Table of Contents

TransactionAwareDataSourceProxy

@Transactional

@Transactional

@Transactional

PROPAGATION_REQUIRED

PROPAGATION_REQUIRES_NEW

PROPAGATION_NESTED

@Transactional

TransactionTemplate

TransactionOperator

TransactionManager

PlatformTransactionManager

ReactiveTransactionManager

DataSource

JdbcTemplate

SELECT

INSERT UPDATE DELETE JdbcTemplate

JdbcTemplate

JdbcTemplate

NamedParameterJdbcTemplate

SQLExceptionTranslator

DataSource

DataSourceUtils

SmartDataSource

AbstractDataSource

SingleConnectionDataSource

DriverManagerDataSource

TransactionAwareDataSourceProxy

DataSourceTransactionManager

JdbcTemplate

SimpleJdbc

SimpleJdbcInsert

SimpleJdbcInsert

SimpleJdbcInsert

SqlParameterSource

SimpleJdbcCall

SimpleJdbcCall

SqlParameters

SimpleJdbcCall

ResultSet

SimpleJdbcCall

SqlQuery

MappingSqlQuery

SqlUpdate

StoredProcedure

DataSource

SessionFactory

LocalEntityManagerFactoryBean

LocalContainerEntityManagerFactoryBean

EntityManagerFactory EntityManager

JpaDialect JpaVendorAdapter

Marshaller Unmarshaller

Marshaller

Unmarshaller

XmlMappingException

Marshaller Unmarshaller

Jaxb2Marshaller

JibxMarshaller

XStreamMarshaller

tx

jdbc

Chapter 1. Transaction Management

`DataSource`

1.1. Advantages of the Spring Framework's Transaction Support Model

1.1.1. Global Transactions

`UserTransaction`

1.1.2. Local Transactions

1.1.3. Spring Framework’s Consistent Programming Model

Do you need an application server for transaction management?

1.2. Understanding the Spring Framework Transaction Abstraction

`TransactionManager` `org.springframework.transaction.PlatformTransactionManager` `org.springframework.transaction.ReactiveTransactionManager`

`PlatformTransactionManager`

Java

```
public interface PlatformTransactionManager extends TransactionManager {

    TransactionStatus getTransaction(TransactionDefinition definition) throws
TransactionException;

    void commit(TransactionStatus status) throws TransactionException;

    void rollback(TransactionStatus status) throws TransactionException;
}
```

Kotlin

```
interface PlatformTransactionManager : TransactionManager {

    @Throws(TransactionException::class)
    fun getTransaction(definition: TransactionDefinition): TransactionStatus

    @Throws(TransactionException::class)
    fun commit(status: TransactionStatus)

    @Throws(TransactionException::class)
    fun rollback(status: TransactionStatus)
}
```

PlatformTransactionManager

PlatformTransactionManager

TransactionException

PlatformTransactionManager

java.lang.RuntimeException

TransactionException

forced

getTransaction(..)

TransactionStatus

TransactionDefinition

TransactionStatus

TransactionStatus

org.springframework.transaction.ReactiveTransactionManager

Java

```
public interface ReactiveTransactionManager extends TransactionManager {

    Mono<ReactiveTransaction> getReactiveTransaction(TransactionDefinition definition)
throws TransactionException;

    Mono<Void> commit(ReactiveTransaction status) throws TransactionException;

    Mono<Void> rollback(ReactiveTransaction status) throws TransactionException;
}
```

Kotlin

```
interface ReactiveTransactionManager : TransactionManager {

    @Throws(TransactionException::class)
    fun getReactiveTransaction(definition: TransactionDefinition):
Mono<ReactiveTransaction>

    @Throws(TransactionException::class)
    fun commit(status: ReactiveTransaction): Mono<Void>

    @Throws(TransactionException::class)
    fun rollback(status: ReactiveTransaction): Mono<Void>
}
```

`ReactiveTransactionManager`

`TransactionDefinition`

`TransactionStatus`

`TransactionStatus`

Java

```
public interface TransactionStatus extends TransactionExecution, SavepointManager,
Flushable {

    @Override
    boolean isNewTransaction();

    boolean hasSavepoint();

    @Override
    void setRollbackOnly();

    @Override
    boolean isRollbackOnly();

    void flush();

    @Override
    boolean isCompleted();
}
```

Kotlin

```
interface TransactionStatus : TransactionExecution, SavepointManager, Flushable {

    override fun isNewTransaction(): Boolean

    fun hasSavepoint(): Boolean

    override fun setRollbackOnly()

    override fun isRollbackOnly(): Boolean

    fun flush()

    override fun isCompleted(): Boolean
}
```

`TransactionManager`

TransactionManager

PlatformTransactionManager

DataSource

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>
```

PlatformTransactionManager DataSource

```
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

DataSource

JtaTransactionManager

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/jee
        https://www.springframework.org/schema/jee/spring-jee.xsd">

    <jee:jndi-lookup id="dataSource" jndi-name="jdbc/jpetstore"/>

    <bean id="txManager"
class="org.springframework.transaction.jta.JtaTransactionManager" />

    <!-- other <bean/> definitions here -->

</beans>
```

JtaTransactionManager DataSource

dataSource <jndi-lookup/> jee

LocalSessionFactoryBean Session

DataSource

DataSource

txManager HibernateTransactionManager DataSourceTransactionManager DataSource HibernateTransactionManager SessionFactory sessionFactory txManager

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list>

<value>org/springframework/samples/petclinic/hibernate/petclinic.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <value>
            hibernate.dialect=${hibernate.dialect}
        </value>
    </property>
</bean>

<bean id="txManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

JtaTransactionManager

```
<bean id="txManager"
class="org.springframework.transaction.jta.JtaTransactionManager"/>
```

1.3. Synchronizing Resources with Transactions

`DataSourceTransactionManager` `DataSource` `HibernateTransactionManager` `SessionFactory`

`TransactionManager`

1.3.1. High-level Synchronization Approach

`JdbcTemplate`

1.3.2. Low-level Synchronization Approach

`DataSourceUtils` `EntityManagerFactoryUtils` `SessionFactoryUtils`

`getConnection()` `DataSource` `org.springframework.jdbc.datasource.DataSourceUtils`

```
Connection conn = DataSourceUtils.getConnection(dataSource);
```

SQLException
CannotGetJdbcConnectionException
DataAccessException
SQLException

DataSourceUtils

JdbcTemplate jdbc.object

1.3.3. TransactionAwareDataSourceProxy

TransactionAwareDataSourceProxy
DataSource DataSource
DataSource

DataSource

1.4. Declarative transaction management

setRollbackOnly()

setRollbackOnly()

setRollbackOnly() TransactionStatus MyApplicationException

java.rmi.RemoteException

1.4.1. Understanding the Spring Framework's Declarative Transaction Implementation

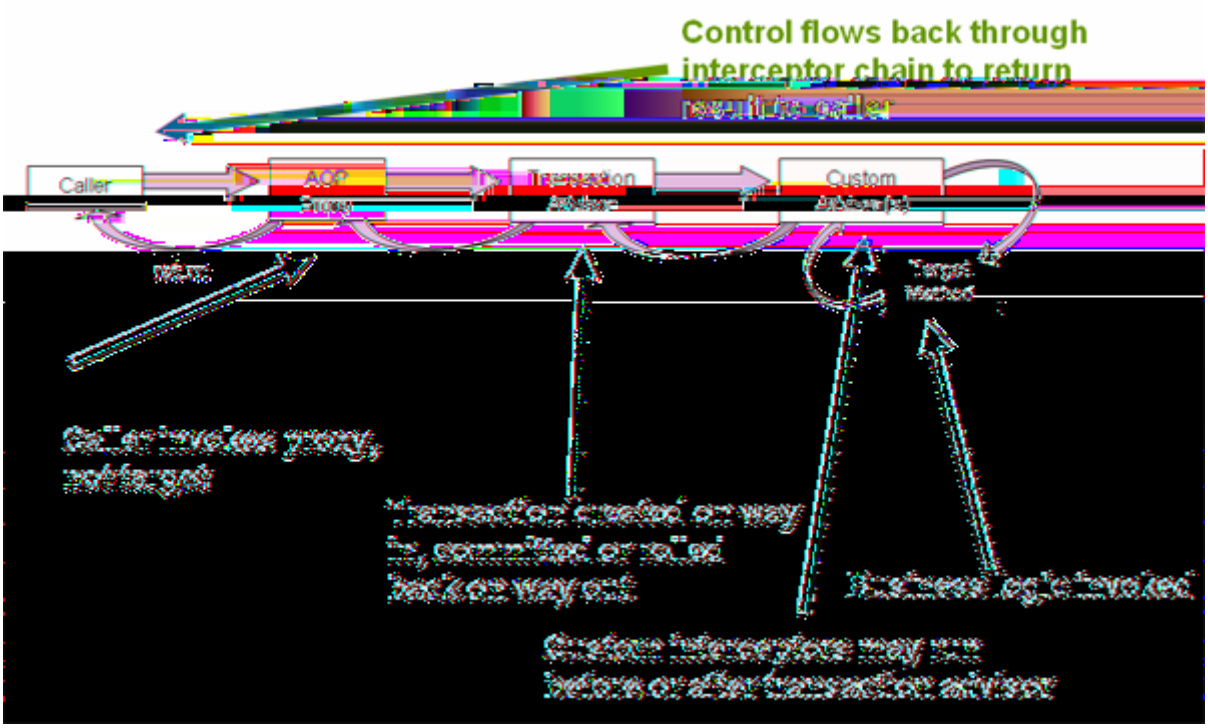
@Transactional @EnableTransactionManagement

TransactionInterceptor TransactionManager

TransactionInterceptor

Publisher Flow void

PlatformTransactionManager ReactiveTransactionManager



1.4.2. Example of Declarative Transaction Implementation

Foo Bar

DefaultFooService UnsupportedOperationException

UnsupportedOperationException FooService

Java

```
// the service interface that we want to make transactional

package x.y.service;

public interface FooService {

    Foo getFoo(String fooName);

    Foo getFoo(String fooName, String barName);

    void insertFoo(Foo foo);

    void updateFoo(Foo foo);

}
```

Kotlin

```
// the service interface that we want to make transactional

package x.y.service

interface FooService {

    fun getFoo(fooName: String): Foo

    fun getFoo(fooName: String, barName: String): Foo

    fun insertFoo(foo: Foo)

    fun updateFoo(foo: Foo)
}
```

Java

```
package x.y.service;

public class DefaultFooService implements FooService {

    @Override
    public Foo getFoo(String fooName) {
        // ...
    }

    @Override
    public Foo getFoo(String fooName, String barName) {
        // ...
    }

    @Override
    public void insertFoo(Foo foo) {
        // ...
    }

    @Override
    public void updateFoo(Foo foo) {
        // ...
    }
}
```

Kotlin

```
package x.y.service

class DefaultFooService : FooService {

    override fun getFoo(fooName: String): Foo {
        // ...
    }

    override fun getFoo(fooName: String, barName: String): Foo {
        // ...
    }

    override fun insertFoo(foo: Foo) {
        // ...
    }

    override fun updateFoo(foo: Foo) {
        // ...
    }
}
```

FooService getFoo(String) getFoo(String, String) insertFoo(Foo) updateFoo(Foo)

```
<!-- from the file 'context.xml' -->
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- this is the service object that we want to make transactional -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- the transactional advice (what 'happens'; see the <aop:advisor/> bean below)
-->
    <tx:advice id="txAdvice" transaction-manager="txManager">
        <!-- the transactional semantics... -->
        <tx:attributes>
```

```
            <!-- all methods starting with 'get' are read-only -->
            <tx:method name="get*" read-only="true"/>
            <!-- other methods use the default transaction settings (see below) -->
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>

    <!-- ensure that the above transactional advice runs for any execution
        of an operation defined by the FooService interface -->
    <aop:config>
        <aop:pointcut id="fooServiceOperation" expression="execution(*
x.y.service.FooService.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
    </aop:config>

    <!-- don't forget the DataSource -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>

    <!-- similarly, don't forget the TransactionManager -->
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- other <bean/> definitions here -->

</beans>
```

fooService
<tx:advice/> <tx:advice/> get

transaction-manager <tx:advice/>
TransactionManager txManager

transaction-manager
<tx:advice/> TransactionManager
transactionManager TransactionManager
transaction-manager

<aop:config/> txAdvice

FooService fooServiceOperation
txAdvice
fooServiceOperation txAdvice

<aop:pointcut/>

```
<aop:config>
    <aop:pointcut id="fooServiceMethods" expression="execution(*
x.y.service.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceMethods"/>
</aop:config>
```

x.y.service

fooService

Java

```
public final class Boot {

    public static void main(final String[] args) throws Exception {
        ApplicationContext ctx = new ClassPathXmlApplicationContext("context.xml",
Boot.class);
        FooService fooService = (FooService) ctx.getBean("fooService");
        fooService.insertFoo (new Foo());
    }
}
```

Kotlin

```
import org.springframework.beans.factory.getBean

fun main() {
    val ctx = ClassPathXmlApplicationContext("context.xml")
    val fooService = ctx.getBean<FooService>("fooService")
    fooService.insertFoo(Foo())
}
```

UnsupportedOperationException insertFoo(..) DefaultFooService

```
<!-- the Spring container is starting up... -->
[AspectJInvocationContextExposingAdvisorAutoProxyCreator] - Creating implicit proxy
for bean 'fooService' with 0 common interceptors and 1 specific interceptors

<!-- the DefaultFooService is actually proxied -->
[JdkDynamicAopProxy] - Creating JDK dynamic proxy for [x.y.service.DefaultFooService]

<!-- ... the insertFoo(..) method is now being invoked on the proxy -->
[TransactionInterceptor] - Getting transaction for x.y.service.FooService.insertFoo

<!-- the transactional advice kicks in here... -->
[DataSourceTransactionManager] - Creating new transaction with name
[x.y.service.FooService.insertFoo]
[DataSourceTransactionManager] - Acquired Connection
[org.apache.commons.dbcp.PoolableConnection@a53de4] for JDBC transaction

<!-- the insertFoo(..) method from DefaultFooService throws an exception... -->
[RuleBasedTransactionAttribute] - Applying rules to determine whether transaction
should rollback on java.lang.UnsupportedOperationException
[TransactionInterceptor] - Invoking rollback for transaction on
x.y.service.FooService.insertFoo due to throwable
[java.lang.UnsupportedOperationException]

<!-- and the transaction is rolled back (by default, RuntimeException instances cause
rollback) -->
[DataSourceTransactionManager] - Rolling back JDBC transaction on Connection
[org.apache.commons.dbcp.PoolableConnection@a53de4]
[DataSourceTransactionManager] - Releasing JDBC Connection after transaction
[DataSourceUtils] - Returning JDBC Connection to DataSource

Exception in thread "main" java.lang.UnsupportedOperationException at
x.y.service.DefaultFooService.insertFoo(DefaultFooService.java:14)
<!-- AOP infrastructure stack trace elements removed for clarity -->
at $Proxy0.insertFoo(Unknown Source)
at Boot.main(Boot.java:11)
```

`ReactiveAdapterRegistry`

`FooService`

Java

```
// the reactive service interface that we want to make transactional

package x.y.service;

public interface FooService {

    Flux<Foo> getFoo(String fooName);

    Publisher<Foo> getFoo(String fooName, String barName);

    Mono<Void> insertFoo(Foo foo);

    Mono<Void> updateFoo(Foo foo);

}
```

Kotlin

```
// the reactive service interface that we want to make transactional

package x.y.service

interface FooService {

    fun getFoo(fooName: String): Flow<Foo>

    fun getFoo(fooName: String, barName: String): Publisher<Foo>

    fun insertFoo(foo: Foo) : Mono<Void>

    fun updateFoo(foo: Foo) : Mono<Void>
}
```

Java

```
package x.y.service;

public class DefaultFooService implements FooService {

    @Override
    public Flux<Foo> getFoo(String fooName) {
        // ...
    }

    @Override
    public Publisher<Foo> getFoo(String fooName, String barName) {
        // ...
    }

    @Override
    public Mono<Void> insertFoo(Foo foo) {
        // ...
    }

    @Override
    public Mono<Void> updateFoo(Foo foo) {
        // ...
    }
}
```

Kotlin

```
package x.y.service

class DefaultFooService : FooService {

    override fun getFoo(fooName: String): Flow<Foo> {
        // ...
    }

    override fun getFoo(fooName: String, barName: String): Publisher<Foo> {
        // ...
    }

    override fun insertFoo(foo: Foo): Mono<Void> {
        // ...
    }

    override fun updateFoo(foo: Foo): Mono<Void> {
        // ...
    }
}
```

TransactionInterceptor

Publisher

1.4.3. Rolling Back a Declarative Transaction

Exception

Exception

RuntimeException Error

Exception

Exception

```
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
    <tx:method name="get*" read-only="true" rollback-for="NoProductInStockException"/>
    <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
```

InstrumentNotFoundException

```
<tx:advice id="txAdvice">
    <tx:attributes>
    <tx:method name="updateStock" no-rollback-for="InstrumentNotFoundException"/>
    <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
```

InstrumentNotFoundException

```
<tx:advice id="txAdvice">
    <tx:attributes>
    <tx:method name="*" rollback-for="Throwable" no-rollback-
for="InstrumentNotFoundException"/>
    </tx:attributes>
</tx:advice>
```

Java

```
public void resolvePosition() {
    try {
        // some business logic...
    } catch (NoProductInStockException ex) {
        // trigger rollback programmatically
        TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
    }
}
```

Kotlin

