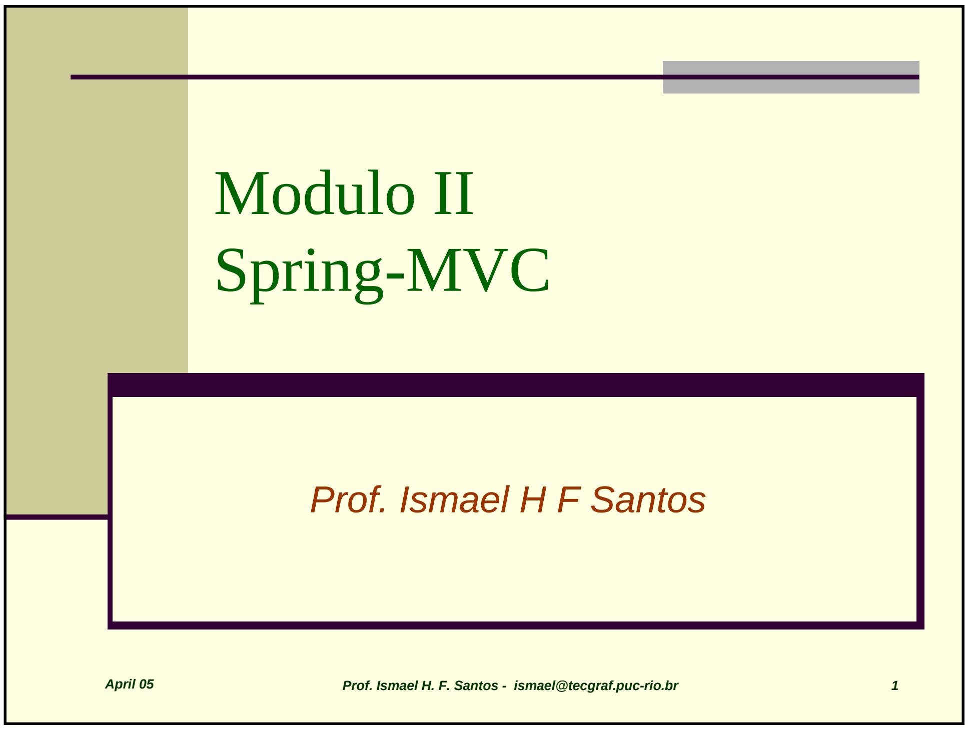
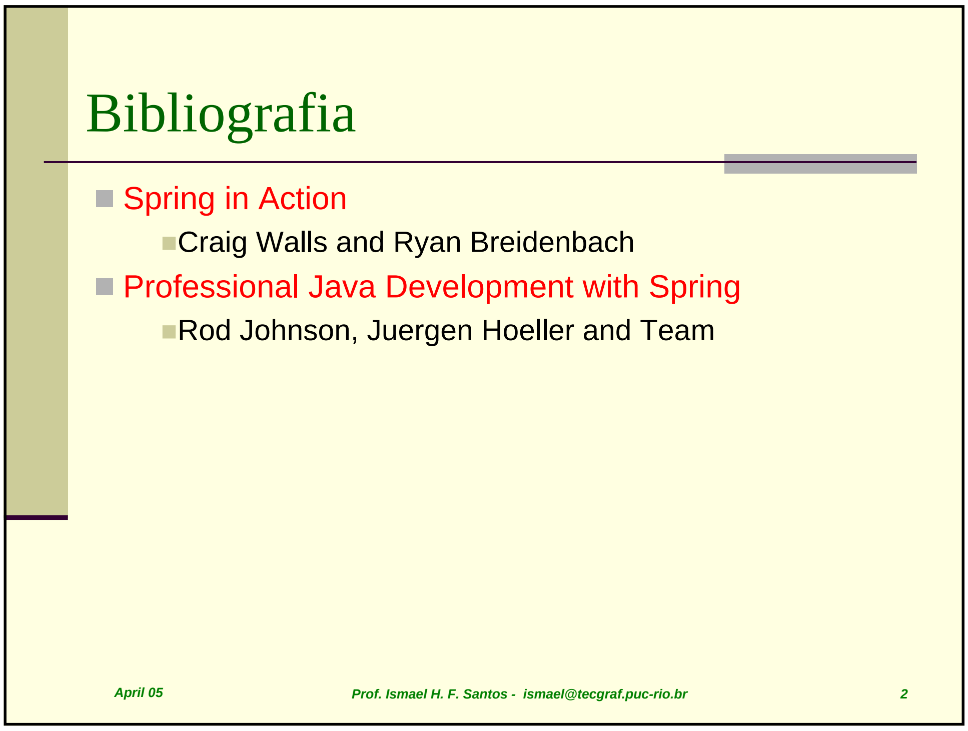


