Spring ve Transaction Yönetimi

Spring ile Transaction Yönetiminin Özellikleri

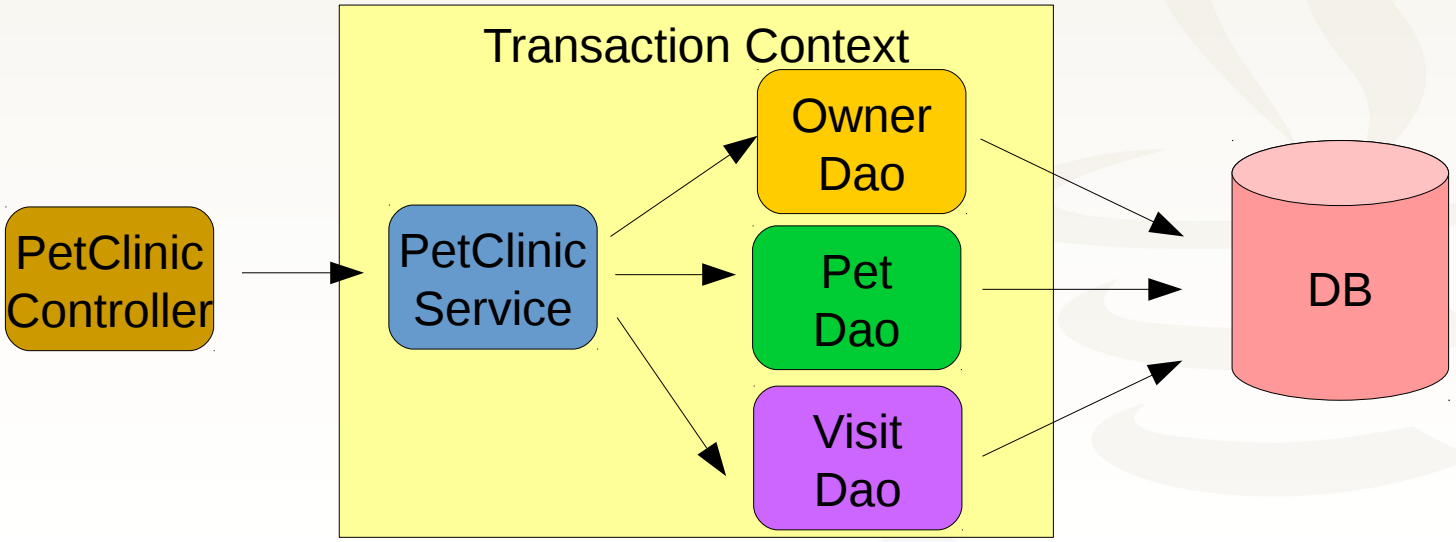
- Spring, değişik veri erişim yöntemlerini aynı anda kullanmayı sağlayan **ortak bir transaction yönetim API**'sine sahiptir
- **Dekleratif** ve **programatik** TX yönetimi mümkündür
- **Global** (dağıtık) ve **lokal** (tek DB) transactionları destekler
- Spring'in temel transaction soyutlaması **PlatformTransactionManager**'dır

Spring ile Transaction Yönetiminin Özellikleri

- **PlatformTransactionManager** sayesinde veri erişimi, TX altyapısından **tamamen bağımsız** hale gelir
- Bu sayede uygulamalar **lokal TX'den global TX'e transparan biçimde geçiş** yapabilir
- Ya da **veri erişim teknolojisi** değiştiği vakit uygulama tarafında bir değişikliğe gerek kalmadan **transaction yönetim altyapısı** rahatlıkla değiştirilebilir
- Bunun için **sadece bean tanımlarında değişiklik** yeterlidir

