



Spring Framework, todos os recursos
do J2EE sem complicação.

Rodrigo Urubatan Ferreira Jardim

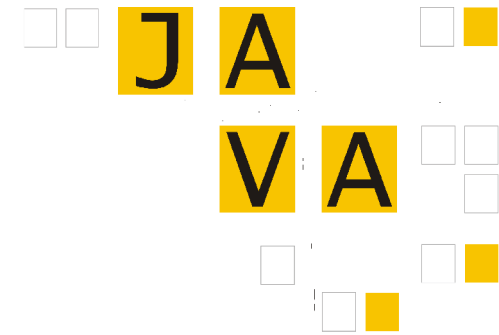
Arquiteto J2EE/Consultor

Immediate Consultoria

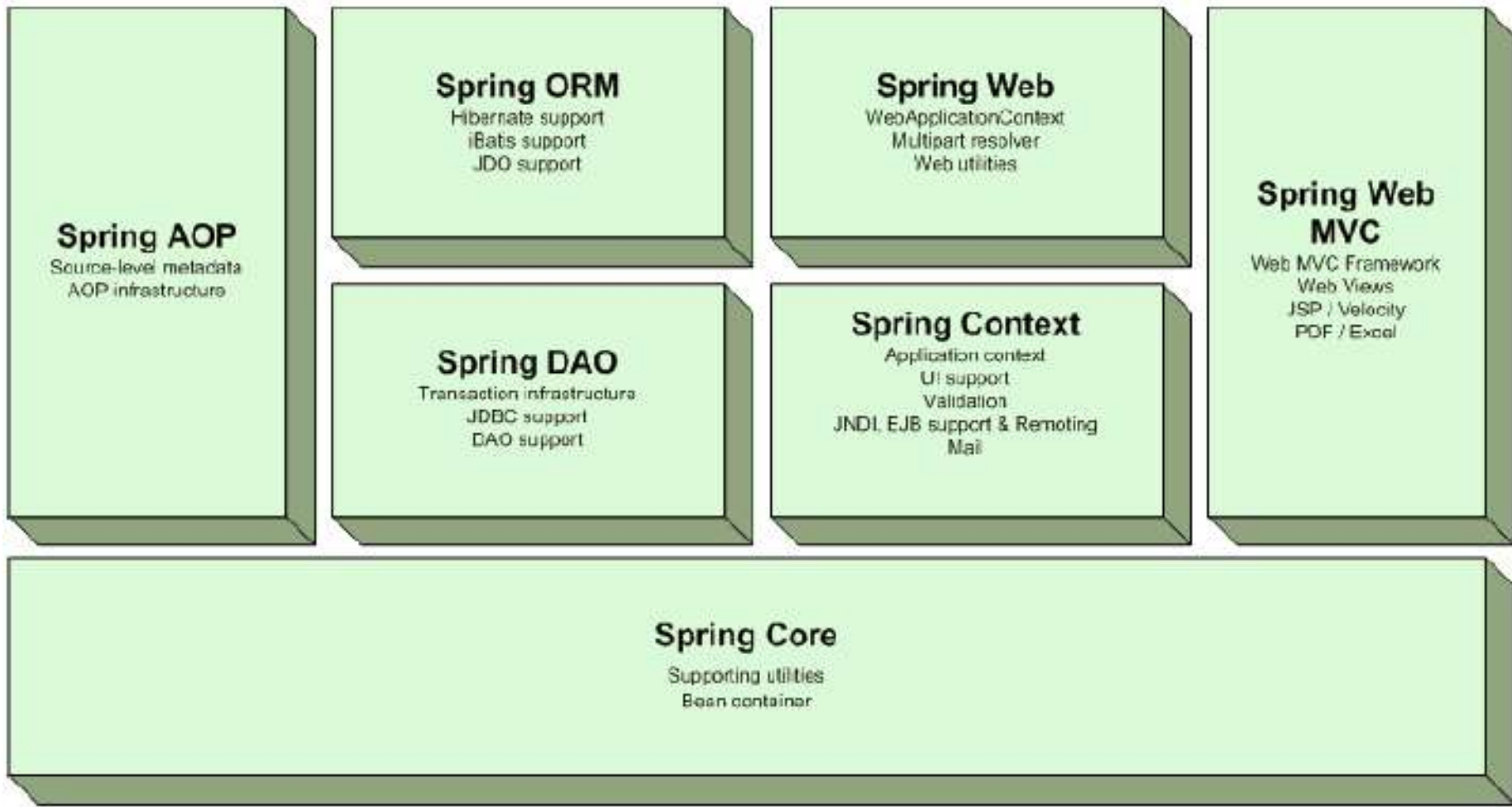
rodrigo@usiinformatica.com.br

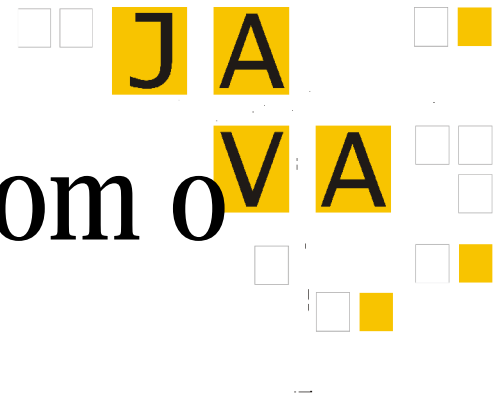
<http://www.usiinformatica.com.br>

<http://www.immediate.com.br>



Spring Framework





Desenvolvendo aplicações com o Spring Framework

Origem e Filosofia

Desenvolvendo para Interfaces e não para Classes Concretas

Configurando o framework

Desenvolvendo aplicações WEB

Enviando e-mails

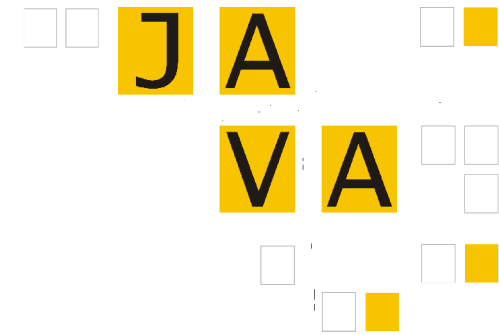
Facilidades para acesso a dados

Acessando EJBs

Editores de propriedades

Utilizando AOP de maneira fácil

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



Origem e Filosofia

Expert One-to-One J2EE Design and Development by Rod Johnson

J2EE deve ser mais fácil de utilizar

Design OO é mais importante do que qualquer tecnologia Utilizada (Incluindo J2EE)

Checked Exceptions são utilizadas em excesso no java

É melhor programar para Interfaces do que para Classes

Programar com o Spring deve ser um prazer

A sua aplicação não deve depender do Spring (ou depender o menos possível)

Desenvolvedores principais: Jürgen Höller e Rod Johnson

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



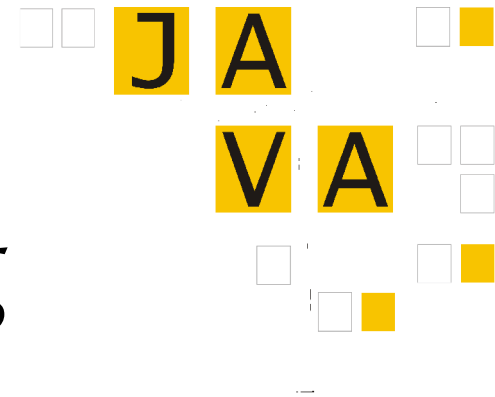
Desenvolvendo para Interfaces e não para classes concretas

