Modulo II
Spring-JDBC

Prof. Ismael H F Santos

Bibliografia

- Spring in Action
  - Craig Walls and Ryan Breidenbach
- Professional Java Development with Spring
  - Rod Johnson, Juergen Hoeller and Team
- Spring – www.springframework.org
- J2EE without EJB – Rod Johnson/ Jurgen Hoeller
- Better, Faster, Lighter Java – Bruce Tate
- Wiring your Web Application with Open Source Java http://www.onjava.com/pub/a/onjava/2004/04/07/wiringwebapps.html

Spring Related Tools and Add-Ons

- ACEGI Security - comprehensive security services for the Spring Framework
- Spring IDE - graphical user interface for the configuration files used by the Spring Framework
- Spring BeanDoc - tool that facilitates documentation and graphing of Spring bean factories and application context files
- XDoclet Spring Tags - support for generating Spring XML config files from annotations in Java classes (you could also use JDK1.5 annotations to achieve this)
- Spring Web Flow - for web applications with demanding page flow requirements
- AppFuse Not really a tool or add-on, but AppFuse is Matt Raible's project to jumpstart your Java web projects. It uses Spring at it's core and studying it is a great way to learn about Spring.
- Spring Framework .NET – Spring Clone for the Dark Side ☺

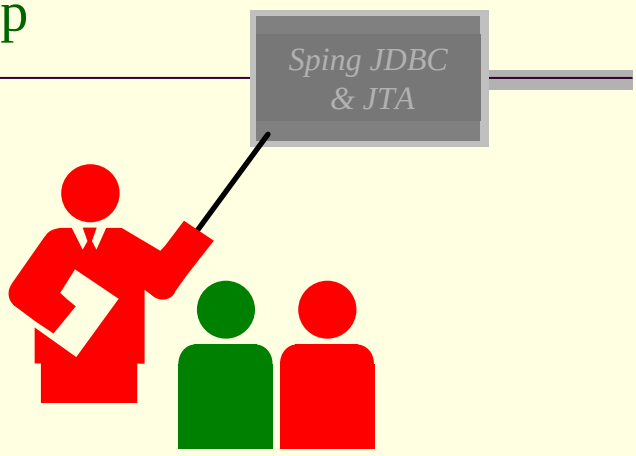
Spring Framework / Spring Related References

- **The Official Spring Reference Manual**
  http://www.springframework.org/docs/reference/
- **Introduction to Spring by Rod Johnson**
- http://www.theserverside.com/articles/article.tss?l=SpringFramework
- **Spring in Action** by Craig Walls and Ryan Breidenbach
- **Pro Spring** by Rob Harrop and Jan Machacek
- **J2EE Without EJB** by Rod Johnson and Juergen Holler
- **Expert One-on-One J2EE Design and Development** by Rod Johnson
- **Spring Developers Notebook** by Bruce Tate and Justin Gehtland
- **Better, Faster, Lighter Java** by Bruce Tate and Justin Gehtland
- **Spring Live** by Matt Raible
- **Professional Java Development with the Spring Framework**
- by many of the core Spring developers: **Coming in July 2005**

Ementa

- Spring DAO
- Spring JDBC

WebApp



O que temos?

- Uma camada de abstração para persistência de dados utilizando um ou mais das seguintes tecnologias
  - JDBC
  - Hibernate, OJB
  - JDO
  - iBatis
- Uma robusta infra-estrutura para o gerenciamento de transações de forma declarativa que suporta tanto transações locais como transações distribuídas através da JTA
- Uma camada simplificadora para tecnologias de distribuição incluindo EJBs, RMI, e Web Services
- Útil suporte a JNDI, JMS, email, e agendamento de tarefas(*task scheduling*)

Spring DAO

- Possui uma API de acesso a dados que ajuda a isolar e melhorar o modo como o dado é servido pela camada de negócio
- Mais uma consistente e rica hierarquia de exceções suficiente para mapear exceções específicas dependentes de tecnologia para outras exceções genéricas
- Além de uma série de templates e warpper classes para se trabalhar com JDBC, Hibernate, JDO, etc.