Transaction Context

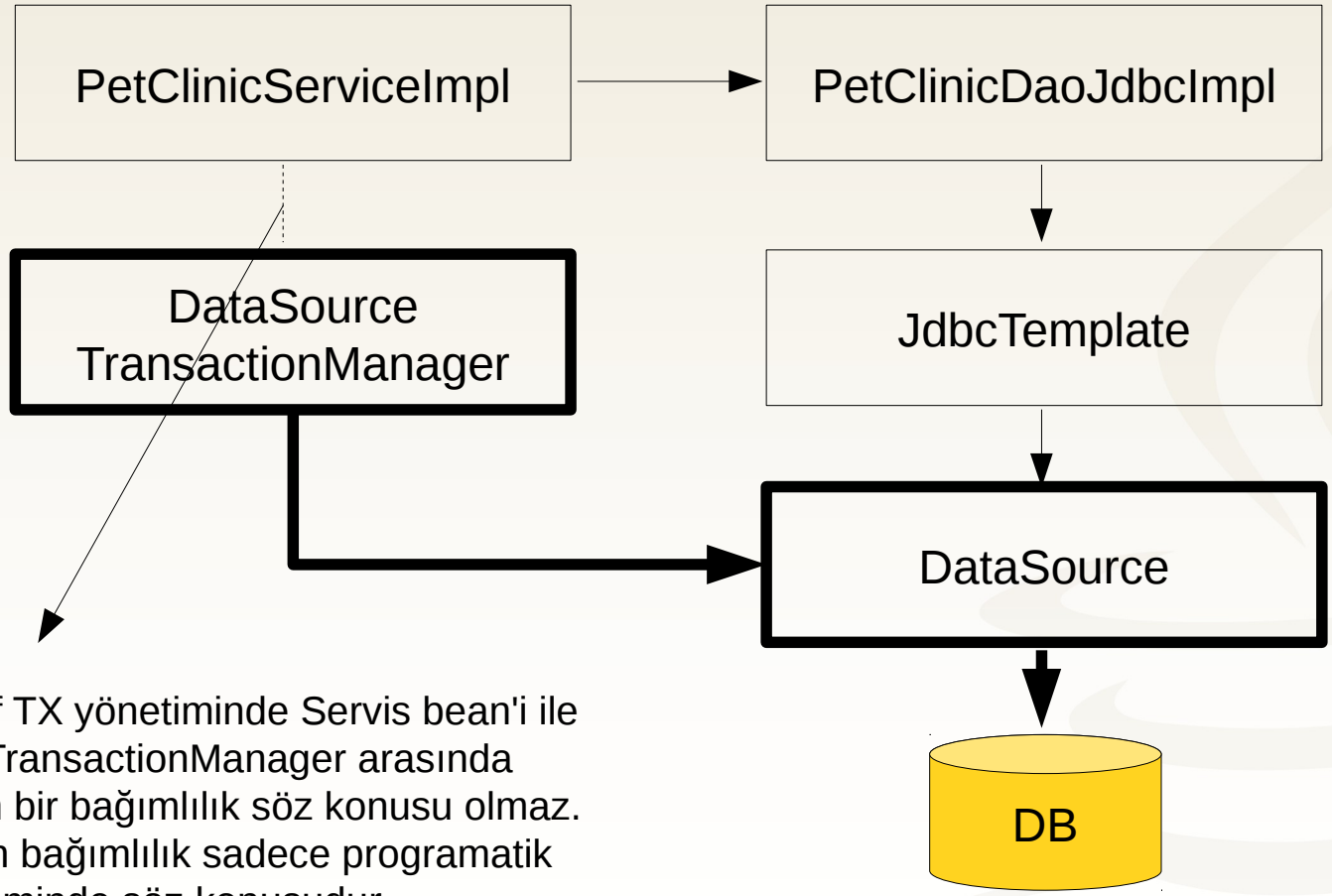
- TX sınırları genellikle servis katmanında belirlenir, servis katmanı bir veya daha fazla DAO bean'ini kullanarak iş mantığını yürütür



PlatformTransaction Manager Konfigürasyonu

- Spring ile transaction yönetiminde **en önemli nokta** veri erişim teknolojisine uygun **PlatformTransactionManager** tanımlanmasıdır
- Farklı veri erişim teknolojileri için **farklı implemantasyonlar** mevcuttur
  - **JDBC**: DataSourceTransactionManager
  - **Hibernate**: HibernateTransactionManager
  - **JPA**: JpaTransactionManager
  - **JTA**: JtaTransactionManager

