链滴

SpringBoot 入门教程 (十) | 声明式事务

作者：JavaFish
原文链接：https://ld246.com/article/1621308216760
来源网站：链滴

前言

如题，今天介绍 SpringBoot 的 声明式事务。

Spring 的事务机制

所有的数据访问技术都有事务处理机制，这些技术提供了 API 用于开启事务、提交事务来完成数据操，或者在发生错误时回滚数据。

而 Spring 的事务机制是用统一的机制来处理不同数据访问技术的事务处理，Spring 的事务机制提供一个 PlatformTransactionManager 接口，不同的数据访问技术的事务使用不同的接口实现，如下：

数据访问技术	实现
JDBC	DataSourceTransactionManager
JPA	JPATransactionManager
Hibernate	HibernateTransactionManager
JDO	JdoTransactionManager
分布式事务	JtaTransactionManager

声明式事务

Spring 支持声明式事务，即使用注解来选择需要使用事务的方法，他使用 @Transactional 注解在法上表明该方法需要事务支持。

被注解的方法在被调用时，Spring 开启一个新的事务，当方法无异常运行结束后，Spring 会提交这事务。如：

```
@Transactional
public void saveStudent(Student student){
      // 数据库操作
}
```

注意，@Transactional 注解来自于 org.springframework.transcation.annotation 包，而不是 javax transaction。

Spring 提供一个 @EnableTranscationManagement 注解在配置类上来开启声明式事务的支持。使了 @EnableTranscationManagement 后，Spring 容器会自动扫描注解 @Transactional 的方法类。@EnableTranscationManagement 的使用方式如下：

```
@Configuration
@EnableTranscationManagement
public class AppConfig{

}
```

注解事务行为

@Transactional 有如下表所示的属性来定制事务行为。

属性	含义
propagation	事务传播行为
isolation	事务隔离级别
readOnly	事务的读写性，boolean型
timeout	超时时间，int型，以秒为单位。
rollbackFor For={SQLException.class})	一组异常类，遇到时回滚。（rollbac
rollbackForCalssName ，类型为 string[]	一组异常类名，遇到回
noRollbackFor	一组异常类，遇到不回滚
norollbackForCalssName 不回滚。	一组异常类名，遇到

类级别使用 @Transactional

@Transactional 不仅可以注解在方法上，还可以注解在类上。注解在类上时意味着此类的所有 public 方法都是开启事务的。*如果类级别和方法级别同时使用了 @Transactional 注解，则使用在类级别的解会重载方法级别的注解。

SpringBoot 的事务支持

1. 自动配置的事务管理器

在使用 JDBC 作为数据访问技术时，配置定义如下：

```
@Bean
@ConditionalOnMissingBean
@ConditionalOnBean(DataSource.class)
public PlatformTransactionManager transactionManager(){
    return new DataSourceTransactionManager(this.dataSource)
}
```

在使用 JPA 作为数据访问技术时，配置定义如下：

```
@Bean
@ConditionalOnMissingBean(PlatformTransactionManager.class)
public PlatformTransactionManager transactionManager(){
    return new JpaTransactionManager()
}
```

2. 自动开启注解事务的支持

SpringBoot 专门用于配置事务的类为 org.springframework.boot.autoconfigure.transcation.TranactionAutoConfiguration，此配置类依赖于 JpaBaseConfiguration 和 DataSourceTransactionManagerAutoConfiguration。

而在 DataSourceTransactionManagerAutoConfiguration 配置里还开启了对声明式事务的支持，码如下：

```
@ConditionalOnMissingBean(AbstractTransactionManagementConfiguration.class)
@Configuration
@EnableTransactionManagement
protected static class TransactionManagementConfiguration{

}
```