```
fun resolvePosition() {
    try {
        // some business logic...
    } catch (ex: NoProductInStockException) {
        // trigger rollback programmatically
        TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
    }
}
```

1.4.4. Configuring Different Transactional Semantics for Different Beans

```
<aop:advisor/>                        pointcut     advice-ref

x.y.service
                                                   Service
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <aop:config>

        <aop:pointcut id="serviceOperation"
                expression="execution(* x.y.service..*Service.*(..))"/>

        <aop:advisor pointcut-ref="serviceOperation" advice-ref="txAdvice"/>

    </aop:config>

    <!-- these two beans will be transactional... -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>
    <bean id="barService" class="x.y.service.extras.SimpleBarService"/>

    <!-- ... and these two beans won't -->
    <bean id="anotherService" class="org.xyz.SomeService"/> <!-- (not in the right
package) -->
    <bean id="barManager" class="x.y.service.SimpleBarManager"/> <!-- (doesn't end in
'Service') -->

    <tx:advice id="txAdvice">
        <tx:attributes>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>

    <!-- other transaction infrastructure beans such as a TransactionManager
omitted... -->

</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <aop:config>

        <aop:pointcut id="defaultServiceOperation"
                expression="execution(* x.y.service.*Service.*(..))"/>

        <aop:pointcut id="noTxServiceOperation"
                expression="execution(* x.y.service.ddl.DefaultDdlManager.*(..))"/>

        <aop:advisor pointcut-ref="defaultServiceOperation" advice-
ref="defaultTxAdvice"/>

        <aop:advisor pointcut-ref="noTxServiceOperation" advice-ref="noTxAdvice"/>

    </aop:config>

    <!-- this bean will be transactional (see the 'defaultServiceOperation' pointcut)
-->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- this bean will also be transactional, but with totally different
transactional settings -->
    <bean id="anotherFooService" class="x.y.service.ddl.DefaultDdlManager"/>

    <tx:advice id="defaultTxAdvice">
        <tx:attributes>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>

    <tx:advice id="noTxAdvice">
        <tx:attributes>
            <tx:method name="*" propagation="NEVER"/>
        </tx:attributes>
    </tx:advice>

    <!-- other transaction infrastructure beans such as a TransactionManager
omitted... -->

</beans>
```

1.4.5. <tx:advice/> Settings

<tx:advice/> <tx:advice/>

REQUIRED.

DEFAULT.

RuntimeException Exception

<tx:method/> <tx:advice/> <tx:attributes/>

Table 1. <tx:method/> settings

Attribute	Required?	Default	Description
name			get* handle* on*Event
propagation		REQUIRED	
isolation		DEFAULT	REQUIRED REQUIRES_NEW
timeout			REQUIRED REQUIRES_NEW
read-only			REQUIRED REQUIRES_NEW

Attribute	Required?	Default	Description

`rollback-for`			`Exception`
			
			`com.foo.MyBusinessException,ServletException`

`no-rollback-for`			`Exception`
			
			`com.foo.MyBusinessException,ServletException`

1.4.6. Using `@Transactional`

`javax.transaction.Transactional`

`@Transactional`

Java

```
// the service class that we want to make transactional
@Transactional
public class DefaultFooService implements FooService {

    Foo getFoo(String fooName) {
        // ...
    }

    Foo getFoo(String fooName, String barName) {
        // ...
    }

    void insertFoo(Foo foo) {
        // ...
    }

    void updateFoo(Foo foo) {
        // ...
    }
}
```

Kotlin

```
// the service class that we want to make transactional
@Transactional
class DefaultFooService : FooService {

    override fun getFoo(fooName: String): Foo {
        // ...
    }

    override fun getFoo(fooName: String, barName: String): Foo {
        // ...
    }

    override fun insertFoo(foo: Foo) {
        // ...
    }

    override fun updateFoo(foo: Foo) {
        // ...
    }
}
```

@EnableTransactionManagement
@Configuration

<tx:annotation-driven/>

```
<!-- from the file 'context.xml' -->
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- this is the service object that we want to make transactional -->
    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- enable the configuration of transactional behavior based on annotations -->
    <tx:annotation-driven transaction-manager="txManager"/><!-- a TransactionManager
is still required --> ①

    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <!-- (this dependency is defined somewhere else) -->
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- other <bean/> definitions here -->

</beans>
```

①

transaction-manager <tx:annotation-driven/>
TransactionManager
transactionManager TransactionManager
transaction-manager

Java

```
// the reactive service class that we want to make transactional
@Transactional
public class DefaultFooService implements FooService {

    Publisher<Foo> getFoo(String fooName) {
        // ...
    }

    Mono<Foo> getFoo(String fooName, String barName) {
        // ...
    }

    Mono<Void> insertFoo(Foo foo) {
        // ...
    }

    Mono<Void> updateFoo(Foo foo) {
        // ...
    }
}
```

Kotlin

```
// the reactive service class that we want to make transactional
@Transactional
class DefaultFooService : FooService {

    override fun getFoo(fooName: String): Flow<Foo> {
        // ...
    }

    override fun getFoo(fooName: String, barName: String): Mono<Foo> {
        // ...
    }

    override fun insertFoo(foo: Foo): Mono<Void> {
        // ...
    }

    override fun updateFoo(foo: Foo): Mono<Void> {
        // ...
    }
}
```

`Publisher`

Method visibility and `@Transactional`

`@Transactional`

`@Transactional`

`@Transactional`

`@Transactional`

`@Transactional`

`@Transactional`

`<tx:annotation-driven/>`

`@Transactional`

`@Transactional`

`proxy-target-class="true"`

`mode="aspectj"`

`@Transactional`

`@PostConstruct`

`mode`

`@Transactional`

Table 2. Annotation driven transaction settings

XML Attribute	Annotation Attribute	Default	Description
transaction-manager	TransactionManagementConfigurer	transactionManager	transactionManager
mode	mode	proxy	proxy aspectj spring-aspects.jar

XML Attribute	Annotation Attribute	Default	Description
proxy-target-class	proxyTargetClass	false	proxy @Transactional proxy-target-class true proxy-target-class false
order	order	Ordered.LOWEST_PRECEDENCE	@Transactional

@Transactional proxy aspectj

proxy-target-class
@Transactional proxy-target-class true proxy-target-class false

@EnableTransactionManagement <tx:annotation-driven/>
@Transactional

WebApplicationContext DispatcherServlet @Transactional

DefaultFooService
@Transactional
updateFoo(Foo)

Java

```
@Transactional(readOnly = true)
public class DefaultFooService implements FooService {

    public Foo getFoo(String fooName) {
        // ...
    }

    // these settings have precedence for this method
    @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
    public void updateFoo(Foo foo) {
        // ...
    }
}
```

Kotlin

```
@Transactional(readOnly = true)
class DefaultFooService : FooService {

    override fun getFoo(fooName: String): Foo {
        // ...
    }

    // these settings have precedence for this method
    @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
    override fun updateFoo(foo: Foo) {
        // ...
    }
}
```

`@Transactional` **Settings**

`@Transactional`

`@Transactional`

`PROPAGATION_REQUIRED.`

`ISOLATION_DEFAULT.`

`RuntimeException` `Exception`

`@Transactional`

Table 3. @Transactional Settings

Property	Type	Description
	`String`	
	`enum Propagation`	
`isolation`	`enum Isolation`	`REQUIRED` `REQUIRES_NEW`

Property	Type	Description
timeout	int	REQUIRED REQUIRES_NEW
readOnly	boolean	REQUIRED REQUIRES_NEW
rollbackFor	Class Throwable.	
rollbackForClassName	Throwable.	
noRollbackFor	Class Throwable.	
noRollbackForClassName	String Throwable.	

. handlePayment(..) BusinessService com.example.BusinessService.handlePayment

Multiple Transaction Managers with `@Transactional`

value transactionManager @Transactional TransactionManager

Java

```
public class TransactionalService {

    @Transactional("order")
    public void setSomething(String name) { ... }

    @Transactional("account")
    public void doSomething() { ... }

    @Transactional("reactive-account")
    public Mono<Void> doSomethingReactive() { ... }
}
```

Kotlin

```
class TransactionalService {

    @Transactional("order")
    fun setSomething(name: String) {
        // ...
    }

    @Transactional("account")
    fun doSomething() {
        // ...
    }

    @Transactional("reactive-account")
    fun doSomethingReactive(): Mono<Void> {
        // ...
    }
}
```

```
<tx:annotation-driven/>

    <bean id="transactionManager1"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        ...
        <qualifier value="order"/>
    </bean>

    <bean id="transactionManager2"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        ...
        <qualifier value="account"/>
    </bean>

    <bean id="transactionManager3"
class="org.springframework.data.r2dbc.connectionfactory.R2dbcTransactionManager">
        ...
        <qualifier value="reactive-account"/>
    </bean>
```

TransactionalService
order account reactive-account
<tx:annotation-driven> transactionManager
TransactionManager

Custom Composed Annotations

@Transactional

Java

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("order")
public @interface OrderTx {
}

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("account")
public @interface AccountTx {
}
```

Kotlin

```
@Target(AnnotationTarget.FUNCTION, AnnotationTarget.TYPE)
@Retention(AnnotationRetention.RUNTIME)
@Transactional("order")
annotation class OrderTx

@Target(AnnotationTarget.FUNCTION, AnnotationTarget.TYPE)
@Retention(AnnotationRetention.RUNTIME)
@Transactional("account")
annotation class AccountTx
```

Java

```
public class TransactionalService {

    @OrderTx
    public void setSomething(String name) {
        // ...
    }

    @AccountTx
    public void doSomething() {
        // ...
    }
}
```

Kotlin

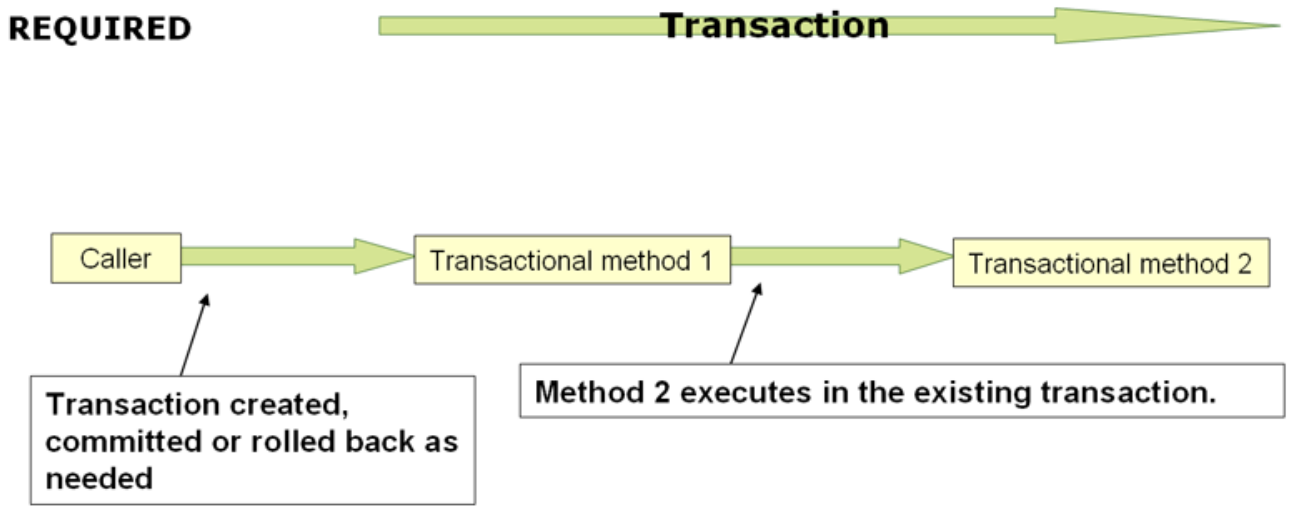
```
class TransactionalService {

    @OrderTx
    fun setSomething(name: String) {
        // ...
    }

    @AccountTx
    fun doSomething() {
        // ...
    }
}
```

1.4.7. Transaction Propagation

Understanding `PROPAGATION_REQUIRED`



`PROPAGATION_REQUIRED`

`validateExistingTransactions` `true`

`PROPAGATION_REQUIRED`

`PROPAGATION_REQUIRED`

UnexpectedRollbackException

UnexpectedRollbackException

Understanding PROPAGATION_REQUIRES_NEW

PROPAGATION_REQUIRES_NEW PROPAGATION_REQUIRED

Understanding PROPAGATION_NESTED

PROPAGATION_NESTED

DataSourceTransactionManager

1.4.8. Advising Transactional Operations

<tx:annotation-driven/>

updateFoo(Foo)

Java

```java
package x.y;

import org.aspectj.lang.ProceedingJoinPoint;
import org.springframework.util.StopWatch;
import org.springframework.core.Ordered;

public class SimpleProfiler implements Ordered {

    private int order;

    // allows us to control the ordering of advice
    public int getOrder() {
        return this.order;
    }

    public void setOrder(int order) {
        this.order = order;
    }

    // this method is the around advice
    public Object profile(ProceedingJoinPoint call) throws Throwable {
        Object returnValue;
        StopWatch clock = new StopWatch(getClass().getName());
        try {
            clock.start(call.toShortString());
            returnValue = call.proceed();
        } finally {
            clock.stop();
            System.out.println(clock.prettyPrint());
        }
        return returnValue;
    }
}
```

Kotlin

```
class SimpleProfiler : Ordered {

    private var order: Int = 0

    // allows us to control the ordering of advice
    override fun getOrder(): Int {
        return this.order
    }

    fun setOrder(order: Int) {
        this.order = order
    }

    // this method is the around advice
    fun profile(call: ProceedingJoinPoint): Any {
        var returnValue: Any
        val clock = StopWatch(javaClass.name)
        try {
            clock.start(call.toShortString())
            returnValue = call.proceed()
        } finally {
            clock.stop()
            println(clock.prettyPrint())
        }
        return returnValue
    }
}
```

`Ordered`

`fooService`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- this is the aspect -->
    <bean id="profiler" class="x.y.SimpleProfiler">
        <!-- run before the transactional advice (hence the lower order number) -->
        <property name="order" value="1"/>
    </bean>

    <tx:annotation-driven transaction-manager="txManager" order="200"/>

    <aop:config>
            <!-- this advice runs around the transactional advice -->
            <aop:aspect id="profilingAspect" ref="profiler">
                <aop:pointcut id="serviceMethodWithReturnValue"
                        expression="execution(!void x.y..*Service.*(..))"/>
                <aop:around method="profile" pointcut-
ref="serviceMethodWithReturnValue"/>
            </aop:aspect>
    </aop:config>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>

    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

</beans>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <bean id="fooService" class="x.y.service.DefaultFooService"/>

    <!-- the profiling advice -->
    <bean id="profiler" class="x.y.SimpleProfiler">
        <!-- run before the transactional advice (hence the lower order number) -->
        <property name="order" value="1"/>
    </bean>

    <aop:config>
        <aop:pointcut id="entryPointMethod" expression="execution(*
x.y..*Service.*(..))"/>
        <!-- runs after the profiling advice (c.f. the order attribute) -->

        <aop:advisor advice-ref="txAdvice" pointcut-ref="entryPointMethod" order="2"/>
        <!-- order value is higher than the profiling aspect -->

        <aop:aspect id="profilingAspect" ref="profiler">
            <aop:pointcut id="serviceMethodWithReturnValue"
                    expression="execution(!void x.y..*Service.*(..))"/>
            <aop:around method="profile" pointcut-ref="serviceMethodWithReturnValue"/>
        </aop:aspect>

    </aop:config>

    <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
            <tx:method name="get*" read-only="true"/>
            <tx:method name="*"/>
        </tx:attributes>
    </tx:advice>

    <!-- other <bean/> definitions such as a DataSource and a TransactionManager here
-->

</beans>
```

fooService

order

1.4.9. Using @Transactional with AspectJ

@Transactional

@Transactional org.springframework.transaction.aspectj.AnnotationTransactionAspect spring-aspects.jar

<tx:annotation-driven/> mode aspectj @Transactional

@Transactional

AnnotationTransactionAspect

Java

```
// construct an appropriate transaction manager
DataSourceTransactionManager txManager = new
DataSourceTransactionManager(getDataSource());

// configure the AnnotationTransactionAspect to use it; this must be done before
executing any transactional methods
AnnotationTransactionAspect.aspectOf().setTransactionManager(txManager);
```

Kotlin

```
// construct an appropriate transaction manager
val txManager = DataSourceTransactionManager(getDataSource())

// configure the AnnotationTransactionAspect to use it; this must be done before
executing any transactional methods
AnnotationTransactionAspect.aspectOf().transactionManager = txManager
```

@Transactional

@Transactional

AnnotationTransactionAspect

1.5. Programmatic Transaction Management

TransactionTemplate TransactionalOperator

TransactionManager

TransactionTemplate
TransactionalOperator
UserTransaction

1.5.1. Using the TransactionTemplate

TransactionTemplate
JdbcTemplate

TransactionTemplate

TransactionTemplate
TransactionCallback

TransactionCallback execute(..) TransactionTemplate

Java

```
public class SimpleService implements Service {

    // single TransactionTemplate shared amongst all methods in this instance
    private final TransactionTemplate transactionTemplate;

    // use constructor-injection to supply the PlatformTransactionManager
    public SimpleService(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
    }

    public Object someServiceMethod() {
        return transactionTemplate.execute(new TransactionCallback() {
            // the code in this method runs in a transactional context
            public Object doInTransaction(TransactionStatus status) {
                updateOperation1();
                return resultOfUpdateOperation2();
            }
        });
    }
}
```

Kotlin

