Spring事务管理

课程介绍

课程安排

- Java事务导引
- Spring事务核心接口
- 编程式事务管理
- 声明式事务管理
- 事务最佳实践
- 课程总结

一、Java事务导引

事务简介

- 什么是事务
  - 事务是正确执行一系列的操作（或动作），使得数据库从一种状态转换成另一种状态，且保证操作全部成功，或者全部失败。
- 事务原则是什么
  - 事务必须服从ISO/IEC所制定的ACID原则。
  - ACID原则的具体内涵如下：

事务简介

- 原子性（ Atomicity ）：
  - 即不可分割性，事务要么全部被执行，要么就全部不被执行。
- 一致性（ Consistency ）：
  - 事务的执行使得数据库从一种正确状态转换成另一种正确状态。
- 隔离性（ Isolation ）：
  - 在事务正确提交之前，它可能的结果不应显示给任何其他事务。
- 持久性（ Durability ）：
  - 事务正确提交后，其结果将永久保存在数据库中。

Java事务

- Java事务的产生
  - 程序操作数据库的需要。以Java编写的程序或系统，实现ACID的操作。
- Java事务实现
  - 通过JDBC相应方法间接来实现对数据库的增、删、改、查，把事务转移到Java程序代码中进行控制；
  - 确保事务—要么全部执行成功，要么撤销不执行。
- 总结：Java事务机制和原理就是操作确保数据库操作的ACID特性。

Java事务实现模式

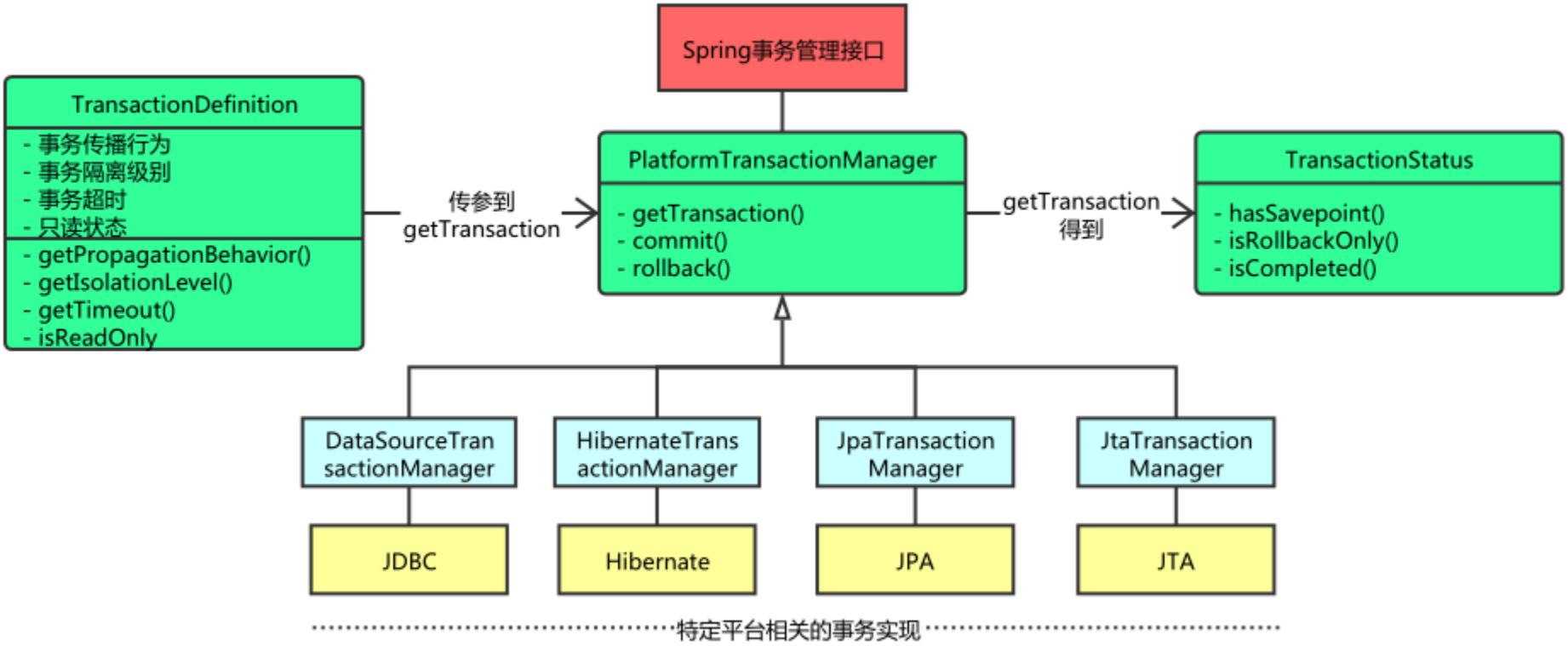
- Java事务的实现
  - 通过Java代码来实现对数据库的事务性操作。
- Java事务类型
  - JDBC事务：用 Connection 对象控制的手动模式和自动模式；
  - JTA(Java Transaction API)事务：与实现无关的，与协议无关的API；
  - 容器事务：应用服务器提供的，且大多是基于JTA完成(通常基于JNDI的，相当复杂的API实现）。