Facilidade para a troca de implementação

Menor acoplamento entre os componentes

Indiferença se o componente utilizado é local ou remoto

Compatibilidade com a futura utilização de AOP e criação de proxies para adicionar verificações e features não incluídas no projeto inicial



Configuração do Spring

BeanFactory - é a casa das BeanDefinitions, ou seja, a unidade básica de configuração do spring framework

XMLBeanFactory - uma implementação de BeanFactory configurada via um arquivo XML

Interfaces para interação com o framework

ApplicationContext

WebApplicationContext

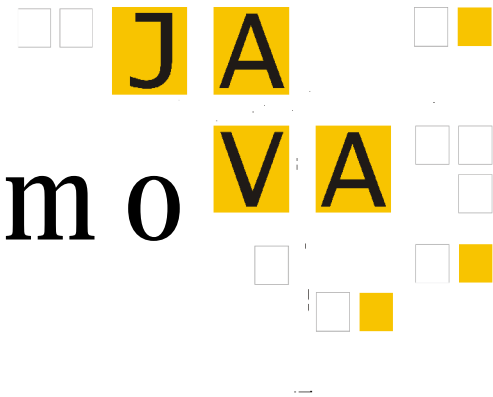


Inversão de Controle/Injeção de Dependências

A Bean Factory do Spring é um Container para Injeção de Dependências

Inversão de Controle é um conceito mais amplo do que Injeção de Dependência

A Inversão de Controle pode ser alcançada de uma maneira mais completa utilizando a BeanFactory junto com o suporte a AOP do Spring Framework



Interfaces para Interação com o Framework

BeanFactoryAware

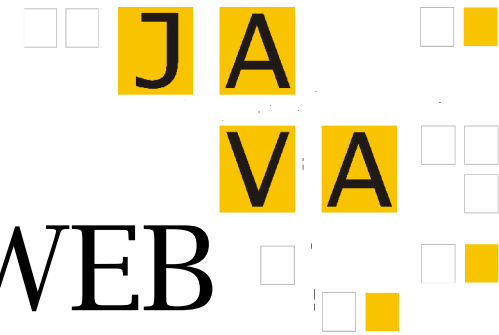
BeanNameAware

FactoryBean

BeanFactoryPostProcessor

PropertyResourceConfigurer

PropertyPlaceholderConfigurer



Desenvolvendo aplicações WEB

Configurando a aplicação

Controllers

Validando formulários

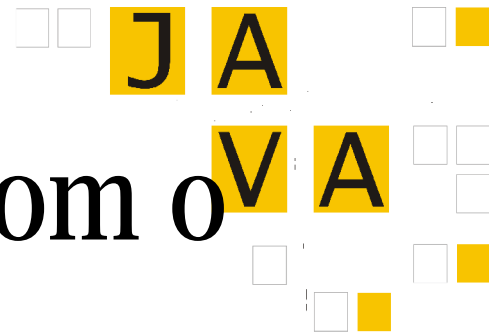
Mapeando requests

Resolvendo Views

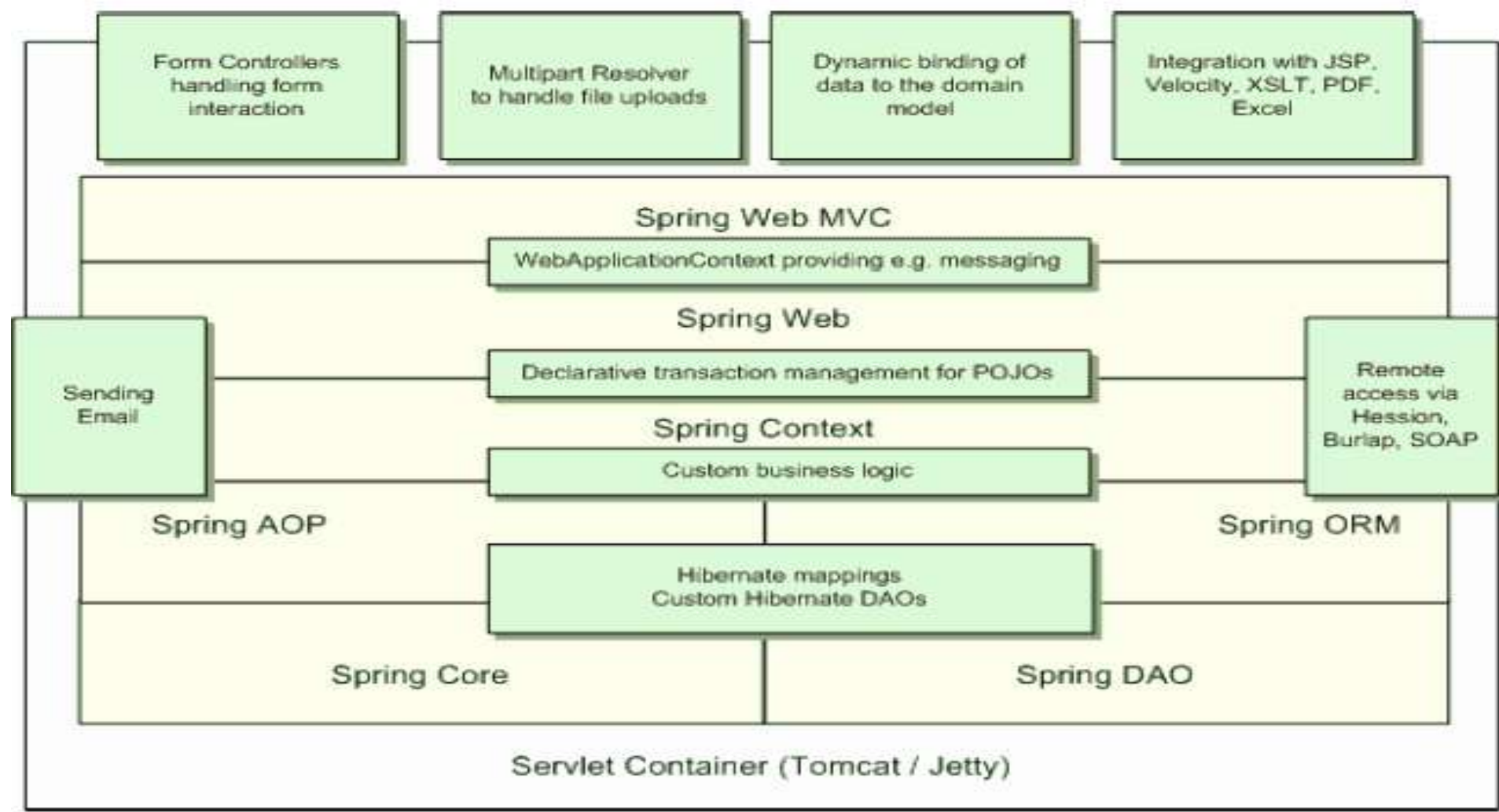
Integrando tecnologias de visualização

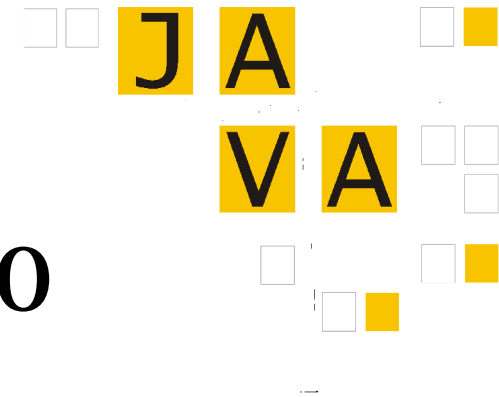
Localizando a aplicação e utilizando temas

