* @Transactional(readOnly = true/false)

1. If we don’t provide any value for readOnly in @Transactional, then the default value will be false.
2. What we write –
3. @Transactional
4. public Student getStudent(int id) {
5. Optional<Student> studentResponse = studentRepository.findById(id);
6. Student student = studentResponse.get();
7. return student;
8. }

What Spring understands –

1. @Transactional(readOnly = false)
2. public Student getStudent(int id) {
3. Optional<Student> studentResponse = studentRepository.findById(id);
4. Student student = studentResponse.get();
5. return student;
6. }
7. Generally, we use @Transactional(readOnly = true) for search or retrieval operation to make sure we can only perform the read-only operation.
8. @Transactional(readOnly = true)
9. public List<Student> getAllStudents() {
10. List<Student> studentResponse = (List<Student>) studentRepository.findAll();
11. return studentResponse;
12. }

4. We can override readOnly behavior using @Modifying  annotation. For example, suppose @Transactional annotation has been used with class level or interface level as below and we want to override readOnly behavior for one method(we don’t want to apply readOnly true for deleteOldBooks() method).

@Repository

@Transactional(readOnly = true)

public interface BookRepository extends CrudRepository<Book,Serializable> {

List<Book> findByBookName(String bookName);

@Modifying

@Transactional

@Query("delete from Book b where b.old= true")

void deleteOldBooks();

}

In the above example, we will able to delete record since we are overriding @Transactional(readOnly = true) nature using @Modifying annotation.

* If we use @Transactional(readOnly = true) to a method which is performing create or update operation then we will not have newly created or updated record into the database but we will have the response data.

Below code will not give any exception but using @Transactional(readOnly = true) while create or update operation doesn’t make any sense. Right!

@Transactional(readOnly = true)

public Student updateStudent(Student entity) {

Student studentResponse = studentRepository.save(entity);

return studentResponse;

}

* @Transactional(rollbackFor)

Spring doesn’t provide default rollback for Checked exception and its child exception(for example Exception or any custom exception which extends Exception class).

@Transactional rollbackFor example using spring boot

By

[**Rakesh**](https://www.netsurfingzone.com/author/admin/)

 -

April 15, 2019

In this post, we will see @Transactional rollbackFor example using spring boot. We will also see what is the effect of using rollbackFor, how to use in the real-time scenario.

Table of Contents

* [Basic points related to @Transactional rollbackFor attribute.](https://www.netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/#Basic_points_related_to_Transactional_rollbackFor_attribute)
* [@Transactional rollbackFor example using spring boot.](https://www.netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/#Transactional_rollbackFor_example_using_spring_boot)
* [Testing of example using postman.](https://www.netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/#Testing_of_example_using_postman)

Basic points related to @Transactional rollbackFor attribute.

We can use rollbackFor attribute as a parameter with @Transnational annotation as follows.

@Transactional(rollbackFor = { RuntimeException.class })

public void saveBook(Book book) {

// some logic

}

The rollbackFor attribute indicates Spring to rollback the transaction for the given exception(the exceptions which we have provided as the value of rollbackFor).

If we don’t provide any value for rollbackFor or we don’t mention rollbackFor as a parameter with @Transnational annotation, spring provides default rollback for RuntimeException/unchecked exception and Error. Let’s see default values for rollbackFor.

What we write.

@Transactional

public void saveBook(Book book) {

// some logic

}

What spring understand.

@Transactional(rollbackFor = { RuntimeException.class, Error.class })

public void saveBook(Book book) {

// some logic

}

In fact, Spring adds some more attributes and corresponding default values, but here we are going to talk about rollbackFor only.

Spring doesn’t provide default rollback for Checked exception and its child exception(for example Exception or any custom exception which extends Exception class).

We can have multiple Exception class values for rollbackFor.

Let’s see some sample code snippet which will behave the same.

First One.

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Autowired

private StoryService storyService;

@Transactional

public void saveBook(Book book) {

Book book1 = null;

// save book

book1 = bookRepository.save(book);

}

}

Second One.

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Autowired

private StoryService storyService;

@Transactional(rollbackFor = {RuntimeException.class})

public void saveBook(Book book) {

Book book1 = null;

// save book

book1 = bookRepository.save(book);

}

}

Note – Observe the difference in above code snippets.

@Transactional rollbackFor example using spring boot.

Prerequisites.

* JDK 1.8
* Oracle 10g
* Eclipse
* maven
* postman

Create maven project, Don’t forget to check ‘Create a simple project (skip)’click on next. Fill all details(GroupId – springtransactionexample, ArtifactId – springtransactionexample and name – springtransactionexample) and click on finish. Keep packaging as the jar.

Modify the pom.xml with below code.

