

Spring Framework e Java 5 trazendo produtividade e Confiabilidade para o desenvolvimento Java EE

Por Rodrigo Urubatan Ferreira
Jardim

Sobre o Palestrante

- Rodrigo Urubatan - SCJP 1.4 e SCWCD
- Trabalha com arquitetura de sistemas J2EE e treinamento
- Já desenvolveu projetos utilizando as linguagens Delphi, C++, PHP, ASP, ColdFusion, Leather, Assembly, Perl, ...
- Trabalha com Java/J2EE a 4 anos e com desenvolvimento de sistemas a 9 anos
- Atualmente colabora com pequenas correções a alguns projetos Open Source como o GUVJ2, Lombok e Velocity e faz parte da coordenação do RSJUG
- Já ministrou palestras em universidades (UCS, ULBRA, UNISC) e diversos eventos (Just Java, FISL, Seminário do RSJUG, Maratona 4 Java, Infosul) tutoriais para o RSJUG e já teve um artigo publicado na revista Mundo Java
- Atualmente trabalha como consultor na AdvancedIT, como gerente de tecnologia e qualidade na Tech Office IT, e ministra cursos e alguns pequenos projetos pela USI Informática
- É o principal desenvolvedor do projeto Spring-Annotation

Spring Framework

- J2EE deveria ser mais fácil de utilizar
- É melhor programar orientado a interfaces do que a classes. Spring reduz a complexidade de programar voltado a interfaces a quase zero.
- JavaBeans oferecem um ótimo meio de configurara aplicações.
- O design OO é mais importante do que qualquer tecnologia, como o J2EE.
- Exceções checadas são utilizadas em excesso no Java. Um framework não deveria te obrigar a tratar exceções, a não ser que fosse possível se recuperar delas.

O que é o Spring Framework

- Basicamente o Spring Framework é um conjunto de componentes reutilizáveis
 - Container de IoC
 - Framework AOP
 - MVC
 - Hierarquia de exceções
 - Facilidades para acesso a dados e utilização de frameworks OR
 - JMS
 - JMX
 - Scripting
 - ...

Inversão de Controle / Injeção de Dependencia

No início de 2004 Martin Fowler perguntou em um forum “quais aspectos do controle deste sistema estamos invertendo” e logo depois ele sugeriu que o nome do pattern/princípio fosse alterado, ou fosse um pouco mais “auto explicativo”, para uma maior explicação do que o Fowler chama de Injeção de dependencias, pode ser consultada a URL

<http://martinfowler.com/articles/injection.html>

Bean Factory

- O que é e para que serve
- Bean Definition
- A classe para o Java Bean
 - Inicialização por construtor
 - Inicialização via factory method estático
 - Inicialização via factory method de instancia
- Identificadores para o Bean (id e name)
- Singleton ou não Singleton

Configuração da Bean Factory

- É por default baseada em XML
- A partir da versão 2.0 utiliza XML Schema para facilitar a configuração
- Propriedades dos beans podem ser configuradas via XML

Lets Play



Exemplo 1

- Criar um projeto Java no Eclipse, adicionar ao build path os jars spring.jar e commons-logging.jar
- Criar uma classe com o nome de Exemplo1 com um método main, e uma propriedade write only de nome mensagem
- Criar um método showMessage que imprime a propriedade mensagem no console
- Criar um arquivo de nome exemplo1.xml para configurar o bean no spring

Exemplo1

- Configurar o valor default da propriedade mensagem para “Bean criado manualmente”
- Adicionar ao XML do spring uma entrada `<bean>` com o id configurado como “exemplo1”, class para a classe criada e com a propriedade mensagem configurada para qualquer valor diferente do default.

