#### Spring Boot

##### 之面向切面编程（AOP）-事务管理

问题描述

Spring Boot 2.2.5

JDK 8

Spring Framework 中的面向切面编程（AOP）特性可以打破传统面向过程、面向对象程序设计的代码执行流程，实现一些“神奇”的功能。事务管理是应用场景之一。

解决

1. 定义@Transactional注解

@Retention(RetentionPolicy.*RUNTIME*)  
@Target({ElementType.*METHOD*})  
public @interface Transactional {  
 @AliasFor("transactionManager")  
 String value() default "";  
  
 @AliasFor("value")  
 String transactionManager() default "";  
  
 Propagation propagation() default Propagation.*REQUIRED*;  
  
 Isolation isolation() default Isolation.*DEFAULT*;  
  
 int timeout() default -1;  
  
 boolean readOnly() default false;  
  
 Class<? extends Throwable>[] rollbackFor() default {};  
  
 String[] rollbackForClassName() default {};  
  
 Class<? extends Throwable>[] noRollbackFor() default {};  
  
 String[] noRollbackForClassName() default {};  
}

自定义注解是为了剥离Spring Tx自带的事务管理器行为。为了方便，照抄Spring Tx中的同名注解@Transactional。

1. 实现TransactionDefinition接口

public class TransactionAttributeWithRollbackRules extends DefaultTransactionAttribute {  
 Collection<Class<? extends Throwable>> rollbackFor = new ArrayList<Class<? extends Throwable>>();  
 Collection<Class<? extends Throwable>> noRollbackFor = new ArrayList<Class<? extends Throwable>>();  
   
 public void addRollbackFor(Class<? extends Throwable>[] rollbackFor) {  
 Collections.*addAll*(this.rollbackFor, rollbackFor);  
 }  
  
 public void addNoRollbackFor(Class<? extends Throwable>[] noRollbackFor) {  
 Collections.*addAll*(this.noRollbackFor, noRollbackFor);  
 }  
  
 @Override  
 public boolean rollbackOn(Throwable ex) {  
 if (ex instanceof RuntimeException || ex instanceof Error) {  
 for (Class<? extends Throwable> t : this.noRollbackFor) {  
 if (t.isAssignableFrom(ex.getClass())) {  
 return false;  
 }  
 }  
 return true;  
 } else {  
 for (Class<? extends Throwable> t : this.rollbackFor) {  
 if (t.isAssignableFrom(ex.getClass())) {  
 return true;  
 }  
 }  
 return false;  
 }  
 }  
   
   
}

1. 定义切面

@Aspect  
public abstract class AbstractTransactionManagementAspect {  
  
 @Autowired  
 private TransactionManager transactionManager;  
  
 @Pointcut  
 protected abstract void transactionalOp();  
  
 public abstract TransactionAttributeWithRollbackRules getTransactionAttribute(  
 JoinPoint jp);  
  
 @Around("transactionalOp()")  
 public Object transact(ProceedingJoinPoint pjp) throws Throwable {  
 TransactionAttributeWithRollbackRules txAttribute = getTransactionAttribute(pjp);  
 TransactionStatus status = ((DataSourceTransactionManager) transactionManager)  
 .getTransaction(txAttribute);  
 try {  
 Object ret = pjp.proceed();  
 ((DataSourceTransactionManager) transactionManager).commit(status);  
 return ret;  
 } catch (Throwable ex) {  
 if (txAttribute.rollbackOn(ex)) {  
 ((DataSourceTransactionManager) transactionManager).rollback(status);  
 } else {  
 ((DataSourceTransactionManager) transactionManager).commit(status);  
 }  
 throw ex;  
 }  
 }  
}

笔记

1. Then, by using the TransactionDefinition and TransactionStatus objects, you can initiate transactions, roll back, and commit.
2. 切面的定义同样遵循面向对象程序设计的继承特性
3. 并发场景下的@Transactional如何实现
4. 事务管理器是由Spring框架实现的，而不是数据库的一部分逻辑
5. AOP核心概念
   1. Aspect
   2. Point cut Expression（九种）
   3. Point cut Signature
   4. Advice（五种）
6. 完整代码：

<https://github.com/MariaLikesFish/spring-aop-transaction>

□