Tratando exceções



Aplicação WEB Completa com o Spring Framework





Configurando a aplicação

O primeiro passo para utilizar o spring em uma aplicação WEB, é informar ao framework quais serão os arquivos de configuração globais.

arquivos `web.xml` e `applicationContext.xml`

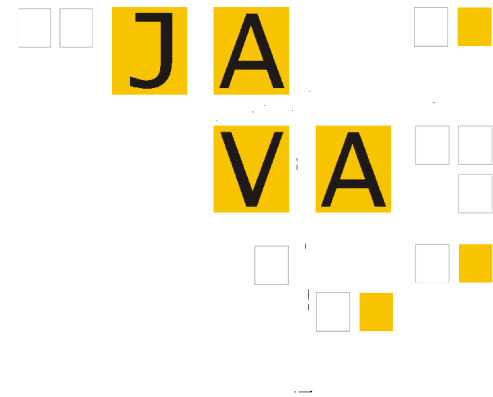
Configurar o listener que ira carregar o contexto.

e opcionalmente configurar um contexto para cada servlet/dispatcher do framework utilizado.

Arquivo `<servletname>-servlet.xml`

Opcionalmente configurar o framework MVC escolhido, caso este não seja o default do spring.

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



web.xml

```
<?xml version="1.0" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<!-- Copyright (c) 2002 by ObjectLearn. All Rights Reserved. -->
<web-app>

<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/applicationContext-hibernate.xml</param-value>
</context-param>

<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<servlet>
<servlet-name>exemplo</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>exemplo</servlet-name>
<url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

Controllers

Controllers no framework MVC do Spring são equivalentes as Actions do Struts ou do WebWork.

No framework MVC do Spring, existem 3 tipos básicos de Controllers

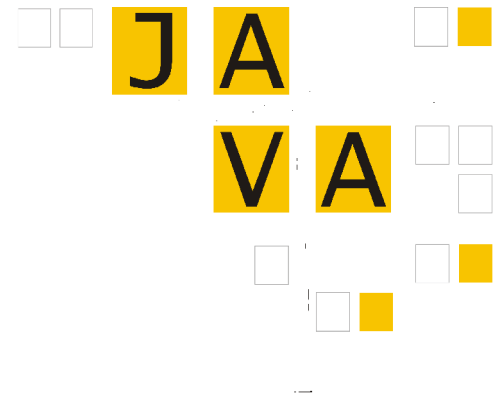
Controllers Simples - Apenas estender a classe `AbstractController`

`MultiActionController` - Um Controller implementa diversas ações, semelhante a uma action implementando a interface `Command` do WebWork

`CommandController` - são Controladores utilizados para facilitar a utilização de formulários e o binding dos parametros destes formulários a estes JavaBeans

`SimpleFormController` - Trás diversas funcionalidades de formulário, validação de input e página de sucesso

`WizardController` - facilita a implementação de formulários multi página, como wizards, controla o estado de cada página, ...



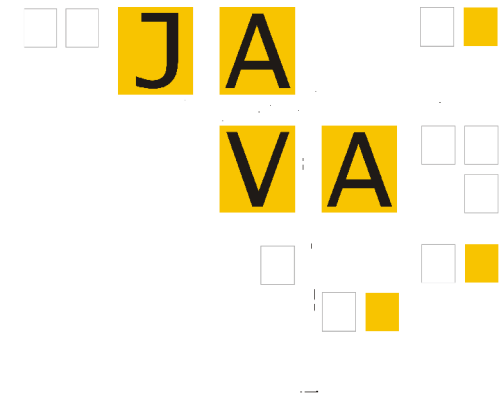
Validando Formulários

Utilizando o framework MVC do Spring, existem duas formas default de validação de formulários:

utilizando o commons-validator do projeto jakarta, para isto basta informar no XML do spring a utilização deste validator para um controller específico.

Utilizar uma classe implementando a interface Validator, desta forma tendo uma validação muito mais flexível e podendo ser realizada mesmo dentro do Dao responsável pelo objeto, facilitando validações do tipo: o objeto já existe, o nome de usuário solicitado já está sendo utilizado, ..., este é o método preferido de validação de formulários no Spring

Ao utilizar outro framework MVC a validação deste deveria ser utilizada, mas o spring pode continuar utilizando o framework de validação nas camadas mais baixas, como os beans de negócio ou acesso a dados.



Mapeando Requests

BeanNameUrlHandleMapping

Direciona os requests para um bean com o mesmo nome do excedente da URL.

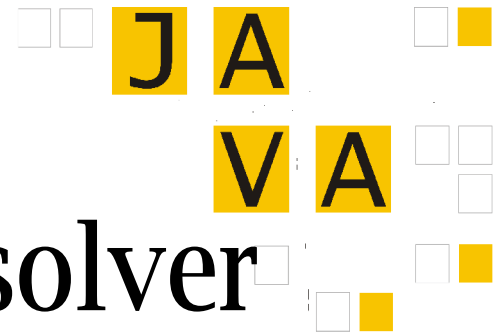
SimpleUrlHandlerMapping

Mapeia requests baseado em WildCards parecidos com os utilizados para diretórios no ANT

HandlerInterceptors

Podem ser adicionados interceptors a qualquer mapeamento de requests, por exemplo para controlar o horario de acesso as páginas ou alguma coisa relacionada a segurança