Exemplo1.java

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
public class Exemplo1 {
private String mensagem = "Bean criado manualmente";
public void setMensagem(String mensagem) {
this.mensagem = mensagem;
}
private void showMessage() {
System.out.println(mensagem);
}
public static void main(String[] args) {
BeanFactory bf = new XmlBeanFactory(new ClassPathResource("/exemplo1.xml"));
Exemplo1 created = new Exemplo1();
Exemplo1 fromSpring = (Exemplo1) bf.getBean("exemplo1");
created.showMessage();
fromSpring.showMessage();
}
}
```

exemplo1.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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean
id="exemplo1"
class="br.com.techoffice.cj2006.exemplo1.Exemplo1">
<property
name="mensagem"
value="Criado pelo spring" />
</bean>
</beans>
```

Exemplo 2

- Criar uma interface de nome RegraDeNegocio com um método executar
- Criar uma classe RegraDeNegociImpl que implemente a interface já criada, com um `System.out.println` dentro do método executar

Exemplo 2

- Criar uma classe Exemplo 2
- Nesta classe criar uma propriedade do tipo RegraDeNegocio
- Criar um arquivo exemplo2.xml
- Configurar uma instancia da classe RegraDeNegocioImpl e uma instancia da classe Exemplo2
- Configurar a propriedade da classe Exemplo2 apontando para o bean que implementa a interface RegraDeNegocio
- Criar um método main na classe exemplo2, onde sera buscado da bean factory o bean exemplo2 e este ira chamar o metodo executar do bean regraDeNegocio

Exemplo2

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class Exemplo2 {
    private RegraDeNegocio regra;

    public Exemplo2(RegraDeNegocio regra) {
        this.regra = regra;
    }

    public void executarRegra() {
        regra.executar();
    }

    public static void main(String args[]) {
        BeanFactory bf = new XmlBeanFactory(new ClassPathResource("/exemplo2.xml"));
        Exemplo2 bean = (Exemplo2) bf.getBean("exemplo2");
        bean.executarRegra();
    }
}
```

RegraDeNegocio

```
package br.com.techoffice.cj2006.exemplo2;
```

```
public interface RegraDeNegocio {
```

```
void executar();
```

```
}
```

RegraDeNegocioImpl

```
public class RegraDeNegocioImpl implements
    RegraDeNegocio {

    public void executar() {
        System.out.println("Regra de negócio executada");
    }

}
```

exemplo2.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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean
id="regraDeNegocio"
class="br.com.techoffice.cj2006.exemplo2.RegraDeNegocioImpl" />
<bean
id="exemplo2"
class="br.com.techoffice.cj2006.exemplo2.Exemplo2">
<constructor-arg ref="regraDeNegocio" />
</bean>
</beans>
```

Recursos avançados da BeanFactory

- Interfaces para o ciclo de vida
 - Inicialização/init-method
 - Finalização/destroy-method
- Sabendo quem você é
 - BeanNameAware
 - BeanFactoryAware
 - ApplicationContextAware
- Extendendo o framework
 - BeanFactoryPostProcessor
 - BeanPostProcessor

Lets Play



Exemplo 3

- Criar uma classe de nome Exemplo3 que contenha os metodos init e destroy
- Esta classe deve implementar a interface BeanNameAware
- Dentro do metodo init a classe deve imprimir “nome” foi criado, e dentro do metodo destroy deve imprimir “nome” foi destruido
- Instanciar um ApplicationContext em vez de uma bean factory e registrar o “shutdownHook”
- Obter do application context uma instancia da classe Exemplo3 que deve ter sido configurada em um arquivo de nome exemplo3.xml

Exemplo3.java

```
public class Exemplo3 implements BeanNameAware{
    private String name;
    public void setBeanName(String name) {
        this.name = name;
    }
    public void init(){
        System.out.println(name + " inicializado");
    }
    public void destroy(){
        System.out.println(name + " destruido");
    }
    public static void main(String[] args){
        ConfigurableApplicationContext appctx = new ClassPathXmlApplicationContext("/exemplo3.xml");
        appctx.registerShutdownHook();
        appctx.getBean("teste");
        System.out.println("aqui podemos executar regras de negócio a vontade");
    }
}
```

