

Spring MVC Framework.

Rodrigo Urubatan Ferreira Jardim

Arquiteto J2EE/Consultor

Immediate Consultoria

rodrigo@usiinformatica.com.br

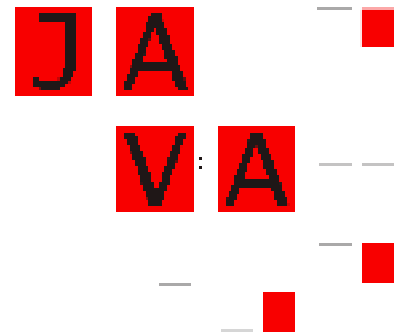
<http://www.usiinformatica.com.br>

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

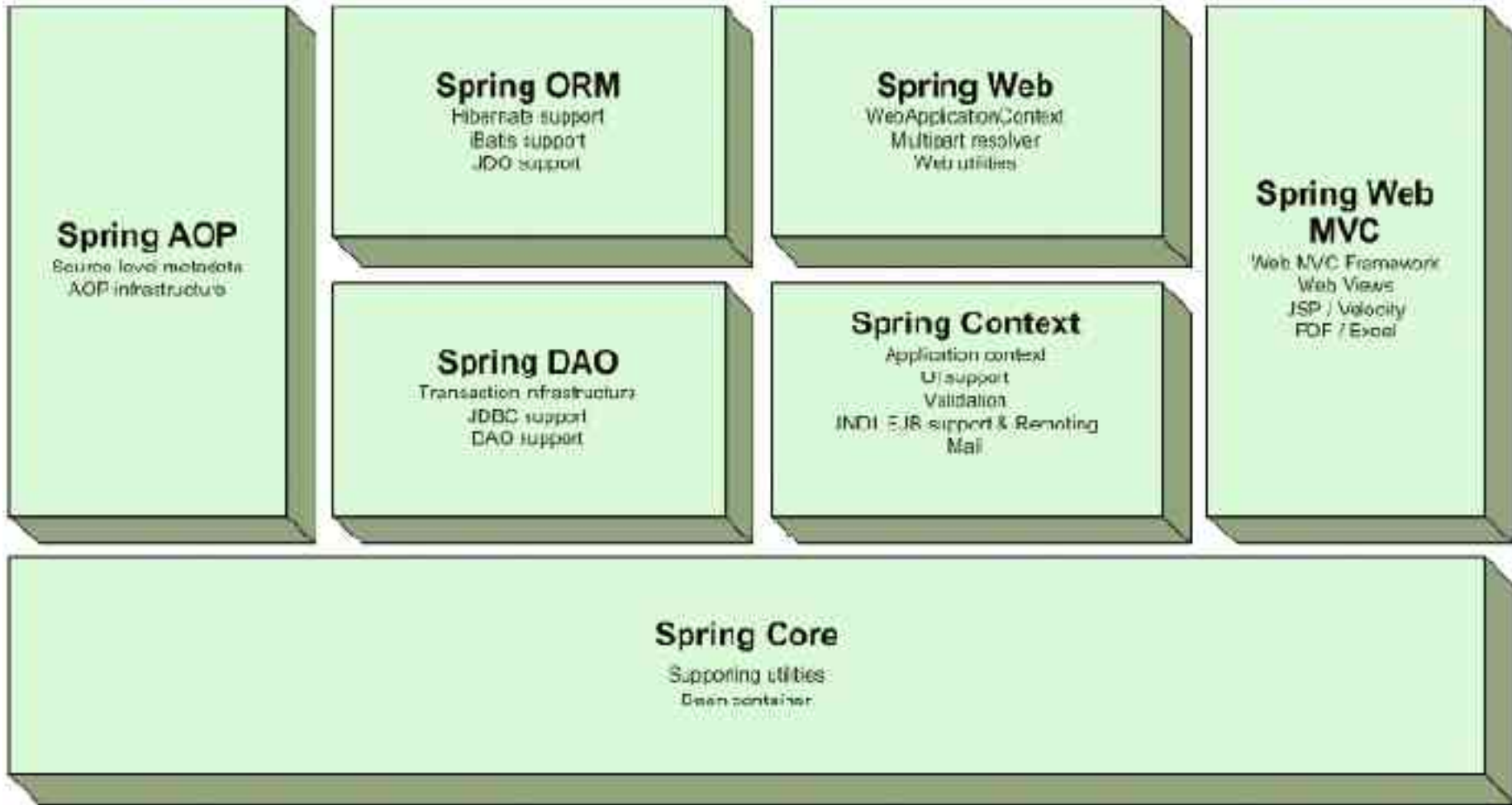


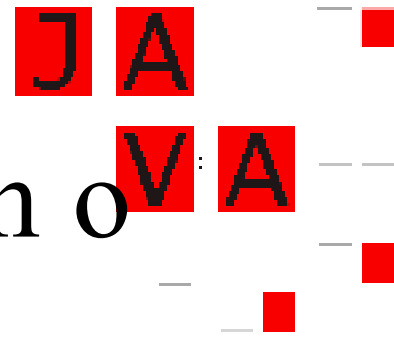
<http://www.immediate.com.br>

<http://www.rsjug.org>



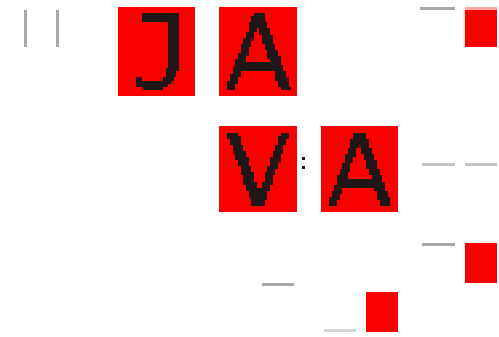
Spring Framework





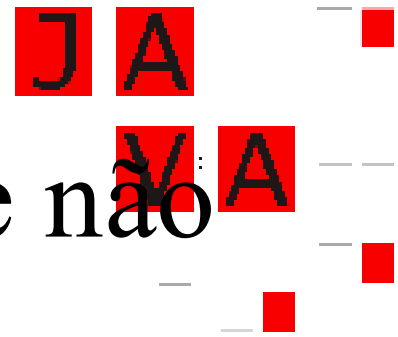
Desenvolvendo aplicações com o Spring Framework

- Histórico do Framework
- Desenvolvendo para Interfaces não para classes concretas
- Inversão de Controle e Injeção de Dependências
- Facilidades AOP do framework
- Desenvolvimento WEB com Spring MVC Framework



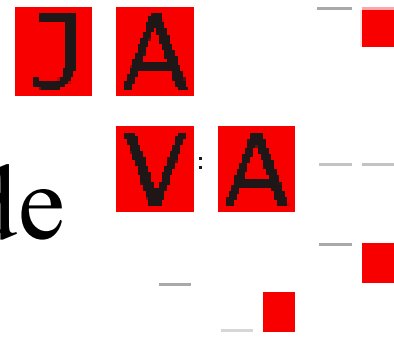
Origem e Filosofia

- 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