PlatformTransactionManager Konfigürasyonu - JDBC



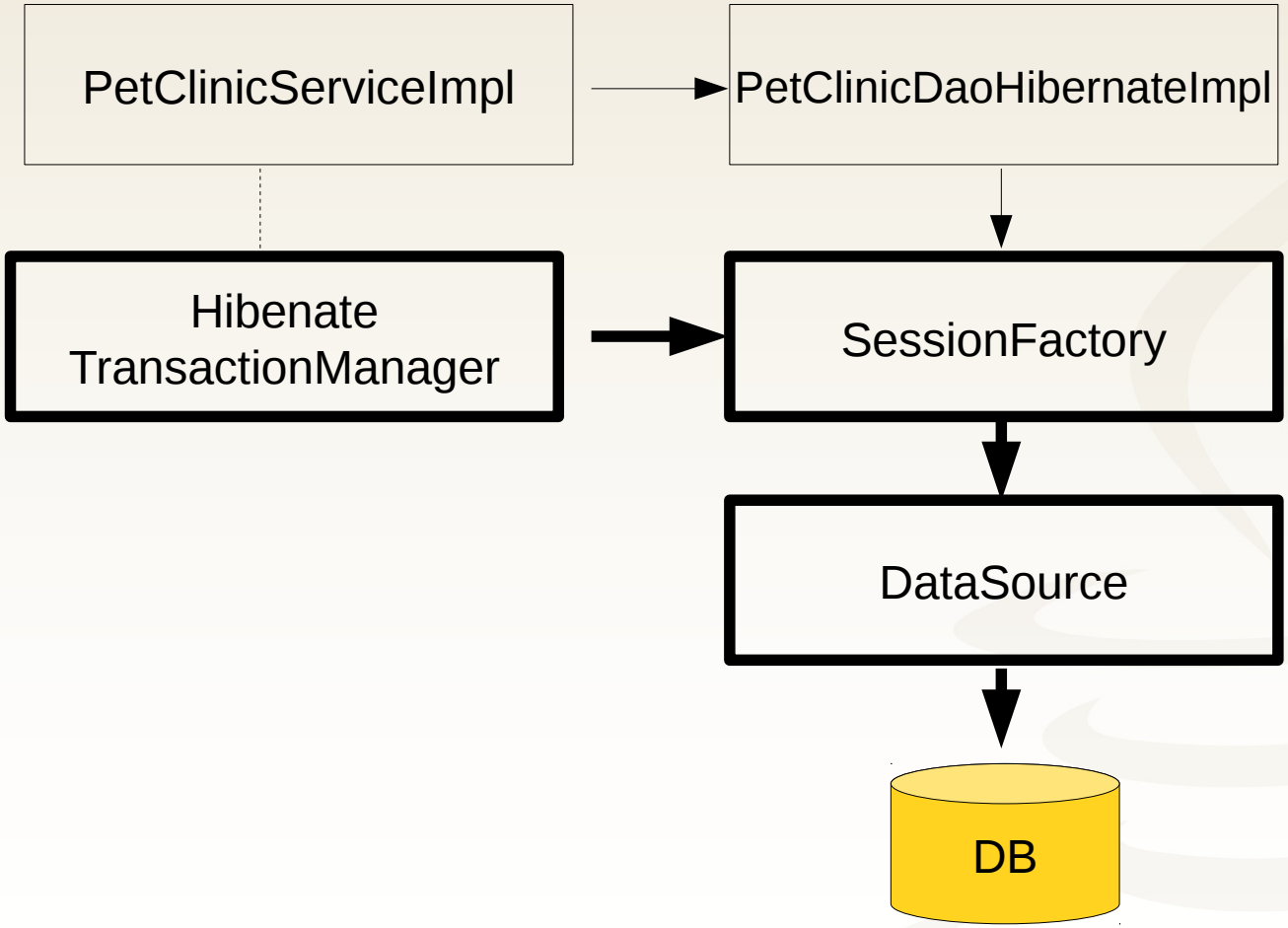
Dekleratif TX yönetiminde Servis bean'i ile PlatformTransactionManager arasında doğrudan bir bağımlılık söz konusu olmaz. Doğrudan bağımlılık sadece programatik TX yönetiminde söz konusudur.