Spring's JDBC Api

- ***Why not just use JDBC***
  - Connection management
  - Exception Handling
  - Transaction management
  - Other persistence mechanisms
- **Spring Provides a Consistent JDBC Abstraction**
  - Spring Helper classes
    - JdbcTemplate
    - Dao Implementations
    - Query and BatchQueries
- ***Examples***
  - *Traditional JDBC Dao: ContactDAO*
  - *Spring JDBC Dao using JdbcTemplate: ContactLoadDAO*

JDBC Tradicional

```
public void updateCustomer(Customer customer) {
    Connection conn = null;
    PreparedStatement ps = null;
    try {
        conn = getConnection();
        ps = conn.prepareStatement("update customer set " +
                "firstName = ?, lastName = ?, ...");
        ps.setString(1, customer.getFirstName());
        ps.setString(2, customer.getLastName());
        ps.executeUpdate();
    } catch (SQLException e) {
        log.error(e);
    } finally {
        try { if (ps != null) ps.close(); }
        catch (SQLException e) {log.error(e);}
        try {if (conn != null) conn.close();}
        catch (SQLException e) {log.error(e);}
    }
}
private Connection getConnection() {
    //... Mais código de encanador
}
```

Spring Transactions

- *Transaction Overview*
  - Atomic, Consistent, Isolated, Durable (ACID)
    - Atomicity and Isolation are of most concern to us as developers. Consistency and Durability are more related to the Transaction source.
- *Similar advantages with Transactions as Spring's JDBC DAO has with DB.*
  - *TransactionTemplate class*
- *Alternative independent of DB Transaction Management and Transaction Management implementation*

Spring Transactions

- Spring Transaction vs Connection.commit() and rollback()
  - PlatformTransactionManager Example

```
PlatformTransactionManager transactionManager;
TransactionDefinition tranDef = new DefaultTransactionDefinition();
TransactionStatus tranStatus =            transactionManager.getTransaction(tranDef);
try {
    …
    transactionManager.commit(tranStatus);
} catch (Exception ex) {
    transactionManager.rollback(tranStatus);
}
```

- *TransactionTemplate could be used in this example as well.*
- *Declarative Transactions*

Spring Transaction

- Consistent abstraction
  - `PlatformTransactionManager`
  - Does not reinvent transaction manager
  - Choose between JTA, JDBC, Hibernate, JDO etc *with simple changes to configuration not Java code*
  - No more rewriting application to scale up from JDBC or Hibernate local transactions to JTA global transactions
  - Use the simplest transaction infrastructure that can possibly work
- Programmatic transaction management
  - Simpler API than JTA
  - Use the same API for JTA, JDBC, Hibernate etc.
  - Write once have transaction management everywhere™

Declarative Transaction Management

- Most popular transaction management option
- Built on same abstraction as programmatic transaction management
- Declarative transaction management for any POJO, without EJB: even without JTA (single database)
- More flexible than EJB CMT
  - Declarative *rollback rules*: roll back on MyCheckedException
  - **Non-invasive: Minimizes dependence on the container**
    - No more passing around EJBContext

Transações com Spring

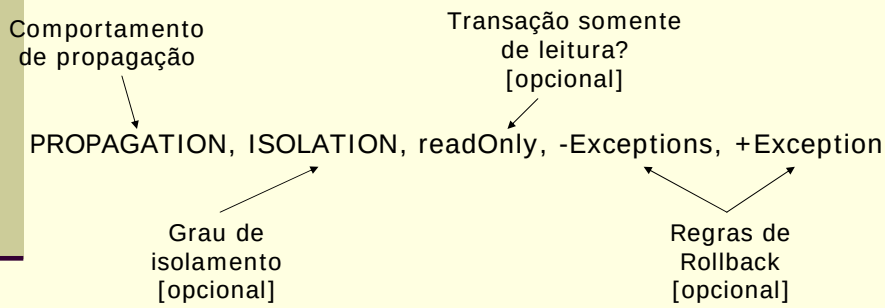
- Suporte para gerenciamento programático e declarativo de transações
- Transações locais são delegadas pelo Spring para o gerente de transações do data-source
- Quando múltiplos recursos estão involvidos (transações distribuídas), Spring delega para o gerente de transações JTA obtido através do JNDI
- Apenas algumas pequenas mudanças são necessárias para trocar entre local e JTA

Transações com Spring

- Gerenciamento Declarativo (+)
  - Usa AOP para encapsular chamadas a objetos transacionais com código de *begin* e *commit* de transações
  - Comportamento de propagação
    - Mandatory, Never, Not Supported, Required, Requires New, Support, Nested
    - Similar a EJBs
  - Também suporta níveis de isolação
    - Default, Read Uncommitted, Read Committed, Repeatable Read, Serializable

Transações com Spring

- Declarando atributos



Os atributos são declarados no arquivo de definição dos beans

Transações com Spring

- Exemplo

```
...
<beans>
<bean id="customerService" class="org.springframework.transaction.
                        interceptor.TransactionProxyFactoryBean">
  <property name="transactionManager">
    <ref bean="transactionManager"/>
  </property>
  <property name="target">
    <ref bean="customerServiceTarget"/>
  </property>
  <property name="transactionAttributes">
    <props>
      <prop key="get*">PROPAGATION_REQUIRED, readOnly</prop>
      <prop key="store*">PROPAGATION_REQUIRED</prop>
    </props>
  </property>
<bean>
...
```

Continua...

