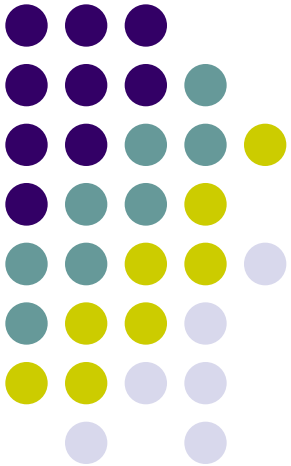
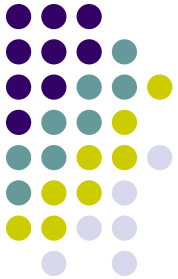
Spring

DataAccess

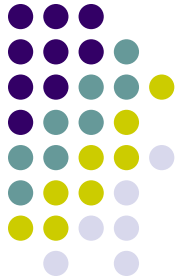


Contenidos



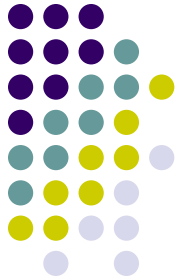
- Intro
- TX
- Data Access
- Ejemplos

Intro



- Spring provee una capa de abstracción para facilitar el acceso a un conjunto de repositorios sin preocuparnos por el manejo de las conexiones, transacciones o gestión de errores en general.

DataSource



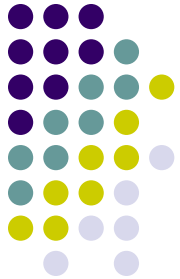
<bean id="*dataSourceXXX*"
class="*org.springframework.jdbc.datasource.DriverManagerDataSource*">

<property name="*driverClassName*" value="*com.mysql.jdbc.Driver*" />
<property name="*url*" value="*jdbc:mysql://localhost/test*" />
<property name="*username*" value="*root*" />
<property name="*password*" value="*root*" />

</bean>

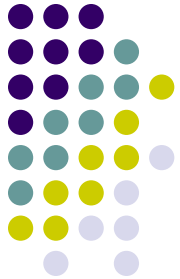
Transaction Management

- DataSourceTransactionManager
- HibernateTransactionManager
- JdoTransactionManager
- JtaTransactionManager



Transaction Management

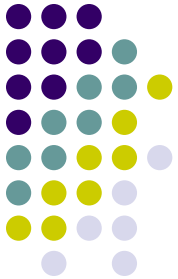
```
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSourceXXX"/>
</bean>
```



Transaction Management

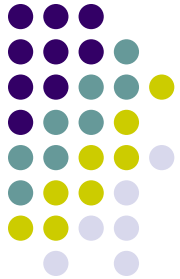
```
<?xml version="1.0" encoding="UTF-8"?>
<beans ……>
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/ds"/>
<bean id="txManager"
class="org.springframework.transaction.jta.JtaTransactionManager" />
</beans>
```

Transaction Management



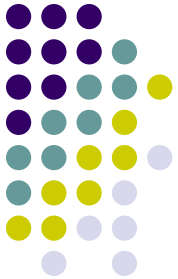
```
<bean id="fooService" class="x.y.service.DefaultFooService"/>
   <tx:advice id="txAdvice" transaction-manager="txManager">
   <tx:attributes>
      <tx:method name="get*" read-only="true"/>
      <tx:method name="*"/>
   </tx:attributes>
</tx:advice>
```

Transaction Management



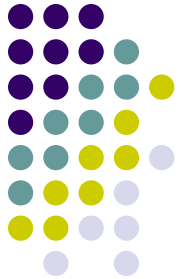
```
<aop:config>
    <aop:pointcut id="fooServiceOperation"       expression="execution(*
x.y.service.FooService.*(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
</aop:config>


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

@Transactional



```
@Repository
public class ClienteDAOImp implements ClienteDAO {
……..

@Service
@Transactional(rollbackFor=Exception.class)
public class ClienteService {
…….