三种事务的差异

- **JDBC事务：控制的局限性在一个数据库连接内，但是其使用简单。**
- **JTA事务：功能强大，可跨越多个数据库或多DAO，使用比较复杂。**
- **容器事务：主要指的是J2EE应用服务器提供的事务管理，局限于EJB应用使用。**

二、Spring事务核心接口

事务接口架构


Spring事务管理接口
TransactionDefinition
- 事务传播行为
- 事务隔离级别
- 事务超时
- 只读状态
- getPropagationBehavior()
- getIsolationLevel()
- getTimeout()
- isReadOnly
传参到
getTransaction
PlatformTransactionManager
- getTransaction()
- commit()
- rollback()
getTransaction
得到
TransactionStatus
- hasSavepoint()
- isRollbackOnly()
- isCompleted()
DataSourceTran
sactionManager
HibernateTrans
actionManager
JpaTransaction
Manager
JtaTransaction
Manager
JDBC
Hibernate
JPA
JTA
特定平台相关的事务实现

Spring事务管理器

- JDBC事务管理器(DataSourceTransactionManager)
  - 本事务管理器是通过调用java.sql.Connection来管理事务。
  - Spring配置示例：
- Hibernate事务管理器(HibernateTransactionManager)
  - 本管理器将事务管理的职责委托给org.hibernate.Transaction对象来管理事务，而后者是从Hibernate Session中获取到的。
  - Spring配置方式：

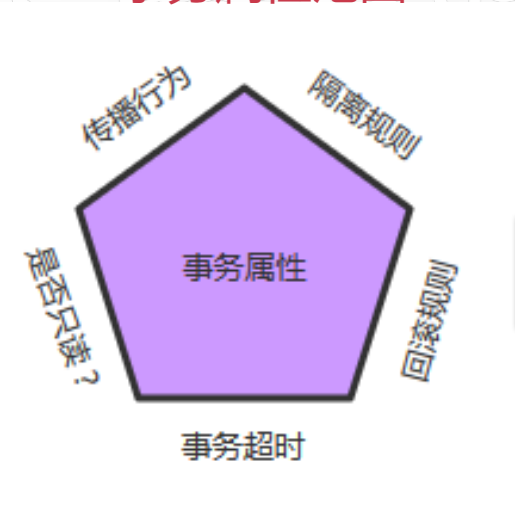
Spring事务管理器

- JPA事务管理器 (JpaTransactionManager)
  - 通过一个JPA实体管理工厂（javax.persistence.EntityManagerFactory接口的任意实现）将与由工厂所产生的JPA EntityManager合作来构建事务。
- JTA事务管理器(JtaTransactionManager)
  - 本管理器将将事务管理的责任委托给javax.transaction.UserTransaction和javax.transaction.TransactionManager对象进行事务处理。

Spring事务属性定义

Spring事务属性

事务属性范围



事务属性定义

```
public interface TransactionDefinition {
    // 返回事务的传播行为
    int getPropagationBehavior();
    // 返回事务的隔离级别，事务管理器根据它来控制
    // 另外一个事务可以看到本事务内的哪些数据
    int getIsolationLevel();
    // 返回事务必须在多少秒内完成
    int getTimeout();
    // 事务是否只读，事务管理器能够根据这个
    //  值进行优化，确保事务是只读的
    bo  n isReadOnly();
}
```

事务传播行为

- 当事务方法被另一个事务方法调用时，必须指定事务应该如何传播；
- Spring的7种传播行为：

传播行为	含义
PROPAGATION_REQUIRED	表示当前方法必须运行在事务中。如果当前事务存在，方法将会在该事务中运行。否则，会启动一个新的事务
PROPAGATION_SUPPORTS	表示当前方法不需要事务上下文，但是如果存在当前事务的话，那么该方法会在这个事务中运行
PROPAGATION_MANDATORY	表示该方法必须在事务中运行，如果当前事务不存在，则会抛出一个异常
PROPAGATION_REQUIRED_NEW	表示当前方法必须运行在它自己的事务中。一个新的事务将被启动。如果存在当前事务，在该方法执行期间，当前事务会被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager
PROPAGATION_NOT_SUPPORTED	表示该方法不应该运行在事务中。如果存在当前事务，在该方法运行期间，当前事务将被挂起。如果使用JTATransactionManager的话，则需要访问TransactionManager
PROPAGATION_NEVER	表示当前方法不应该运行在事务上下文中。如果当前正有一个事务在运行，则会抛出异常
PROPAGATION_NESTED	表示如果当前已经存在一个事务，那么该方法将会在嵌套事务中运行。嵌套的事务可以独立于当前事务进行单独地提交或回滚。如果当前事务不存在，那么其行为与PROPAGATION_REQUIRED一样。注意各厂商对这种传播行为的支持是有所差异的。可以参考资源管理器的文档来确认它们是否支持嵌套事务