Exemplo3.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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-
    2.0.xsd">
<bean
id="teste"
class="br.com.techoffice.cj2006.exemplo3.Exemplo3"
init-method="init" destroy-method="destroy">
</bean>
</beans>
```

Escopos padrão

- O que é Escopo de um Bean
- Singleton
- Prototype
- Quando utilizar um ou outro
- Outros escopos

Exemplo 4

- Criar uma classe Exemplo4 com um campo contador estático que seja incrementado toda vez que uma instancia da classe for criada, e um campo id que guarda o valor que este contador tinha no momento em que cada instancia foi criada
- Adicionar um método showId que imprime o ID da classe no console
- Criar um arquivo exemplo4.xml com 2 instancias da classe Exemplo4 configurados uma configurada com o escopo “singleton” e outra configurada com o escopo prototype
- Dentro do método main, buscar 5 vezes cada uma das instancias e chamar o método showId de cada uma delas.

Exemplo4.java

```
public class Exemplo4 {
    private static int counter = 1;
    private int id;
    public Exemplo4() {
        id = counter++;
    }
    public void showId() {
        System.out.println("bean ID: " + id);
    }
    public static void main(String[] args) {
        BeanFactory bf = new XmlBeanFactory(new ClassPathResource(
            "/exemplo4.xml"));
        for (int i = 0; i < 5; i++) {
            for (String name : new String[] { "testesing", "testeprot" }) {
                Exemplo4 bean = (Exemplo4) bf.getBean(name);
                System.out.println(name);
                bean.showId();
            }
        }
    }
}
```

Exemplo4.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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd">
<bean id="testesing"
class="br.com.techoffice.cj2006.exemplo4.Exemplo4"
scope="singleton"/>
<bean id="testeprot"
class="br.com.techoffice.cj2006.exemplo4.Exemplo4"
scope="prototype"/>
</beans>
```

Lets Play



Acesso a dados

- Hierarquia de excessões detalhada
- Suporte a Dao
- Templates

Exemplo 5

- Criar um arquivo exemplo5.xml
- Configurar um dataSource
- Iniciar o servidor do hsqldb
- Criar uma tabela exemplo5 com um campo id e um campo nome
- Criar uma classe que estende SqlUpdate para realizar as inserções no banco de dados
- Criar uma classe Exemplo5Data com um campo id e um campo nome
- Criar uma classe que estenda MappingSqlQuery para realizar as consultas e transformar os resultados em objetos
- Criar uma classe Exemplo5 com uma propriedade insertCommand e uma propriedade queryCommand respectivamente do tipo SqlUpdate e MappingSqlQuery
- Na classe Exemplo5 criar um método executarInserts que vai inserir algumas linhas e um método executarConsulta que vai listar os dados
- No método main, buscar de uma beanFactory uma instancia da classe Exemplo5 e chamar em sequencia os metodos executarInserts e executarConsulta

Banco de dados

- Servidor
- Java -cp hsqldb.jar org.hsqldb.Server -database.0 mydb -dbname.0 xdb
- Ferramenta de administração:
- java
org.hsqldb.util.DatabaseManagerSwing

Lets Play



Exemplo5.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"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
<property name="driverClassName" value="org.hsqldb.jdbcDriver" />
<property name="url" value="jdbc:hsqldb:hsqldb://localhost/xdB" />
<property name="username" value="sa" />
<property name="password" value="" />
</bean>
<bean id="insert" class="br.com.techoffice.cj2006.exemplo5.InsertBean">
<constructor-arg ref="dataSource"/>
</bean>
<bean id="query" class="br.com.techoffice.cj2006.exemplo5.QueryBean">
<constructor-arg ref="dataSource"/>
</bean>
<bean id="teste" class="br.com.techoffice.cj2006.exemplo5.Exemplo5">
<property name="insertCommand" ref="insert"/>
<property name="queryCommand" ref="query"/>
</bean>
</beans>
```

Exemplo5.java