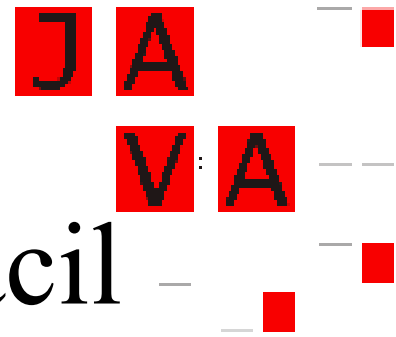
Desenvolvendo para Interfaces e não para classes concretas

- 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



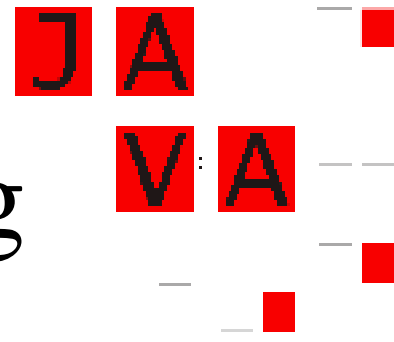
Inversão de Controle e Injeção de dependências

- O que é Inversão de Controle
- O que é Injeção de dependências
- Formas de Injeção de dependência
- BeanFactory o Container de injeção de dependências do Spring Framework.



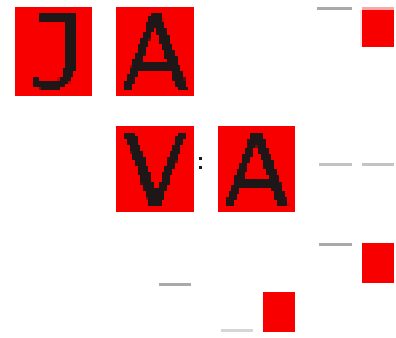
Utilizando AOP de maneira fácil

- Conceitos AOP
 - Aspect - um conceito distribuído 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 métodos ou campos a objetos
 - TargetObject - um objeto que contém 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



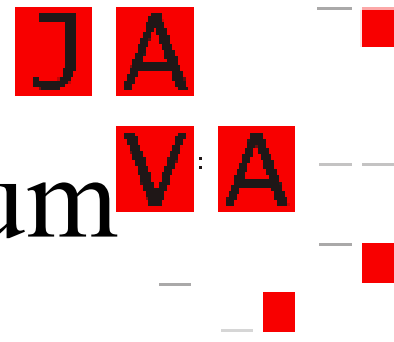
Capacidades AOP do Spring Framework

- 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)