Modulo II Spring-MVC

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

Ementa

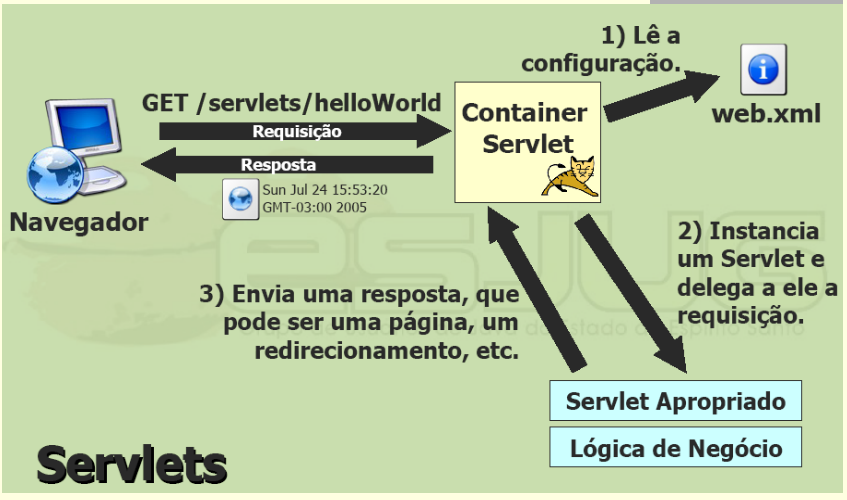
- Spring in the Web Tier
- Spring in the Middle Tier
- Spring-MVC

WebApp

Histórico Programação Web

- 1995: lançamento oficial, no início eram as Applets;
- 1996: Java Servlets;
- 1997: Swing/JFC;
- 1999: JavaServer Pages (JSP);
- 1999: plataforma J2EE;
- 2004: JavaServer Faces;
- 2005: Java completa 10 anos – de ponta a ponta;
- 2006: Java EE 5.

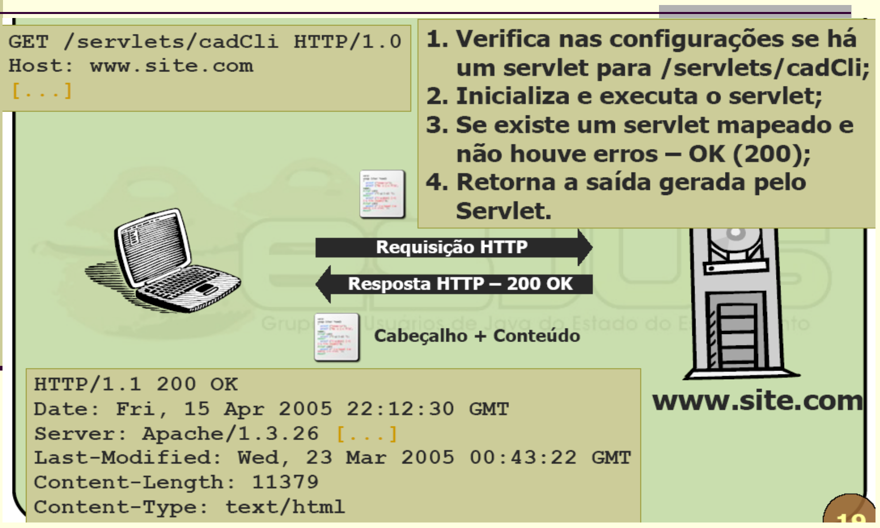
Histórico Programação Web


1) Lê a configuração.
web.xml
GET /servlets/helloWorld
Requisição
Container Servlet
Resposta
Sun Jul 24 15:53:20 GMT-03:00 2005
Navegador
2) Instancia um Servlet e delega a ele a requisição.
3) Envia uma resposta, que pode ser uma página, um redirecionamento, etc.
Servlet Apropriado
Lógica de Negócio
Servlets

Container Web – Java

- Container = gerenciador de objetos com ciclo de vida específico;
- Tem parte das funcionalidades de um Servidor de Aplicações J2EE;
  - Ex.: Tomcat, Jetty, Resin, WebLogic, Oracle AS, WebSphere, JBoss, etc.
- JSR 53 = Servlet 2.3 e JSP 1.2;
- JSR 152 = JSP 2.0;
- JSR 154 = Servlet 2.4;
- JSR 245 = JSP 2.1;
- JSR 315 = Servlet 3.0;
  - Os containers implementam as especificações.