事务隔离级别

- 事务隔离级别
  - 隔离级别定义了一个事务可能受其他并发事务影响的程度。
  - 隔离级别分为：

隔离级别	含义
ISOLATION_DEFAULT	使用后端数据库默认的隔离级别。
ISOLATION_READ_UNCOMMITTED	最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。
ISOLATION_READ_COMMITTED	允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。
ISOLATION_REPEATABLE_READ	对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
ISOLATION_SERIALIZABLE	最高的隔离级别，完全服从ACID的隔离级别，确保阻止脏读、不可重复读以及幻读，也是最慢的事务隔离级别，因为它通常是通过完全锁定事务相关的数据库表来实现的。

事务中注意的问题

- **事务是否只读**
  - 利用数据库事务的“只读”属性，进行特定优化处理。
- **注意：**
  - 事务的是否“只读”属性，不同的数据库厂商支持不同。
  - 通常而言：只读属性的应用要参考厂商的具体支持说明，比如，

    Oracle的“readOnly”不起作用，不影响其增删改查；

    Mysql的“readOnly”为true，只能查，增删改则出异常。

事务中注意的问题

- **事务超时**
  - 事务超时就是事务的一个定时器，在特定时间内事务如果没有执行完毕，那么就会自动回滚，而不是一直等待其结束。
- **设计事务时注意点：**
  - 为了使应用程序很好地运行，事务不能运行太长的时间。因为事务可能涉及对后端数据库的锁定，所以长时间的事务会不必要的占用数据库资源。

事务中注意的问题

- **事务回滚**
  - **默认情况下，事务只有遇到运行期异常时才会回滚，而在遇到检查型异常时不会回滚。**
- **自定义回滚策略：**
  - 声明事务在遇到特定的检查型异常时像遇到运行期异常那样回滚；
  - 声明事务遇到特定的异常不回滚，即使这些异常是运行期异常。

Spring事务状态

Spring事务状态

- 事务接口
  - 通过事务管理器获取TransactionStatus实例；
  - 控制事务在回滚或提交的时候需要应用对应的事务状态；
  - Spring事务接口：

```
//Spring事务状态接口：
//通过调用PlatformTransactionManager的getTransaction()
//获取事务状态实例
public interface TransactionStatus{
    boolean isNewTransaction(); // 是否是新的事务
    boolean hasSavepoint(); // 是否有恢复点
    void setRollbackOnly();  // 设置为只回滚
    boolean isRollbackOnly(); // 是否为只回滚
    boolean isCompleted; // 是否已完成
}
```

三、编程式事务管理

编程式事务实现方式

- 模板事务(TransactionTemplate)的方式
  - 此为Spring官方团队推荐的编程式事务管理方式；
  - 主要工具为JdbcTemplate类。
- 平台事务管理器(PlatformTransactionManager)方式
  - 类似应用JTA UserTransaction API方式，但异常处理更简洁；
  - 辅助类为：TransactionDefinition和TransactionStatus。

编程式事务实现案例

- 模板事务(TransactionTemplate)案例
  - 步骤：获取模板对象；选择事务结果类型；业务数据操作处理。
- 平台事务管理器(PlatformTransactionManager)案例
  - 步骤：获取事务管理器；获取事务属性对象；
    获取事务状态对象；创建JDBC模板对象；
    业务数据操作处理。

编程事务总结

- 需要有效的数据源，具体数据源根据实际情况创建
- 创建编程事务管理对象：
  - 事务模板（TransactionTemplate）
  - 事务管理器（PlateformTransactionManager）
- 业务逻辑处理
  - 基于JdbcTemplate完成业务处理。

四、声明式事务管理

声明式事务实现方式

- **声明式事务管理的配置类型:**
  - **5种类型：独立代理；共享代理；拦截器；tx拦截器；全注释。**
- **声明式事务管理配置实现方式:**
  - **5种类型的配置实现参考（配置代码案例）**
  - **声明式事务管理完整案例：书籍管理操作。**

五、事务管理最佳实践

编程事务管理和声明事务管理区别

- 编程式事务允许用户在代码中精确定义事务的边界;
- 声明式事务有助于用户将操作与事务规则进行解耦
  - 基于AOP交由Spring容器实现；
  - 实现关注点聚焦在业务逻辑上。
- 概括而言：
  - 编程式事务侵入到了业务代码里面，但是提供了更加详细的事务管理；而声明式事务由于基于AOP，所以既能起到事务管理的作用，又可以不影响业务代码的具体实现。

两种事务的选择

- 小型应用、事务操作少：
  - 建议**编程式事务管理**实现：TransactionTemplate；
  - 简单、显式操作、直观明显、可以设置事务名称。
- 大型应用，事务操作量多：
  - 业务复杂度高、关联性紧密，建议**声明式事务管理**实现；
  - 关注点聚焦到业务层面，实现业务和事务的解耦。

通用事务问题的解决方案

- **事务管理器类型**
  - **基于不同的数据源选择对应的事务管理器；**
  - **选择正确的PlatformTransactionManager实现类；**
  - **全局事务的选择：JtaTransactionManager。**

Spring事务管理总结

Spring事务管理总结

- 事务与Spring事务管理
- Spring事务核心接口类
- 编程式事务实现
- 声明式事务实现
- 事务的最佳实践

更多内容...

- 请关注——