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-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>

<groupId>springtransactionexample</groupId>

<artifactId>springtransactionexample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springtransactionexample</name>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.5.2.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

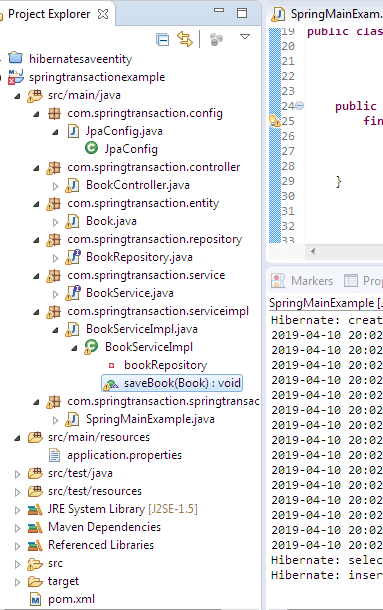
<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

</dependencies>

</project>

Create the below directory structure.

[](https://netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/attachment/rollbackfor)

Define the following classes and interfaces.

Book.java

package com.springtransaction.entity;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name = "book")

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int bookId;

@Column(name="book\_name")

private String bookName;

@Column(name="auther\_name")

private String autherName;

@Column(name="price")

private int price;

public String getAutherName() {

return autherName;

}

public void setAutherName(String autherName) {

this.autherName = autherName;

}

public int getPrice() {

return price;

}

public void setPrice(int price) {

this.price = price;

}

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public int getBookId() {

return bookId;

}

public void setBookId(int bookId) {

this.bookId = bookId;

}

}

BookService.java – interface

package com.springtransaction.service;

import java.util.List;

import org.springframework.stereotype.Component;

import com.springtransaction.entity.Book;

@Component

public interface BookService {

public void saveBook(Book book);

}

BookRepository.java – interface

package com.springtransaction.repository;

import java.io.Serializable;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

import com.springtransaction.entity.Book;

@Repository

public interface BookRepository extends CrudRepository<Book,Serializable> {

}

BookServiceImpl.java

package com.springtransaction.serviceimpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import com.springtransaction.entity.Book;

import com.springtransaction.repository.BookRepository;

import com.springtransaction.service.BookService;

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Transactional(rollbackFor = {RuntimeException.class})

public void saveBook(Book book) {

Book book1 = null;

// save book

book1 = bookRepository.save(book);

}

}

BookController.java

package com.springtransaction.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.RestController;

import com.springtransaction.entity.Book;

import com.springtransaction.service.BookService;

@RestController

@RequestMapping("/book")

public class BookController {

@Autowired

private BookService bookService;

@RequestMapping(value = "/savebook",method = RequestMethod.POST)

@ResponseBody

public Book saveBook(@RequestBody Book book) {

bookService.saveBook(book);

return book;

}

}

JpaConfig.java

package com.springtransaction.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@Configuration

@EnableJpaRepositories(basePackages = "com.springtransaction.repository")

public class JpaConfig {

}

SpringMainExample.java

package com.springtransaction.springtransaction;

import org.hibernate.SessionFactory;

import org.hibernate.jpa.HibernateEntityManagerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.annotation.Bean;

import com.springtransaction.entity.Book;

import com.springtransaction.service.BookService;

@SpringBootApplication(scanBasePackages={"com.springtransaction"})

@EntityScan("com.springtransaction.\*")

public class SpringMainExample {

public static void main(final String[] args) {

final ConfigurableApplicationContext configurableApplicationContext = SpringApplication

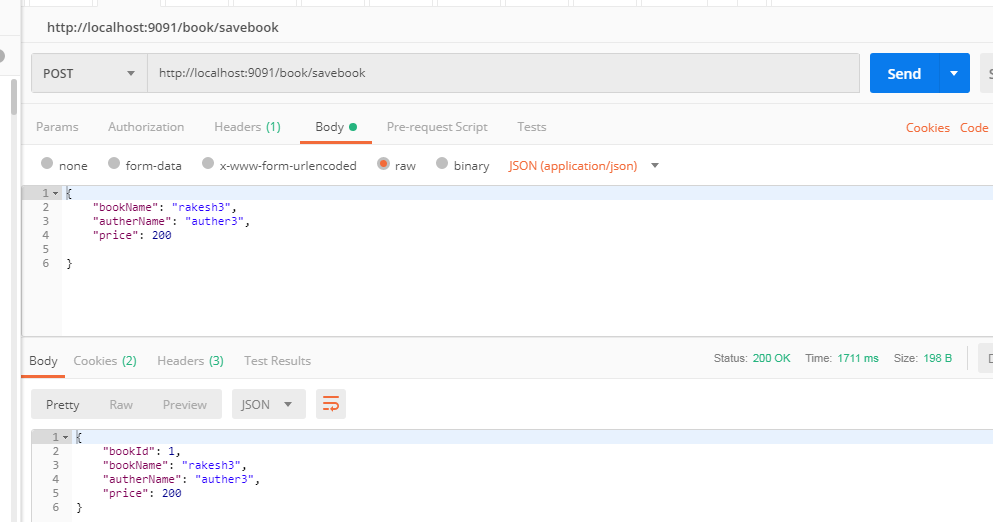
.run(SpringMainExample.class, args);

