

Jul 26, 2015

The magic of @Transactional and its performance

I was planning to write a small article about the `rollbackFor` parameter for `@Transactional` annotation. But when I was preparing the code examples, I decided to do some benchmarks. Let's start with the brief annotation description.

@Transactional

`@Transactional` is annotation which allows you to work with databases' transactions in the declarative way.

```
@Transactional
public void operation() {
    ... //Code which works with database
}
```

This is pretty convenient - you don't need to think about direct transactions management and exceptions handling. Everything will be done automatically in the proxy class. This image (from [spring documentation](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html) (<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html>)) show how class hierarchy looks like:



So, it looks pretty clear. [JavaDoc](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html) (<http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html>) about this annotation is always there and gives enough information to get started. But there's one unclear thing with exception handling - here's an example:

```
@Transactional
public void operation() {
    entityManager.persist(new User("Dima"));
    throw new Exception();
}
```

Will it be committed? - Yes, it will!



The documentation says:

Although EJB container default behavior automatically rolls back the transaction on a system exception (usually a runtime exception), EJB CMT does not roll back the transaction automatically on an application exception (that is, a checked exception other than `java.rmi.RemoteException`). While the Spring default behavior for declarative transaction management follows EJB convention (roll back is automatic only on unchecked exceptions), it is often useful to customize this behavior.

[Spring Documentation \(http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html#transaction-declarative\)](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html#transaction-declarative).

Okaaay... So, if you expect that checked exception will be thrown in your code, you better use `rollbackFor` in this way:

```
@Transactional(rollbackFor = Exception.class)
public void operation() {
    entityManager.persist(new User("Dima"));
    throw new Exception();
}
```

Benchmarking

Now it's clear. Let's check how much we pay for the AOP magic. I wrote 4 simple methods to compare performance of **insert** operations using different ways to manage transactions and ORM layers:

```
@Service
public class UserService {

    @Autowired
    PlatformTransactionManager transactionManager;

    @Autowired
    UserRepository repository;

    @Autowired
    @Qualifier("count")
    Integer count;

    @Autowired
    EntityManagerFactory entityManagerFactory;

    @Autowired
    DataSource dataSource;

    @Transactional
    public void transactional() {
        IntStream.range(0, count).forEach(i -> repository.save(new User("name")));
    }

    public void transactionManager() {
        DefaultTransactionDefinition def = new DefaultTransactionDefinition();
        def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
        TransactionStatus transactionStatus = null;
        try {
            transactionStatus = transactionManager.getTransaction(def);
            IntStream.range(0, count).forEach(i -> repository.save(new User("name")));
            transactionManager.commit(transactionStatus);
        } catch (Exception e) {
            if (transactionStatus != null) {
                transactionManager.rollback(transactionStatus);
            }
            throw new RuntimeException(e);
        }
    }

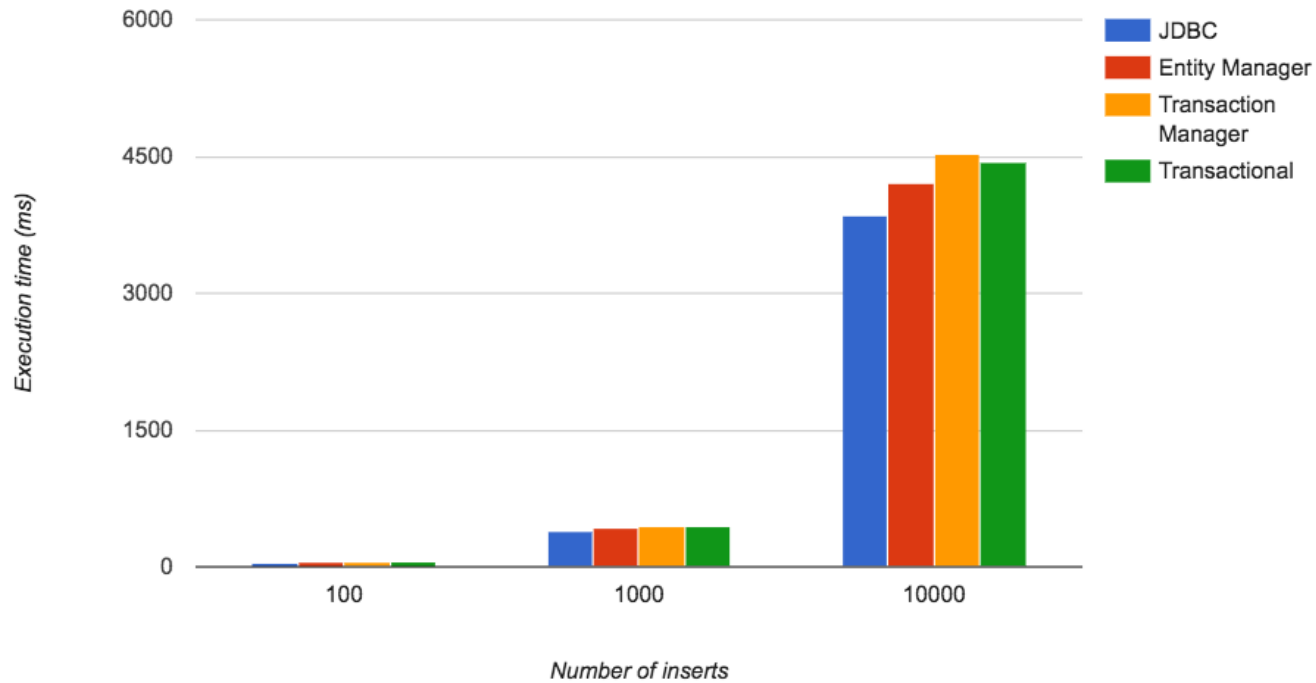
    public void jdbc() {
        try (Connection connection = dataSource.getConnection()) {
            connection.setAutoCommit(false);
            try (PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO User (name) VALUES (?)")) {

                IntStream.range(0, count).forEach(i -> {
                    try {
                        preparedStatement.setString(1, "name");
                        preparedStatement.executeUpdate();
                    } catch (SQLException e) {
                        throw new RuntimeException(e);
                    }
                });
            }
            connection.commit();
        }
    }
}
```

```
        } catch (Exception e1) {
            connection.rollback();
            throw new RuntimeException(e1);
        }
    } catch (Exception e1) {
        throw new RuntimeException(e1);
    }
}

public void entityManager() {
    EntityManager entityManager = entityManagerFactory.createEntityManager();
    try {
        entityManager.getTransaction().begin();
        IntStream.range(0, count).forEach(i -> entityManager.persist(new User("name")));
        entityManager.getTransaction().commit();
    } catch (Exception e) {
        entityManager.getTransaction().rollback();
    } finally {
        entityManager.close();
    }
}
}
```