```
<bean id="handlerMapping"
class="org.springframework.web.servlet.handl
er.SimpleUrlHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="officeHoursInterceptor"/>
    </list>
  </property>
  <property name="mappings">
    <props>
      <prop
        key="/*.form">editAccountFormController</
        prop>
      <prop
        key="/*.view">editAccountFormController</
        prop>
    </props>
  </property>
</bean>
<bean id="officeHoursInterceptor"
class="samples.TimeBasedAccessInterceptor">
  <property name="openingTime">
    <value>9</value>
  </property>
  <property name="closingTime">
    <value>18</value>
  </property>
</bean>
```



Resolvendo Views - viewResolver

A resolução de views no framework MVC do Spring, é bem flexível e suporta diversos tipos de recursos.

Esta é realizada como a maior parte do framework por um Bean configurado no contexto.

Os viewResolvers que vem implementados por default são os seguintes:

- AbstractCachingViewResolver
- ResourceBundleViewResolver
- UrlBasedViewResolver
- InternalResourceViewResolver
- VelocityViewResolver



Tecnologias de Visualização

Configurando um dos viewResolvers disponiveis, pode-se utilizar o spring com as seguintes tecnologias de View:

JSP/JSTL

Tiles

Velocity

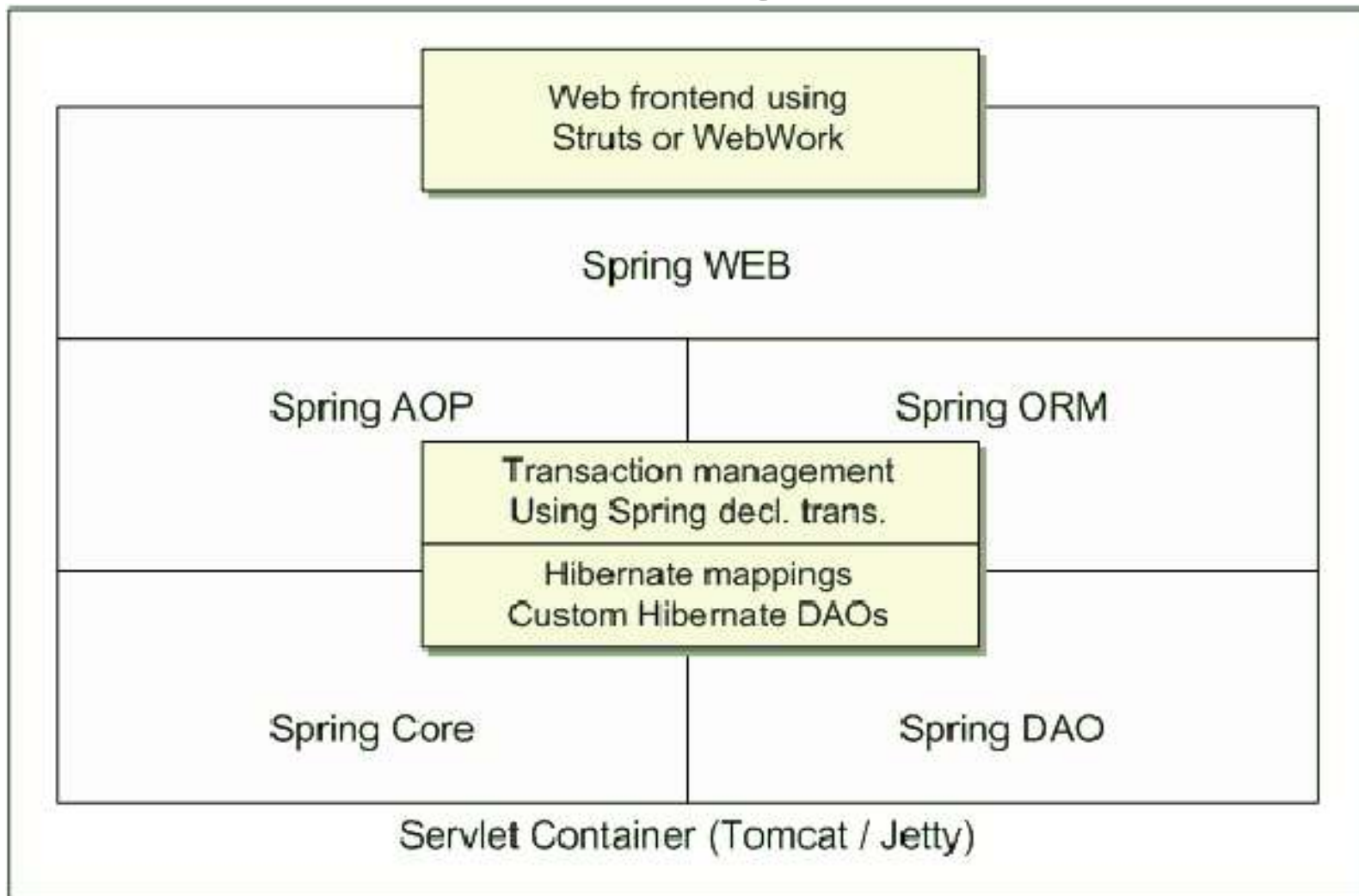
XSLT

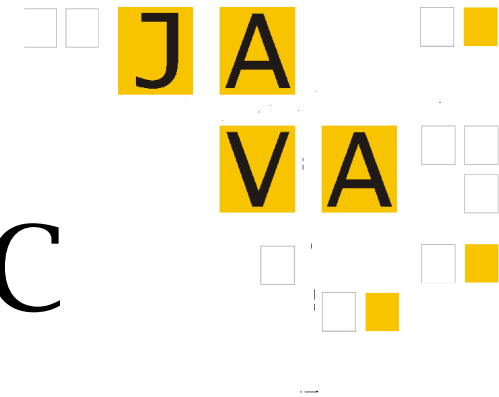
PDF

Excel



Integrando com outro Controlador MVC





Outros Frameworks MVC

O spring é facilmente integravel a outros frameworks MVC não obrigando a utilização do próprio.

os seguintes frameworks podem se beneficiar facilmente das facilidades do Spring:

Struts

Tapestry

WebWork (1/2)



Localizando a aplicação e utilizando temas

A maior parte do Spring Framework é localizável.

existem os seguintes localeResolvers já implementados:

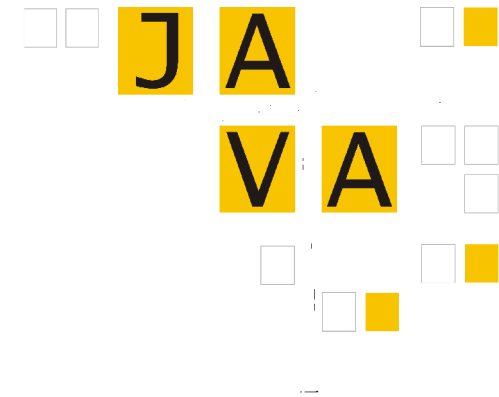
AcceptHeaderLocaleResolver

CookieLocaleResolver

SessionLocaleResolver

LocaleChangeInterceptor

este interceptor pode ser adicionado ao handler resolver, e irá detectar um parâmetro com o nome siteLanguage no request para alterar o locale do usuário chamando no localeResolver do contexto o método setLocale



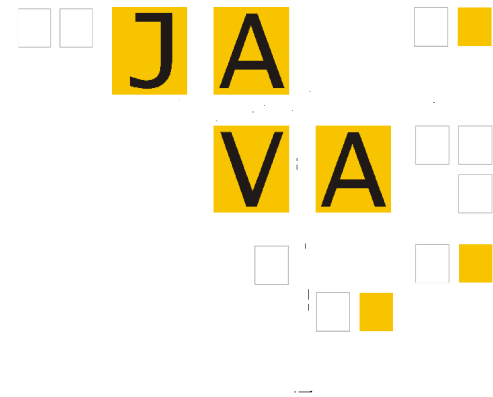
Tratando Exceções

SimpleMappingExceptionHandler

Habilita a mesma feature da API servlet, possibilita mapear um nome de classe de exceção que pode ser jogada de um controller para uma view.

ExceptionHandler

implementando esta interface em qualquer bean do contexto, o que se resume ao método: `resolveException(Exception, Handler)` pode-se programaticamente ficar sabendo qual exceção foi gerada e em qual controller do sistema, o que traz muito mais flexibilidade.

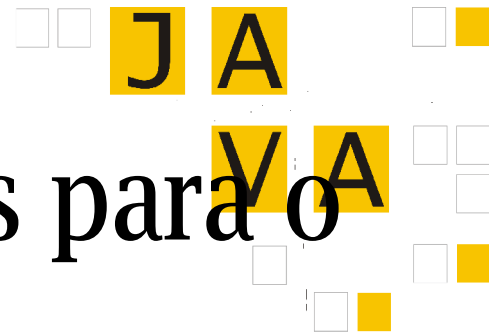


Enviando e-mails

A API javamail não é exatamente fácil de ser utilizada.

O spring possui uma API de abstração para o envio de mensagens/e-mails, estes podem ser enviados utilizando JavaMail ou então o MailSender desenvolvido pelo Jason Hunter, do pacote `com.oreilly.servlet`.

Qualquer uma destas implementações é totalmente compatível e utilizável dentro do ambiente IoC do Spring, podendo até ser mascarada através do framework de AOP, como veremos depois.



Configuração dos componentes para o e-mail.

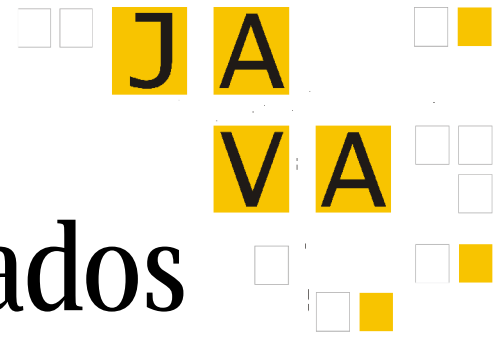
```
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
  <property name="host">
    <value>mail.mycompany.com</value>
  </property>
</bean>
<bean id="mailMessage" class="org.springframework.mail.SimpleMailMessage">
  <property name="from">
    <value>customerservice@mycompany.com</value>
  </property>
  <property name="subject">
    <value>Your order</value>
  </property>
</bean>
<bean id="orderManager" class="com.mycompany.businessapp.support.OrderManagerImpl">
  <property name="mailSender">
    <ref bean="mailSender"/>
  </property>
  <property name="message">
    <ref bean="mailMessage"/>
  </property>
</bean>
```



Como utilizar a configuração feita.