```
// use constructor-injection to supply the PlatformTransactionManager
class SimpleService(transactionManager: PlatformTransactionManager) : Service {

    // single TransactionTemplate shared amongst all methods in this instance
    private val transactionTemplate = TransactionTemplate(transactionManager)

    fun someServiceMethod() = transactionTemplate.execute<Any?> {
        updateOperation1()
        resultOfUpdateOperation2()
    }
}
```

`TransactionCallbackWithoutResult`

Java

```
transactionTemplate.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        updateOperation1();
        updateOperation2();
    }
});
```

Kotlin

```
transactionTemplate.execute(object : TransactionCallbackWithoutResult() {
    override fun doInTransactionWithoutResult(status: TransactionStatus) {
        updateOperation1()
        updateOperation2()
    }
})
```

`setRollbackOnly()` `TransactionStatus`

Java

```
transactionTemplate.execute(new TransactionCallbackWithoutResult() {

    protected void doInTransactionWithoutResult(TransactionStatus status) {
        try {
            updateOperation1();
            updateOperation2();
        } catch (SomeBusinessException ex) {
            status.setRollbackOnly();
        }
    }
});
```

Kotlin

```
transactionTemplate.execute(object : TransactionCallbackWithoutResult() {

    override fun doInTransactionWithoutResult(status: TransactionStatus) {
        try {
            updateOperation1()
            updateOperation2()
        } catch (ex: SomeBusinessException) {
            status.setRollbackOnly()
        }
    }
})
```

Specifying Transaction Settings

`TransactionTemplate` `TransactionTemplate` `TransactionTemplate:`

Java

```
public class SimpleService implements Service {

    private final TransactionTemplate transactionTemplate;

    public SimpleService(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);

        // the transaction settings can be set here explicitly if so desired

this.transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_READ_UNCOMM
ITTED);
        this.transactionTemplate.setTimeout(30); // 30 seconds
        // and so forth...
    }
}
```

Kotlin

```
class SimpleService(transactionManager: PlatformTransactionManager) : Service {

    private val transactionTemplate = TransactionTemplate(transactionManager).apply {
        // the transaction settings can be set here explicitly if so desired
        isolationLevel = TransactionDefinition.ISOLATION_READ_UNCOMMITTED
        timeout = 30 // 30 seconds
        // and so forth...
    }
}
```

TransactionTemplate

```
<bean id="sharedTransactionTemplate"
        class="org.springframework.transaction.support.TransactionTemplate">
    <property name="isolationLevelName" value="ISOLATION_READ_UNCOMMITTED"/>
    <property name="timeout" value="30"/>
</bean>
```

sharedTransactionTemplate

TransactionTemplate

TransactionTemplate

TransactionTemplate

TransactionTemplate

TransactionTemplate

1.5.2. Using the `TransactionOperator`

`TransactionOperator`

`TransactionOperator`

`TransactionOperator`

Java

```
public class SimpleService implements Service {

    // single TransactionOperator shared amongst all methods in this instance
    private final TransactionalOperator transactionalOperator;

    // use constructor-injection to supply the ReactiveTransactionManager
    public SimpleService(ReactiveTransactionManager transactionManager) {
        this.transactionOperator = TransactionalOperator.create(transactionManager);
    }

    public Mono<Object> someServiceMethod() {

        // the code in this method runs in a transactional context

        Mono<Object> update = updateOperation1();

        return
update.then(resultOfUpdateOperation2).as(transactionalOperator::transactional);
    }
}
```

Kotlin

```
// use constructor-injection to supply the ReactiveTransactionManager
class SimpleService(transactionManager: ReactiveTransactionManager) : Service {

    // single TransactionalOperator shared amongst all methods in this instance
    private val transactionalOperator =
TransactionalOperator.create(transactionManager)

    suspend fun someServiceMethod() = transactionalOperator.executeAndAwait<Any?> {
        updateOperation1()
        resultOfUpdateOperation2()
    }
}
```

`TransactionalOperator`

`mono.as(transactionalOperator::transactional)`

`transactionalOperator.execute(TransactionCallback<T>)`

`setRollbackOnly()`

`ReactiveTransaction`

Java

```
transactionalOperator.execute(new TransactionCallback<>() {

    public Mono<Object> doInTransaction(ReactiveTransaction status) {
        return updateOperation1().then(updateOperation2)
                    .doOnError(SomeBusinessException.class, e ->
status.setRollbackOnly());
        }
    }
});
```

Kotlin

```
transactionalOperator.execute(object : TransactionCallback() {

    override fun doInTransactionWithoutResult(status: ReactiveTransaction) {
        updateOperation1().then(updateOperation2)
                    .doOnError(SomeBusinessException.class, e ->
status.setRollbackOnly())
    }
})
```

Cancel Signals

`Subscriber` `Subscription` `Publisher`

next() take(long) timeout(Duration)

TransactionalOperator

Publisher Flux Publisher

Specifying Transaction Settings

TransactionalOperator TransactionalOperator

TransactionalOperator:

Java

```
public class SimpleService implements Service {

    private final TransactionalOperator transactionalOperator;

    public SimpleService(ReactiveTransactionManager transactionManager) {
        DefaultTransactionDefinition definition = new DefaultTransactionDefinition();

        // the transaction settings can be set here explicitly if so desired

definition.setIsolationLevel(TransactionDefinition.ISOLATION_READ_UNCOMMITTED);
        definition.setTimeout(30); // 30 seconds
        // and so forth...

        this.transactionalOperator = TransactionalOperator.create(transactionManager,
definition);
    }
}
```

Kotlin

```
class SimpleService(transactionManager: ReactiveTransactionManager) : Service {

    private val definition = DefaultTransactionDefinition().apply {
        // the transaction settings can be set here explicitly if so desired
        isolationLevel = TransactionDefinition.ISOLATION_READ_UNCOMMITTED
        timeout = 30 // 30 seconds
        // and so forth...
    }
    private val transactionalOperator = TransactionalOperator(transactionManager,
definition)
}
```

1.5.3. Using the `TransactionManager`

Using the `PlatformTransactionManager`

`org.springframework.transaction.PlatformTransactionManager` `PlatformTransactionManager` `TransactionDefinition` `TransactionStatus`

Java

```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
// explicitly setting the transaction name is something that can be done only
programmatically
def.setName("SomeTxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

TransactionStatus status = txManager.getTransaction(def);
try {
    // put your business logic here
}
catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
}
txManager.commit(status);
```

Kotlin

```
val def = DefaultTransactionDefinition()
// explicitly setting the transaction name is something that can be done only
programmatically
def.setName("SomeTxName")
def.propagationBehavior = TransactionDefinition.PROPAGATION_REQUIRED

val status = txManager.getTransaction(def)
try {
    // put your business logic here
} catch (ex: MyException) {
    txManager.rollback(status)
    throw ex
}

txManager.commit(status)
```

Using the `ReactiveTransactionManager`

`org.springframework.transaction.ReactiveTransactionManager` `ReactiveTransactionManager` `TransactionDefinition` `ReactiveTransaction`

Java

```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
// explicitly setting the transaction name is something that can be done only
programmatically
def.setName("SomeTxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

Mono<ReactiveTransaction> reactiveTx = txManager.getReactiveTransaction(def);

reactiveTx.flatMap(status -> {

    Mono<Object> tx = ...; // put your business logic here

    return tx.then(txManager.commit(status))
            .onErrorResume(ex -> txManager.rollback(status).then(Mono.error(ex)));
});
```

Kotlin

```
val def = DefaultTransactionDefinition()
// explicitly setting the transaction name is something that can be done only
programmatically
def.setName("SomeTxName")
def.propagationBehavior = TransactionDefinition.PROPAGATION_REQUIRED

val reactiveTx = txManager.getReactiveTransaction(def)
reactiveTx.flatMap { status ->

    val tx = ... // put your business logic here

    tx.then(txManager.commit(status))
            .onErrorResume { ex -> txManager.rollback(status).then(Mono.error(ex)) }
}
```

1.6. Choosing Between Programmatic and Declarative Transaction Management

`TransactionTemplate`

1.7. Transaction-bound Events

`@EventListener`
`@TransactionalEventListener`

Java

```
@Component
public class MyComponent {

    @TransactionalEventListener
    public void handleOrderCreatedEvent(CreationEvent<Order> creationEvent) {
        // ...
    }
}
```

Kotlin

```
@Component
class MyComponent {

    @TransactionalEventListener
    fun handleOrderCreatedEvent(creationEvent: CreationEvent<Order>) {
        // ...
    }
}
```

`@TransactionalEventListener` `phase`

BEFORE_COMMIT
AFTER_COMMIT AFTER_ROLLBACK AFTER_COMPLETION

fallbackExecution
true

1.8. Application server-specific integration

JtaTransactionManager
UserTransaction TransactionManager
TransactionManager

JtaTransactionManager

JtaTransactionManager

```
<tx:jta-transaction-manager/>
```

JtaTransactionManager

1.8.1. IBM WebSphere

WebSphereUowTransactionManager UOWManager

PROPAGATION_REQUIRES_NEW

1.8.2. Oracle WebLogic Server

WebLogicJtaTransactionManager
JtaTransactionManager
JtaTransactionManager

1.9. Solutions to Common Problems

1.9.1. Using the Wrong Transaction Manager for a Specific `DataSource`

`PlatformTransactionManager`

`org.springframework.transaction.jta.JtaTransactionManager`

`DataSource`

1.10. Further Resources

Java Transaction Design Strategies

Chapter 2. DAO Support

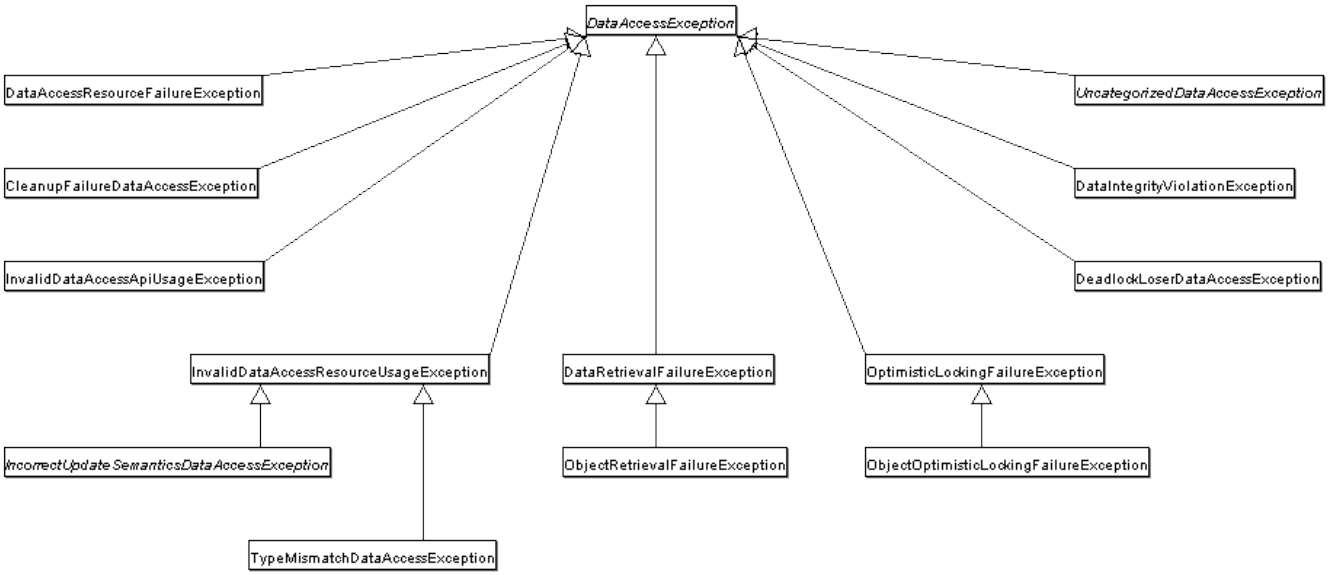
2.1. Consistent Exception Hierarchy

SQLException

DataAccessException

HibernateExceptions PersistenceExceptions convertHibernateAccessException(..) convertJpaAccessException() SessionFactoryUtils org.springframework.dao PersistenceExceptions

DataAccessException



2.2. Annotations Used to Configure DAO or Repository Classes

`@Repository`

`@Repository`

Java

```
@Repository ①
public class SomeMovieFinder implements MovieFinder {
    // ...
}
```

① `@Repository`

Kotlin

```
@Repository ①
class SomeMovieFinder : MovieFinder {
    // ...
}
```

① `@Repository`

`DataSource` `EntityManager`

`@Autowired` `@Inject` `@Resource` `@PersistenceContext`

Java

```
@Repository
public class JpaMovieFinder implements MovieFinder {

    @PersistenceContext
    private EntityManager entityManager;

    // ...
}
```

Kotlin

```
@Repository
class JpaMovieFinder : MovieFinder {

    @PersistenceContext
    private lateinit var entityManager: EntityManager

    // ...
}
```

`SessionFactory`

Java

```
@Repository
public class HibernateMovieFinder implements MovieFinder {

    private SessionFactory sessionFactory;

    @Autowired
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    // ...
}
```

Kotlin

```
@Repository
class HibernateMovieFinder(private val sessionFactory: SessionFactory) : MovieFinder {
    // ...
}
```

`DataSource`
`JdbcTemplate`
`SimpleJdbcCall` `DataSource`
`DataSource`

Java

```
@Repository
public class JdbcMovieFinder implements MovieFinder {

    private JdbcTemplate jdbcTemplate;

    @Autowired
    public void init(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    // ...
}
```

Kotlin

```
@Repository
class JdbcMovieFinder(dataSource: DataSource) : MovieFinder {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    // ...
}
```

Chapter 3. Data Access with JDBC

Table 4. Spring JDBC - who does what?

Action	Spring	You

3.1. Choosing an Approach for JDBC Database Access

`JdbcTemplate` `SimpleJdbcInsert` `SimpleJdbcCall`

`JdbcTemplate`

`NamedParameterJdbcTemplate` `JdbcTemplate` `?`

`SimpleJdbcInsert` `SimpleJdbcCall`

MappingSqlQuery SqlUpdate StoredProcedure

execute(…) update(…) findObject(…)

3.2. Package Hierarchy

core org.springframework.jdbc.core JdbcTemplate

org.springframework.jdbc.core.simple SimpleJdbcInsert SimpleJdbcCall org.springframework.jdbc.core.namedparam NamedParameterJdbcTemplate

SimpleJdbc

datasource org.springframework.jdbc.datasource DataSource DataSource

org.springfamework.jdbc.datasource.embedded

object org.springframework.jdbc.object

org.springframework.jdbc.core

support org.springframework.jdbc.support SQLException

org.springframework.dao

SQLExceptionTranslator

3.3. Using the JDBC Core Classes to Control Basic JDBC Processing and Error Handling

JdbcTemplate

NamedParameterJdbcTemplate

SQLExceptionTranslator

3.3.1. Using JdbcTemplate

JdbcTemplate

JdbcTemplate

ResultSet

org.springframework.dao

JdbcTemplate
Connection JdbcTemplate
PreparedStatementCreator
CallableStatementCreator
RowCallbackHandler ResultSet

JdbcTemplate
DataSource

DataSource

DEBUG
JdbcTemplate
JdbcTemplate

JdbcTemplate
JdbcTemplate

Querying (SELECT)

Java

```
int rowCount = this.jdbcTemplate.queryForObject("select count(*) from t_actor",
Integer.class);
```

Kotlin

```
val rowCount = jdbcTemplate.queryForObject<Int>("select count(*) from t_actor")!!
```

Java

```
int countOfActorsNamedJoe = this.jdbcTemplate.queryForObject(
        "select count(*) from t_actor where first_name = ?", Integer.class, "Joe");
```

Kotlin

```
val countOfActorsNamedJoe = jdbcTemplate.queryForObject<Int>(
        "select count(*) from t_actor where first_name = ?", arrayOf("Joe"))!!
```

`String`

Java

```
String lastName = this.jdbcTemplate.queryForObject(
        "select last_name from t_actor where id = ?",
        String.class, 1212L);
```

Kotlin

```
val lastName = this.jdbcTemplate.queryForObject<String>(
        "select last_name from t_actor where id = ?",
        arrayOf(1212L))!!
```

Java

```
Actor actor = jdbcTemplate.queryForObject(
        "select first_name, last_name from t_actor where id = ?",
        (resultSet, rowNum) -> {
            Actor newActor = new Actor();
            newActor.setFirstName(resultSet.getString("first_name"));
            newActor.setLastName(resultSet.getString("last_name"));
            return newActor;
        },
        1212L);
```

Kotlin

```
val actor = jdbcTemplate.queryForObject(
            "select first_name, last_name from t_actor where id = ?",
            arrayOf(1212L)) { rs, _ ->
        Actor(rs.getString("first_name"), rs.getString("last_name"))
    }
```

Java

```
List<Actor> actors = this.jdbcTemplate.query(
        "select first_name, last_name from t_actor",
        (resultSet, rowNum) -> {
            Actor actor = new Actor();
            actor.setFirstName(resultSet.getString("first_name"));
            actor.setLastName(resultSet.getString("last_name"));
            return actor;
        });
```

Kotlin

```
val actors = jdbcTemplate.query("select first_name, last_name from t_actor") { rs, _
->
        Actor(rs.getString("first_name"), rs.getString("last_name"))
```

`RowMapper`

Java

```
private final RowMapper<Actor> actorRowMapper = (resultSet, rowNum) -> {
    Actor actor = new Actor();
    actor.setFirstName(resultSet.getString("first_name"));
    actor.setLastName(resultSet.getString("last_name"));
    return actor;
};

public List<Actor> findAllActors() {
    return this.jdbcTemplate.query( "select first_name, last_name from t_actor",
actorRowMapper);
}
```

Kotlin