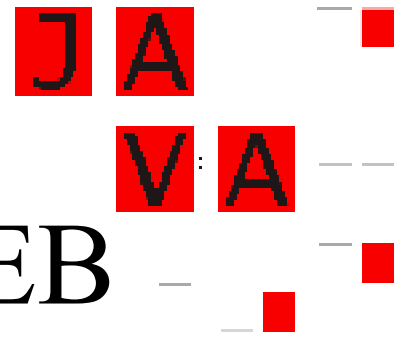
Adicionando uma interface a um objeto via AOP

```
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.au
        toproxy.BeanNameAutoProxyCreator">
    <property
        name="beanNames"><value>*</value></prop
            erty>
    <property name="interceptorNames">
        <list>
            <value>isModifiedInterceptor</value>
        </list>
    </property>
</bean>
```

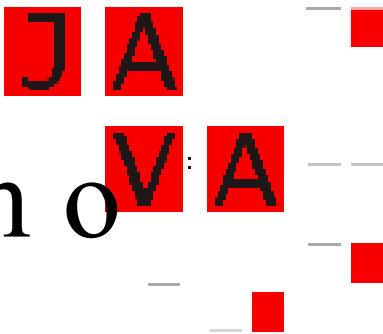


Desenvolvendo aplicações WEB

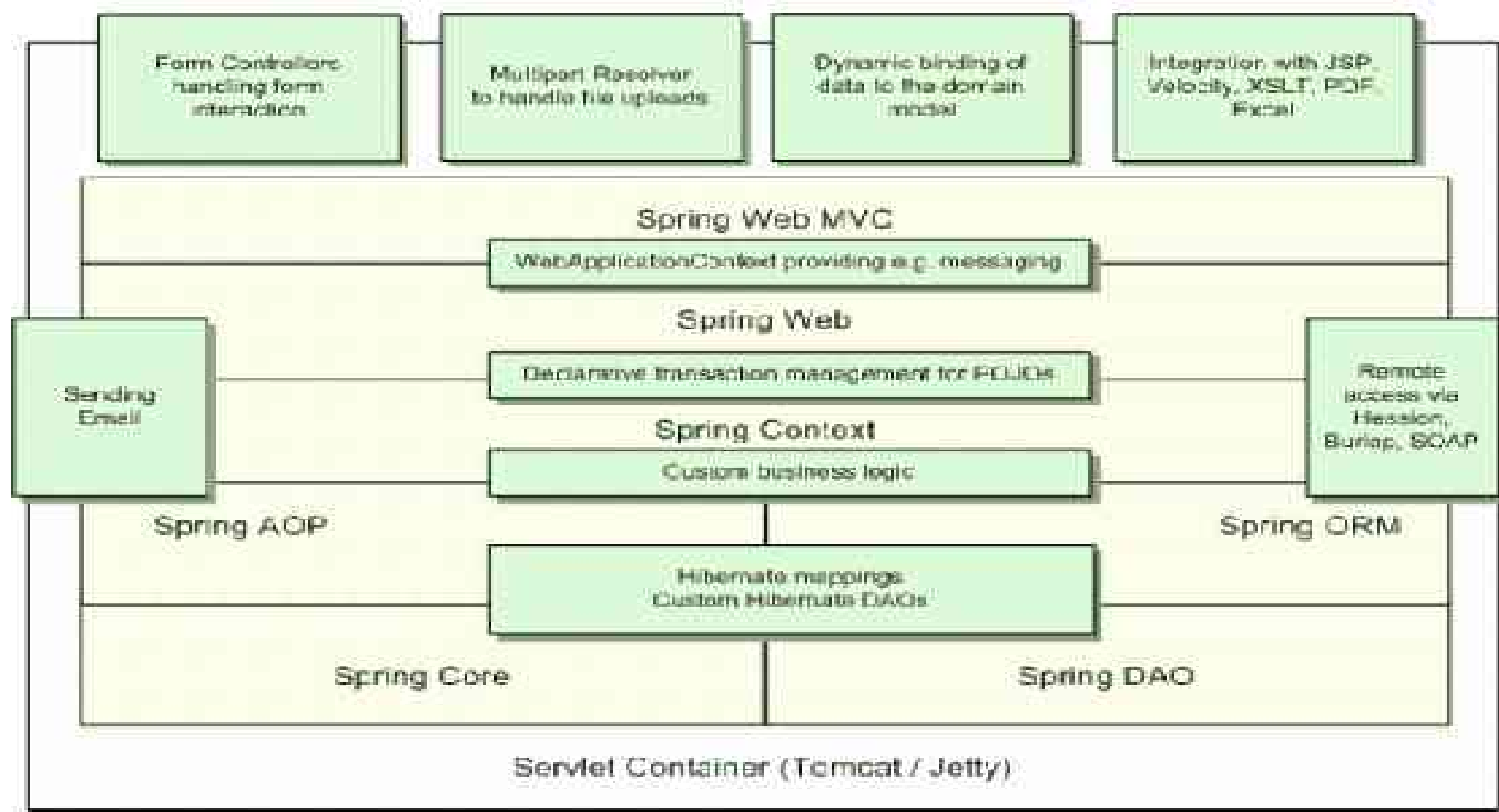
- Configuração do framework
- Controllers
- Validação de formulários
- Mapeando requests
- Resolvendo Views
- Integrando tecnologias de visualização
- Localização e utilização de temas
- Tratamento de exceções
- Exposição de objetos de negocio como serviços remotos via XML-RPC, Hessian e Burlap
- Chamada de objetos remotos utilizando XML-RPC, Hessian, Burlap e EJBs