}

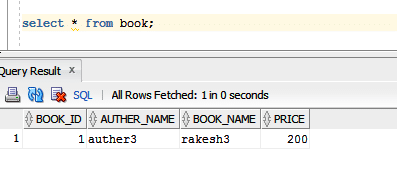
}

We are good now. Let’s run the application.

Testing of example using postman.



Let’s check the DB.



We have the book record in the database. Yes, we don’t have any problem in code, no exception occurs, so the record is inserted in database and transaction completed. What will happen if we have some problem with saveBook()? Suppose we have some logic which throws NullpointerException after saving the book. Let’s modify the BookServiceImpl.java something like below.

package com.springtransaction.serviceimpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import com.springtransaction.entity.Book;

import com.springtransaction.repository.BookRepository;

import com.springtransaction.service.BookService;

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Transactional(rollbackFor = {RuntimeException.class})

public void saveBook(Book book) {

Book book1 = null;

// save book

book1 = bookRepository.save(book);

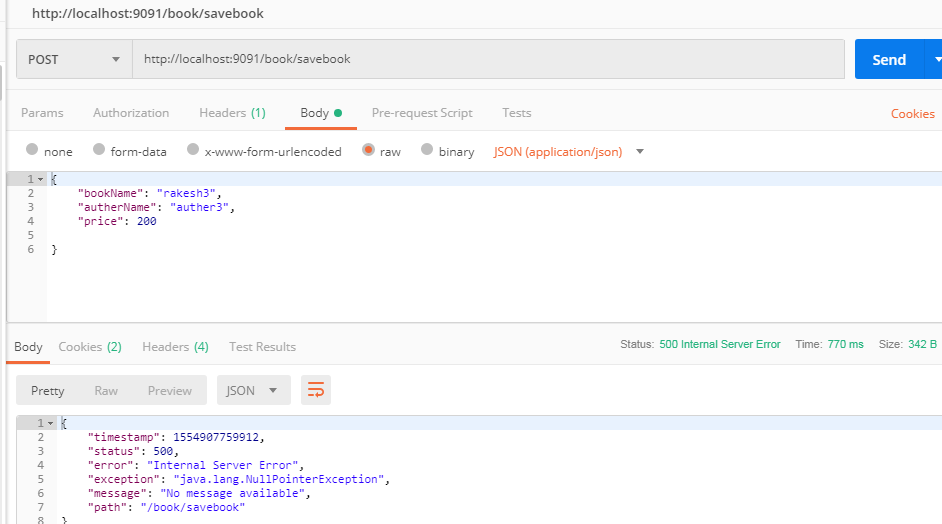
String s1 = null;

System.out.println(s1.length());

}

}

Test from the postman.

[](https://netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/attachment/rollbackfor2)

Think! In DB we will have a record for book or not? The answer is no. Since we are using @Transactional annotation and rollbackFor = {RuntimeException.class}, our transaction will rollback because we have NullpointerException in saveBook () method. Again one question may come in mind, we have defined RuntimeException.class for rollbackFor, why the transaction is getting rollback even for NullpointerException? Don’t forget NullpointerException is a child of RuntimeException. Spring will take care transaction rollback for RuntimeExeception as well as its all child exception.

Again if we don’t provide rollbackFor = {RuntimeException.class} with @Transactional annotation what will happen? We have already discussed by default we will have  rollbackFor = {RuntimeException.class}, spring will take care.

**Real-time use of rollbackFor.**

So far we have learned, Spring provides a default value for rollbackFor i.e RuntimeException and Error. Even we don’t use rollbackFor attribute our transaction will be rollbacked for any RuntimeException.

So what is the use of **rollbackFor**? Don’t forget spring is taking care of the only RuntimeException, not for Checked Exception. What will happen is some checked exception comes in our application.

If we do not use rollbackFor attribute for the checked exception, the transaction will not be rollbacked for the checked exception, which we may don’t want. If we want to rollback the transaction for all kind of exception including checked and unchecked we need to specify the rollbackFor attribute something like below.

**rollbackFor = {Exception.class}**

Now we are saying to spring, hey spring if you see any exception checked or unchecked please rollback the transaction(don’t save the record in DB).

Let’s see an example where we will use rolbackFor attribute with a value RuntimeException.class and will throw checked exception intentionally. Let’s see what happens.

Modify the classes and interface as below.