所以在 SpringBoot 中，无须显式开启使用 @EnableTransactionManagement 注解。

实战

演示如何使用 Transactional 使用异常导致数据回滚与使用异常导致数据不回滚。

1. 准备工作：

SpringBoot 2.1.3

JDK 1.8

IDEA

2. pom.xml 依赖：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.3.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.nasus</groupId>
    <artifactId>transaction</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>transaction</name>
    <description>transaction Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <!-- JPA 相关 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <!-- web 启动类 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

```
        <!-- mysql 连接类 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!-- lombok 插件，简化实体代码 -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.20</version>
        </dependency>
        <!-- 单元测试 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

代码注释很清楚，没啥好说的。

3. application.yaml 配置：

```
spring:
  # \u6570\u636E\u5E93\u76F8\u5173
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&serverTi
ezone=UTC&useSSL=true
    username: root
    password: 123456
  # jpa \u76F8\u5173
  jpa:
    hibernate:
      ddl-auto: update   # ddl-auto:\u8BBE\u4E3A create \u8868\u793A\u6BCF\u6B21\u90FD\u
1CD\u65B0\u5EFA\u8868
    show-sql: true
```

4. 实体类：

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    @Id
    @GeneratedValue
    private Integer id;

    private String name;

    private Integer age;
}
```

5. dao 层

```
import com.nasus.transaction.domain.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentRepository extends JpaRepository<Student, Integer> {
}
```

6. service 层

```
import com.nasus.transaction.domain.Student;

public interface StudentService {

    Student saveStudentWithRollBack(Student student);

    Student saveStudentWithoutRollBack(Student student);

}
```

实现类：

```
import com.nasus.transaction.domain.Student;
import com.nasus.transaction.repository.StudentRepository;
import com.nasus.transaction.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class StudentServiceImpl implements StudentService {

    @Autowired
```

```
    // 直接注入 StudentRepository 的 bean
    private StudentRepository studentRepository;

    // 使用 @Transactional 注解的 rollbackFor 属性，指定特定异常时，触发回滚
    @Transactional(rollbackFor = {IllegalArgumentException.class})
    @Override
    public Student saveStudentWithRollBack(Student student) {
        Student s = studentRepository.save(student);
        if ("高斯林".equals(s.getName())){
            //硬编码，手动触发异常
            throw new IllegalArgumentException("高斯林已存在，数据将回滚");
        }
        return s;
    }

    // 使用 @Transactional 注解的 noRollbackFor 属性，指定特定异常时，不触发回滚
    @Transactional(noRollbackFor = {IllegalArgumentException.class})
    @Override
    public Student saveStudentWithoutRollBack(Student student) {
        Student s = studentRepository.save(student);
        if ("高斯林".equals(s.getName())){
            throw new IllegalArgumentException("高斯林已存在，数据将不会回滚");
        }
        return s;
    }

}
```

代码注释同样很清楚，没啥好说的。

7. controller 层

```
import com.nasus.transaction.domain.Student;
import com.nasus.transaction.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/student")
public class StudentController {

    // 注入 studentservice 的 bean
    @Autowired
    private StudentService studentService;

    // 测试回滚情况
    @PostMapping("/withRollBack")
    public Student saveStudentWithRollBack(@RequestBody Student student){
        return studentService.saveStudentWithRollBack(student);
    }
```