```
public interface OrderManager {  
  
    void placeOrder(Order order);  
  
}
```

```
public class OrderManagerImpl implements OrderManager {  
    private JavaMailSender mailSender;  
    private SimpleMailMessage message;  
    public void setMailSender(JavaMailSender mailSender) {  
        this.mailSender = mailSender;  
    }  
    public void setMessage(SimpleMailMessage message) {  
        this.message = message;  
    }  
    public void placeOrder(Order order) {  
        SimpleMailMessage msg = new SimpleMailMessage  
        (this.message);  
        msg.setTo(order.getCustomer().getEmailAddress());  
        msg.setText(  
            "Dear "  
            + order.getCustomer().getFirstName()  
            + order.getCustomer().getLastName()  
            + ", thank you for placing order. Your order  
            number is "  
            + order.getOrderNumber());  
        try{  
            mailSender.send(msg);  
        }  
        catch(MailException ex) {  
            System.err.println(ex.getMessage());  
        }  
    }  
}
```



Facilidades para acesso a dados

Tratando exceções

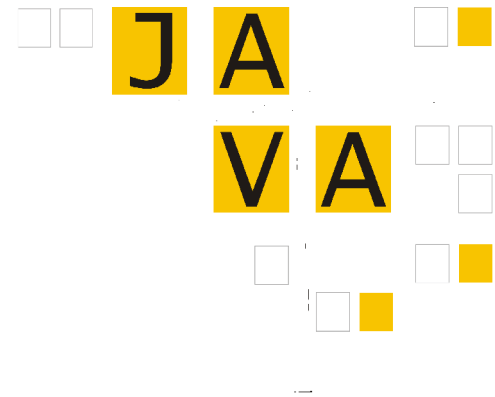
Transações declarativas

Transações programáticas

Utilizando JDBC

Utilizando Hibernate

Utilizando JDO



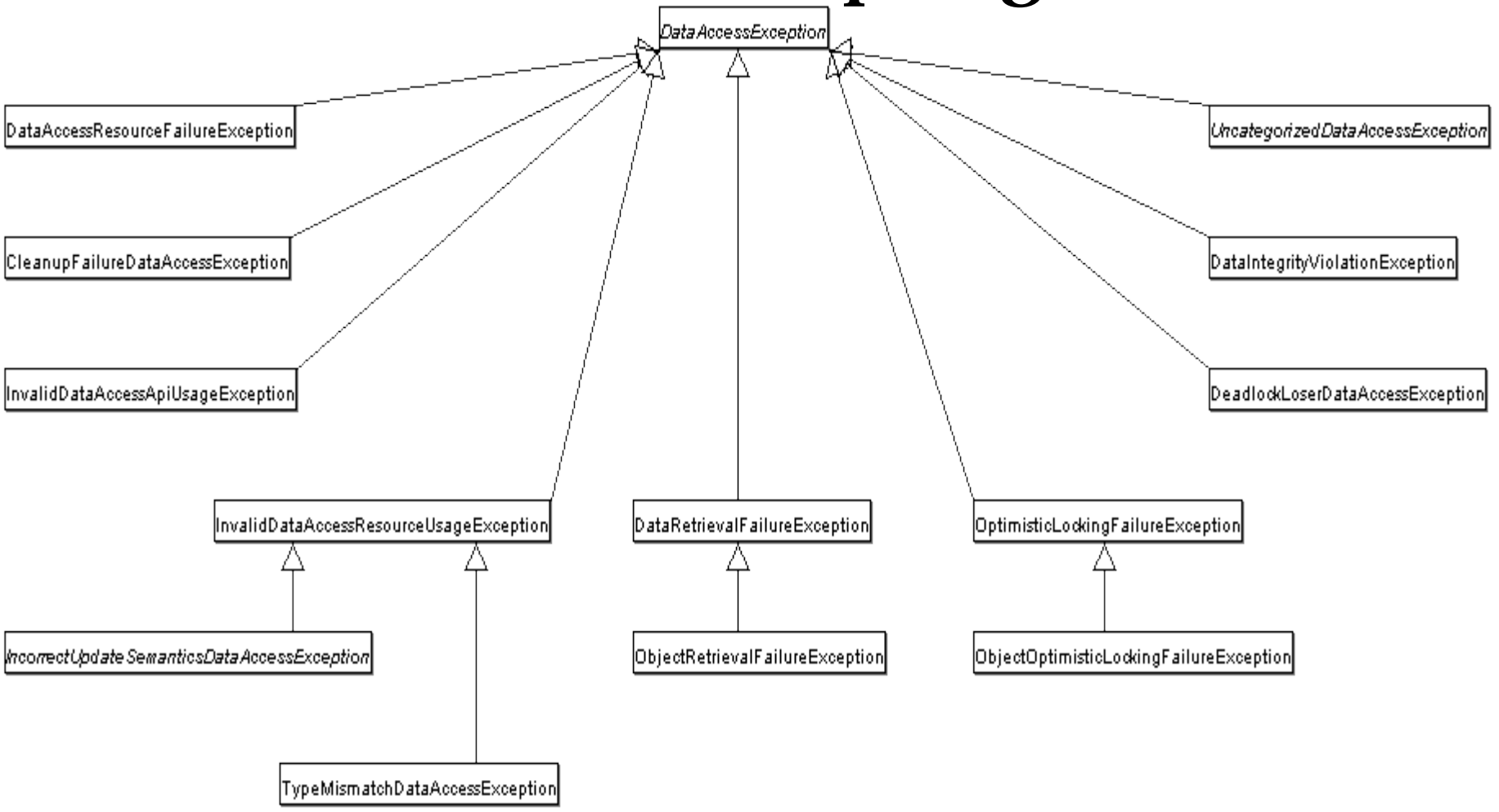
Tratando Exceções

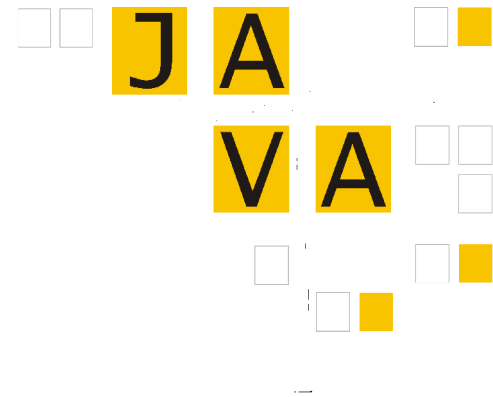
`java.sql.SQLException` normalmente não diz absolutamente nada de útil para a aplicação. é muito chato e normalmente não muito útil ter que tratar exceções declaradas que na maioria das vezes durante o acesso a dados da aplicação não podem ser recuperadas.

O spring prove uma hierarquia muito clara de exceções de acesso a dados, descendentes de `RuntimeException`.



Hierarquia de Exceções de Acesso a dados do Spring.





Transações declarativas

O spring suporta transações declarativas mesmo em ambientes sem suporte para JTA

são suportados diversos níveis de isolamento de transações

ISOLATION_READ_UNCOMMITTED

ISOLATION_READ_COMMITTED

ISOLATION_REPEATABLE_READ

ISOLATION_SERIALIZABLE

E diversos níveis de propagação de transações

PROPAGATION_NEVER

PROPAGATION_NOT_SUPPORTED

PROPAGATION_REQUIRES_NEW

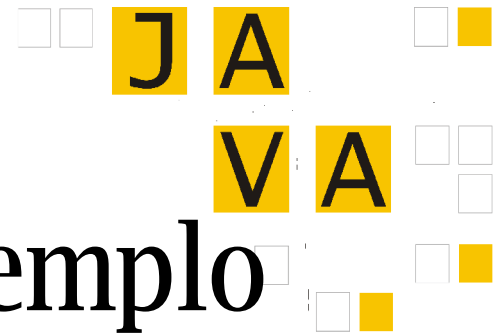
PROPAGATION_MANDATORY

PROPAGATION_SUPPORTS

PROPAGATION_REQUIRED

Como tudo no spring, estes atributos podem ser definidos de maneira programática ou declarativa.

<http://www.usiinformatica.com.br> <http://www.rsjug.org>

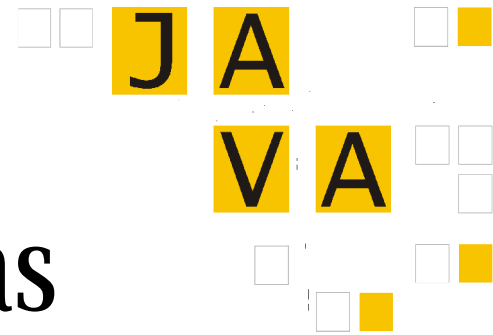


Transações Declarativas - exemplo