BookServiceImpl.java

package com.springtransaction.serviceimpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import com.springtransaction.entity.Book;

import com.springtransaction.repository.BookRepository;

import com.springtransaction.service.BookService;

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Transactional(rollbackFor = {RuntimeException.class})

public void saveBook(Book book) throws Exception{

Book book1 = null;

// save book

book1 = bookRepository.save(book);

throw new Exception("this is checked exception throwing intentinally");

}

}

BookService.java

package com.springtransaction.service;

import java.util.List;

import org.springframework.stereotype.Component;

import com.springtransaction.entity.Book;

@Component

public interface BookService {

public void saveBook(Book book) throws Exception;

}

BookController.java

package com.springtransaction.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.RestController;

import com.springtransaction.entity.Book;

import com.springtransaction.service.BookService;

@RestController

@RequestMapping("/book")

public class BookController {

@Autowired

private BookService bookService;

@RequestMapping(value = "/savebook",method = RequestMethod.POST)

@ResponseBody

public Book saveBook(@RequestBody Book book) throws Exception{

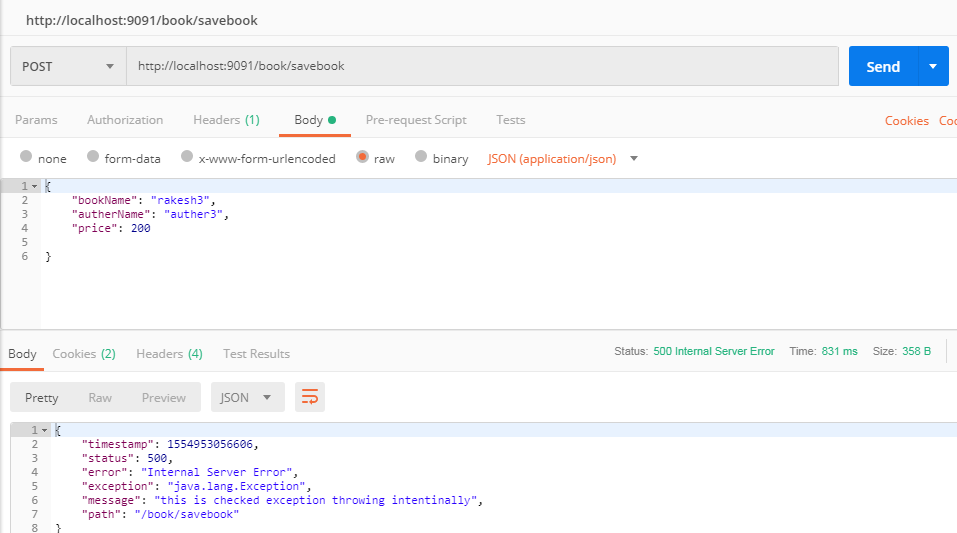
bookService.saveBook(book);

return book;

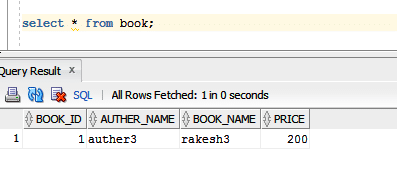
}

}

If you run above changes and test from the postman, even we are using @Transactional annotation with rollbackFor = RuntimeException.class, we will have one book record in DB. Why? Because we are throwing checked exception in BookServiceImpl.java but we are saying to spring hey spring rollback the transaction for only RuntimeException/unchecked exception not for the checked exception.

[](https://netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/attachment/rollbackfor3)

Let’s check the DB.



Let’s modify the BookServiceImpl.java something like below.

**rollbackFor = {Exception.class}**

BookServiceImpl.java

package com.springtransaction.serviceimpl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Isolation;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import com.springtransaction.entity.Book;

import com.springtransaction.repository.BookRepository;

import com.springtransaction.service.BookService;

@Service

public class BookServiceImpl implements BookService {

@Autowired

private BookRepository bookRepository;

@Transactional(rollbackFor = {Exception.class})

public void saveBook(Book book) throws Exception{

Book book1 = null;

// save book

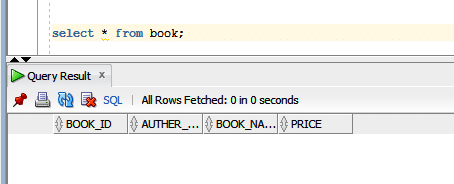
book1 = bookRepository.save(book);

throw new Exception("this is checked exception throwing intentinally");

}

}

test from the postman and see the DB we will not any record.

[](https://netsurfingzone.com/spring/transactional-rollbackfor-example-using-spring-boot/attachment/rollbackfor4)

That is the expected result. We are telling to spring just rollback the transaction for any exception.

That’s all about @Transactional rollbackFor example using spring boot.