```
val actorMapper = RowMapper<Actor> { rs: ResultSet, rowNum: Int ->
    Actor(rs.getString("first_name"), rs.getString("last_name"))
}

fun findAllActors(): List<Actor> {
    return jdbcTemplate.query("select first_name, last_name from t_actor",
actorMapper)
}
```

Updating (`INSERT`, `UPDATE`, and `DELETE`) with `JdbcTemplate`

`update(..)`

Java

```
this.jdbcTemplate.update(
        "insert into t_actor (first_name, last_name) values (?, ?)",
        "Leonor", "Watling");
```

Kotlin

```
jdbcTemplate.update(
        "insert into t_actor (first_name, last_name) values (?, ?)",
        "Leonor", "Watling")
```

Java

```
this.jdbcTemplate.update(
        "update t_actor set last_name = ? where id = ?",
        "Banjo", 5276L);
```

Kotlin

```
jdbcTemplate.update(
        "update t_actor set last_name = ? where id = ?",
        "Banjo", 5276L)
```

Java

```
this.jdbcTemplate.update(
        "delete from t_actor where id = ?",
        Long.valueOf(actorId));
```

Kotlin

```
jdbcTemplate.update("delete from t_actor where id = ?", actorId.toLong())
```

Other `JdbcTemplate` Operations

`execute(..)`

Java

```
this.jdbcTemplate.execute("create table mytable (id integer, name varchar(100))");
```

Kotlin

```
jdbcTemplate.execute("create table mytable (id integer, name varchar(100))")
```

Java

```
this.jdbcTemplate.update(
        "call SUPPORT.REFRESH_ACTORS_SUMMARY(?)",
        Long.valueOf(unionId));
```

Kotlin

```
jdbcTemplate.update(
        "call SUPPORT.REFRESH_ACTORS_SUMMARY(?)",
        unionId.toLong())
```

`JdbcTemplate` Best Practices

`JdbcTemplate`
`JdbcTemplate`
`JdbcTemplate`
`DataSource`

`JdbcTemplate`
`NamedParameterJdbcTemplate` `DataSource`
`DataSource` `JdbcTemplate`
`DataSource`

Java

```
public class JdbcCorporateEventDao implements CorporateEventDao {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    // JDBC-backed implementations of the methods on the CorporateEventDao follow...
}
```

Kotlin

```
class JdbcCorporateEventDao(dataSource: DataSource) : CorporateEventDao {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    // JDBC-backed implementations of the methods on the CorporateEventDao follow...
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="corporateEventDao" class="com.example.JdbcCorporateEventDao">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <context:property-placeholder location="jdbc.properties"/>

</beans>
```

@Repository
DataSource @Autowired

Java

```
@Repository ①
public class JdbcCorporateEventDao implements CorporateEventDao {

    private JdbcTemplate jdbcTemplate;

    @Autowired ②
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource); ③
    }

    // JDBC-backed implementations of the methods on the CorporateEventDao follow...
}
```

① @Repository

② DataSource @Autowired

③ JdbcTemplate DataSource

Kotlin

```
@Repository ①
class JdbcCorporateEventDao(dataSource: DataSource) : CorporateEventDao { ②

    private val jdbcTemplate = JdbcTemplate(dataSource) ③

    // JDBC-backed implementations of the methods on the CorporateEventDao follow...
}
```

① @Repository

② DataSource

③ JdbcTemplate DataSource

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Scans within the base package of the application for @Component classes to
configure as beans -->
    <context:component-scan base-package="org.springframework.docs.test" />

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <context:property-placeholder location="jdbc.properties"/>

</beans>
```

JdbcDaoSupport

setDataSource(..) JdbcDaoSupport

JdbcDaoSupport

JdbcTemplate
JdbcTemplate
JdbcTemplate DataSources
JdbcTemplate

3.3.2. Using NamedParameterJdbcTemplate

NamedParameterJdbcTemplate

'?' NamedParameterJdbcTemplate JdbcTemplate
JdbcTemplate
NamedParameterJdbcTemplate JdbcTemplate

NamedParameterJdbcTemplate

Java

```
// some JDBC-backed DAO class...
private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

public void setDataSource(DataSource dataSource) {
    this.namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
}

public int countOfActorsByFirstName(String firstName) {

    String sql = "select count(*) from T_ACTOR where first_name = :first_name";

    SqlParameterSource namedParameters = new MapSqlParameterSource("first_name",
firstName);

    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Integer.class);
}
```

Kotlin

```
private val namedParameterJdbcTemplate = NamedParameterJdbcTemplate(dataSource)

fun countOfActorsByFirstName(firstName: String): Int {
    val sql = "select count(*) from T_ACTOR where first_name = :first_name"
    val namedParameters = MapSqlParameterSource("first_name", firstName)
    return namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Int::class.java)!!
}
```

sql
namedParameters
MapSqlParameterSource

NamedParameterJdbcTemplate Map
NamedParameterJdbcOperations NamedParameterJdbcTemplate

Map

Java

```
// some JDBC-backed DAO class...
private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

public void setDataSource(DataSource dataSource) {
    this.namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
}

public int countOfActorsByFirstName(String firstName) {

    String sql = "select count(*) from T_ACTOR where first_name = :first_name";

    Map<String, String> namedParameters = Collections.singletonMap("first_name",
firstName);

    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Integer.class);
}
```

Kotlin

```
// some JDBC-backed DAO class...
private val namedParameterJdbcTemplate = NamedParameterJdbcTemplate(dataSource)

fun countOfActorsByFirstName(firstName: String): Int {
    val sql = "select count(*) from T_ACTOR where first_name = :first_name"
    val namedParameters = mapOf("first_name" to firstName)
    return namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Int::class.java)!!
}
```

NamedParameterJdbcTemplate
SqlParameterSource
MapSqlParameterSource
SqlParameterSource NamedParameterJdbcTemplate
MapSqlParameterSource java.util.Map

SqlParameterSource BeanPropertySqlParameterSource

Java

```
public class Actor {

    private Long id;
    private String firstName;
    private String lastName;

    public String getFirstName() {
        return this.firstName;
    }

    public String getLastName() {
        return this.lastName;
    }

    public Long getId() {
        return this.id;
    }

    // setters omitted...

}
```

Kotlin

```
data class Actor(val id: Long, val firstName: String, val lastName: String)
```

`NamedParameterJdbcTemplate`

Java

```
// some JDBC-backed DAO class...
private NamedParameterJdbcTemplate namedParameterJdbcTemplate;

public void setDataSource(DataSource dataSource) {
    this.namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
}

public int countOfActors(Actor exampleActor) {

    // notice how the named parameters match the properties of the above 'Actor' class
    String sql = "select count(*) from T_ACTOR where first_name = :firstName and
last_name = :lastName";

    SqlParameterSource namedParameters = new
BeanPropertySqlParameterSource(exampleActor);

    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Integer.class);
}
```

Kotlin

```
// some JDBC-backed DAO class...
private val namedParameterJdbcTemplate = NamedParameterJdbcTemplate(dataSource)

private val namedParameterJdbcTemplate = NamedParameterJdbcTemplate(dataSource)

fun countOfActors(exampleActor: Actor): Int {
    // notice how the named parameters match the properties of the above 'Actor' class
    val sql = "select count(*) from T_ACTOR where first_name = :firstName and
last_name = :lastName"
    val namedParameters = BeanPropertySqlParameterSource(exampleActor)
    return namedParameterJdbcTemplate.queryForObject(sql, namedParameters,
Int::class.java)!!
}
```

`NamedParameterJdbcTemplate` `JdbcTemplate` `JdbcTemplate` `JdbcTemplate` `getJdbcOperations()` `JdbcTemplate` `JdbcOperations`

`JdbcTemplate` `NamedParameterJdbcTemplate`

3.3.3. Using `SQLExceptionTranslator`

`SQLExceptionTranslator`

SQLExceptions org.springframework.dao.DataAccessException

SQLErrorCodeSQLExceptionTranslator SQLExceptionTranslator SQLState

SQLErrorCodes SQLErrorCodesFactory SQLErrorCodes sql-error-codes.xml DatabaseProductName DatabaseMetaData

SQLErrorCodeSQLExceptionTranslator

SQLErrorCodeSQLExceptionTranslator

SQLExceptionTranslator customSqlExceptionTranslator SQLErrorCodes

CustomSQLErrorCodesTranslation customTranslations SQLErrorCodes

SQLExceptionSubclassTranslator

SQLStateSQLExceptionTranslator

SQLErrorCodesFactory Error sql-error-codes.xml SQLErrorCodes

SQLErrorCodeSQLExceptionTranslator

Java

```
public class CustomSQLErrorCodesTranslator extends SQLErrorCodeSQLExceptionTranslator
{

    protected DataAccessException customTranslate(String task, String sql,
SQLException sqlEx) {
        if (sqlEx.getErrorCode() == -12345) {
            return new DeadlockLoserDataAccessException(task, sqlEx);
        }
        return null;
    }
}
```

Kotlin

```
class CustomSQLErrorCodesTranslator : SQLErrorCodeSQLExceptionTranslator() {

    override fun customTranslate(task: String, sql: String?, sqlEx: SQLException):
DataAccessException? {
        if (sqlEx.errorCode == -12345) {
                return DeadlockLoserDataAccessException(task, sqlEx)
            }
            return null;
    }
}
```

`-12345`

`JdbcTemplate` `setExceptionTranslator` `JdbcTemplate`

Java

```
private JdbcTemplate jdbcTemplate;

public void setDataSource(DataSource dataSource) {

    // create a JdbcTemplate and set data source
    this.jdbcTemplate = new JdbcTemplate();
    this.jdbcTemplate.setDataSource(dataSource);

    // create a custom translator and set the DataSource for the default translation
lookup
    CustomSQLErrorCodesTranslator tr = new CustomSQLErrorCodesTranslator();
    tr.setDataSource(dataSource);
    this.jdbcTemplate.setExceptionTranslator(tr);

}

public void updateShippingCharge(long orderId, long pct) {
    // use the prepared JdbcTemplate for this update
    this.jdbcTemplate.update("update orders" +
        " set shipping_charge = shipping_charge * ? / 100" +
        " where id = ?", pct, orderId);
}
```

Kotlin

```
// create a JdbcTemplate and set data source
private val jdbcTemplate = JdbcTemplate(dataSource).apply {
    // create a custom translator and set the DataSource for the default translation
lookup
    exceptionTranslator = CustomSQLErrorCodesTranslator().apply {
        this.dataSource = dataSource
    }
}

fun updateShippingCharge(orderId: Long, pct: Long) {
    // use the prepared JdbcTemplate for this update
    this.jdbcTemplate!!.update("update orders" +
            " set shipping_charge = shipping_charge * ? / 100" +
            " where id = ?", pct, orderId)
}
```

sql-error-codes.xml

3.3.4. Running Statements

DataSource JdbcTemplate JdbcTemplate

Java

```
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

public class ExecuteAStatement {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public void doExecute() {
        this.jdbcTemplate.execute("create table mytable (id integer, name
varchar(100))");
    }
}
```

Kotlin

```
import javax.sql.DataSource
import org.springframework.jdbc.core.JdbcTemplate

class ExecuteAStatement(dataSource: DataSource) {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    fun doExecute() {
        jdbcTemplate.execute("create table mytable (id integer, name varchar(100))")
    }
}
```

3.3.5. Running Queries

`queryForObject(..)` Type `InvalidDataAccessApiUsageException` `int` `String`

Java

```
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

public class RunAQuery {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public int getCount() {
        return this.jdbcTemplate.queryForObject("select count(*) from mytable",
Integer.class);
    }

    public String getName() {
        return this.jdbcTemplate.queryForObject("select name from mytable",
String.class);
    }
}
```

Kotlin

```
import javax.sql.DataSource
import org.springframework.jdbc.core.JdbcTemplate

class RunAQuery(dataSource: DataSource) {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    val count: Int
        get() = jdbcTemplate.queryForObject("select count(*) from mytable")!!

    val name: String?
        get() = jdbcTemplate.queryForObject("select name from mytable")
}
```

`queryForList(..)` `List` `Map`

Java

```
private JdbcTemplate jdbcTemplate;

public void setDataSource(DataSource dataSource) {
    this.jdbcTemplate = new JdbcTemplate(dataSource);
}

public List<Map<String, Object>> getList() {
    return this.jdbcTemplate.queryForList("select * from mytable");
}
```

Kotlin

```
private val jdbcTemplate = JdbcTemplate(dataSource)

fun getList(): List<Map<String, Any>> {
    return jdbcTemplate.queryForList("select * from mytable")
}
```

```
[{name=Bob, id=1}, {name=Mary, id=2}]
```

3.3.6. Updating the Database

Java

```
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

public class ExecuteAnUpdate {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public void setName(int id, String name) {
        this.jdbcTemplate.update("update mytable set name = ? where id = ?", name,
id);
    }
}
```

Kotlin

```
import javax.sql.DataSource
import org.springframework.jdbc.core.JdbcTemplate

class ExecuteAnUpdate(dataSource: DataSource) {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    fun setName(id: Int, name: String) {
        jdbcTemplate.update("update mytable set name = ? where id = ?", name, id)
    }
}
```

3.3.7. Retrieving Auto-generated Keys

`update()`

`PreparedStatementCreator`

`KeyHolder`

`PreparedStatement`

Java

```
final String INSERT_SQL = "insert into my_test (name) values(?)";
final String name = "Rob";

KeyHolder keyHolder = new GeneratedKeyHolder();
jdbcTemplate.update(connection -> {
    PreparedStatement ps = connection.prepareStatement(INSERT_SQL, new String[] { "id"
});
    ps.setString(1, name);
    return ps;
}, keyHolder);

// keyHolder.getKey() now contains the generated key
```

Kotlin

```
val INSERT_SQL = "insert into my_test (name) values(?)"
val name = "Rob"

val keyHolder = GeneratedKeyHolder()
jdbcTemplate.update({
    it.prepareStatement (INSERT_SQL, arrayOf("id")).apply { setString(1, name) }
}, keyHolder)

// keyHolder.getKey() now contains the generated key
```

3.4. Controlling Database Connections

`DataSource`

`DataSourceUtils`

`SmartDataSource`

`AbstractDataSource`

`SingleConnectionDataSource`

`DriverManagerDataSource`

`TransactionAwareDataSourceProxy`

`DataSourceTransactionManager`

3.4.1. Using `DataSource`

`DataSource` `DataSource`

DataSource

DriverManagerDataSource SimpleDriverDataSource

DriverManagerDataSource
DataSource

DriverManagerDataSource

DriverManagerDataSource

DriverManager

DriverManagerDataSource

Java

```
DriverManagerDataSource dataSource = new DriverManagerDataSource();
dataSource.setDriverClassName("org.hsqldb.jdbcDriver");
dataSource.setUrl("jdbc:hsqldb:hsql://localhost:");
dataSource.setUsername("sa");
dataSource.setPassword("");
```

Kotlin

```
val dataSource = DriverManagerDataSource().apply {
    setDriverClassName("org.hsqldb.jdbcDriver")
    url = "jdbc:hsqldb:hsql://localhost:"
    username = "sa"
    password = ""
}
```

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<context:property-placeholder location="jdbc.properties"/>
```

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<context:property-placeholder location="jdbc.properties"/>
```

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-
method="close">
    <property name="driverClass" value="${jdbc.driverClassName}"/>
    <property name="jdbcUrl" value="${jdbc.url}"/>
    <property name="user" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<context:property-placeholder location="jdbc.properties"/>
```

3.4.2. Using `DataSourceUtils`

`DataSourceUtils` `static`

`DataSourceTransactionManager`

3.4.3. Implementing SmartDataSource

SmartDataSource
DataSource

3.4.4. Extending AbstractDataSource

AbstractDataSource abstract DataSource
DataSource AbstractDataSource
DataSource

3.4.5. Using SingleConnectionDataSource

SingleConnectionDataSource SmartDataSource
Connection

close
suppressClose true

Connection

SingleConnectionDataSource

DriverManagerDataSource

3.4.6. Using DriverManagerDataSource

DriverManagerDataSource DataSource
Connection

DataSource
Connection.close() DataSource
commons-dbcp
DriverManagerDataSource

3.4.7. Using TransactionAwareDataSourceProxy

TransactionAwareDataSourceProxy DataSource
DataSource
DataSource

DataSource

JdbcTemplate DataSourceUtils

TransactionAwareDataSourceProxy

3.4.8. Using DataSourceTransactionManager

DataSourceTransactionManager PlatformTransactionManager

DataSourceUtils.getConnection(DataSource) DataSource.getConnection org.springframework.dao SQLExceptions JdbcTemplate

DataSourceTransactionManager

JdbcTemplate DataSourceUtils.applyTransactionTimeout(..)

JtaTransactionManager

3.5. JDBC Batch Operations

3.5.1. Basic Batch Operations with JdbcTemplate

JdbcTemplate BatchPreparedStatementSetter batchUpdate getBatchSize setValues getBatchSize t_actor

Java