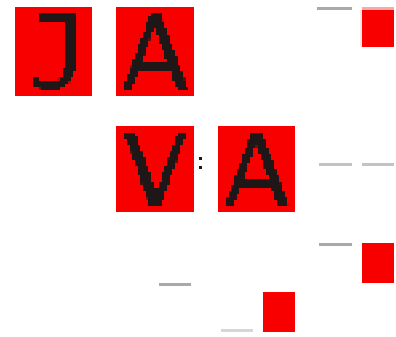
<http://www.immediate.com.br>

<http://www.rsjug.org>



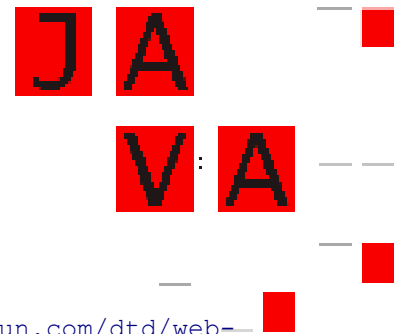
Aplicação WEB Completa com o Spring Framework





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.

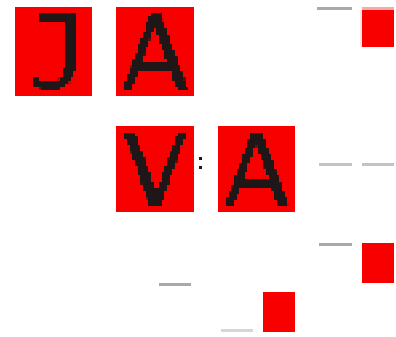


web.xml

```
<?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-web.xml,/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>
<servlet-name>caucho</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet </servlet-class>
<load-on-startup>4</load-on-startup>
</servlet>
<servlet>
<servlet-name>axis</servlet-name>
<servlet-class>org.apache.axis.transport.http.AxisServlet </servlet-class>
<load-on-startup>5</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>exemplo</servlet-name>
<url-pattern>*.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>caucho</servlet-name>
<url-pattern>/caucho/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>axis</servlet-name>
<url-pattern>/axis/*</url-pattern>
</servlet-mapping>
```

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

<http://www.rsjug.org>



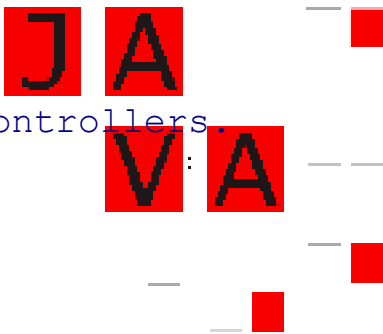
Controllers

- 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, ...

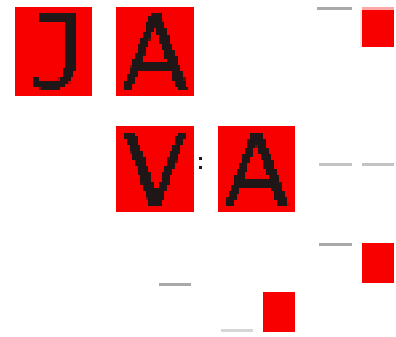


RSJUG
.org

```
<bean id="delegateMAComponent"
class="justjava.example.web.controllers.
ExampleDelegateMultiAction"
autowire="byName"/>
<bean id="delegateMA"
class="org.springframework.web.servlet.m
vc.multiaction.MultiActionController"
>
<property name="delegate">
<ref bean="delegateMAComponent"/>
</property>
</bean>
<bean id="exampleForm"
class="justjava.example.web.controlle
rs.ExampleForm"
autowire="byName">
<property name="formView">
<value>userForm</value>
</property>
<property name="successView">
<value>user-created</value>
</property>
<property name="commandName">
<value>User</value>
</property>
<property name="commandClass">
<value>justjava.example.web.vo.User</val
ue>
</property>
</bean>
```