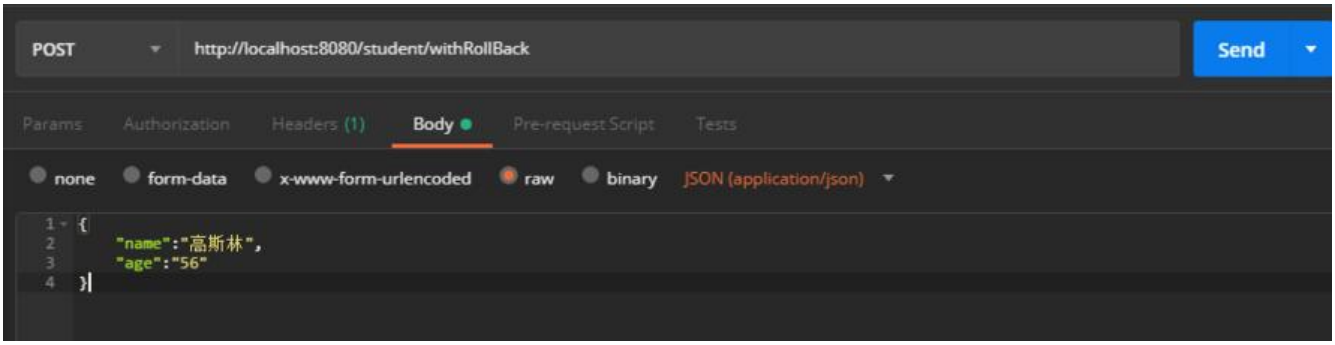
```
    // 测试不回滚情况
    @PostMapping("/withOutRollBack")
    public Student saveStudentWithoutRollBack(@RequestBody Student student){
        return studentService.saveStudentWithoutRollBack(student);
    }
}
```

Postman 测试结果

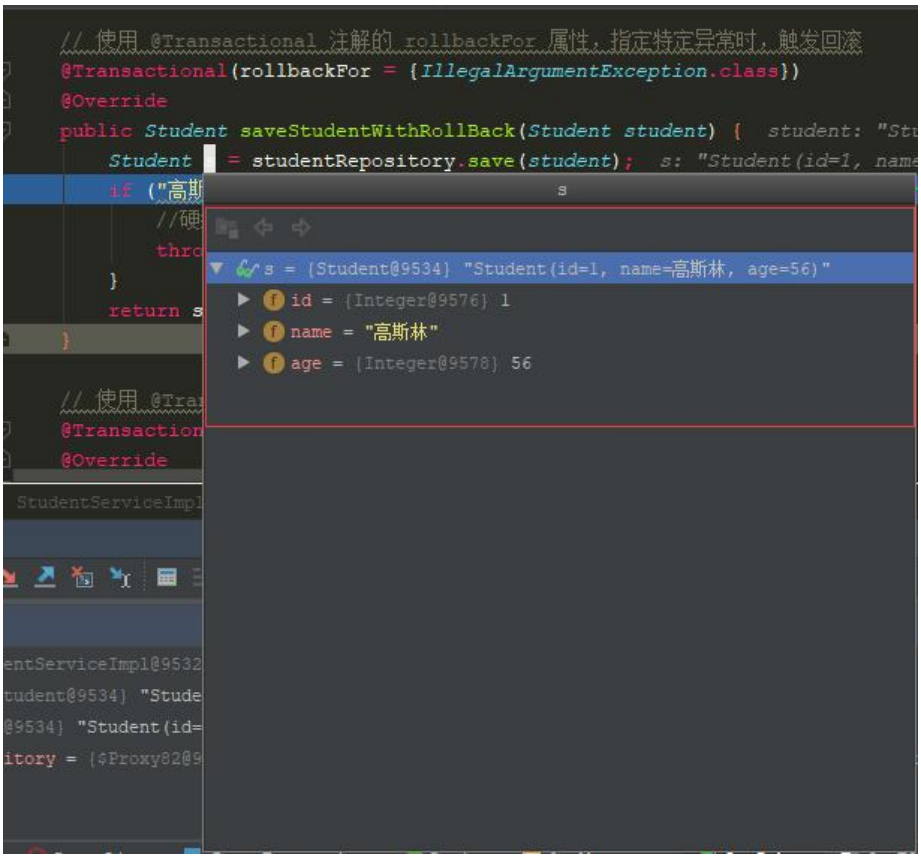
为了更清楚地理解回滚，以 debug (调试模式) 启动程序。并在 StudentServiceImpl 的 saveStudentWithRollBack 方法上打上断点。