```
public class Exemplo5 {
    private MappingSqlQuery queryCommand;
    private SqlUpdate insertCommand;
    public void setInsertCommand(SqlUpdate insertCommand) {
        this.insertCommand = insertCommand;
    }
    public void setQueryCommand(MappingSqlQuery queryCommand) {
        this.queryCommand = queryCommand;
    }
    public void executarInserts(){
        insertCommand.update("Rodrigo");
        insertCommand.update("Urubatan");
        insertCommand.update("Conexão");
    }
    public void executarConsulta(){
        List<Exemplo5Data> result = queryCommand.execute();
        for(Exemplo5Data ex : result){
            System.out.format("ID: %s, Nome: %s\n", ex.getId(),ex.getNome());
        }
    }
    public static void main(String[] args){
        BeanFactory bf = new XmlBeanFactory(new ClassPathResource(
            "/exemplo5.xml"));
        Exemplo5 bean = (Exemplo5) bf.getBean("teste");
        bean.executarInserts();
        bean.executarConsulta();
    }
}
```

Exemplo5Data

```
public class Exemplo5Data {  
    private int id;  
    private String nome;  
    public Exemplo5Data(int id, String nome) {  
        super();  
        this.id = id;  
        this.nome = nome;  
    }  
    public int getId() {  
        return id;  
    }  
    public String getNome() {  
        return nome;  
    }  
}
```

InsertBean

```
public class InsertBean extends SqlUpdate {  
  
    public InsertBean(DataSource ds) {  
        super(ds, "insert into  
            exemplo5(nome) values (?)", new  
            int[] {Types.VARCHAR});  
    }  
  
}
```

QueryBean

```
public class QueryBean extends MappingSqlQuery {

    public QueryBean(DataSource ds) {
        super(ds, "select id,nome from exemplo5");
    }

    @Override
    protected Object mapRow(ResultSet rs, int rowNum) throws
        SQLException {
        return new Exemplo5Data(rs.getInt("id"),rs.getString("nome"));
    }

}
```

Controle de transações

- Suporte a controle de transações programático ou declarativo
- Transparencia do provedor de transações (JDBC Nativo, JTA, ...)
- Suporte aos mesmos níveis de transação e isolamento disponíveis em containers Java EE

Exemplo 6 – transações programáticas

- Alterar o exemplo5.xml para incluir um platform transaction manager
- Alterar o Exemplo5 para executar o metodo executarInserts dentro de um transactionCallback passado para um TransactionTemplate

Alterações exemplo5.xml

```
<bean id="txManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
      >
<property name="dataSource" ref="dataSource" />
</bean>
<bean id="teste" class="br.com.techoffice.cj2006.exemplo6.Exemplo6">
<property name="insertCommand" ref="insert" />
<property name="queryCommand" ref="query" />
<property name="transactionManager" ref="txManager" />
</bean>
```

Alterações Exemplo5.java

```
public void executarInserts() {
    new TransactionTemplate(transactionManager)
        .execute(new TransactionCallback() {
            public Object doInTransaction(TransactionStatus status) {
                insertCommand.update("Rodrigo");
                insertCommand.update("Urubatan");
                insertCommand.update("Conexão");
                insertCommand.update("Java");
                insertCommand.update("2006");
                return null;
            }
        });
}
```

Exemplo 7 – Transações declarativas

- Alterar o exemplo5.xml para incluir um transaction manager
- Alterar o exemplo5.xml para inserir o controle de transações via AOP

Alterações exemplo5.xml