```
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public int[] batchUpdate(final List<Actor> actors) {
        return this.jdbcTemplate.batchUpdate(
                "update t_actor set first_name = ?, last_name = ? where id = ?",
                new BatchPreparedStatementSetter() {
                    public void setValues(PreparedStatement ps, int i) throws
SQLException {
                        Actor actor = actors.get(i);
                        ps.setString(1, actor.getFirstName());
                        ps.setString(2, actor.getLastName());
                        ps.setLong(3, actor.getId().longValue());
                    }
                    public int getBatchSize() {
                        return actors.size();
                    }
                });
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    fun batchUpdate(actors: List<Actor>): IntArray {
        return jdbcTemplate.batchUpdate(
                "update t_actor set first_name = ?, last_name = ? where id = ?",
                object: BatchPreparedStatementSetter {
                    override fun setValues(ps: PreparedStatement, i: Int) {
                        ps.setString(1, actors[i].firstName)
                        ps.setString(2, actors[i].lastName)
                        ps.setLong(3, actors[i].id)
                    }

                    override fun getBatchSize() = actors.size
                })
    }

    // ... additional methods
}
```

`InterruptibleBatchPreparedStatementSetter` `isBatchExhausted`

3.5.2. Batch Operations with a List of Objects

`JdbcTemplate` `NamedParameterJdbcTemplate`

`SqlParameterSource` `SqlParameterSourceUtils.createBatch` `String` `Map`

Java

```
public class JdbcActorDao implements ActorDao {

    private NamedParameterTemplate namedParameterJdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.namedParameterJdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
    }

    public int[] batchUpdate(List<Actor> actors) {
        return this.namedParameterJdbcTemplate.batchUpdate(
                "update t_actor set first_name = :firstName, last_name = :lastName
where id = :id",
                SqlParameterSourceUtils.createBatch(actors));
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val namedParameterJdbcTemplate = NamedParameterJdbcTemplate(dataSource)

    fun batchUpdate(actors: List<Actor>): IntArray {
        return this.namedParameterJdbcTemplate.batchUpdate(
                "update t_actor set first_name = :firstName, last_name = :lastName
where id = :id",
                SqlParameterSourceUtils.createBatch(actors));
    }

        // ... additional methods
}
```

?

?

Java

```
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public int[] batchUpdate(final List<Actor> actors) {
        List<Object[]> batch = new ArrayList<Object[]>();
        for (Actor actor : actors) {
            Object[] values = new Object[] {
                    actor.getFirstName(), actor.getLastName(), actor.getId()};
            batch.add(values);
        }
        return this.jdbcTemplate.batchUpdate(
                "update t_actor set first_name = ?, last_name = ? where id = ?",
                batch);
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    fun batchUpdate(actors: List<Actor>): IntArray {
        val batch = mutableListOf<Array<Any>>()
        for (actor in actors) {
            batch.add(arrayOf(actor.firstName, actor.lastName, actor.id))
        }
        return jdbcTemplate.batchUpdate(
                "update t_actor set first_name = ?, last_name = ? where id = ?",
batch)
    }

    // ... additional methods
}
```

`int`

`-2`

PreparedStatement

null

ParameterMetaData.getParameterType

spring.jdbc.getParameterType.ignore true

spring.properties

3.5.3. Batch Operations with Multiple Batches

batchUpdate

Collection

ParameterizedPreparedStatementSetter

Java

```
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public int[][] batchUpdate(final Collection<Actor> actors) {
        int[][] updateCounts = jdbcTemplate.batchUpdate(
                "update t_actor set first_name = ?, last_name = ? where id = ?",
                actors,
                100,
                (PreparedStatement ps, Actor actor) -> {
                    ps.setString(1, actor.getFirstName());
                    ps.setString(2, actor.getLastName());
                    ps.setLong(3, actor.getId().longValue());
                });
        return updateCounts;
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val jdbcTemplate = JdbcTemplate(dataSource)

    fun batchUpdate(actors: List<Actor>): Array<IntArray> {
        return jdbcTemplate.batchUpdate(
                    "update t_actor set first_name = ?, last_name = ? where id = ?",
                    actors, 100) { ps, argument ->
            ps.setString(1, argument.firstName)
            ps.setString(2, argument.lastName)
            ps.setLong(3, argument.id)
        }
    }

    // ... additional methods
}
```

int

-2

3.6. Simplifying JDBC Operations with the SimpleJdbc Classes

SimpleJdbcInsert SimpleJdbcCall

3.6.1. Inserting Data by Using SimpleJdbcInsert

SimpleJdbcInsert
SimpleJdbcInsert
setDataSource
SimpleJdbcInsert
withTableName fluid
SimpleJdbcInsert

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.insertActor = new SimpleJdbcInsert(dataSource).withTableName("t_actor");
    }

    public void add(Actor actor) {
        Map<String, Object> parameters = new HashMap<String, Object>(3);
        parameters.put("id", actor.getId());
        parameters.put("first_name", actor.getFirstName());
        parameters.put("last_name", actor.getLastName());
        insertActor.execute(parameters);
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val insertActor = SimpleJdbcInsert(dataSource).withTableName("t_actor")

    fun add(actor: Actor) {
        val parameters = mutableMapOf<String, Any>()
        parameters["id"] = actor.id
        parameters["first_name"] = actor.firstName
        parameters["last_name"] = actor.lastName
        insertActor.execute(parameters)
    }

    // ... additional methods
}
```

`execute` `java.util.Map` `Map`

3.6.2. Retrieving Auto-generated Keys by Using `SimpleJdbcInsert`

`id` `Actor` `SimpleJdbcInsert` `usingGeneratedKeyColumns`

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.insertActor = new SimpleJdbcInsert(dataSource)
                .withTableName("t_actor")
                .usingGeneratedKeyColumns("id");
    }

    public void add(Actor actor) {
        Map<String, Object> parameters = new HashMap<String, Object>(2);
        parameters.put("first_name", actor.getFirstName());
        parameters.put("last_name", actor.getLastName());
        Number newId = insertActor.executeAndReturnKey(parameters);
        actor.setId(newId.longValue());
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val insertActor = SimpleJdbcInsert(dataSource)
            .withTableName("t_actor").usingGeneratedKeyColumns("id")

    fun add(actor: Actor): Actor {
        val parameters = mapOf(
                "first_name" to actor.firstName,
                "last_name" to actor.lastName)
        val newId = insertActor.executeAndReturnKey(parameters);
        return actor.copy(id = newId.toLong())
    }

    // ... additional methods
}
```

`id` `Map` `executeAndReturnKey` `java.lang.Number` `java.lang.Number` `KeyHolder` `executeAndReturnKeyHolder`

3.6.3. Specifying Columns for a `SimpleJdbcInsert`

`usingColumns`

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.insertActor = new SimpleJdbcInsert(dataSource)
                .withTableName("t_actor")
                .usingColumns("first_name", "last_name")
                .usingGeneratedKeyColumns("id");
    }

    public void add(Actor actor) {
        Map<String, Object> parameters = new HashMap<String, Object>(2);
        parameters.put("first_name", actor.getFirstName());
        parameters.put("last_name", actor.getLastName());
        Number newId = insertActor.executeAndReturnKey(parameters);
        actor.setId(newId.longValue());
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val insertActor = SimpleJdbcInsert(dataSource)
            .withTableName("t_actor")
            .usingColumns("first_name", "last_name")
            .usingGeneratedKeyColumns("id")

    fun add(actor: Actor): Actor {
        val parameters = mapOf(
                "first_name" to actor.firstName,
                "last_name" to actor.lastName)
        val newId = insertActor.executeAndReturnKey(parameters);
        return actor.copy(id = newId.toLong())
    }

    // ... additional methods
}
```

3.6.4. Using `SqlParameterSource` to Provide Parameter Values

`Map`

`SqlParameterSource`

`BeanPropertySqlParameterSource`

`BeanPropertySqlParameterSource`

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.insertActor = new SimpleJdbcInsert(dataSource)
                .withTableName("t_actor")
                .usingGeneratedKeyColumns("id");
    }

    public void add(Actor actor) {
        SqlParameterSource parameters = new BeanPropertySqlParameterSource(actor);
        Number newId = insertActor.executeAndReturnKey(parameters);
        actor.setId(newId.longValue());
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val insertActor = SimpleJdbcInsert(dataSource)
            .withTableName("t_actor")
            .usingGeneratedKeyColumns("id")

    fun add(actor: Actor): Actor {
        val parameters = BeanPropertySqlParameterSource(actor)
        val newId = insertActor.executeAndReturnKey(parameters)
        return actor.copy(id = newId.toLong())
    }

    // ... additional methods
}
```

`MapSqlParameterSource` `Map` `addValue`

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcInsert insertActor;

    public void setDataSource(DataSource dataSource) {
        this.insertActor = new SimpleJdbcInsert(dataSource)
                .withTableName("t_actor")
                .usingGeneratedKeyColumns("id");
    }

    public void add(Actor actor) {
        SqlParameterSource parameters = new MapSqlParameterSource()
                .addValue("first_name", actor.getFirstName())
                .addValue("last_name", actor.getLastName());
        Number newId = insertActor.executeAndReturnKey(parameters);
        actor.setId(newId.longValue());
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val insertActor = SimpleJdbcInsert(dataSource)
            .withTableName("t_actor")
            .usingGeneratedKeyColumns("id")

    fun add(actor: Actor): Actor {
        val parameters = MapSqlParameterSource()
                    .addValue("first_name", actor.firstName)
                    .addValue("last_name", actor.lastName)
        val newId = insertActor.executeAndReturnKey(parameters)
        return actor.copy(id = newId.toLong())
    }

    // ... additional methods
}
```

3.6.5. Calling a Stored Procedure with SimpleJdbcCall

SimpleJdbcCall in out

ARRAY STRUCT

VARCHAR

DATE

first_name last_name birth_date out

```
CREATE PROCEDURE read_actor (
    IN in_id INTEGER,
    OUT out_first_name VARCHAR(100),
    OUT out_last_name VARCHAR(100),
    OUT out_birth_date DATE)
BEGIN
    SELECT first_name, last_name, birth_date
    INTO out_first_name, out_last_name, out_birth_date
    FROM t_actor where id = in_id;
END;
```

in_id id out

SimpleJdbcCall SimpleJdbcInsert

StoredProcedure

SimpleJdbcCall

DataSource

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcCall procReadActor;

    public void setDataSource(DataSource dataSource) {
        this.procReadActor = new SimpleJdbcCall(dataSource)
                .withProcedureName("read_actor");
    }

    public Actor readActor(Long id) {
        SqlParameterSource in = new MapSqlParameterSource()
                .addValue("in_id", id);
        Map out = procReadActor.execute(in);
        Actor actor = new Actor();
        actor.setId(id);
        actor.setFirstName((String) out.get("out_first_name"));
        actor.setLastName((String) out.get("out_last_name"));
        actor.setBirthDate((Date) out.get("out_birth_date"));
        return actor;
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val procReadActor = SimpleJdbcCall(dataSource)
            .withProcedureName("read_actor")


    fun readActor(id: Long): Actor {
        val source = MapSqlParameterSource().addValue("in_id", id)
        val output = procReadActor.execute(source)
        return Actor(
                id,
                output["out_first_name"] as String,
                output["out_last_name"] as String,
                output["out_birth_date"] as Date)
    }

        // ... additional methods
}
```

`SqlParameterSource`

execute Map out out_first_name out_last_name out_birth_date

execute Actor out out out

LinkedCaseInsensitiveMap JdbcTemplate setResultsMapCaseInsensitive true JdbcTemplate SimpleJdbcCall

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcCall procReadActor;

    public void setDataSource(DataSource dataSource) {
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.setResultsMapCaseInsensitive(true);
        this.procReadActor = new SimpleJdbcCall(jdbcTemplate)
                .withProcedureName("read_actor");
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private var procReadActor = SimpleJdbcCall(JdbcTemplate(dataSource).apply {
        isResultsMapCaseInsensitive = true
    }).withProcedureName("read_actor")

    // ... additional methods
}
```

out

3.6.6. Explicitly Declaring Parameters to Use for a `SimpleJdbcCall`

`SimpleJdbcCall` `declareParameters` `SqlParameter` `SqlParameter`

`withoutProcedureColumnMetaDataAccess`

`useInParameterNames`

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcCall procReadActor;

    public void setDataSource(DataSource dataSource) {
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.setResultsMapCaseInsensitive(true);
        this.procReadActor = new SimpleJdbcCall(jdbcTemplate)
                .withProcedureName("read_actor")
                .withoutProcedureColumnMetaDataAccess()
                .useInParameterNames("in_id")
                .declareParameters(
                        new SqlParameter("in_id", Types.NUMERIC),
                        new SqlOutParameter("out_first_name", Types.VARCHAR),
                        new SqlOutParameter("out_last_name", Types.VARCHAR),
                        new SqlOutParameter("out_birth_date", Types.DATE)
                );
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

        private val procReadActor = SimpleJdbcCall(JdbcTemplate(dataSource).apply {
            isResultsMapCaseInsensitive = true
        }).withProcedureName("read_actor")
                .withoutProcedureColumnMetaDataAccess()
                .useInParameterNames("in_id")
                .declareParameters(
                        SqlParameter("in_id", Types.NUMERIC),
                        SqlOutParameter("out_first_name", Types.VARCHAR),
                        SqlOutParameter("out_last_name", Types.VARCHAR),
                        SqlOutParameter("out_birth_date", Types.DATE)
    )

        // ... additional methods
}
```

3.6.7. How to Define `SqlParameters`

`SimpleJdbc` `SqlParameter` `java.sql.Types`

Java

```
new SqlParameter("in_id", Types.NUMERIC),
new SqlOutParameter("out_first_name", Types.VARCHAR),
```

Kotlin

```
SqlParameter("in_id", Types.NUMERIC),
SqlOutParameter("out_first_name", Types.VARCHAR),
```

`SqlParameter` `SqlQuery` `SqlQuery`

`SqlOutParameter` `out` `SqlInOutParameter` `InOut`

SqlParameter SqlInOutParameter
StoredProcedure

SqlOutParameter

out RowMapper
REF SqlReturnType

3.6.8. Calling a Stored Function by Using SimpleJdbcCall

withFunctionName

executeFunction

executeObject out
get_actor_name

```
CREATE FUNCTION get_actor_name (in_id INTEGER)
RETURNS VARCHAR(200) READS SQL DATA
BEGIN
    DECLARE out_name VARCHAR(200);
    SELECT concat(first_name, ' ', last_name)
        INTO out_name
        FROM t_actor where id = in_id;
    RETURN out_name;
END;
```

SimpleJdbcCall

Java

```
public class JdbcActorDao implements ActorDao {

    private JdbcTemplate jdbcTemplate;
    private SimpleJdbcCall funcGetActorName;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.setResultsMapCaseInsensitive(true);
        this.funcGetActorName = new SimpleJdbcCall(jdbcTemplate)
                .withFunctionName("get_actor_name");
    }

    public String getActorName(Long id) {
        SqlParameterSource in = new MapSqlParameterSource()
                .addValue("in_id", id);
        String name = funcGetActorName.executeFunction(String.class, in);
        return name;
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

    private val jdbcTemplate = JdbcTemplate(dataSource).apply {
        isResultsMapCaseInsensitive = true
    }
    private val funcGetActorName = SimpleJdbcCall(jdbcTemplate)
            .withFunctionName("get_actor_name")

    fun getActorName(id: Long): String {
        val source = MapSqlParameterSource().addValue("in_id", id)
        return funcGetActorName.executeFunction(String::class.java, source)
    }

    // ... additional methods
}
```

`executeFunction` `String`

3.6.9. Returning a ResultSet or REF Cursor from a SimpleJdbcCall

out

SimpleJdbcCall returningResultSet RowMapper

RowMapper

execute

t_actor

```
CREATE PROCEDURE read_all_actors()
BEGIN
 SELECT a.id, a.first_name, a.last_name, a.birth_date FROM t_actor a;
END;
```

RowMapper BeanPropertyRowMapper newInstance

Java

```
public class JdbcActorDao implements ActorDao {

    private SimpleJdbcCall procReadAllActors;

    public void setDataSource(DataSource dataSource) {
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        jdbcTemplate.setResultsMapCaseInsensitive(true);
        this.procReadAllActors = new SimpleJdbcCall(jdbcTemplate)
                .withProcedureName("read_all_actors")
                .returningResultSet("actors",
                BeanPropertyRowMapper.newInstance(Actor.class));
    }

    public List getActorsList() {
        Map m = procReadAllActors.execute(new HashMap<String, Object>(0));
        return (List) m.get("actors");
    }

    // ... additional methods
}
```

Kotlin

```
class JdbcActorDao(dataSource: DataSource) : ActorDao {

        private val procReadAllActors = SimpleJdbcCall(JdbcTemplate(dataSource).apply
{
            isResultsMapCaseInsensitive = true
        }).withProcedureName("read_all_actors")
                .returningResultSet("actors",
                        BeanPropertyRowMapper.newInstance(Actor::class.java))

    fun getActorsList(): List<Actor> {
        val m = procReadAllActors.execute(mapOf<String, Any>())
        return m["actors"] as List<Actor>
    }

    // ... additional methods
}
```

`execute` `Map`

3.7. Modeling JDBC Operations as Java Objects

`org.springframework.jdbc.object`

`StoredProcedure`
`JdbcTemplate`
`JdbcTemplate`

3.7.1. Understanding `SqlQuery`

`SqlQuery`
`newRowMapper(..)` `RowMapper`
`ResultSet`
`SqlQuery` `MappingSqlQuery`

`SqlQuery` `MappingSqlQueryWithParameters` `UpdatableSqlQuery`

3.7.2. Using `MappingSqlQuery`

`MappingSqlQuery` `mapRow(..)` `ResultSet` `t_actor` `Actor`

Java

```
public class ActorMappingQuery extends MappingSqlQuery<Actor> {

    public ActorMappingQuery(DataSource ds) {
        super(ds, "select id, first_name, last_name from t_actor where id = ?");
        declareParameter(new SqlParameter("id", Types.INTEGER));
        compile();
    }