Transações com Spring

- Exemplo (cont)

```
...
<bean id="customerServiceTarget" class="CustomerServiceImpl">
  <property name="customerDao">
    <ref bean="customerDao"/>
  </property>
</bean>
<bean id="transactionManager" class="org.springframework.orm.
                        hibernate.HibernateTransactionManager">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
</bean>
<bean id="customerDao" class="HibernateCustomerDao">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
</bean>
...
```


Make ServiceImpl POJO transactional

```
public class ServiceImpl implements Service {
    private int timeout;
    private AccountDao accountDao;

    public void setTimeout(int timeout) {
        this.timeout = timeout;
    }

    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    public void doSomething() throws ServiceWithdrawnException {
    }
}

<bean id="serviceTarget" class="com.mycompany.service.ServiceImpl">
    <property name="timeout"><value>30</timeout></property>
    <property name="accountDao"><ref local="accountDao"/></property>
</bean>
```

Make ServiceImpl transactional

```
<bean id="service"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"/>
    <property name="target">
        <ref local="serviceTarget"/>
    </property>
    <property name="transactionManager">
        <ref local="localTransactionManager"/>
    </property>
    <property name="transactionAttributes">
        <props>
            <prop key="do*">
               PROPAGATION_REQUIRED,-ServiceWithdrawnException
            </prop>
        </props>
    </property>
</bean>
```


Make ServiceImpl transactional

- Rollback rule means that we don't need to call `setRollbackOnly()`
  - Of course Spring also supports programmatic rollback
- Can run this from a JUnit test case
  - Doesn't depend on a heavyweight container
- Can work with JTA, JDBC, Hibernate, JDO, iBATIS transactions…
  - Just change definition of transaction manager

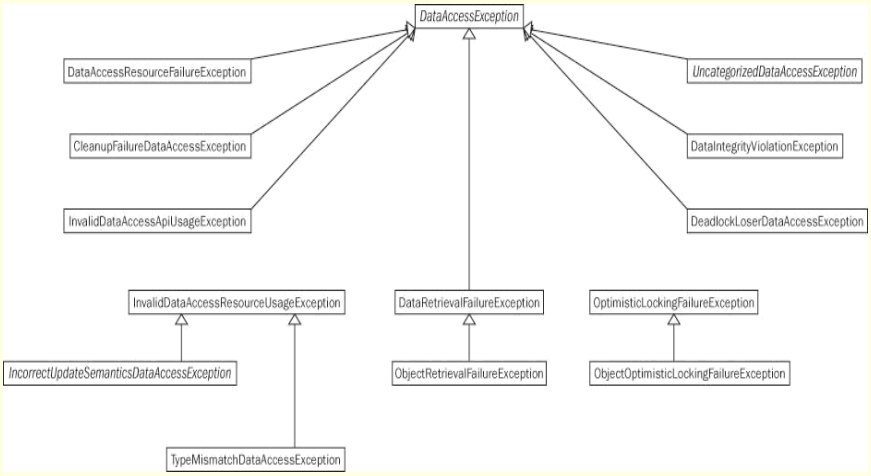
Make ServiceImpl transactional

- Alternative approaches, simpler in large applications:
  - Use “auto proxy creator” to apply similar transaction attributes to multiple beans
  - Use metadata or another pointcut approach to apply transactional behaviour to multiple classes

Spring DAO

- Integrated with Spring transaction management
  - Unique synergy
  - Gestalt again…
- Doesn’t reinvent the wheel.
  - *There are good solutions for O/R mapping, we make them easier to use*
- Out-of-the-box support for
  - JDBC
  - Hibernate
  - JDO
  - iBATIS
- Model allows support for other technologies (TopLink etc)
- Consistent `DataAccessException` hierarchy allows truly technology-agnostic DAOs