It runs in Spring Boot project with the default configuration. You can check the source code of this benchmark out on [github](https://github.com/dimafeng/dimafeng-examples) (<https://github.com/dimafeng/dimafeng-examples>) (`gradlew transactional:run`). That's what I've got:



This chart shows the best execution time from 20 attempts for 100, 1000, and 10000 inserts. As you see, **@Transactional + Spring Data** works almost the same as working with **EntityManager** directly. For 10000 inserts, the time difference between **@Transactional** and **EntityManager** was 250ms - it's 4% of total execution time, and it's much less than the difference between pure **JDBC** and **EntityManager**.

[@_ \(mailto:?\)](mailto:dimafeng.com)
 subject=The%20magic%20of%20%40Transactional%20and%20its%20performance&body=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransac

[_ \(https://twitter.com/share?\)](https://twitter.com/share?url=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F&text=The%20magic%20of%20%40Transactional%20and%20its%20performance)
 url=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F&text=The%20magic%20of%20%40Transactional%20and%20its%20performance

[f \(https://facebook.com/sharer/sharer.php?\)](https://facebook.com/sharer/sharer.php?u=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F)
 u=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F)



(<https://plus.google.com/share?url=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F>)



(<https://www.linkedin.com/shareArticle?mini=true&url=http%3A%2F%2Fdimafeng.com%2F2015%2F07%2F26%2Ftransactional%2F>)

Related Posts

01/29 Scala testing with a human face (<http://dimafeng.com/2016/01/29/scala-spring-test/>)

01/02 Scala with a human face (<http://dimafeng.com/2016/01/02/scala-spring/>)

11/27 Dynamic bean definition for automatic FilterRegistrationBeanFactory unregistration (<http://dimafeng.com/2015/11/27/dynamic-bean-definition/>)

08/29 Spring @Configuration vs @Component (http://dimafeng.com/2015/08/29/spring-configuration_vs_component/)

08/16 CGLIB: signer information does not match signer information of other classes (<http://dimafeng.com/2015/08/16/cglib/>)

07/22 WebApplicationContext in the stand-alone application (<http://dimafeng.com/2015/07/22/web-app-context/>)

5 Comments dimafeng's blog

Login

Recommend Tweet Share

Sort by Best



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Ravish Sinha • 9 months ago

good article.

^ | v • Reply • Share ›



Empty • 2 years ago

Nice article!

But will it rollback changes of in-memory structures like queues or maps that does not rely on transaction manager? Or it works only with database transactions?

^ | v • Reply • Share ›



dimafeng Mod → Empty • 2 years ago

Thank you!

In case of rollback, only the DB transaction will be rolled back and all Java-managed objects will be kept changed.

^ | v • Reply • Share ›



heldev • 2 years ago

Hi, thank you for the article!

Such a big difference between JDBC and the rest seems a bit unexpected.
Is there any chance that it's caused by the missed execution statement (e.g. preparedStatement.execute()) in the example listing?

^ | v • Reply • Share ›



dimafeng Mod ↗ heldev • 2 years ago


Thank you for your comment!
You are exactly right! Just a stupid mistake I didn't pay attention to. Now this looks much better.

^ | v • Reply • Share ›

ALSO ON DIMAFENG'S BLOG

Linux command line cheat sheet

4 comments • 3 years ago

 Dmitry Drozdov — It's better to use tail -F instead of tail -f. That way it would handle log rotation.


Vim essentials (Part 1)

2 comments • 4 years ago

 dimafeng — Thank you :)


5 drawbacks to liking SBT



9 comments • 2 years ago

 Laurent B. — Felt exactly the same until I started writing SBT builds in plain scala (project/build.scala). Things suddenly went fine as ...

Integration testing using docker

6 comments • 3 years ago

 dimafeng — Hi Itay, A basic example would look like this: class ComposeSpec extends FlatSpec with ForAllTestContainer { override val container = ...

 Subscribe  Add Disqus to your site Add Disqus Add  Disqus' Privacy Policy Privacy Policy Privacy Policy

Software Engineer with proven expertise in object-oriented analysis and design and exceptional record overseeing all facets of Software Development Life Cycle, from analysis and design to implementation and maintenance