    @Override
    protected Actor mapRow(ResultSet rs, int rowNumber) throws SQLException {
        Actor actor = new Actor();
        actor.setId(rs.getLong("id"));
        actor.setFirstName(rs.getString("first_name"));
        actor.setLastName(rs.getString("last_name"));
        return actor;
    }
}
```

Kotlin

```
class ActorMappingQuery(ds: DataSource) : MappingSqlQuery<Actor>(ds, "select id,
first_name, last_name from t_actor where id = ?") {

    init {
        declareParameter(SqlParameter("id", Types.INTEGER))
        compile()
    }

    override fun mapRow(rs: ResultSet, rowNumber: Int) = Actor(
            rs.getLong("id"),
            rs.getString("first_name"),
            rs.getString("last_name")
    )
}
```

`MappingSqlQuery` `Actor` `DataSource` `DataSource` `PreparedStatement`

`declareParameter` `SqlParameter` `SqlParameter` `java.sql.Types` `compile()`

Java

```
private ActorMappingQuery actorMappingQuery;

@Autowired
public void setDataSource(DataSource dataSource) {
    this.actorMappingQuery = new ActorMappingQuery(dataSource);
}

public Customer getCustomer(Long id) {
    return actorMappingQuery.findObject(id);
}
```

Kotlin

```
private val actorMappingQuery = ActorMappingQuery(dataSource)

fun getCustomer(id: Long) = actorMappingQuery.findObject(id)
```

`id` `findObject` `id` `execute`

Java

```
public List<Actor> searchForActors(int age, String namePattern) {
    List<Actor> actors = actorSearchMappingQuery.execute(age, namePattern);
    return actors;
}
```

Kotlin

```
fun searchForActors(age: Int, namePattern: String) =
            actorSearchMappingQuery.execute(age, namePattern)
```

3.7.3. Using `SqlUpdate`

`SqlUpdate` `RdbmsOperation` `update(..)` `execute(..)`

SQLUpdate

SqlUpdate

execute

Java

```
import java.sql.Types;
import javax.sql.DataSource;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.SqlUpdate;

public class UpdateCreditRating extends SqlUpdate {

    public UpdateCreditRating(DataSource ds) {
        setDataSource(ds);
        setSql("update customer set credit_rating = ? where id = ?");
        declareParameter(new SqlParameter("creditRating", Types.NUMERIC));
        declareParameter(new SqlParameter("id", Types.NUMERIC));
        compile();
    }

    /**
     * @param id for the Customer to be updated
     * @param rating the new value for credit rating
     * @return number of rows updated
     */
    public int execute(int id, int rating) {
        return update(rating, id);
    }
}
```

Kotlin

```
import java.sql.Types
import javax.sql.DataSource
import org.springframework.jdbc.core.SqlParameter
import org.springframework.jdbc.object.SqlUpdate

class UpdateCreditRating(ds: DataSource) : SqlUpdate() {

    init {
        setDataSource(ds)
        sql = "update customer set credit_rating = ? where id = ?"
        declareParameter(SqlParameter("creditRating", Types.NUMERIC))
        declareParameter(SqlParameter("id", Types.NUMERIC))
        compile()
    }

    /**
    * @param id for the Customer to be updated
    * @param rating the new value for credit rating
    * @return number of rows updated
    */
    fun execute(id: Int, rating: Int): Int {
        return update(rating, id)
    }
}
```

3.7.4. Using `StoredProcedure`

`StoredProcedure` `abstract` `execute(..)` `protected` `sql` `StoredProcedure` `SqlParameter`

Java

```
new SqlParameter("in_id", Types.NUMERIC),
new SqlOutParameter("out_first_name", Types.VARCHAR),
```

Kotlin

```
SqlParameter("in_id", Types.NUMERIC),
SqlOutParameter("out_first_name", Types.VARCHAR),
```

java.sql.Types

SqlParameter

SqlQuery

SqlQuery

SqlOutParameter out

SqlInOutParameter InOut

in

in

out RowMapper

REF SqlReturnType

StoredProcedure sysdate()

StoredProcedure StoredProcedure

StoredProcedure

SqlOutParameter execute()

Map Map

Java

```
import java.sql.Types;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.object.StoredProcedure;

public class StoredProcedureDao {

    private GetSysdateProcedure getSysdate;

    @Autowired
    public void init(DataSource dataSource) {
        this.getSysdate = new GetSysdateProcedure(dataSource);
    }

    public Date getSysdate() {
        return getSysdate.execute();
    }

    private class GetSysdateProcedure extends StoredProcedure {

        private static final String SQL = "sysdate";

        public GetSysdateProcedure(DataSource dataSource) {
            setDataSource(dataSource);
            setFunction(true);
            setSql(SQL);
            declareParameter(new SqlOutParameter("date", Types.DATE));
            compile();
        }

        public Date execute() {
            // the 'sysdate' sproc has no input parameters, so an empty Map is
supplied...
            Map<String, Object> results = execute(new HashMap<String, Object>());
            Date sysdate = (Date) results.get("date");
            return sysdate;
        }
    }

}
```

Kotlin

```
import java.sql.Types
import java.util.Date
import java.util.Map
import javax.sql.DataSource
import org.springframework.jdbc.core.SqlOutParameter
import org.springframework.jdbc.object.StoredProcedure

class StoredProcedureDao(dataSource: DataSource) {

    private val SQL = "sysdate"

    private val getSysdate = GetSysdateProcedure(dataSource)

    val sysdate: Date
        get() = getSysdate.execute()

    private inner class GetSysdateProcedure(dataSource: DataSource) :
StoredProcedure() {

        init {
            setDataSource(dataSource)
            isFunction = true
            sql = SQL
            declareParameter(SqlOutParameter("date", Types.DATE))
            compile()
        }

        fun execute(): Date {
            // the 'sysdate' sproc has no input parameters, so an empty Map is
supplied...
            val results = execute(mutableMapOf<String, Any>())
            return results["date"] as Date
        }
    }
}
```

`StoredProcedure`

Java

```
import java.util.HashMap;
import java.util.Map;
import javax.sql.DataSource;
import oracle.jdbc.OracleTypes;
import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.object.StoredProcedure;

public class TitlesAndGenresStoredProcedure extends StoredProcedure {

    private static final String SPROC_NAME = "AllTitlesAndGenres";

    public TitlesAndGenresStoredProcedure(DataSource dataSource) {
        super(dataSource, SPROC_NAME);
        declareParameter(new SqlOutParameter("titles", OracleTypes.CURSOR, new
TitleMapper()));
        declareParameter(new SqlOutParameter("genres", OracleTypes.CURSOR, new
GenreMapper()));
        compile();
    }

    public Map<String, Object> execute() {
        // again, this sproc has no input parameters, so an empty Map is supplied
        return super.execute(new HashMap<String, Object>());
    }
}
```

Kotlin

```
import java.util.HashMap
import javax.sql.DataSource
import oracle.jdbc.OracleTypes
import org.springframework.jdbc.core.SqlOutParameter
import org.springframework.jdbc.object.StoredProcedure

class TitlesAndGenresStoredProcedure(dataSource: DataSource) :
StoredProcedure(dataSource, SPROC_NAME) {

    companion object {
        private const val SPROC_NAME = "AllTitlesAndGenres"
    }

    init {
        declareParameter(SqlOutParameter("titles", OracleTypes.CURSOR, TitleMapper()))
        declareParameter(SqlOutParameter("genres", OracleTypes.CURSOR, GenreMapper()))
        compile()
    }

    fun execute(): Map<String, Any> {
        // again, this sproc has no input parameters, so an empty Map is supplied
        return super.execute(HashMap<String, Any>())
    }
}
```

declareParameter(..)
TitlesAndGenresStoredProcedure RowMapper

RowMapper

TitleMapper ResultSet Title
ResultSet

Java

```
import java.sql.ResultSet;
import java.sql.SQLException;
import com.foo.domain.Title;
import org.springframework.jdbc.core.RowMapper;

public final class TitleMapper implements RowMapper<Title> {

    public Title mapRow(ResultSet rs, int rowNum) throws SQLException {
        Title title = new Title();
        title.setId(rs.getLong("id"));
        title.setName(rs.getString("name"));
        return title;
    }
}
```

Kotlin

```
import java.sql.ResultSet
import com.foo.domain.Title
import org.springframework.jdbc.core.RowMapper

class TitleMapper : RowMapper<Title> {

    override fun mapRow(rs: ResultSet, rowNum: Int) =
            Title(rs.getLong("id"), rs.getString("name"))
}
```

`GenreMapper` `ResultSet` `Genre` `ResultSet`

Java

```
import java.sql.ResultSet;
import java.sql.SQLException;
import com.foo.domain.Genre;
import org.springframework.jdbc.core.RowMapper;

public final class GenreMapper implements RowMapper<Genre> {

    public Genre mapRow(ResultSet rs, int rowNum) throws SQLException {
        return new Genre(rs.getString("name"));
    }
}
```

Kotlin

```
import java.sql.ResultSet
import com.foo.domain.Genre
import org.springframework.jdbc.core.RowMapper

class GenreMapper : RowMapper<Genre> {

    override fun mapRow(rs: ResultSet, rowNum: Int): Genre {
        return Genre(rs.getString("name"))
    }
}
```

`execute(..)`

`execute(Map)`

Java

```
import java.sql.Types;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import javax.sql.DataSource;
import oracle.jdbc.OracleTypes;
import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.StoredProcedure;

public class TitlesAfterDateStoredProcedure extends StoredProcedure {

    private static final String SPROC_NAME = "TitlesAfterDate";
    private static final String CUTOFF_DATE_PARAM = "cutoffDate";

    public TitlesAfterDateStoredProcedure(DataSource dataSource) {
        super(dataSource, SPROC_NAME);
        declareParameter(new SqlParameter(CUTOFF_DATE_PARAM, Types.DATE);
        declareParameter(new SqlOutParameter("titles", OracleTypes.CURSOR, new
TitleMapper()));
        compile();
    }

    public Map<String, Object> execute(Date cutoffDate) {
        Map<String, Object> inputs = new HashMap<String, Object>();
        inputs.put(CUTOFF_DATE_PARAM, cutoffDate);
        return super.execute(inputs);
    }
}
```

Kotlin

```
import java.sql.Types
import java.util.Date
import javax.sql.DataSource
import oracle.jdbc.OracleTypes
import org.springframework.jdbc.core.SqlOutParameter
import org.springframework.jdbc.core.SqlParameter
import org.springframework.jdbc.object.StoredProcedure

class TitlesAfterDateStoredProcedure(dataSource: DataSource) :
StoredProcedure(dataSource, SPROC_NAME) {

    companion object {
        private const val SPROC_NAME = "TitlesAfterDate"
        private const val CUTOFF_DATE_PARAM = "cutoffDate"
    }

    init {
        declareParameter(SqlParameter(CUTOFF_DATE_PARAM, Types.DATE))
        declareParameter(SqlOutParameter("titles", OracleTypes.CURSOR, TitleMapper()))
        compile()
    }

    fun execute(cutoffDate: Date) = super.execute(
            mapOf<String, Any>(CUTOFF_DATE_PARAM to cutoffDate))
}
```

3.8. Common Problems with Parameter and Data Value Handling

3.8.1. Providing SQL Type Information for Parameters

`NULL`

`JdbcTemplate`

`int`

`java.sql.Types`

`SqlParameterValue`

SqlParameterSource BeanPropertySqlParameterSource MapSqlParameterSource

3.8.2. Handling BLOB and CLOB objects

JdbcTemplate SimpleJdbc LobHandler LobHandler LobCreator getLobCreator

LobCreator LobHandler

- byte[] getBlobAsBytes setBlobAsBytes
- InputStream getBlobAsBinaryStream setBlobAsBinaryStream

- String getClobAsString setClobAsString
- InputStream getClobAsAsciiStream setClobAsAsciiStream
- Reader getClobAsCharacterStream setClobAsCharacterStream

JdbcTemplate AbstractLobCreatingPreparedStatementCallback setValues LobCreator

lobHandler DefaultLobHandler

Java

```
final File blobIn = new File("spring2004.jpg");
final InputStream blobIs = new FileInputStream(blobIn);
final File clobIn = new File("large.txt");
final InputStream clobIs = new FileInputStream(clobIn);
final InputStreamReader clobReader = new InputStreamReader(clobIs);

jdbcTemplate.execute(
    "INSERT INTO lob_table (id, a_clob, a_blob) VALUES (?, ?, ?)",
    new AbstractLobCreatingPreparedStatementCallback(lobHandler) {  ①
        protected void setValues(PreparedStatement ps, LobCreator lobCreator) throws
SQLException {
            ps.setLong(1, 1L);
            lobCreator.setClobAsCharacterStream(ps, 2, clobReader,
(int)clobIn.length());  ②
            lobCreator.setBlobAsBinaryStream(ps, 3, blobIs, (int)blobIn.length());  ③
        }
    }
);

blobIs.close();
clobReader.close();
```

① lobHandler DefaultLobHandler

② setClobAsCharacterStream

③ setBlobAsBinaryStream

Kotlin

```
val blobIn = File("spring2004.jpg")
val blobIs = FileInputStream(blobIn)
val clobIn = File("large.txt")
val clobIs = FileInputStream(clobIn)
val clobReader = InputStreamReader(clobIs)

jdbcTemplate.execute(
        "INSERT INTO lob_table (id, a_clob, a_blob) VALUES (?, ?, ?)",
        object: AbstractLobCreatingPreparedStatementCallback(lobHandler) {  ①
            override fun setValues(ps: PreparedStatement, lobCreator: LobCreator) {
                ps.setLong(1, 1L)
                lobCreator.setClobAsCharacterStream(ps, 2, clobReader,
clobIn.length().toInt())  ②
                lobCreator.setBlobAsBinaryStream(ps, 3, blobIs,
blobIn.length().toInt())  ③
            }
        }
)
blobIs.close()
clobReader.close()
```

① lobHandler DefaultLobHandler

② setClobAsCharacterStream

③ setBlobAsBinaryStream

setBlobAsBinaryStream setClobAsAsciiStream setClobAsCharacterStream LobCreator DefaultLobHandler.getLobCreator() contentLength DefaultLobHandler

JdbcTemplate lobHandler DefaultLobHandler

Java

```
List<Map<String, Object>> l = jdbcTemplate.query("select id, a_clob, a_blob from
lob_table",
    new RowMapper<Map<String, Object>>() {
        public Map<String, Object> mapRow(ResultSet rs, int i) throws SQLException {
            Map<String, Object> results = new HashMap<String, Object>();
            String clobText = lobHandler.getClobAsString(rs, "a_clob");  ①
            results.put("CLOB", clobText);
            byte[] blobBytes = lobHandler.getBlobAsBytes(rs, "a_blob");  ②
            results.put("BLOB", blobBytes);
            return results;
        }
    });
```

① `getClobAsString`

② `getBlobAsBytes`

Kotlin

```
val l = jdbcTemplate.query("select id, a_clob, a_blob from lob_table") { rs, _ ->
    val clobText = lobHandler.getClobAsString(rs, "a_clob")  ①
    val blobBytes = lobHandler.getBlobAsBytes(rs, "a_blob")  ②
    mapOf("CLOB" to clobText, "BLOB" to blobBytes)
}
```

① `getClobAsString`

② `getBlobAsBytes`

3.8.3. Passing in Lists of Values for IN Clause

`select * from T_ACTOR where id in (1, 2, 3)`

`NamedParameterJdbcTemplate` `JdbcTemplate` `java.util.List`

`in`

`java.util.List` `in` `select * from`

T_ACTOR where (id, last_name) in ((1, 'Johnson'), (2, 'Harrop'\))

3.8.4. Handling Complex Types for Stored Procedure Calls

SqlReturnType
SqlTypeValue

SqlReturnType getTypeValue
SqlOutParameter
STRUCT ITEM_TYPE

Java

```
public class TestItemStoredProcedure extends StoredProcedure {

    public TestItemStoredProcedure(DataSource dataSource) {
        // ...
        declareParameter(new SqlOutParameter("item", OracleTypes.STRUCT, "ITEM_TYPE",
            (CallableStatement cs, int colIndx, int sqlType, String typeName) -> {
                STRUCT struct = (STRUCT) cs.getObject(colIndx);
                Object[] attr = struct.getAttributes();
                TestItem item = new TestItem();
                item.setId(((Number) attr[0]).longValue());
                item.setDescription((String) attr[1]);
                item.setExpirationDate((java.util.Date) attr[2]);
                return item;
            }));
        // ...
    }
```

Kotlin

```
class TestItemStoredProcedure(dataSource: DataSource) : StoredProcedure() {

    init {
        // ...
        declareParameter(SqlOutParameter("item", OracleTypes.STRUCT, "ITEM_TYPE") {
cs, colIndx, sqlType, typeName ->
            val struct = cs.getObject(colIndx) as STRUCT
            val attr = struct.getAttributes()
            TestItem((attr[0] as Long, attr[1] as String, attr[2] as Date)
        })
        // ...
    }
}
```

SqlTypeValue TestItem
SqlTypeValue createTypeValue

StructDescriptor ArrayDescriptor
StructDescriptor

Java

```
final TestItem testItem = new TestItem(123L, "A test item",
        new SimpleDateFormat("yyyy-M-d").parse("2010-12-31"));