Servlet Container

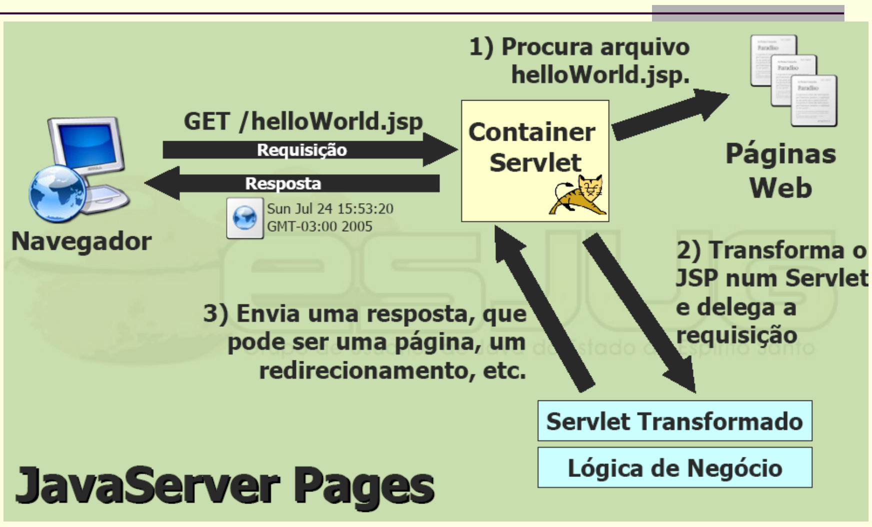


```
GET /servlets/cadCli HTTP/1.0
Host: www.site.com
[...]
```

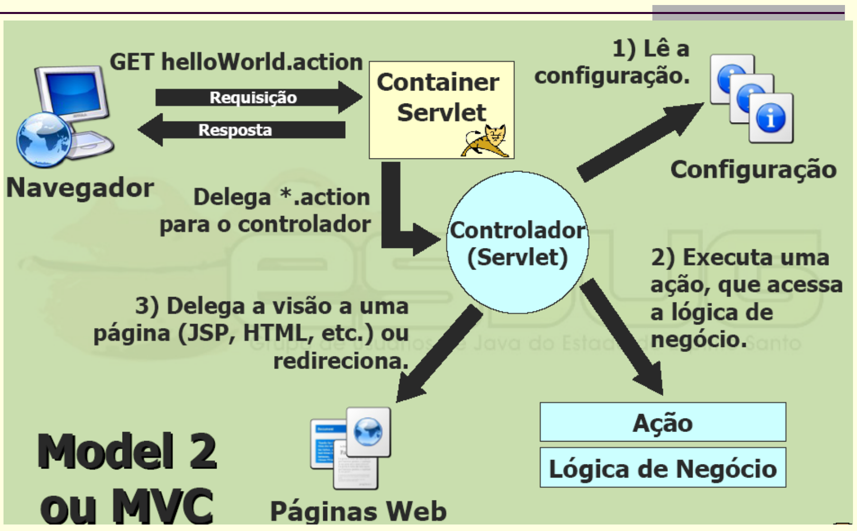
1. Verifica nas configurações se há um servlet para /servlets/cadCli;
2. Inicializa e executa o servlet;
3. Se existe um servlet mapeado e não houve erros – OK (200);
4. Retorna a saída gerada pelo Servlet.

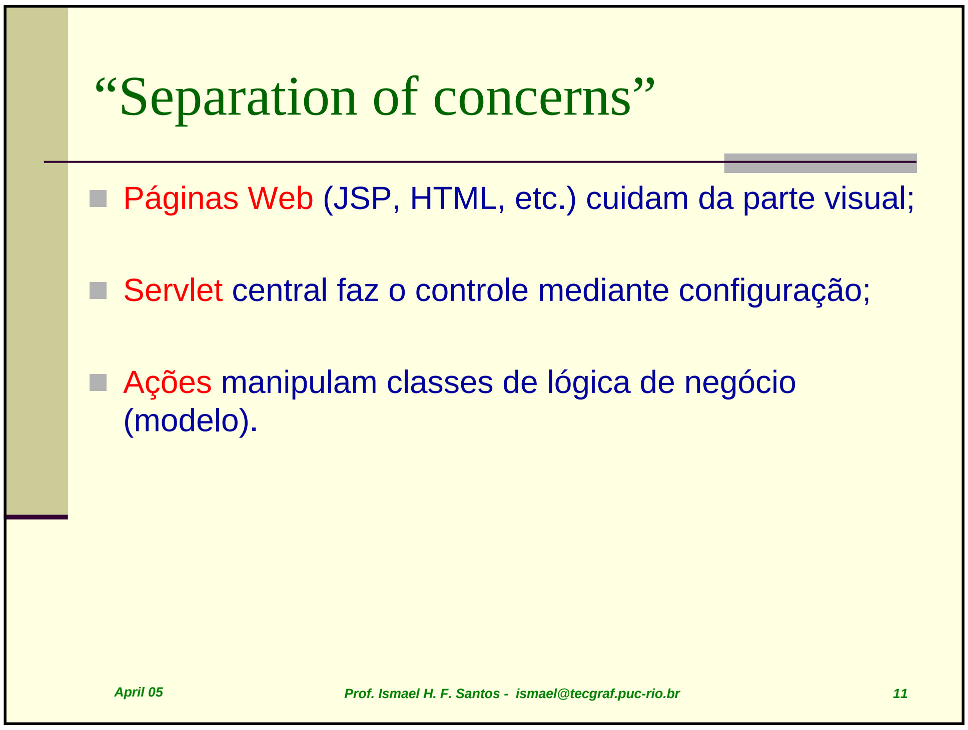
```
HTTP/1.1 200 OK
Date: Fri, 15 Apr 2005 22:12:30 GMT
Server: Apache/1.3.26 [...]
Last-Modified: Wed, 23 Mar 2005 00:43:22 GMT
Content-Length: 11379
Content-Type: text/html
```