@Transactional
public void guardarXXXX
```

@Transactional



Propagation

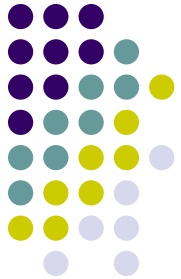
Isolation level

Read/Write

Timeout

Triggers rollback

@Transactional

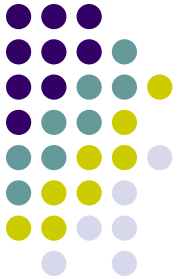


```
<tx:annotation-driven />
```

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

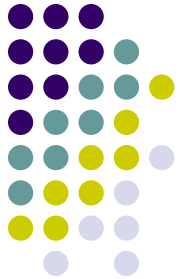
JDBC

```
@Repository
public class JdbcMovieFinder implements MovieFinder {

  private JdbcTemplate jdbcTemplate;

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

// ...
}
```

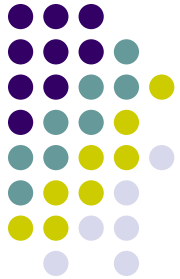


JDBC

```
int rowCount = jdbcTemplate.queryForInt("select count(*) from t_actor");

int rowCount = jdbcTemplate.queryForInt(
"select count(*) from t_actor where first_name = ?", "Joe");

String lastName = jdbcTemplate.queryForObject(
"select last_name from t_actor where id = ?",
new Object[]{1212}, String.class);
```



JDBC

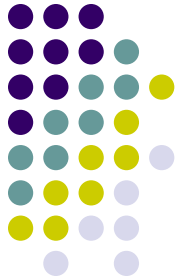


```
Actor actor = jdbcTemplate.queryForObject(
  "select first_name, last_name from t_actor where id = ?",
  new Object[]{1212}, new RowMapper<Actor>() {
     public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
       Actor actor = new Actor();
       actor.setFirstName(rs.getString("first_name"));
       actor.setLastName(rs.getString("last_name"));
      return actor;
    }
});
```

JDBC

```
List<Actor> actors = this.jdbcTemplate.query(
   "select first_name, last_name from t_actor",
     new RowMapper<Actor>() {
        public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
           Actor actor = new Actor();
           actor.setFirstName(rs.getString("first_name"));
           actor.setLastName(rs.getString("last_name"));
           return actor;
    }
});
```

JDBC

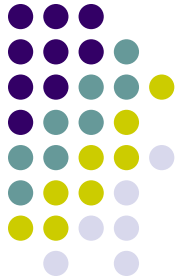


```
this.jdbcTemplate.update("insert into t_actor (first_name, last_name) values (?, ?)",
"Leonor", "Watling");
```

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

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

JDBC



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

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

Otras opciones

SimpleJdbcInsert
SimpleJdbcCall

Hibernate

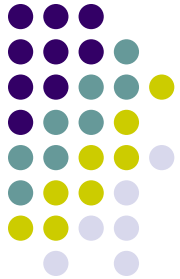
```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

 <property name="dataSource"><ref bean="dataSourceXXX" /></property>
 <property name="hibernateProperties">
  <props>
    <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
    <prop key="hibernate.show_sql">true</prop>
    <prop key="hibernate.hbm2ddl.auto">update</prop>
  </props>
 </property>

 <property name="annotatedClasses">
  <list>
     <value>xxxxxxxxxx</value>
   </list>
 </property>
</bean>
```

Hibernate

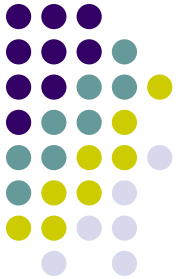
```
<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
<property name="sessionFactory" ref="sessionFactory" />
</bean>

<bean
class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
```

Hibernate

```
@Repository
public class ClienteDAOImp implements ClienteDAO {

  @Autowired
  private SessionFactory sessionFactory;

  public Long guardarCliente(Cliente cliente) {
     sessionFactory.getCurrentSession().save(cliente);
     return cliente.getId();
  }

……..
```

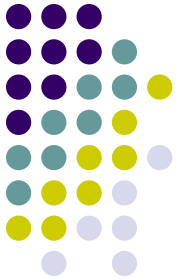


Data Access Utils



- JDBC (JdbcTemplate)
- Hibernate (HibernateTemplate)
- iBatis (SqlMapClientTemplate)
- JDO (JdoTemplate)
- TopLink (TopLinkTemplate)
- Tx (TransactionTemplate)

Data Access Utils



Connection conn = DataSourceUtils.getConnection(dataSource);

SimpleJdbcTemplate jdbcTemplate = new SimpleJdbcTemplate (dataSource);

HibernateTemplate hibernateTemplate = new HibernateTemplate(sessionFactory);

JdbcDaoSupport

HibernateDaoSupport

Etc

Ejercicio

1. *Crear un DAO para las cuentas bancarias mediante JDBC (Alta, baja, modificación y listado)*
2. *Crear un DAO para Cliente mediante Hibernate (Alta, baja, modificación y listado)*
3. *Crear una clase BancoService que utiliza los DAOs según corresponda (utilizar @Transactional..)*
4. *Modificar el DAO de hibernate para que trabaje mediante un GenericDAO*