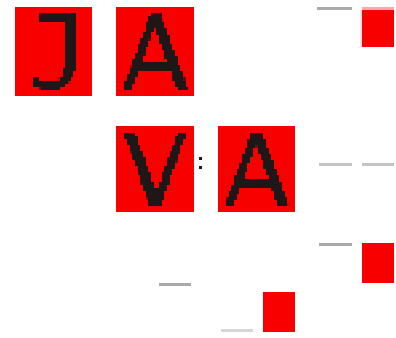


```
<bean id="exampleWizard"
class="justjava.example.web.controllers.
ExampleWizard"
autowire="byName">
<property name="commandName">
<value>User</value>
</property>
<property name="commandClass">
<value>justjava.example.web.vo.User</val
ue>
</property>
<property name="pages">
<list>
<value>userWiz1</value>
<value>userWiz2</value>
<value>userWiz3</value>
<value>userWizFinish</value>
</list>
</property>
</bean>
<bean id="exampleMA"
class="justjava.example.web.controllers.
ExampleMultiAction"
autowire="byName"/>
```



Validando Formulários

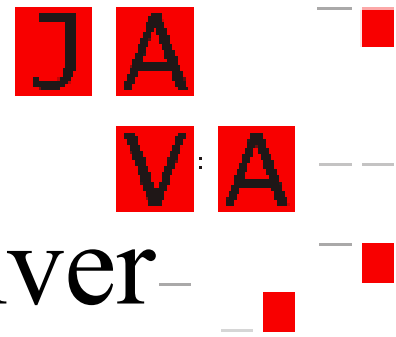
- Utilizando o framework MVC do Spring, existem duas formas default de validação de formulários:
 - utilizando o commons-validator do projeto jakarta, para isto basta informar no XML do spring a utilização deste validator para um controller específico.
 - Utilizar uma classe implementando a interface 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 ja existe, o nome de usuário solicitado ja esta sendo utilizado, ..., este é o metodo preferido de validação de formulários no Spring
- Ao utilizar outro framework MVC a validação deste devera 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

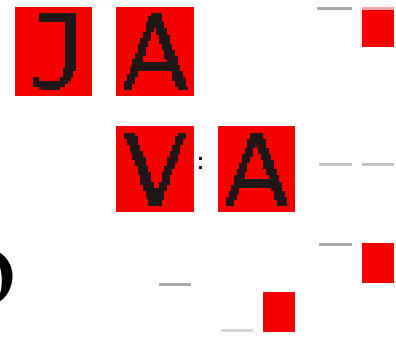
- BeanNameUrlHandleMapping
 - Direciona os requests para um bean com o mesmo nome do excedente da URL.
- SimpleUrlHandlerMapping
 - Mapeia requests baseado em WildCards parecidos com os utilizados para diretórios no ANT
- HandlerInterceptors
 - Podem ser adicionados interceptors a qualquer mapeamento de requests, por exemplo para controlar o horario de acesso as páginas ou alguma coisa relacionada a segurança

```
<bean id="urlMapping"  
class="org.springframework.web.se  
rvlet.handler.SimpleUrlHandler  
Mapping">  
<property name="mappings">  
<props>  
<prop  
key="/delegate/*.html">delegat  
eMA</prop>  
<prop  
key="/form.html">exampleForm</  
prop>  
<prop  
key="/wiz.html">exampleWizard<  
/prop>  
<prop  
key="/ma/*.html">exampleMA</pr  
op>  
</props>  
</property>  
</bean>
```



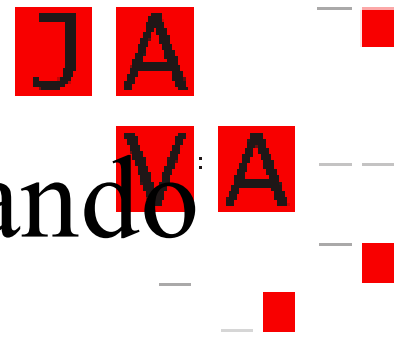
Resolvendo Views - viewResolver

- 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



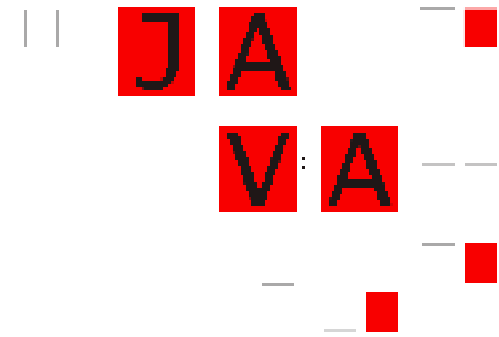
Tecnologias de Visualização

- Configurando um dos viewResolvers disponiveis, pode-se utilizar o spring com as seguintes tecnologias de View:
 - JSP/JSTL
 - Tiles
 - Velocity
 - XSLT
 - PDF
 - Excel