Spring DAO: Consistent exception hierarchy



Consistent Abstract Classes for DAO Support

- Extend your DAO classes with the proper xxxDAOSupport class that matches your persistence mechanism.
  - **JdbcDaoSupport**
    - Super class for JDBC data access objects.
    - Requires a DataSource to be set, providing a JdbcTemplate based on it to subclasses.
  - **HibernateDaoSupport**
    - Super class for Hibernate data access objects.
    - Requires a SessionFactory to be set, providing a HibernateTemplate based on it to subclasses.
  - **JdoDaoSupport**
    - Super class for JDO data access objects.
    - Requires a PersistenceManagerFactory to be set, providing a JdoTemplate based on it to subclasses.
  - **SqlMapDaoSupport**
    - Supper class for iBATIS SqlMap data access object.
    - Requires a DataSource to be set, providing a SqlMapTemplate

Spring DAO

- Usando um template jdbc do spring

```
public void updateCustomer(Customer customer) {
    String sql = "update customer set " +
        "firstName = ?, lastName = ?, ...";
    Object[] params = new Object[] {
        customer.getFirstName(),
        customer.getLastName(),
        ...};
    int[] types = new int[] {
        Types.VARCHAR,
        Types.VARCHAR,
        ...};
    jdbcTemplate.update(sql, params, types);
}
```

O jdbcTemplate pode ser injetado pelo container...

Spring DAO

- Operações também podem ser modeladas como objetos

```
public class UpdateCustomer extends SqlUpdate {
    public UpdateCustomer(DataSource ds) {
        setDataSource(ds);
        setSql("update customer set… values (?,?..)");
        declareParameter(new SqlParameter(Types.VARCHAR));
        declareParameter(new SqlParameter(Types.VARCHAR));
        //...
        compile();
    }
    public int update(Customer customer) {
        Object[] params = new Object[] {
             customer.getFirstName(),
             customer.getLastName()
        };
        return update(params);
    }
}
```

Spring DAO

- Usando o objeto UpdateCustomer

```
public class JdbcCustomerDao extends JdbcDaoSupport
                                        implements CustomerDao {

    private UpdateCustomer updateCustomer;

    protected void initDao() throws Exception {
        super.initDao();
        updateCustomer = new UpdateCustomer(getDataSource());
    }

    public void updateCustomer(Customer customer) {
        updateCustomer.update(customer);
    }
}
```

A classe UpdateCustomer pode ser uma *inner class*

Spring

Spring DAO Templates

- Built in code templates that support JDBC, Hibernate, JDO, and iBatis SQL Maps
- Simplifies data access coding by reducing redundant code and helps avoid common errors.
- Alleviates opening and closing connections in your DAO code.
- No more ThreadLocal or passing Connection/Session objects.
- Transaction management is handled by a wired bean
- You are dropped into the template with the resources you need for data access – Session, PreparedStatement, etc.
- Code only needs to be implemented in callback methods.
  - doInXXX(Object)
- Optional separate JDBC framework

Ex: Code without a template

```
public class OrderHibernateDAO implements IOrderDAO {
public Order saveOrder(Order order) throws OrderException{
   Session s = null;
   Transaction tx = null;
   try{
        s = ...  // get a new Session object
        tx = s.beginTransaction();
        s.save(order);
        tx.commit();
   } catch (HibernateException he){
        // log, rollback, and convert to OrderException
   } catch (SQLException sqle){
        // log, rollback, and convert to OrderException
   } finally {
        s.close();  // needs a try/catch block
   }
   return order;
}
```


Ex: Spring DAO Template Example

```
public class OrderHibernateDAO extends HibernateDaoSupport
   implements IOrderDAO {
...
public Order saveOrder(final Order order) {
    HibernateTemplate hib = getHibernateTemplate();
   return (Order)hib.execute(new HibernateCallback() {
        public Object doInHibernate(Session session)
               throws HibernateException, SQLException {
               session.save(order);
               return order;
        }
   });
}
...
}
```

Ex 2: Spring DAO Template Example

```
public class OrderHibernateDAO extends HibernateDaoSupport
   implements IOrderDAO {
...
public List findOrdersByCustomerId(int id) {
  HibernateTemplate hib = getHibernateTemplate();
   return hib.findByNamedQuery("OrdersByCustomerID",
                                  new Integer(id));
}
public Order findOrderById(int orderId ) {
  HibernateTemplate hib = getHibernateTemplate();
   return (Order) hib.load(Order.class,
                             new Integer(orderId));
   }
...
}
```


Spring DAO: JDBC

- Class library offers simpler programming model than raw JDBC
  - Two flavours of usage:
    - Callbacks (JdbcTemplate)
    - JDBC objects: Model queries, updates and stored procedures as objects
- **No more try/catch/finally blocks**
- **No more leaked connections**
  - Spring will *always* close a connection: no scope for programmer error
- *Meaningful* exception hierarchy
  - No more vendor code lookups
    - Spring autodetects database and knows what Oracle, DB2 error codes mean
    - More portable code
  - More readable code
    - `catch (BadSqlGrammarException ex)`
- Stored procedure support
- Can refactor to clean up JDBC without adopting Spring overall
  - *Incremental adoption: Step by step*