测试前数据库结果：

1. Postman 测试回滚



debug 模式下可见数据已保存，且获得 id 为 1。：



继续执行抛出异常 IllegalArgumentException，将导致数据回滚：

```
java.lang.IllegalArgumentException: 高斯林已存在，数据将回滚
    at com.nasus.transaction.service.impl.StudentServiceImpl.saveStudentWithRollBack(StudentServiceImpl.java:36) ~[classes/:na]
    at com.nasus.transaction.service.impl.StudentServiceImpl$$FastClassBySpringCGLIB$$13c7bf42.invoke(<generated>) ~[classes/:na]
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218) ~[spring-core-5.1.5.RELEASE.jar:5.1.5.RELEASE]
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:749) ~[spring-aop-5.1.5.RELEASE.jar:5.1.5
.RELEASE]
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163) ~[spring-aop-5.1.5.RELEASE.jar:5.1.5
.RELEASE]
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:294) ~[spring-tx-5.1.5
.RELEASE.jar:5.1.5.RELEASE]
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:98) ~[spring-tx-5.1.5.RELEASE.jar:5.1.5
.RELEASE]
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186) ~[spring-aop-5.1.5.RELEASE.jar:5.1.5
.RELEASE]
```

测试后数据库结果：并无新增数据，回滚成功。

2. Postman 测试不回滚

测试前数据库结果：

遇到 IllegalArgumentException 异常数据不会回滚：

```
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into student (age, name, id) values (?, ?, ?)
2019-02-20 15:10:44.663 ERROR 13808 --- [nio-8080-exec-8] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet]
 in context with path [] threw exception [Request processing failed; nested exception is java.lang.IllegalArgumentException: 高斯林已存在，数据将不会回滚]
 with root cause

java.lang.IllegalArgumentException: 高斯林已存在，数据将不会回滚
  at com.nasus.transaction.service.impl.StudentServiceImpl.saveStudentWithoutRollBack(StudentServiceImpl.java:47) ~[classes/:na]
```

测试后数据库结果：新增数据，数据不回滚。

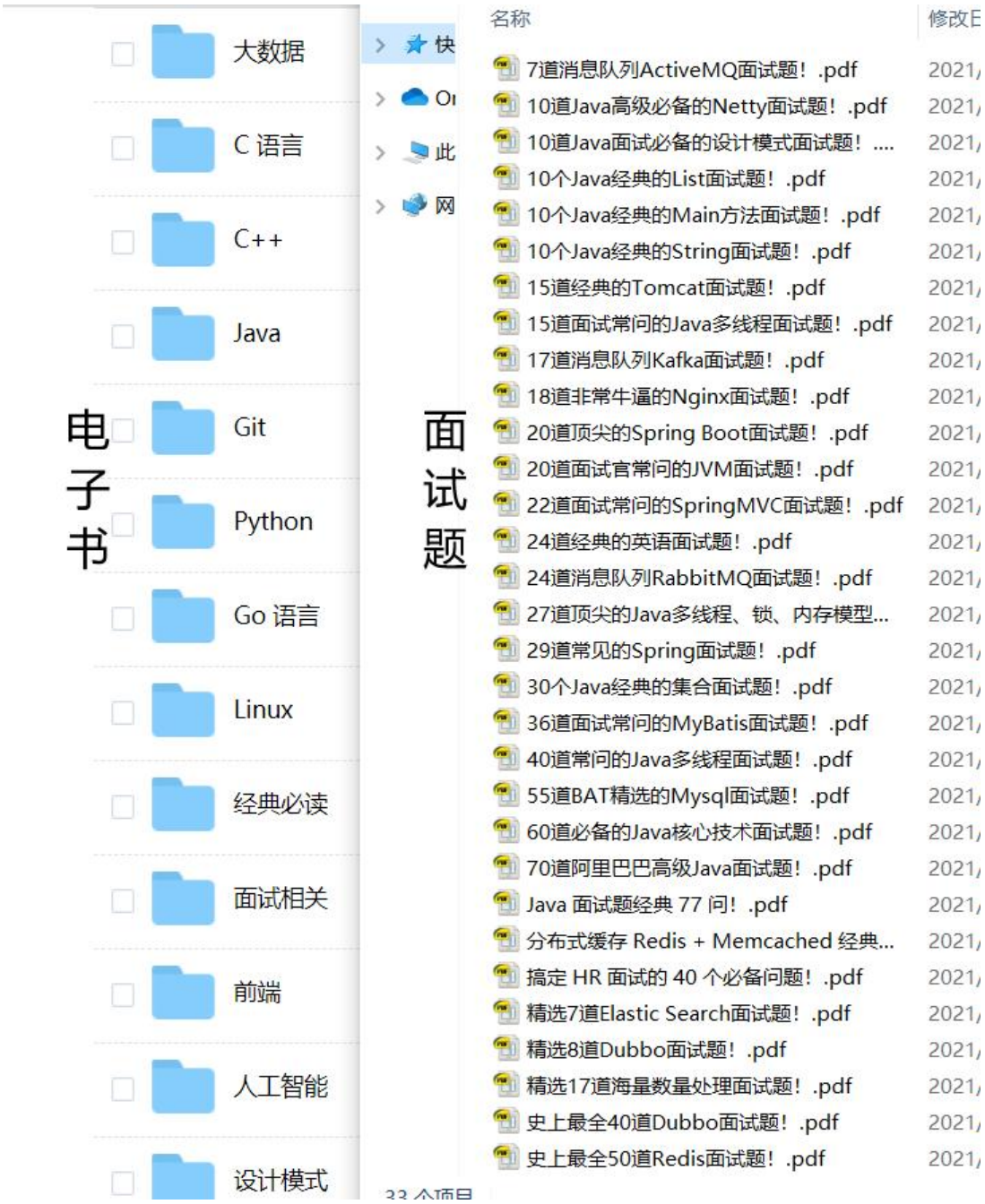
id	name	age
1	aaa	21
2	bbb	22
3	ccc	23
4	高斯林	56