SqlTypeValue value = new AbstractSqlTypeValue() {
    protected Object createTypeValue(Connection conn, int sqlType, String typeName)
throws SQLException {
        StructDescriptor itemDescriptor = new StructDescriptor(typeName, conn);
        Struct item = new STRUCT(itemDescriptor, conn,
        new Object[] {
            testItem.getId(),
            testItem.getDescription(),
            new java.sql.Date(testItem.getExpirationDate().getTime())
        });
        return item;
    }
};
```

Kotlin

```
val (id, description, expirationDate) = TestItem(123L, "A test item",
        SimpleDateFormat("yyyy-M-d").parse("2010-12-31"))

val value = object : AbstractSqlTypeValue() {
    override fun createTypeValue(conn: Connection, sqlType: Int, typeName: String?):
Any {
        val itemDescriptor = StructDescriptor(typeName, conn)
        return STRUCT(itemDescriptor, conn,
                arrayOf(id, description, java.sql.Date(expirationDate.time)))
    }
}
```

SqlTypeValue Map execute

SqlTypeValue
ARRAY
SqlTypeValue ARRAY
ARRAY

Java

```
final Long[] ids = new Long[] {1L, 2L};

SqlTypeValue value = new AbstractSqlTypeValue() {
    protected Object createTypeValue(Connection conn, int sqlType, String typeName)
throws SQLException {
        ArrayDescriptor arrayDescriptor = new ArrayDescriptor(typeName, conn);
        ARRAY idArray = new ARRAY(arrayDescriptor, conn, ids);
        return idArray;
    }
};
```

Kotlin

```
class TestItemStoredProcedure(dataSource: DataSource) : StoredProcedure() {

    init {
        val ids = arrayOf(1L, 2L)
        val value = object : AbstractSqlTypeValue() {
            override fun createTypeValue(conn: Connection, sqlType: Int, typeName:
String?): Any {
                val arrayDescriptor = ArrayDescriptor(typeName, conn)
                return ARRAY(arrayDescriptor, conn, ids)
            }
        }
    }
}
```

3.9. Embedded Database Support

`org.springframework.jdbc.datasource.embedded`

`DataSource`

3.9.1. Why Use an Embedded Database?

3.9.2. Creating an Embedded Database by Using Spring XML

`ApplicationContext`

`embedded-database` `spring-jdbc`

```
<jdbc:embedded-database id="dataSource" generate-name="true">
    <jdbc:script location="classpath:schema.sql"/>
    <jdbc:script location="classpath:test-data.sql"/>
</jdbc:embedded-database>
```

`schema.sql` `test-data.sql`

`javax.sql.DataSource`

3.9.3. Creating an Embedded Database Programmatically

`EmbeddedDatabaseBuilder`

Java

```
EmbeddedDatabase db = new EmbeddedDatabaseBuilder()
        .generateUniqueName(true)
        .setType(H2)
        .setScriptEncoding("UTF-8")
        .ignoreFailedDrops(true)
        .addScript("schema.sql")
        .addScripts("user_data.sql", "country_data.sql")
        .build();

// perform actions against the db (EmbeddedDatabase extends javax.sql.DataSource)

db.shutdown()
```

Kotlin

```
val db = EmbeddedDatabaseBuilder()
        .generateUniqueName(true)
        .setType(H2)
        .setScriptEncoding("UTF-8")
        .ignoreFailedDrops(true)
        .addScript("schema.sql")
        .addScripts("user_data.sql", "country_data.sql")
        .build()

// perform actions against the db (EmbeddedDatabase extends javax.sql.DataSource)

db.shutdown()
```

`EmbeddedDatabaseBuilder`

`EmbeddedDatabaseBuilder`

Java

```
@Configuration
public class DataSourceConfig {

    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
                .generateUniqueName(true)
                .setType(H2)
                .setScriptEncoding("UTF-8")
                .ignoreFailedDrops(true)
                .addScript("schema.sql")
                .addScripts("user_data.sql", "country_data.sql")
                .build();
    }
}
```

Kotlin

```
@Configuration
class DataSourceConfig {

    @Bean
    fun dataSource(): DataSource {
        return EmbeddedDatabaseBuilder()
                .generateUniqueName(true)
                .setType(H2)
                .setScriptEncoding("UTF-8")
                .ignoreFailedDrops(true)
                .addScript("schema.sql")
                .addScripts("user_data.sql", "country_data.sql")
                .build()
    }
}
```

3.9.4. Selecting the Embedded Database Type

Using HSQL

`type` `embedded-database` `HSQL` `setType(EmbeddedDatabaseType)` `EmbeddedDatabaseType.HSQL`

Using H2

`type` `embedded-database` `H2` `setType(EmbeddedDatabaseType)` `EmbeddedDatabaseType.H2`

Using Derby

`type` `embedded-database` `DERBY` `setType(EmbeddedDatabaseType)` `EmbeddedDatabaseType.DERBY`

3.9.5. Testing Data Access Logic with an Embedded Database

`ApplicationContext`

Java

```
public class DataAccessIntegrationTestTemplate {

    private EmbeddedDatabase db;

    @BeforeEach
    public void setUp() {
        // creates an HSQL in-memory database populated from default scripts
        // classpath:schema.sql and classpath:data.sql
        db = new EmbeddedDatabaseBuilder()
                .generateUniqueName(true)
                .addDefaultScripts()
                .build();
    }

    @Test
    public void testDataAccess() {
        JdbcTemplate template = new JdbcTemplate(db);
        template.query( /* ... */ );
    }

    @AfterEach
    public void tearDown() {
        db.shutdown();
    }

}
```

Kotlin

```
class DataAccessIntegrationTestTemplate {

    private lateinit var db: EmbeddedDatabase

    @BeforeEach
    fun setUp() {
        // creates an HSQL in-memory database populated from default scripts
        // classpath:schema.sql and classpath:data.sql
        db = EmbeddedDatabaseBuilder()
                .generateUniqueName(true)
                .addDefaultScripts()
                .build()
    }

    @Test
    fun testDataAccess() {
        val template = JdbcTemplate(db)
        template.query( /* ... */)
    }

    @AfterEach
    fun tearDown() {
        db.shutdown()
    }
}
```

3.9.6. Generating Unique Names for Embedded Databases

@Configuration

ApplicationContext

EmbeddedDatabaseFactory

<jdbc:embedded-database>

EmbeddedDatabaseBuilder

testdb

<jdbc:embedded-database>

id

dataSource

`EmbeddedDatabaseFactory.setGenerateUniqueDatabaseName()`

`EmbeddedDatabaseBuilder.generateUniqueName()`

`<jdbc:embedded-database generate-name="true" … >`

3.9.7. Extending the Embedded Database Support

`EmbeddedDatabaseConfigurer`

`DataSourceFactory` `DataSource`

3.10. Initializing a `DataSource`

`org.springframework.jdbc.datasource.init`
`DataSource`
`DataSource`

3.10.1. Initializing a Database by Using Spring XML

`DataSource`

`initialize-database` `spring-jdbc`

```
<jdbc:initialize-database data-source="dataSource">
    <jdbc:script location="classpath:com/foo/sql/db-schema.sql"/>
    <jdbc:script location="classpath:com/foo/sql/db-test-data.sql"/>
</jdbc:initialize-database>
```

`classpath*:/com/foo/**/sql/*-data.sql`

```
<jdbc:initialize-database data-source="dataSource"
    enabled="#{systemProperties.INITIALIZE_DATABASE}"> ①
    <jdbc:script location="..."/>
</jdbc:initialize-database>
```

① enabled INITIALIZE_DATABASE

```
<jdbc:initialize-database data-source="dataSource" ignore-failures="DROPS">
    <jdbc:script location="..."/>
</jdbc:initialize-database>
```

DROP

DROP

DROP … IF EXISTS

DROP

CREATE

ignore-failures NONE DROPS ALL

; ;

```
<jdbc:initialize-database data-source="dataSource" separator="@@"> ①
    <jdbc:script location="classpath:com/myapp/sql/db-schema.sql" separator=";"/> ②
    <jdbc:script location="classpath:com/myapp/sql/db-test-data-1.sql"/>
    <jdbc:script location="classpath:com/myapp/sql/db-test-data-2.sql"/>
</jdbc:initialize-database>
```

① @@

② db-schema.sql ;

test-data @@ db-schema.sql

; @@

db-schema

DataSourceInitializer

Initialization of Other Components that Depend on the Database

DataSource
init-method @PostConstruct
afterPropertiesSet()
InitializingBean

Lifecycle
SmartLifecycle
SmartLifecycle autoStartup Lifecycle
ConfigurableApplicationContext.start()

ApplicationEvent
ContextRefreshedEvent

SmartLifecycle

BeanFactory

<import/>

DataSource
ApplicationContext
DataSource

Chapter 4. Object Relational Mapping (ORM) Data Access

4.1. Introduction to ORM with Spring

Easier testing.

`SessionFactory` `DataSource`

Common data access exceptions.

`DataAccessException`

General resource management.

`SessionFactory` `EntityManagerFactory` `DataSource`

`Session`

`Session`

`Session`

`SessionFactory`

Integrated transaction management.

`@Transactional`

4.2. General ORM Integration Considerations

4.2.1. Resource and Transaction Management

`JdbcTemplate`

`SQLException`

`DataAccessException`

`JdbcTemplate`

4.2.2. Exception Translation

`HibernateException` `PersistenceException` `IllegalArgumentException` `IllegalStateException`

`@Repository`

Java

```
@Repository
public class ProductDaoImpl implements ProductDao {

    // class body here...

}
```

Kotlin

```
@Repository
class ProductDaoImpl : ProductDao {
    // class body here...
}
```

```
<beans>

    <!-- Exception translation bean post processor -->
    <bean
class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor
"/>

    <bean id="myProductDao" class="product.ProductDaoImpl"/>

</beans>
```

`PersistenceExceptionTranslator` `@Repository`

4.3. Hibernate

4.3.1. `SessionFactory` Setup in a Spring Container

`DataSource` `SessionFactory`

`DataSource` `SessionFactory`

```
<beans>

    <bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:hsql://localhost:9001"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
    </bean>

    <bean id="mySessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <property name="dataSource" ref="myDataSource"/>
        <property name="mappingResources">
            <list>
                <value>product.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <value>
                hibernate.dialect=org.hibernate.dialect.HSQLDialect
            </value>
        </property>
    </bean>

</beans>
```

BasicDataSource DataSource

```
<beans>
    <jee:jndi-lookup id="myDataSource" jndi-name="java:comp/env/jdbc/myds"/>
</beans>
```

SessionFactory JndiObjectFactoryBean <jee:jndi-lookup>

LocalSessionFactoryBuilder

@Bean FactoryBean

LocalSessionFactoryBean LocalSessionFactoryBuilder

SimpleAsyncTaskExecutor LocalSessionFactoryBean
bootstrapExecutor LocalSessionFactoryBuilder
buildSessionFactory

EntityManagerFactory

4.3.2. Implementing DAOs Based on the Plain Hibernate API

Session
Session

Java

```
public class ProductDaoImpl implements ProductDao {

    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    public Collection loadProductsByCategory(String category) {
        return this.sessionFactory.getCurrentSession()
                .createQuery("from test.Product product where product.category=?")
                .setParameter(0, category)
                .list();
    }
}
```

Kotlin

```
class ProductDaoImpl(private val sessionFactory: SessionFactory) : ProductDao {

    fun loadProductsByCategory(category: String): Collection<*> {
        return sessionFactory.currentSession
                .createQuery("from test.Product product where product.category=?")
                .setParameter(0, category)
                .list()
    }
}
```

SessionFactory

static HibernateUtil

static

HibernateTemplate

setSessionFactory(..)

```
<beans>

    <bean id="myProductDao" class="product.ProductDaoImpl">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>

</beans>
```

HibernateException

LocalSessionFactoryBean

SessionFactory.getCurrentSession()

Session

HibernateTransactionManager

Session

JtaTransactionManager

4.3.3. Declarative Transaction Demarcation

`@Transactional`

Java

```
public class ProductServiceImpl implements ProductService {

    private ProductDao productDao;

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    @Transactional
    public void increasePriceOfAllProductsInCategory(final String category) {
        List productsToChange = this.productDao.loadProductsByCategory(category);
        // ...
    }

    @Transactional(readOnly = true)
    public List<Product> findAllProducts() {
        return this.productDao.findAllProducts();
    }
}
```

Kotlin

```
class ProductServiceImpl(private val productDao: ProductDao) : ProductService {

    @Transactional
    fun increasePriceOfAllProductsInCategory(category: String) {
        val productsToChange = productDao.loadProductsByCategory(category)
        // ...
    }

    @Transactional(readOnly = true)
    fun findAllProducts() = productDao.findAllProducts()
}
```

`PlatformTransactionManager`

`<tx:annotation-driven/>` `@Transactional`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
        https://www.springframework.org/schema/tx/spring-tx.xsd
        http://www.springframework.org/schema/aop
        https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- SessionFactory, DataSource, etc. omitted -->

    <bean id="transactionManager"
            class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <tx:annotation-driven/>

    <bean id="myProductService" class="product.SimpleProductService">
        <property name="productDao" ref="myProductDao"/>
    </bean>

</beans>
```

4.3.4. Programmatic Transaction Demarcation

PlatformTransactionManager setTransactionManager(..) productDAO setProductDao(..)

```
<beans>

    <bean id="myTxManager"
class="org.springframework.orm.hibernate5.HibernateTransactionManager">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>

    <bean id="myProductService" class="product.ProductServiceImpl">
        <property name="transactionManager" ref="myTxManager"/>
        <property name="productDao" ref="myProductDao"/>
    </bean>

</beans>
```

Java

```
public class ProductServiceImpl implements ProductService {

    private TransactionTemplate transactionTemplate;
    private ProductDao productDao;

    public void setTransactionManager(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
    }

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    public void increasePriceOfAllProductsInCategory(final String category) {
        this.transactionTemplate.execute(new TransactionCallbackWithoutResult() {
            public void doInTransactionWithoutResult(TransactionStatus status) {
                List productsToChange =
this.productDao.loadProductsByCategory(category);
                // do the price increase...
            }
        });
    }
}
```

Kotlin

```
class ProductServiceImpl(transactionManager: PlatformTransactionManager,
                         private val productDao: ProductDao) : ProductService {

    private val transactionTemplate = TransactionTemplate(transactionManager)

    fun increasePriceOfAllProductsInCategory(category: String) {
        transactionTemplate.execute {
            val productsToChange = productDao.loadProductsByCategory(category)
            // do the price increase...
        }
    }
}
```

TransactionInterceptor
TransactionTemplate
TransactionTemplate
TransactionStatus
TransactionInterceptor

4.3.5. Transaction Management Strategies

TransactionTemplate TransactionInterceptor PlatformTransactionManager HibernateTransactionManager SessionFactory ThreadLocal Session JtaTransactionManager PlatformTransactionManager

JtaTransactionManager LocalSessionFactoryBean SessionFactory

JtaTransactionManager

HibernateTransactionManager JtaTransactionManager

HibernateTransactionManager Connection DataSource

HibernateTransactionManager SessionFactory DataSource dataSource LocalSessionFactoryBean DataSource dataSource HibernateTransactionManager

4.3.6. Comparing Container-managed and Locally Defined Resources

SessionFactory

SessionFactory SessionFactory SessionFactory

DataSource

TransactionManagerLookup

SessionFactory

DataSource

SessionFactory

HibernateTransactionManager JtaTransactionManager

SessionFactory

4.3.7. Spurious Application Server Warnings with Hibernate

XADataSource

PlatformTransactionManager

```
java.sql.SQLException: The transaction is no longer active - status: 'Committed'. No
further JDBC access is allowed within this transaction.
```

PlatformTransactionManager

PlatformTransactionManager

JndiObjectFactoryBean <jee:jndi-lookup>

JtaTransactionManager

PlatformTransactionManager

jtaTransactionManager LocalSessionFactoryBean.