PlatformTransactionManager Konfigürasyonu - JDBC

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTr
ansactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>


<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManage
rDataSource">
  <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 Konfigürasyonu - Hibernate

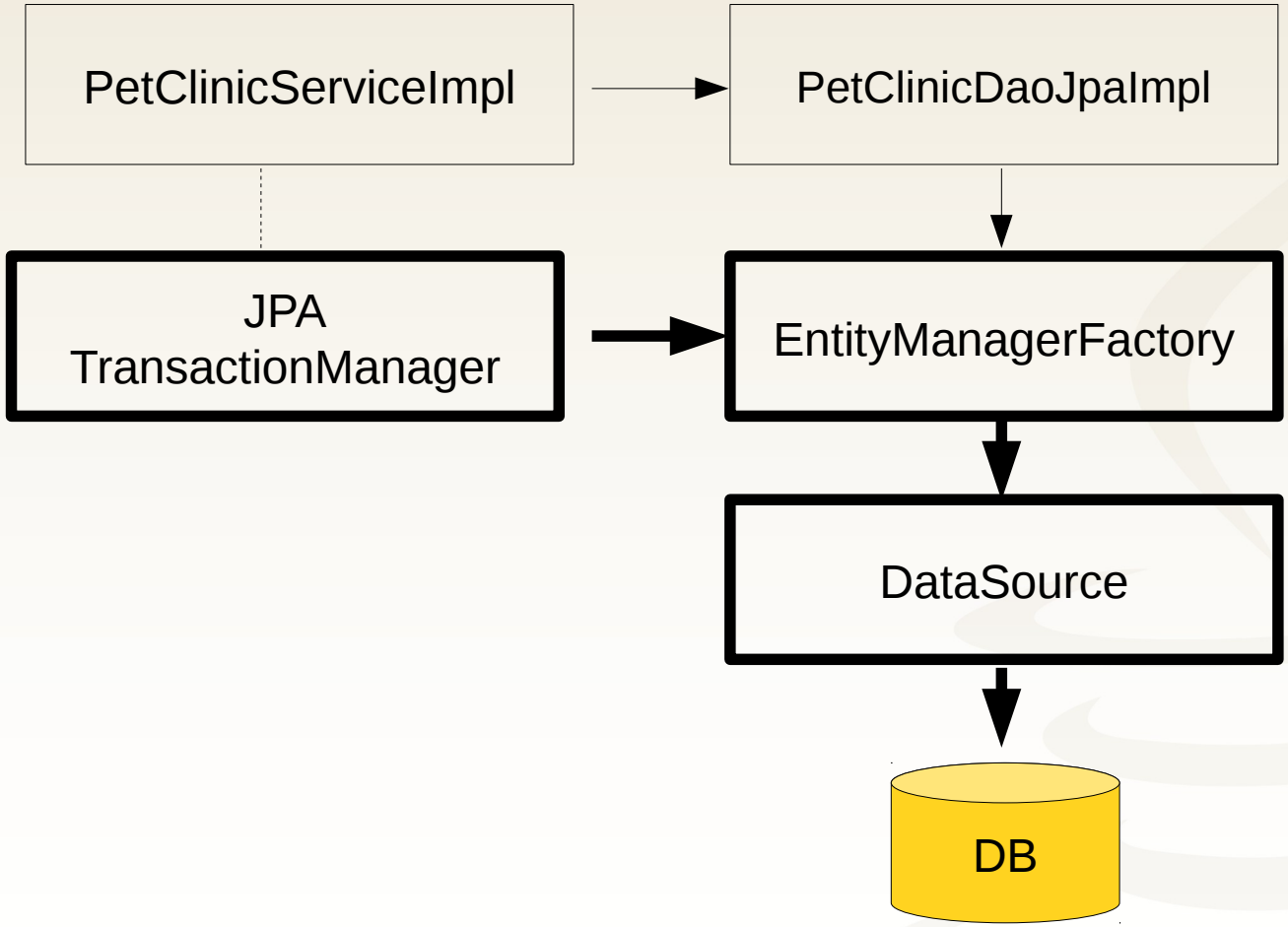


PlatformTransactionManager Konfigürasyonu - Hibernate

```
<bean id="transactionManager"
class="org.springframework.orm.hibernate4.HibernateTransactionManager">
  <property name="sessionFactory" ref="sessionFactory" />
</bean>

<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.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>
```