源码下载

https://github.com/turoDog/Demo/tree/master/springboot_transaction_demo

后语

以上为 SpringBoot 声明式事务的教程。


电子书
大数据
C 语言
C++
Java
Git
Python
Go 语言
Linux
经典必读
面试相关
前端
人工智能
设计模式
面试题
名称
修改日
7道消息队列ActiveMQ面试题！.pdf
10道Java高级必备的Netty面试题！.pdf
10道Java面试必备的设计模式面试题！....
10个Java经典的List面试题！.pdf
10个Java经典的Main方法面试题！.pdf
10个Java经典的String面试题！.pdf
15道经典的Tomcat面试题！.pdf
15道面试常问的Java多线程面试题！.pdf
17道消息队列Kafka面试题！.pdf
18道非常牛逼的Nginx面试题！.pdf
20道顶尖的Spring Boot面试题！.pdf
20道面试官常问的JVM面试题！.pdf
22道面试常问的SpringMVC面试题！.pdf
24道经典的英语面试题！.pdf
24道消息队列RabbitMQ面试题！.pdf
27道顶尖的Java多线程、锁、内存模型...
29道常见的Spring面试题！.pdf
30个Java经典的集合面试题！.pdf
36道面试常问的MyBatis面试题！.pdf
40道常问的Java多线程面试题！.pdf
55道BAT精选的Mysql面试题！.pdf
60道必备的Java核心技术面试题！.pdf
70道阿里巴巴高级Java面试题！.pdf
Java 面试题经典 77 问！.pdf
分布式缓存 Redis + Memcached 经典...
搞定 HR 面试的 40 个必备问题！.pdf
精选7道Elastic Search面试题！.pdf
精选8道Dubbo面试题！.pdf
精选17道海量数量处理面试题！.pdf
史上最全40道Dubbo面试题！.pdf
史上最全50道Redis面试题！.pdf
2021/

JavaFish

微信扫描二维码，关注我的公众号