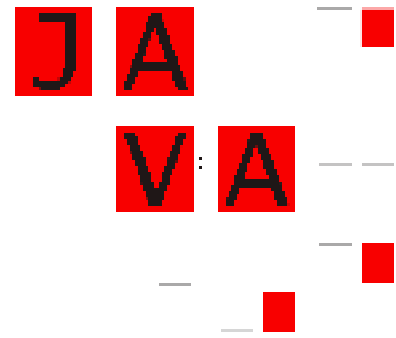
Localizando a aplicação e utilizando temas

- A maior parte do Spring Framework é localizavel.
- existem os seguintes localeResolvers ja implementados:
 - AcceptHeaderLocaleResolver
 - CookieLocaleResolver
 - SessionLocaleResolver
- LocaleChangeInterceptor
 - este interceptor pode ser adicionado ao handler resolver, e ira detectar um parâmetro com o nome siteLanguage no request para alterar o locale do usuário chamando no localeResolver do contexto o método setLocale



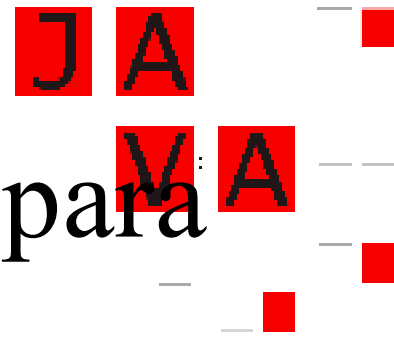
Tratando Exceções

- 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.
- HandlerExceptionHandler
 - 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.



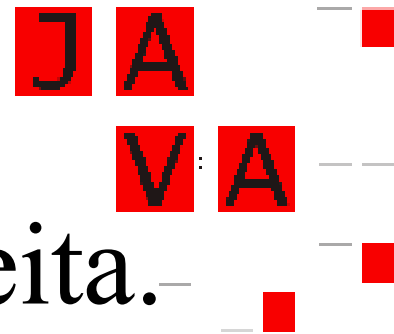
Enviando e-mails

- 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.

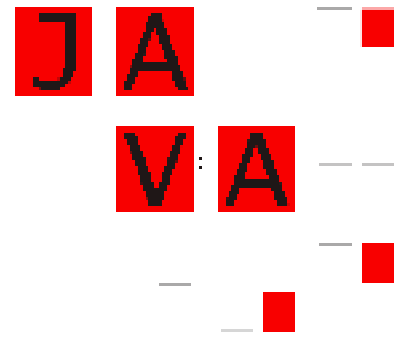
```
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
  <property name="host">
    <value>mail.mycompany.com</value>
  </property>
</bean>
<bean id="mailMessage" class="org.springframework.mail.SimpleMailMessage">
  <property name="from">
    <value>customerservice@mycompany.com</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.

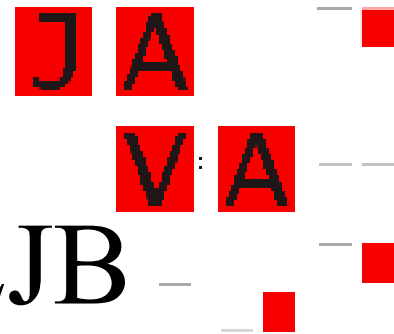
```
public interface OrderManager {  
  
    void placeOrder(Order order);  
  
}
```

```
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());  
        }  
    }  
}
```



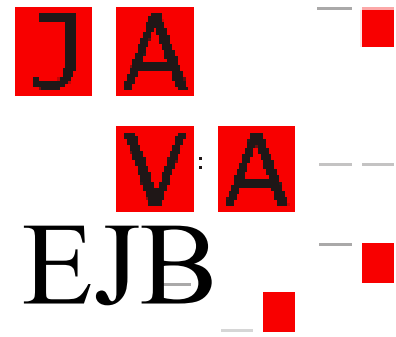
Acessando EJBs

- 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.



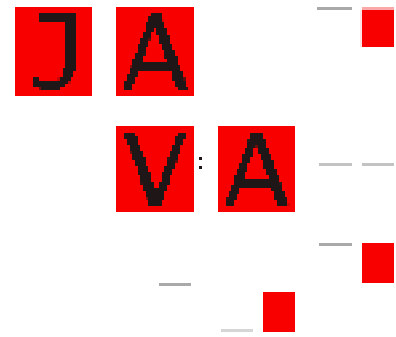
Configurando o acesso a um EJB