PlatformTransactionManager

JtaTransactionManager

PlatformTransactionManager

TransactionManagerLookup

PlatformTransactionManager

PlatformTransactionManager

JtaTransactionManager
afterCompletion

afterTransactionCompletion
close()
close()

Connection.close()
Connection

PlatformTransactionManager

JtaTransactionManager
beforeCompletion

Session

afterCompletion

4.4. JPA

org.springframework.orm.jpa

4.4.1. Three Options for JPA Setup in a Spring Environment

EntityManagerFactory

LocalEntityManagerFactoryBean

LocalContainerEntityManagerFactoryBean

Using LocalEntityManagerFactoryBean

LocalEntityManagerFactoryBean EntityManagerFactory

PersistenceProvider

```
<beans>
    <bean id="myEmf"
class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="myPersistenceUnit"/>
    </bean>
</beans>
```

DataSource

Obtaining an EntityManagerFactory from JNDI

EntityManagerFactory

```
<beans>
    <jee:jndi-lookup id="myEmf" jndi-name="persistence/myPersistenceUnit"/>
</beans>
```

META-INF/persistence.xml persistence-unit-ref

web.xml

DataSource

META-INF/persistence.xml EntityManager

EntityManagerFactory

JtaTransactionManager

@PersistenceUnit @PersistenceContext

Using LocalContainerEntityManagerFactoryBean

LocalSessionFactoryBean
LocalContainerEntityManagerFactoryBean

LocalContainerEntityManagerFactoryBean EntityManagerFactory

LocalContainerEntityManagerFactoryBean PersistenceUnitInfo persistence.xml dataSourceLookup loadTimeWeaver

LocalContainerEntityManagerFactoryBean

```
<beans>
    <bean id="myEmf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="someDataSource"/>
        <property name="loadTimeWeaver">
            <bean
class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
        </property>
    </bean>
</beans>
```

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
    <persistence-unit name="myUnit" transaction-type="RESOURCE_LOCAL">
        <mapping-file>META-INF/orm.xml</mapping-file>
        <exclude-unlisted-classes/>
    </persistence-unit>
</persistence>
```

<exclude-unlisted-classes/>

<exclude-unlisted-classes>true</exclude-unlisted-classes/> <exclude-unlisted-classes>false</exclude-unlisted-classes/>

exclude-unlisted-classes

LocalContainerEntityManagerFactoryBean

DataSource

EntityManagerFactory

persistenceXmlLocation LocalContainerEntityManagerFactoryBean

META-INF/persistence.xml

When is load-time weaving required?

LoadTimeWeaver ClassTransformer

ClassTransformers

LoadTimeWeaver

ClassTransformer

LoadTimeWeaver

LoadTimeWeaver

@EnableLoadTimeWeaving context:load-time-weaver

LocalContainerEntityManagerFactoryBean

```
<context:load-time-weaver/>
<bean id="emf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    ...
</bean>
```

`loadTimeWeaver`

```
<bean id="emf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="loadTimeWeaver">
        <bean
class="org.springframework.instrument.classloading.ReflectiveLoadTimeWeaver"/>
    </property>
</bean>
```

Dealing with Multiple Persistence Units

`PersistenceUnitManager`

`META-INF/persistence.xml`

```
<bean id="pum"
class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenceUnitManager">
    <property name="persistenceXmlLocations">
        <list>
            <value>org/springframework/orm/jpa/domain/persistence-multi.xml</value>
            <value>classpath:/my/package/**/custom-persistence.xml</value>
            <value>classpath*:META-INF/persistence.xml</value>
        </list>
    </property>
    <property name="dataSources">
        <map>
            <entry key="localDataSource" value-ref="local-db"/>
            <entry key="remoteDataSource" value-ref="remote-db"/>
        </map>
    </property>
    <!-- if no datasource is specified, use this one -->
    <property name="defaultDataSource" ref="remoteDataSource"/>
</bean>

<bean id="emf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitManager" ref="pum"/>
    <property name="persistenceUnitName" value="myCustomUnit"/>
</bean>
```

`PersistenceUnitInfo`

`PersistenceUnitPostProcessor`
`PersistenceUnitManager`
`LocalContainerEntityManagerFactoryBean`

Background Bootstrapping

`LocalContainerEntityManagerFactoryBean`
`bootstrapExecutor`

```
<bean id="emf"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="bootstrapExecutor">
        <bean class="org.springframework.core.task.SimpleAsyncTaskExecutor"/>
    </property>
</bean>
```

`EntityManagerFactory`
`EntityManagerFactoryInfo`

`createEntityManager`

4.4.2. Implementing DAOs Based on JPA: EntityManagerFactory and EntityManager

EntityManagerFactory EntityManager
EntityManager EntityManager
EntityManager
EntityManager

EntityManagerFactory EntityManager @PersistenceUnit @PersistenceContext PersistenceAnnotationBeanPostProcessor @PersistenceUnit

Java

```
public class ProductDaoImpl implements ProductDao {

    private EntityManagerFactory emf;

    @PersistenceUnit
    public void setEntityManagerFactory(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public Collection loadProductsByCategory(String category) {
        try (EntityManager em = this.emf.createEntityManager()) {
            Query query = em.createQuery("from Product as p where p.category = ?1");
            query.setParameter(1, category);
            return query.getResultList();
        }
    }
}
```

Kotlin

```
class ProductDaoImpl : ProductDao {

    private lateinit var emf: EntityManagerFactory

    @PersistenceUnit
    fun setEntityManagerFactory(emf: EntityManagerFactory) {
        this.emf = emf
    }

    fun loadProductsByCategory(category: String): Collection<*> {
        val em = this.emf.createEntityManager()
        val query = em.createQuery("from Product as p where p.category = ?1");
        query.setParameter(1, category);
        return query.resultList;
    }
}
```

`EntityManagerFactory`

```
<beans>

    <!-- bean post-processor for JPA annotations -->
    <bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>

    <bean id="myProductDao" class="product.ProductDaoImpl"/>

</beans>
```

`PersistenceAnnotationBeanPostProcessor`

`context:annotation-config`

`CommonAnnotationBeanPostProcessor`

```
<beans>

    <!-- post-processors for all standard config annotations -->
    <context:annotation-config/>

    <bean id="myProductDao" class="product.ProductDaoImpl"/>

</beans>
```

EntityManager

EntityManager

Java

```
public class ProductDaoImpl implements ProductDao {

    @PersistenceContext
    private EntityManager em;

    public Collection loadProductsByCategory(String category) {
        Query query = em.createQuery("from Product as p where p.category =
:category");
        query.setParameter("category", category);
        return query.getResultList();
    }
}
```

Kotlin

```
class ProductDaoImpl : ProductDao {

    @PersistenceContext
    private lateinit var em: EntityManager

    fun loadProductsByCategory(category: String): Collection<*> {
        val query = em.createQuery("from Product as p where p.category = :category")
        query.setParameter("category", category)
        return query.resultList
    }
}
```

@PersistenceContext type PersistenceContextType.TRANSACTION EntityManager PersistenceContextType.EXTENDED EntityManager

`EntityManager`

`EntityManager`

Method- and field-level Injection

`@PersistenceUnit`

`@PersistenceContext`

`EntityManager`

`EntityManager`

`EntityManagerFactory`

4.4.3. Spring-driven JPA transactions

`JpaTransactionManager`

`JpaTransactionManager`

`DataSource` `JpaDialect`

`Connection`

`JpaDialect`

HibernateTransactionManager

LocalSessionFactoryBean

4.4.4. Understanding JpaDialect and JpaVendorAdapter

JpaTransactionManager AbstractEntityManagerFactoryBean JpaDialect jpaDialect JpaDialect

Connection

PersistenceExceptions DataAccessExceptions

DefaultJpaDialect

LocalContainerEntityManagerFactoryBean JpaVendorAdapter JpaDialect HibernateJpaVendorAdapter EclipseLinkJpaVendorAdapter EntityManagerFactory

JpaTransactionManager

JpaDialect JpaVendorAdapter

4.4.5. Setting up JPA with JTA Transaction Management

JpaTransactionManager

JtaTransactionManager JpaTransactionManager

DataSource

DataSource

EntityManagerFactory

jpaProperties
LocalContainerEntityManagerFactoryBean

HibernateJpaVendorAdapter
on-close
HibernateJpaVendorAdapter
prepareConnection
hibernate.connection.handling_mode
DELAYED_ACQUISITION_AND_RELEASE_AFTER_STATEMENT

EntityManagerFactory

LocalContainerEntityManagerFactoryBean EntityManagerFactory

4.4.6. Native Hibernate Setup and Native Hibernate Transactions for JPA Interaction

LocalSessionFactoryBean
HibernateTransactionManager @PersistenceContext
SessionFactory EntityManagerFactory
Session EntityManager

LocalContainerEntityManagerFactoryBean JpaTransactionManager
SessionFactory.getCurrentSession() HibernateTemplate
@PersistenceContext EntityManager

HibernateJpaVendorAdapter

LocalSessionFactoryBuilder
@Bean FactoryBean

> LocalSessionFactoryBean LocalSessionFactoryBuilder
> LocalContainerEntityManagerFactoryBean
>
> LocalSessionFactoryBean bootstrapExecutor
> LocalSessionFactoryBuilder
> buildSessionFactory

Chapter 5. Marshalling XML by Using Object-XML Mappers

5.1. Introduction

5.1.1. Ease of configuration

5.1.2. Consistent Interfaces

`Marshaller` `Unmarshaller`

5.1.3. Consistent Exception Hierarchy

`XmlMappingException`

5.2. Marshaller and Unmarshaller

5.2.1. Understanding Marshaller

org.springframework.oxm.Marshaller

Java

```
public interface Marshaller {

    /**
     * Marshal the object graph with the given root into the provided Result.
     */
    void marshal(Object graph, Result result) throws XmlMappingException, IOException;
}
```

Kotlin

```
interface Marshaller {

    /**
    * Marshal the object graph with the given root into the provided Result.
    */
    @Throws(XmlMappingException::class, IOException::class)
    fun marshal(
            graph: Any,
            result: Result
    )
}
```

Marshaller javax.xml.transform.Result

Result implementation	Wraps XML representation
DOMResult	org.w3c.dom.Node
SAXResult	org.xml.sax.ContentHandler
StreamResult	java.io.File java.io.OutputStream java.io.Writer

marshal()
Marshaller

5.2.2. Understanding Unmarshaller

Marshaller org.springframework.oxm.Unmarshaller

Java

```
public interface Unmarshaller {

    /**
     * Unmarshal the given provided Source into an object graph.
     */
    Object unmarshal(Source source) throws XmlMappingException, IOException;
}
```

Kotlin

```
interface Unmarshaller {

    /**
    * Unmarshal the given provided Source into an object graph.
    */
    @Throws(XmlMappingException::class, IOException::class)
    fun unmarshal(source: Source): Any
}
```

javax.xml.transform.Source
Result Source

Source implementation	Wraps XML representation
DOMSource	org.w3c.dom.Node
SAXSource	org.xml.sax.InputSource org.xml.sax.XMLReader
StreamSource	java.io.File java.io.InputStream java.io.Reader

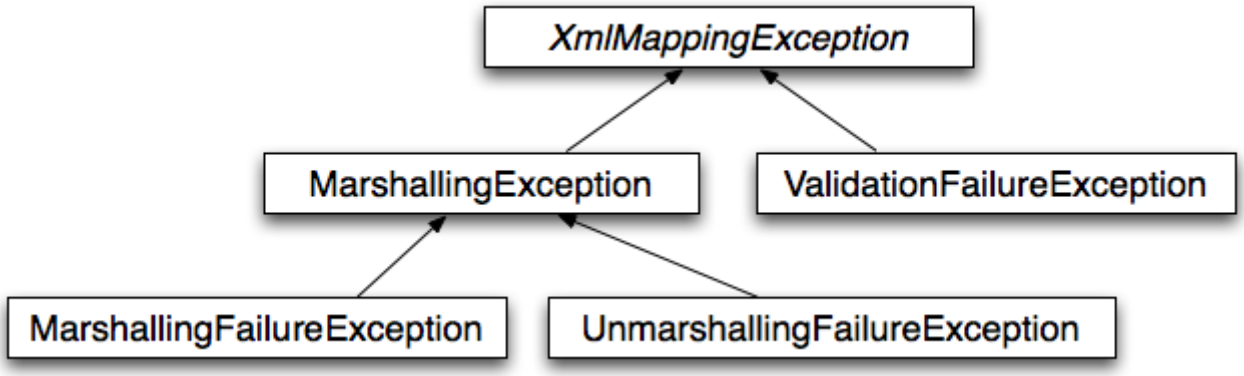
Marshaller Unmarshaller

applicationContext.xml

5.2.3. Understanding XmlMappingException

XmlMappingException

MarshallingFailureException UnmarshallingFailureException



5.3. Using Marshaller and Unmarshaller

Java

```
public class Settings {

    private boolean fooEnabled;

    public boolean isFooEnabled() {
        return fooEnabled;
    }

    public void setFooEnabled(boolean fooEnabled) {
        this.fooEnabled = fooEnabled;
    }
}
```

Kotlin

```
class Settings {
    var isFooEnabled: Boolean = false
}
```

`saveSettings()` `settings.xml` `loadSettings()` `main()`

Java

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.oxm.Marshaller;
import org.springframework.oxm.Unmarshaller;

public class Application {

    private static final String FILE_NAME = "settings.xml";
    private Settings settings = new Settings();
    private Marshaller marshaller;
    private Unmarshaller unmarshaller;

    public void setMarshaller(Marshaller marshaller) {
        this.marshaller = marshaller;
    }

    public void setUnmarshaller(Unmarshaller unmarshaller) {
        this.unmarshaller = unmarshaller;
    }

    public void saveSettings() throws IOException {
        try (FileOutputStream os = new FileOutputStream(FILE_NAME)) {
            this.marshaller.marshal(settings, new StreamResult(os));
        }
    }

    public void loadSettings() throws IOException {
        try (FileInputStream is = new FileInputStream(FILE_NAME)) {
            this.settings = (Settings) this.unmarshaller.unmarshal(new
StreamSource(is));
        }
    }

    public static void main(String[] args) throws IOException {
        ApplicationContext appContext =
                new ClassPathXmlApplicationContext("applicationContext.xml");
        Application application = (Application) appContext.getBean("application");
        application.saveSettings();
        application.loadSettings();
    }
}
```

Kotlin

```
class Application {

    lateinit var marshaller: Marshaller

    lateinit var unmarshaller: Unmarshaller

    fun saveSettings() {
        FileOutputStream(FILE_NAME).use { outputStream -> marshaller.marshal(settings,
StreamResult(outputStream)) }
    }

    fun loadSettings() {
        FileInputStream(FILE_NAME).use { inputStream -> settings =
unmarshaller.unmarshal(StreamSource(inputStream)) as Settings }
    }
}

private const val FILE_NAME = "settings.xml"

fun main(args: Array<String>) {
    val appContext = ClassPathXmlApplicationContext("applicationContext.xml")
    val application = appContext.getBean("application") as Application
    application.saveSettings()
    application.loadSettings()
}
```

Application marshaller unmarshaller applicationContext.xml

```
<beans>
    <bean id="application" class="Application">
        <property name="marshaller" ref="xstreamMarshaller" />
        <property name="unmarshaller" ref="xstreamMarshaller" />
    </bean>
    <bean id="xstreamMarshaller"
class="org.springframework.oxm.xstream.XStreamMarshaller"/>
</beans>
```

XStreamMarshaller Marshaller Unmarshaller xstreamMarshaller marshaller unmarshaller

settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings foo-enabled="false"/>
```

5.4. XML Configuration Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:oxm="http://www.springframework.org/schema/oxm" ①
xsi:schemaLocation="http://www.springframework.org/schema/beans
  https://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/oxm
https://www.springframework.org/schema/oxm/spring-oxm.xsd"> ②
```

① oxm

② oxm

jaxb2-marshaller

jibx-marshaller

```
<oxm:jaxb2-marshaller id="marshaller"
contextPath="org.springframework.ws.samples.airline.schema"/>
```

5.5. JAXB

jaxb.properties

Marshaller Unmarshaller Marshaller Unmarshaller org.springframework.oxm.jaxb

5.5.1. Using Jaxb2Marshaller

Jaxb2Marshaller Marshaller Unmarshaller

contextPath

classesToBeBound

```
<beans>
    <bean id="jaxb2Marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
        <property name="classesToBeBound">
            <list>
                <value>org.springframework.oxm.jaxb.Flight</value>
                <value>org.springframework.oxm.jaxb.Flights</value>
            </list>
        </property>
        <property name="schema" value="classpath:org/springframework/oxm/schema.xsd"/>
    </bean>

    ...

</beans>
```

XML Configuration Namespace

jaxb2-marshaller org.springframework.oxm.jaxb.Jaxb2Marshaller

```
<oxm:jaxb2-marshaller id="marshaller"
contextPath="org.springframework.ws.samples.airline.schema"/>
```

class-to-be-bound

```
<oxm:jaxb2-marshaller id="marshaller">
    <oxm:class-to-be-bound
name="org.springframework.ws.samples.airline.schema.Airport"/>
    <oxm:class-to-be-bound
name="org.springframework.ws.samples.airline.schema.Flight"/>
    ...
</oxm:jaxb2-marshaller>
```

Attribute	Description	Required
id		
contextPath		

5.6. JiBX

org.springframework.oxm.jibx

5.6.1. Using JibxMarshaller

JibxMarshaller Marshaller Unmarshaller targetClass bindingName Flights

```
<beans>
    <bean id="jibxFlightsMarshaller"
class="org.springframework.oxm.jibx.JibxMarshaller">
        <property name="targetClass">org.springframework.oxm.jibx.Flights</property>
    </bean>
    ...
</beans>
```

JibxMarshaller JibxMarshaller targetClass

XML Configuration Namespace

jibx-marshaller org.springframework.oxm.jibx.JibxMarshaller

```
<oxm:jibx-marshaller id="marshaller" target-
class="org.springframework.ws.samples.airline.schema.Flight"/>
```

Attribute	Description	Required
id		
target-class		
bindingName		

5.7. XStream

`org.springframework.oxm.xstream`

5.7.1. Using `XStreamMarshaller`

`XStreamMarshaller`

```
<beans>
    <bean id="xstreamMarshaller"
class="org.springframework.oxm.xstream.XStreamMarshaller">
        <property name="aliases">
            <props>
                <prop key="Flight">org.springframework.oxm.xstream.Flight</prop>
            </props>
        </property>
    </bean>
    ...
</beans>
```

XStreamMarshaller

XStreamMarshaller
supportedClasses XStreamMarshaller

```
<bean id="xstreamMarshaller"
class="org.springframework.oxm.xstream.XStreamMarshaller">
    <property name="supportedClasses"
value="org.springframework.oxm.xstream.Flight"/>
    ...
</bean>
```

CatchAllConverter

Chapter 6. Appendix

6.1. XML Schemas

tx

jdbc

6.1.1. The tx Schema

tx

'spring-tx.xsd'

tx

tx

tx

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx" ①
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx
https://www.springframework.org/schema/tx/spring-tx.xsd ②
        http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- bean definitions here -->

</beans>
```

① tx

②

tx

aop

aop aop

6.1.2. The jdbc Schema

jdbc

jdbc

jdbc

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc" ①
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/jdbc
https://www.springframework.org/schema/jdbc/spring-jdbc.xsd"> ②

    <!-- bean definitions here -->

</beans>
```

① jdbc

②