Integração ao Hibernate

- Spring prove um bean SessionFactory que simplifica a configuração e o gerênciamento de sessões em objetos de negócio
- Uma classe HibernateTemplate
- Uma classe HibernateDaoSupport que pode ser herdada para maior abstração
- Gerenciamento e mapeamento das HibernateException´s
- Facilmente plugável ao framework de Transações do Spring

Integração ao Hibernate

- Exemplo de configuração

```
...
<beans>
  <bean id="sessionFactory" class="org.springframework.orm.
                                  hibernate.LocalSessionFactoryBean">
  <property name="dataSource">
    <ref bean="dataSource"/>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">
          net.sf.hibernate.dialect.MySQLDialect
       </prop>
    </props>
  </property>
  <property name="mappingResources">
    <list><value>Customer.hbm.xml</value></list>
  </property>
</bean>
```

Continua...

Integração ao Hibernate

- Exemplo de configuração (cont)

```
<bean id="customerDao" class="HibernateCustomerDao">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
</bean>
<bean id="dataSource" class="org.springframework.jndi.
                                    JndiObjectFactoryBean">
  <property name="jndiName">
    <value>java:comp/env/jdbc/myDataSource</value>
  </property>
</bean>
```


Integração ao Hibernate

- Customer DAO...

```
public class HibernateCustomerDao extends HibernateDaoSupport
                                          implements CustomerDao {

    public void updateCustomer(Customer customer) {
        getHibernateTemplate().update(customer);
    }
}
```

- Simples e limpo!
- Existe suporte semelhante para outras tecnologias ORM como iBatis, JDO e OJB

Spring DAO: Hibernate

- Manages Hibernate sessions
  - No more custom ThreadLocal sessions
  - Sessions are managed within Spring transaction management
  - Works with JTA if desired
  - Works within EJB container with CMT if desired
- HibernateTemplate makes common operations easy
  - Simpler, consistent exception handling
  - Many operations become one-liners
  - **Less, simpler, code compared to using Hibernate alone**
- Portability: Switch between Hibernate, JDO and other transparent persistence technologies without changing DAO interfaces
  - Can even switch to JDBC where transparent update is not implied
- Mixed use of Hibernate and JDBC *within the same transaction*

HibernateTemplate DAO example

```
public class MyHibernateDao implements MyDao {

    private HibernateTemplate hibernateTemplate;

    public MyHibernateDao (net.sf.hibernate.SessionFactory sessionFactory) {
            hibernateTemplate = new HibernateTemplate(sessionFactory);
    }

    public Collection getWorkflows() {
            return hibernateTemplate.find("from Workflow");
    }


<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource"><ref local="dataSource"/></property>
    <property name="mappingResources">
            <value>mycompany/mappings.hbm.xml</value>
    </property>
    <property name="hibernateProperties">
            <props>
                    <prop key="hibernate.dialect">net.sf.hibernate.dialect.HSQLDialect</prop>
            </props>
    </property>
</bean>

<bean id="myDao" class="com.mycompany.MyHibernateDao"
    autowire="constructor"
>
```

Spring

Consistent Exception Handling

- Spring has it's own exception handling hierarchy for DAO logic.
- No more copy and pasting redundant exception logic!
- Exceptions from JDBC, or a supported ORM, are wrapped up into an appropriate, and consistent, DataAccessException and thrown.
- This allows you to decouple exceptions in your business logic. These exceptions are treated as unchecked exceptions that you can handle in your business tier if needed. No need to try/catch in your DAO.
- Define your own exception translation by subclassing classes such as SQLErrorCodeSQLExceptionTranslator

Spring

Spring and Testing

- Easier test driven development (TDD)
- Integration testing
  - Can use a standalone Spring configuration with mock objects for testing.
  - Consider XMLApplicationContext or FileSystemApplicationContext.
- Unit testing
  - Allows you to test outside the container without using the Spring container.
- Easy to test POJOs

Even More Spring Components

- JavaMail helpers
- Scheduling support via Quartz
- Convenience implementation classes for
  - Remoting support – JAXRPC, RMI, Hessian, and Burlap
- EJB support for easier access.
- Acegi Security System for Spring
  - http://acegisecurity.sourceforge.net/
  - Very good framework!
- Eclipse Plugin – Spring IDE for Eclipse
- Coming soon
  - JMS implementation classes
  - JMX support