation:hibernate配置文件的位置 -->  <!-- <property name="configLocation" value="classpath:hibernate.cfg.xml"></property> -->  <property name=*"dataSource"* ref=*"dataSource"*></property>  <property name=*"hibernateProperties"*>  <props>  <prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  <prop key=*"hibernate.format\_sql"*>true</prop>  <prop key=*"hibernate.hbm2ddl.auto"*>update</prop>  <prop key=*"hibernate.current\_session\_context\_class"*>thread</prop>  </props>  </property>  <!-- 映射文件位置 -->  <property name=*"mappingLocations"* value=*"classpath:com/gyf/borrowsys/domain/\*.hbm.xml"*></property>  </bean> |

### 第三步：spring整合struts

编写action类，并将其配置给spring ，spring可以注入service

编写struts.xml

表单jsp页面

#### web.xml 配置

1.确定配置文件contextConfigLocation

2.配置监听器 ContextLoaderListener

3.配置前端控制器 StrutsPrepareAndExecuteFitler

|  |
| --- |
| <!-- 加载spring的配置文件，初始化相关的bean -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>    <!-- struts的过滤器 -->  <filter>  <filter-name>struts2</filter-name>  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>  </filter>    <filter-mapping>  <filter-name>struts2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

#### action和spring配置文件

|  |
| --- |
| **action中service默认会根据名称注入**  **默认情况下框架使用的自动装配策略是name，也就是说框架会去 Spring中寻找与action属性名字相同的bean** |
|  |

#### actoin对象由spring创建