```
<bean id="txManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
      >
<property name="dataSource" ref="dataSource" />
</bean>
<tx:advice id="txAdvice" transaction-manager="txManager">
<tx:attributes>
<tx:method name="*" />
</tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">
<aop:pointcut id="inserts" expression="execution(*
      br.com.techoffice.cj2006.exemplo7.InsertBean.*(..))" />
<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation" />
</aop:config>
```

Spring MVC

- Amplamente customizável, mas não tão fácil de customizar
- Interface Controller
- Implementações de controller padrão
- Resolvendo URLs
- Resolvendo Views

Lets Play



Exemplo 8 – um controller simples

- Criar um projeto web no eclipse
- Copiar os jars spring.jar, commons-pool.jar, commons-dbcj.jar, commons-logging.jar, hsqldb.jar, aspectjweaver.jar, aspectjrt.jar e cglib-nodep.jar para o diretório WEB-INF/lib do projeto.
- Criar uma classe que implemente Controller
- Criar um arquivo /WEB-INF/app-servlet.xml
- Registrar o servlet do spring no web.xml
- Registrar o controller criado no app-servlet.xml
- Criar um arquivo hello.jsp
- Redirecionar do controller para o hello.jsp retornando um ModelAndView(“/hello.jsp”);

Exemplo8Controller

```
public class Exemplo8Controller implements
    Controller {
    public ModelAndView handleRequest(HttpServletRequest
        request,
        HttpServletResponse response) throws Exception {
        return new ModelAndView("/hello.jsp", "mensagem", "Ola
            Conexão Java!");
    }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>conexaojava_exemploweb</display-name>
<servlet>
<servlet-name>app</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>app</servlet-name>
<url-pattern>*.sp</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

App-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd
       http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop.xsd">
<bean name="/hello.sp"
      class="br.com.techoffice.cj2006.exemplo8.Exemplo8Controller"/>
</beans>
```

Hello.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>CJ 2006</title>
</head>
<body>
    ${mensagem}
</body>
</html>
```

Suporte a view technologies

- JSP/JSTL
 - Taglib para formulários
- Velocity
- XSL
- Document (Word e Excel)
- PDF

Lets Play



Exemplo 9 – utilizando velocity

- Configurar o suporte a velocity no app-servlet.xml
- Criar uma classe que implemente Controller e no metodo handleRequest retorne um new ModelAndView(“home”);
- Criar um arquivo /WEB-INF/velocity/hello.vm
- Adicionar o velocity.jar e o commons-collection.jar ao WEB-INF/lib

Hello.vm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
  "http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-  
  8859-1">  
<title>CJ 2006</title>  
</head>  
<body>  
  ${mensagem}  
</body>  
</html>
```

Exemplo9Controller.java

```
public class Exemplo9Controller implements
    Controller{
public ModelAndView handleRequest(HttpServletRequest
    request, HttpServletResponse response) throws
    Exception {
return new ModelAndView("hello", "mensagem", "Ola
    Conexão Java! Velocity!!!");
}
}
```

App-servlet.xml

```
<bean name="/hellovel.sp" class="br.com.techoffice.cj2006.exemplo9.Exemplo9Controller"
  />
<bean id="velocityConfig"
  class="org.springframework.web.servlet.view.velocity.VelocityConfigurer">
  <property name="resourceLoaderPath" value="/WEB-INF/velocity/" />
</bean>
<bean id="velocityViewResolver"
  class="org.springframework.web.servlet.view.velocity.VelocityViewResolver">
  <property name="cache" value="true" />
  <property name="prefix" value="" />
  <property name="suffix" value=".vm" />
  <property name="exposeSpringMacroHelpers" value="true" />
</bean>
<bean id="defaultViewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
```

Usando menos XML - AutoWire

- O que é autowire
- Vantagens e desvantagens
- Tipos de autowire

Lets Play



Exemplo 10 – autowire por nome

- Alterar no exemplo2.xml o nome do bean de regra de negócio para “regra”
- Remover o parametro do contrutor do bean exemplo 2
- Configurar para o exemplo2 o autowire para “byName”
- Alterar a classe Exemplo2, remover o parametro de contrutor e criar um setter para a propriedade “regra”

Exemplo10.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-
       2.0.xsd">
  <bean id="regra"
        class="br.com.techoffice.cj2006.exemplo10.RegraDeNegocioImpl"
        />
  <bean id="exemplo"
        class="br.com.techoffice.cj2006.exemplo10.Exemplo10"
        autowire="byName"></bean>
</beans>
```

