Aprameyah
Technologies Pvt. Ltd.

Spring Transactions

Aprameyah
Technologies Pvt. Ltd.

Spring Transactions

- Spring provides the equivalent of container managed transactions of EJB

- Spring DOES NOT provide a distributed transaction manager like JTA

- Spring allows transaction propagation using transactional attributes

Aprameyah
Technologies Pvt. Ltd.

DataSources

- For Spring to manage the transactions it should also be managing the connections aswell.

- Data sources have to be defined in spring context

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
   <property name="driverClassName"
                        value="org.apache.derby.jdbc.ClientDriver" />
   <property name="url"
         value="jdbc:derby:…" />
   <property name="username" value="APP" />
   <property name="password" value="app" />
</bean>
```

Aprameyah
Technologies Pvt. Ltd.

Transaction Manager

- Spring provides out of the box transaction managers for standalone jdbc and hibernate transactions.

- The transaction manager has to control the datasource. So we create a transaction manager with a datasource

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

Creating An Advice

- The transaction should be applied as an AOP advice. We create an advice using the transaction manager. There is a dedicated tx:advice namespace to aid this standard construct

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <!-- the transactional semantics... -->
    <tx:attributes>
       <!-- all methods starting with 'get' are read-only -->
       <tx:method name="get*" propagation="SUPPORTS"/>
       <tx:method name="*" propagation="REQUIRED"/>
     </tx:attributes>
</tx:advice>
```

Determining Pointcuts

- We have the transaction manager, datasource and an advice. We need to now bind the advice using a pointcut. Is done with the aop namespace

```
<aop:config>
   <aop:pointcut id="userOperations" expression="execution(*
com.mydomain.service.UserManager.*(..))"/>
   <aop:advisor advice-ref="txAdvice" pointcut-ref="userOperations"/>
</aop:config>
```

Use The Datasource

- As a last step we need to make sure that our data access classes use connection from the datasource

```
@Autowired
DataSource dataSource;
```

and then

```
dataSource.getConnection().createStatement();
```

Aprameyah
Technologies Pvt. Ltd.

Lets Try It

- Create a method “generateOrder” that takes a user id and creates a record in the Order table. Throw an exception if User name is “admin” because admin cannot have orders

- Create a method “generateOrders” in UserManager that generates 5 orders for each user in the database with the help of above method.

- Configure it such that failure to generate order for any single user should not break order generation for other users