```
<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

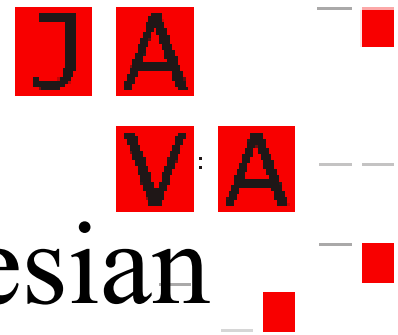
```
public interface MyBusinessInterface {
public void executarCompra(String userName, Collection items);
}
public class MySessionBean extends AbstractStatelessSessionBean implements MyBusinessInterface {

/* (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());

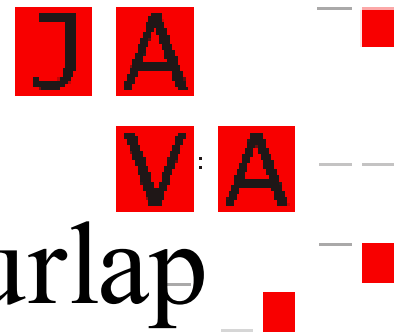
    ms.send(smm);
}
/* (non-Javadoc)
 * @see org.springframework.ejb.support.AbstractStatelessSessionBean#onEjbCreate()
 */
protected void onEjbCreate() throws CreateException {
}
}
```



Expondo objetos remotos via Hessian

- Hessian é um protocolo binário para comunicação via HTTP desenvolvido pela BEA.
- A publicação de serviços neste protocolo via o container IoC do Spring Framework é extremamente fácil, basta declarar um bean do tipo `HessianServiceExporter` e informar a este qual o bean deve ser exposto como serviço e a interface que este deve utilizar.

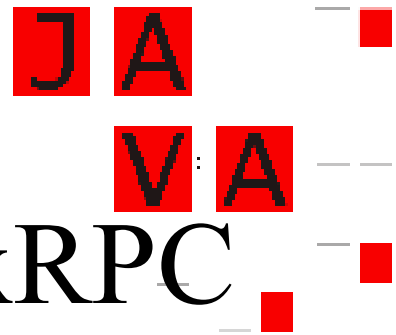
```
<bean name="/User-hessian"  
      class="org.springframework  
            k.remoting.caucho.Hessian  
            ServiceExporter">  
  <property  
    name="service"><ref  
      bean="userBO"/></property  
  >  
  <property  
    name="serviceInterface">  
  <value>justjava.example.web  
    .business.UserBO</value>  
  </property>  
</bean>
```



Expondo objetos remotos via Burlap

- Burlap é o equivalente em XML ao Hessian, também desenvolvido pela BEA.
- A publicação de serviços neste protocolo via o container IoC do Spring Framework é extremamente fácil, basta declarar um bean do tipo BurlapServiceExporter e informar a este qual o bean deve ser exposto como serviço e a interface que este deve utilizar.

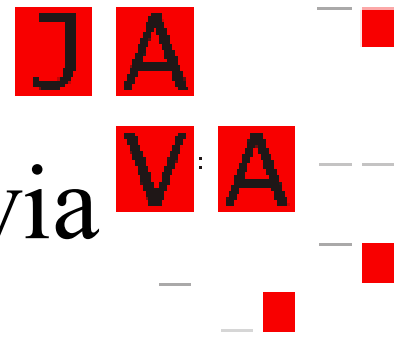
```
<bean name="/Pedido-burlap"
      class="org.springframework
      k.remoting.caucho.BurlapS
      erviceExporter">
  <property
    name="service"><ref
      bean="pedidoBO"/></propert
      y>
  <property
    name="serviceInterface">
  <value>justjava.example.web
    .business.PedidoBO</value
    >
  </property>
</bean>
```



Expondo objetos remotos via JaxRPC

- Jax-RPC é a implementação de WebServices em padrão Java.
- O SpringFramework prove uma maneira fácil de expor serviços neste protocolo, e compatível com qualquer implementação deste.
- Neste exemplo, será utilizado o Axis.
- Neste caso basta declarar normalmente o servlet do Axis e criar um Bean que estenda a classe ServletEndpointSupport para ser exposto como serviço, e logo após expolo como serviço na implementação de WebServices escolhida.
- No caso do axis, isto se faz utilizando o arquivo server-config.wsdd.