```
<bean id="myTransactionManager"  
class="org.springframework.orm.hibernate.HibernateTransactionManager">  
  <property name="sessionFactory">  
    <ref local="sessionFactory"/>  
  </property>  
</bean>
```

```
<bean id="myTransactionInterceptor"  
class="org.springframework.transaction.interceptor.TransactionInterceptor">  
  <property name="transactionManager">  
    <ref bean="myTransactionManager"/>  
  </property>  
  <property name="transactionAttributeSource">
```

```
<value>org.javanuke.core.model.dao.hibernate.HibernateAbstractDao.*=PROPAGATION_REQUI  
RED</value>  
  </property>  
</bean>
```



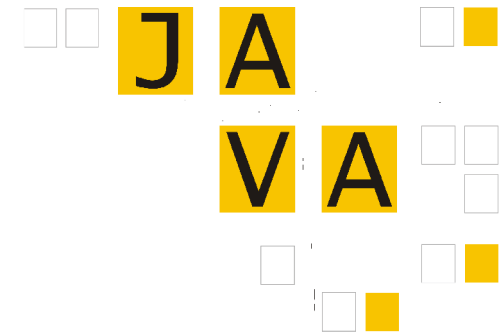
Transações Programáticas

```
TransactionTemplate tt = new TransactionTemplate();
Object result = tt.execute(new TransactionCallback() {
    public Object doInTransaction(TransactionStatus status) {
        updateOperation1();
        return resultOfUpdateOperation2();
    }
});
```

```
tt.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        updateOperation1();
        updateOperation2();
    }
});
```

Como todos os templates no spring, a idéia do TransactionTemplate é liberar o código da aplicação de obter e liberar recursos, e facilitar o tratamento de exceções.

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



Utilizando JDBC

JdbcTemplate

PreparedStatementCreator

CallableStatementCreator

RowCallbackHandler

SQLExceptionTranslator

MappingSqlQuery

SqlFunction

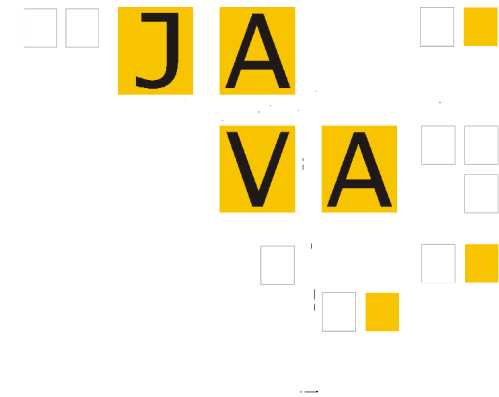
StoredProcedure

RdbmsOperation

Com a classe JdbcTemplate já temos todas as vantagens da abstração do JDBC do Spring, podendo executar consultas e qualquer tipo de SQL diretamente.

pode-se definir um mapeamento customizado para as exceções do banco de dados, ou então apenas estender com a definição de erros específicos de um banco de dados.

Pode-se também mapear facilmente o resultado de uma consulta para objetos Java, ou então, representar a própria consulta como um objeto Java.



Utilizando Hibernate

HibernateTemplate

HibernateTransaction
Manager

SessionFactoryUtils

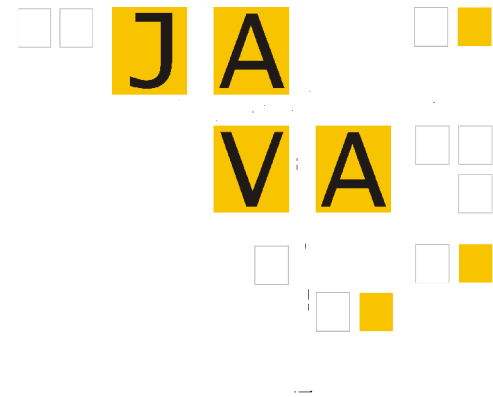
HibernateCallBack

Como no JDBC puro, o HibernateTemplate prove uma série de facilidades para a utilização do Hibernate a partir do spring framework.

Todas as operações de gerenciamento de dados de um HibernateTemplate são executads por uma implementação da interface HibernateCallBack.

o HibernateTransactionManager é utilizado para controlar as transações do hibernate a partir do AOP do spring framework.

SessionFactoryUtils, é utilizado para a utilização do Hibernate de maneira programatica.



Utilizando JDO

JdoTemplate

JdoCallback

JdoTransactionManager

LocalPersistenceManagerFactoryBean

Como no JDBC e no Hibernate o JdoTemplate traz as facilidades do spring para a utilização do JDO.

todas as operações de alteração de dados do JdoTemplate, são executadas dentro de uma implementação de JdoCallback.

JdoTransactionManager une o framework AOP do Spring a utilização do JDO.

LocalPersistenceManagerFactoryBean é a maneira preferida de se obter um PersistenceManagerFactory para a utilização da aplicação, ja que muitas implementações de JDO não suportam a utilização de um PMF através de JNDI.

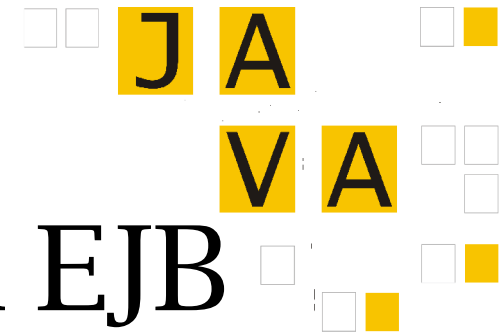
Acessando EJBs

Acessar EJBs a partir do container IoC do spring é exatamente a mesma coisa que acessar qualquer outro componente de negócios, ou seja, é como utilizar um POJO.

O spring prove suporte para a utilização de session beans locais ou remotos, bastando para isto, trocar a factory utilizada na configuração, de
`org.springframework.ejb.access.LocalStatelessSessionProxyFactoryBean` para
`org.springframework.ejb.access.RemoteStatelessSessionProxyFactoryBean`

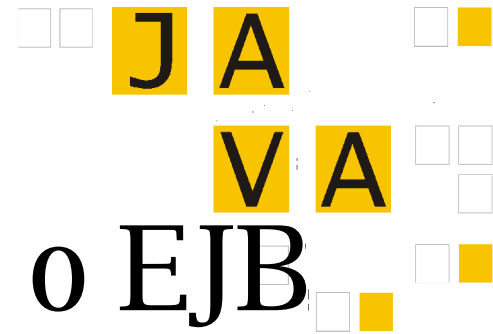
e também prove facilidades para ser utilizado a partir de um EJB, que será criado pelo EJB Container, e não pelo Spring.

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



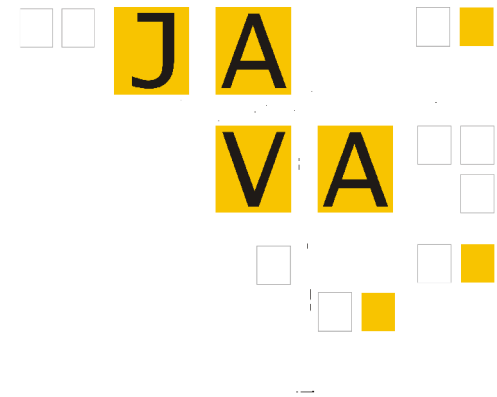
Configurando o acesso a um EJB