PlatformTransactionManager Konfigürasyonu - JPA



PlatformTransactionManager Konfigürasyonu - JPA

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="packagesToScan"
              value="com.javaegitimleri.petclinic.model"/>
    <property name="jpaProperties">
        <value>
            hibernate.dialect=org.hibernate.dialect.H2Dialect
        </value>
    </property>

    <property name="persistenceProviderClass"
              value="org.hibernate.jpa.HibernatePersistenceProvider"/>
</bean>

<bean id="transactionManager"
        class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"/>
</bean>
```

Dekleratif Transaction Yönetimi

- Uygulama içinde TX yönetimi ile ilgili **kod yazılmaz**
- Servis metot çağrısı geldiği vakit Spring Container tarafından **yeni bir TX başlatılır**
- Metot başarılı sonlandığı vakit **TX commit** edilir
- **Sınıf veya metot düzeyinde** TX yönetimi yapılabilir
- En sık **tercih edilen yöntem**dir

Dekleratif Transaction Yönetimi ve Rollback

- Transactional bir metot içerisinde bir exception meydana geldiğinde **exception türü**ne bakılır
- Default olarak runtime exception'larda **TX rollback** edilir
- Checked exception'larda ise **TX commit** edilir
- Ancak çoğunlukla **bu davranış değiştirilir**

Dekleratif Transaction Yönetimi

- Dekleratif TX yönetimi @**Transactional** anotasyonu ile yapılır
- **Sınıf veya metot düzeyinde** kullanılabilir
- Sadece **public metotlar**da kullanılmalıdır
- `<tx:annotation-driven/>` elemanı @Transactional anotasyonlarını devreye sokar

@Transactional ile Dekleratif TX Yönetimi

```
@Transactional
public class DefaultFooService implements FooService {

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

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

@Transactional ile Dekleratif TX Yönetimi

```
public class DefaultFooService implements FooService {

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

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

@Transactional ile Dekleratif TX Yönetimi

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

<tx:annotation-driven/>

<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

   <property name="dataSource" ref="dataSource"/>

</bean>
```

@Transactional Default Değerleri

- **@Transactional** anotasyonundaki attribute'ların default değerleri:
  - **Propagation** REQUIRED
  - **Isolation** DEFAULT
  - **Transaction** read/write
  - **Timeout** sistem default
  - **Rollback** Herhangi bir RuntimeException

@Transactional Default Değerleri

```
@Transactional(rollbackFor = Exception.class)
public class DefaultFooService implements FooService {

   @Transactional(readOnly = true)
   public Foo getFoo(String fooName) {
      // ...
   }

   @Transactional(propagation = Propagation.REQUIRES_NEW)
   public void updateFoo(Foo foo) {
      // ...
   }
}
```

Hangi @Transactional'ı Kullanmalı ?

- **JTA 1.2** API'si ile gelen bir **@Transactional** anotasyonu da mevcuttur
- Spring bu anotasyonu da **desteklemektedir**
- JTA'nın anotasyonunda **sadece propagation ve rollback kuralları** değiştirilebilmektedir

Dekleratif Yöntemde Exception Fırlatmadan Rollback

```
@Transactional
public void foo() {
    try {

// is mantigi...

    } catch (Exception ex) {
TransactionAspectSupport.
currentTransactionStatus().setRollbackOnly();
    }
}
```

Gerektiğinde uygulama içerisinden **setRollbackOnly()** çağrılarak exception fırlatmadan da TX rollback yaptırılabilir

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com