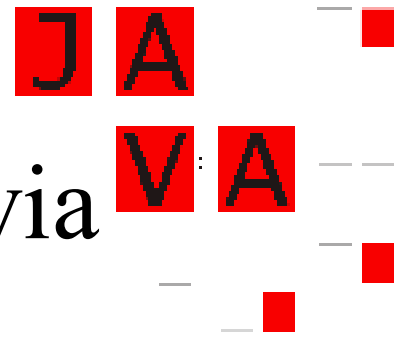
```
<service name="PedidoService"
  provider="java:RPC">
  <parameter name="allowedMethods"
    value="*" />
  <parameter name="className"
    value="justjava.example.web.xmlrpc.Pe
    didoService" />
  <beanMapping qname="justjava:Pedido"
    xmlns:jpetstore="urn:JustJava"
    languageSpecificType="java:justjava.e
    xample.web.vo.Pedido" />
  <beanMapping qname="justjava:Item"
    xmlns:jpetstore="urn:JustJava"
    languageSpecificType="java:justjava.e
    xample.web.vo.Item" />
  <beanMapping qname="justjava:Endereco"
    xmlns:jpetstore="urn:JustJava"
    languageSpecificType="java:justjava.e
    xample.web.vo.Endereco" />
  <beanMapping qname="justjava:User"
    xmlns:jpetstore="urn:JustJava"
    languageSpecificType="java:justjava.e
    xample.web.vo.User" />
</service>
```



Acessando objetos remotos via Hessian

- A chamada de objetos remotos pelo protocolo Hessian é tão fácil quanto a exposição deste serviço.
- Para esta, basta conhecer a interface pública do serviço e a URL, e declarar um Bean como no exemplo.

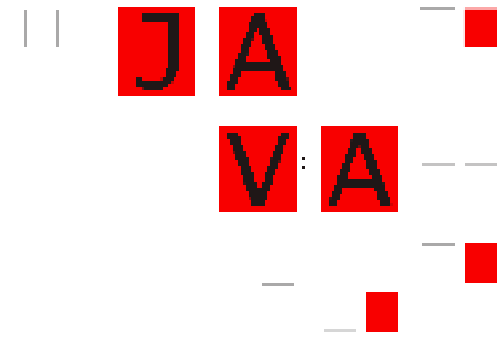
```
<bean id="userBO"  
class="org.springframework.  
remoting.caucho.HessianPr  
oxyFactoryBean">  
  
<property  
name="serviceUrl">  
  
<value>http://localhost:808  
0/spring/caucho/User-  
hessian</value>  
  
</property>  
  
<property  
name="ServiceInterface">  
  
<value>justjava.example.web  
.business.UserBO</value>  
  
</property>  
</bean>
```



Acessando objetos remotos via Burlap

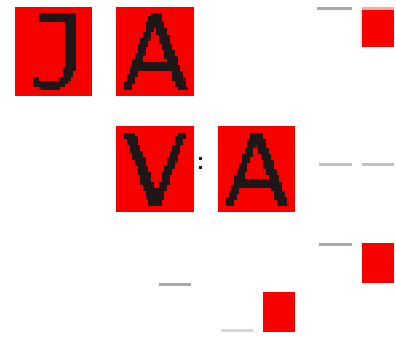
- A chamada de objetos remotos pelo protocolo Burlap é tão fácil quanto a exposição deste serviço.
- Para esta, basta conhecer a interface pública do serviço e a URL, e declarar um Bean como no exemplo.

```
<bean id="pedidoBO"  
class="org.springframework.  
remoting.caucho.BurlapPro  
xyFactoryBean">  
  
<property  
name="serviceUrl">  
  
<value>http://localhost:808  
0/spring/caucho/Pedido-  
burlap</value>  
  
</property>  
  
<property  
name="ServiceInterface">  
  
<value>justjava.example.web  
.business.PedidoBO</value  
>  
  
</property>  
  
</bean>
```



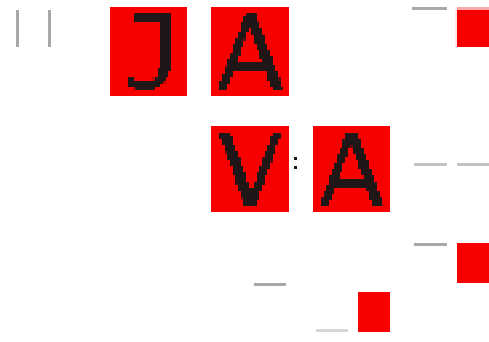
O que esta por vir

- 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



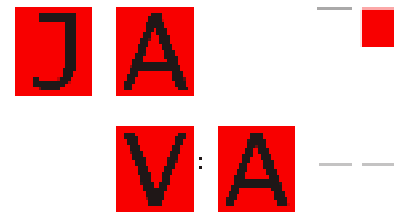
Projetos Relacionados

- Rich Client Platform (Sandbox)
 - Spring RCP
- Validation (Sandbox)
 - Commons-Validator
 - Attribute Based
- Security
 - Acegi Security System for Spring
<http://acegisecurity.sourceforge.net>



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

<http://www.rsjug.org>



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>





J

A

V

A

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>

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

<http://www.rsjug.org>