```
<bean id="mySessionBean"  
  class="org.springframework.ejb.access.LocalStatelessSessionProxyFactoryBean">  
  <property name="jndiName">  
    <value>mySessionBean</value>  
  </property>  
  <property name="businessInterface">  
    <value>br.com.usiinformatica.palestra.spring.ejb.MyBusinessInterface</value>  
  </property>  
</bean>  
<bean id="myController" class = "br.com.usiinformatica.palestra.spring.ejb.EjbController">  
  <property name="mySessionBean">  
    <ref bean="mySessionBean"/>  
  </property>  
</bean>
```



Implementando o Cliente para o EJB

```
public class EjbController {
private MyBusinessInterface mySessionBean;
/**
 * @return Returns the mySessionBean.
 */
public MyBusinessInterface getMySessionBean() {
return this.mySessionBean;
}
/**
 * @param mySessionBean The mySessionBean to set.
 */
public void setMySessionBean( MyBusinessInterface mySessionBean ) {
this.mySessionBean = mySessionBean;
}
public void metodoDeNegocio(){
mySessionBean.executarCompra("usuario",Collections.EMPTY_LIST);
}
}
```



Implementando o EJB

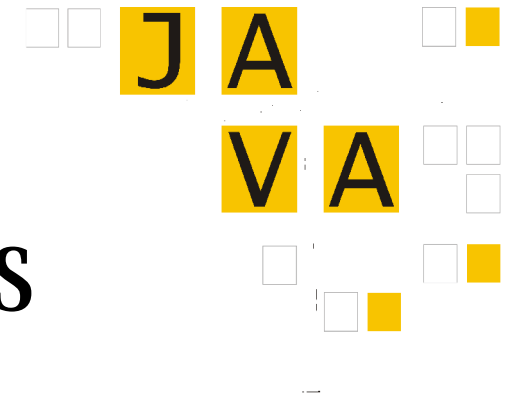
```
public interface MyBusinessInterface {
public void executarCompra(String userName, Collection items);
}
public class MySessionBean extends AbstractStatelessSessionBean implements MyBusinessInterface {

/* (non-Javadoc)
 * @see br.com.usiinformatica.palestra.spring.ejb.MyBusinessInterface#executarCompra(java.lang.String, java.util.Collection)
 */
public void executarCompra( String userName, Collection items ) {
    MailSender ms = ( MailSender ) getBeanFactory().getBean("mailSender");

    SimpleMailMessage smm = ( SimpleMailMessage ) getBeanFactory().getBean("mailMessage");

    smm.setText("Nova compra do usuário: " + userName + "\n Contendo: " + items.toString());

    ms.send(smm);
}
/* (non-Javadoc)
 * @see org.springframework.ejb.support.AbstractStatelessSessionBean#onEjbCreate()
 */
protected void onEjbCreate() throws CreateException {
}
}
```



Editores de propriedades

O spring se compromete a não re inventar a roda, então, por que inventar uma outra maneira de mapear valores complexos para propriedades?

O Spring, utiliza os PropertyEditors do próprio java, para ensinar o Spring a converter um tipo novo de dados, ou alterar a forma como ele faz isto, basta registrar com o PropertyEditorManager um novo property editor para uma classe qualquer.

Já são configurados por default, os property editors padrão do Java, e property editors para os seguintes tipos de objetos: Classes, Files/Arquivos, Properties (arquivos properties), Arrays de Strings, Locales e URLs.

O Spring utiliza os PropertyEditors, para converter os valores enviados para o framework MVC pelo request, e ate para converter os valores das propriedades setadas como String no arquivo XML de configuração

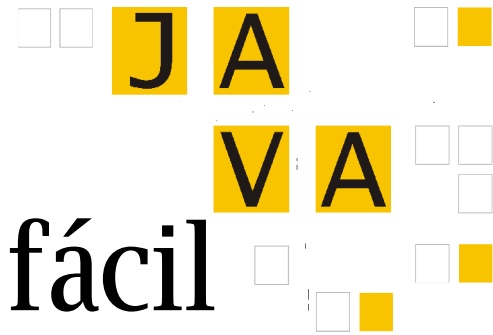


Criando um Novo editor de propriedades.