Exemplo10.java

```
public class Exemplo10 {  
    private RegraDeNegocio regra;  
  
    public void setRegra(RegraDeNegocio regra) {  
        this.regra = regra;  
    }  
    public void executarRegra() {  
        regra.executar();  
    }  
    public static void main(String args[]){  
        BeanFactory bf = new XmlBeanFactory(new ClassPathResource("/exemplo10.xml"));  
        Exemplo10 bean = (Exemplo10) bf.getBean("exemplo");  
        bean.executarRegra();  
    }  
}
```

Usando ainda menos XML - Anotações

- Controle de transações via anotação
- Propriedades obrigatórias
- Suporte a `@AspectJ`
- `@Configurable`

Lets Play



Exemplo 11 – controle de transações

- Anotar o método executar inserts do Exemplo7 com @Transactional
- Alterar o exemplo7.xml para utilizar o gerenciamento de transações via anotações

Exemplo11.java

```
@Transactional
```

```
public void executarInserts () {  
    insertCommand.update ("Rodrigo");  
    insertCommand.update ("Urubatan");  
    insertCommand.update ("Conexão");  
    insertCommand.update ("Java");  
    insertCommand.update ("2006");  
}
```

Exemplo11.xml

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
<property name="driverClassName" value="org.hsqldb.jdbcDriver" />
<property name="url" value="jdbc:hsqldb:hsql://localhost/xdB" />
<property name="username" value="sa" />
<property name="password" value="" />
</bean>
<bean id="insert" class="br.com.techoffice.cj2006.exemplo11.InsertBean">
<constructor-arg ref="dataSource" />
</bean>
<bean id="query" class="br.com.techoffice.cj2006.exemplo11.QueryBean">
<constructor-arg ref="dataSource" />
</bean>
<bean id="teste" class="br.com.techoffice.cj2006.exemplo11.Exemplo7">
<property name="insertCommand" ref="insert" />
<property name="queryCommand" ref="query" />
</bean>
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
<property name="dataSource" ref="dataSource" />
</bean>
<tx:annotation-driven proxy-target-class="true"/>
```

Usando muito menos XML – Spring-Annotation

- O que é o Spring-Annotation?
- Configurando a bean factory sem XML
- Configuração mista, o que deve ser XML e o que não deve
- SpringMVC sem XML
- Suporte a JSF no Spring-Annotation

Lets Play



Exemplo 12 – Spring MVC

- Adicionar os jars spring-annotation-web e spring-annotation-base no WEB-INF/lib
- Fazer uma copia do controller do exemplo 9
- Anotar a classe com `@Bean` e `@RequestMapping`
- Alterar o `ap-servlet.xml` para incluir o suporte a anotações.

App-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:sa="https://spring-
  annotation.dev.java.net/context"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
  http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
  https://spring-annotation.dev.java.net/context https://spring-
    annotation.dev.java.net/context.xsd">
<sa:annotation-autoload />
<import resource="classpath*:applicationContext.xml"/>
```

Exemplo12Controller.java

```
@Bean
@UrlMapping ("/testespa.sp")
public class Exemplo12Controller implements
    Controller{
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws
        Exception {
    return new ModelAndView("hello", "mensagem", "Ola
        Conexão Java! Velocity!!!");
    }
}
```

Referencias

- <http://www.springframework.org>
- <http://blog.urubatan.com.br>
- <http://www.guj.com.br>
- <https://spring-annotation.dev.java.net/nonav/>
- <http://www.portaljava.com>