```
public class Cliente {
    private String nome;
    private String endereco;
    ...
}
public class ClientePropertyEditor extends PropertyEditorSupport {

    public String getAsText() {
        Cliente c = ( Cliente ) getValue();
        return c.getNome() + ":" + c.getEndereco();
    }

    public void setAsText( String arg0 ) throws IllegalArgumentException {
        Cliente c = new Cliente();
        if ( arg0 != null ) {
            String[] props = arg0.split( ":" );
            c.setEndereco( props[1] );
            c.setNome( props[0] );
        }
        else {
            c.setEndereco( null );
            c.setNome( null );
        }
    }
}
```



Utilizando AOP de maneira fácil

Conceitos AOP

Aspect - um conceito distribuido pela aplicação, por exemplo, gerenciamento de transações

Advice - uma ação executada em um JointPoint

JointPoint - a especificação de onde uma ação deve ser executada

Pointcut - um conjunto de JointPoints especificando quando um advice deve ser executado

Introduction - a adição de metodos ou campos a objetos

TargetObject - um objeto que contem um advice

AOP Proxy - um objeto criado pelo framework AOP

Weaving - o ato de juntar aspectos para criar um TargetObject

Capacidades AOP do Spring Framework

Utilização do AOP



Capacidades AOP do Spring Framework

O spring traz um framework AOP muito poderoso e totalmente integrado com a BeanFactory utilizada por toda a aplicação, mas não suporta todos os recursos AOP suportados por outras implementações de AOP.

O spring não suporta interceptação de campos, apenas de métodos.

O suporte a AOP do spring implementa as interfaces definidas pela AOP Alliance.

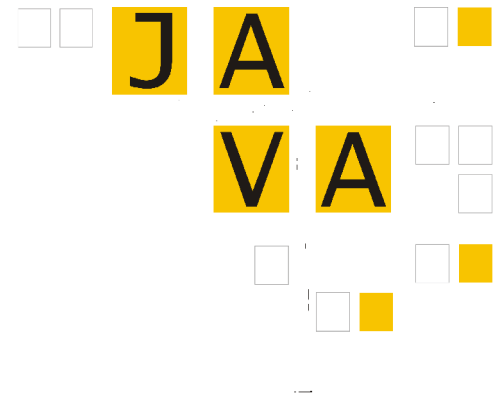
Por default são suportados os seguintes tipos de Advices:

MethodInterceptor

ThrowsAdvice

BeforeAdvice

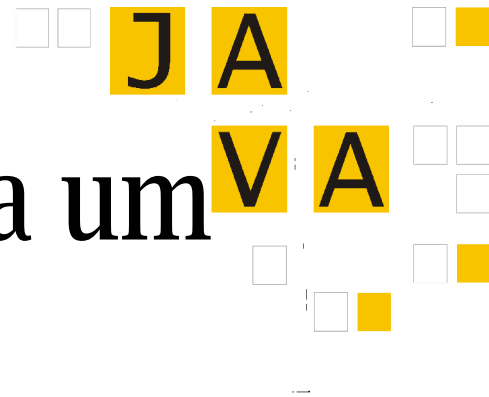
AfterReturningAdvice



Utilização do AOP

Um exemplo clássico de utilização transparente do suporte a AOP do spring framework, é o gerenciamento de transações declarativas nativo do Spring, que é implementado como um Advice AOP.

podemos ver também, diversas implementações de logging, controle de exceções, adição de interfaces em objetos (por exemplo, uma interface IsModified)

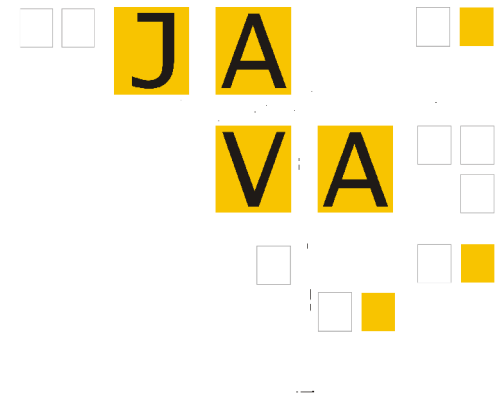


Adicionando uma interface a um objeto via AOP

```
public class IsModifiedInterceptor extends
    DelegatingIntroductionInterceptor
    implements IsModified {
    private boolean modified;
    public boolean isModified() {
        return modified;
    }
    public void resetModified(){
        modified=false;
    }
    public Object invoke(MethodInvocation
        invocation) throws Throwable {
        if (invocation.getMethod().getName().
            indexOf("set") == 0)
            modified=true;
        return super.invoke(invocation);
    }
}
```

```
<bean id="isModifiedInterceptor"
    class="IsModifiedInterceptor"/>
```

```
<bean id="modifiedBeanNameProxyCreator"
    class="org.springframework.aop.framework.a
    utoproxy.BeanNameAutoProxyCreator">
    <property
        name="beanNames"><value>*</value></pro
        perty>
    <property name="interceptorNames">
        <list>
            <value>isModifiedInterceptor</value>
        </list>
    </property>
</bean>
```



O que esta por vir

Spring 1.1

JMS Support

JMX Support

declarative Rules-Based validator

AOP pointcut expression Language, JSR-175 preview

Spring 1.2

OGNL Support

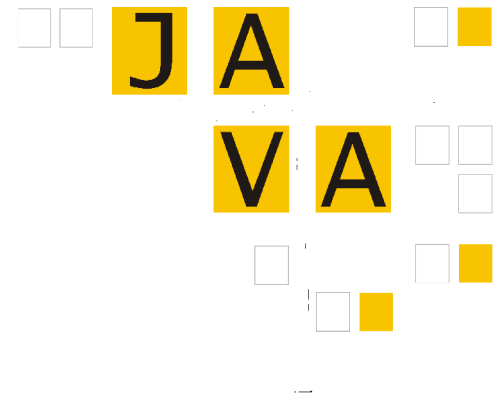
JCA Support

enhanced RMI support

Spring 1.3?

JSF

Portlets



Projetos Relacionados

Rich Client Platform (Sandbox)

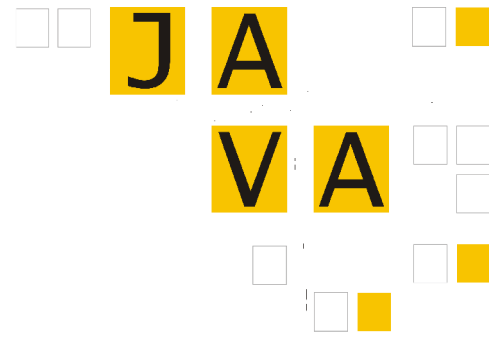
Spring RCP

Validation (Sandbox)

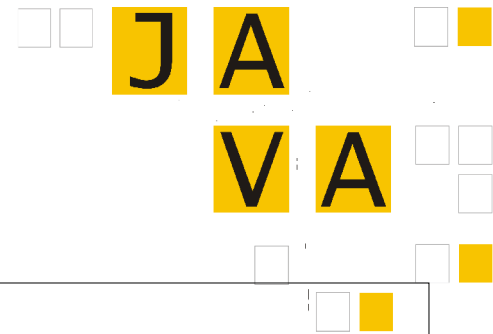
Commons-Validator
Attribute Based

Security

Acegi Security System for Spring
<http://acegisecurity.sourceforge.net>



?



Site do Spring framework

<http://www.springframework.org>

Javadoc do Sprign Framework

<http://www.springframework.org/docs/api/>

Spring + Hibernate

<http://hibernate.bluemars.net/110.html>

Spring + WebWork 1

<http://wiki.opensymphony.com/space/Spring+Framework+Integration>

The Server Side

<http://www.theserverside.com>

Spring

<http://www.usiinformatica.com.br> <http://www.rsjug.org>



Rodrigo Urubatan Ferreira Jardim
Consultor/Desenvolvedor J2EE
Immediate Consultoria

Sun Certified Web Component Developer for J2EE 1.4
Sun Certified Programmer for the Java 2 Platform 1.4

rodrigo@usiinformatica.com.br
<http://www.usiinformatica.com.br>

<http://www.immediate.com.br>