Java Server Pages


1) Procura arquivo helloWorld.jsp.
GET /helloWorld.jsp
Requisição
Resposta
Container Servlet
Páginas Web
Navegador
Sun Jul 24 15:53:20 GMT-03:00 2005
2) Transforma o JSP num Servlet e delega a requisição
3) Envia uma resposta, que pode ser uma página, um redirecionamento, etc.
Servlet Transformado
Lógica de Negócio
JavaServer Pages

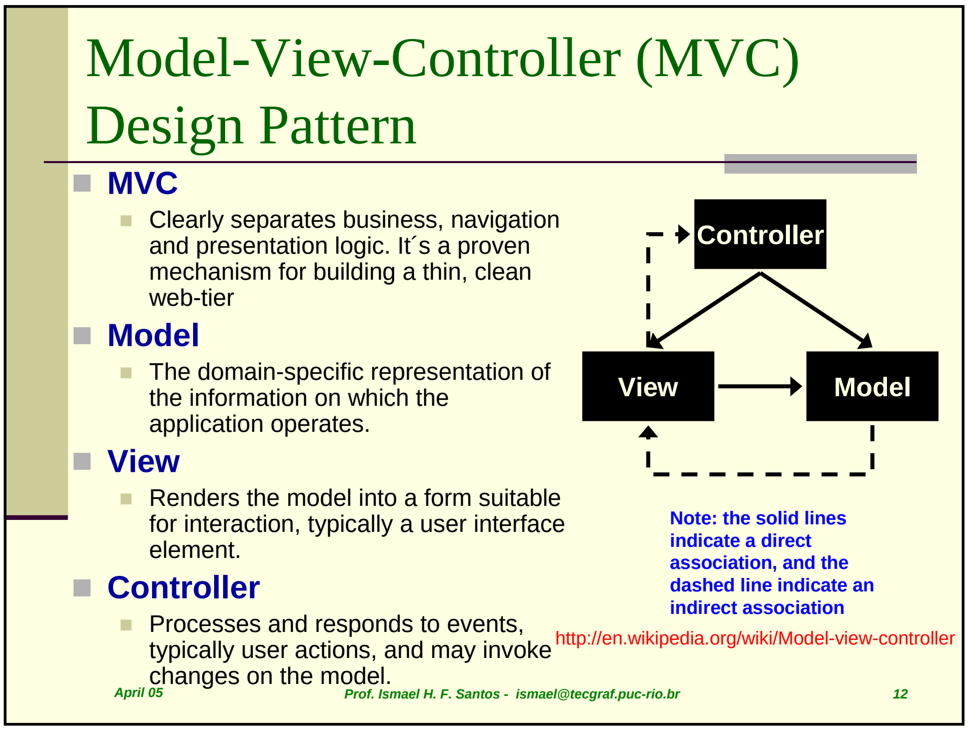
Servlet Controller





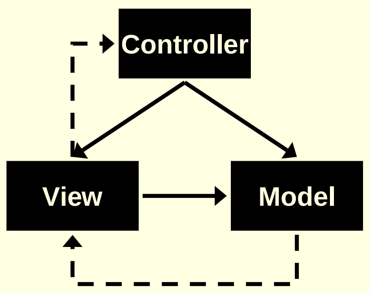
“Separation of concerns”

- Páginas Web (JSP, HTML, etc.) cuidam da parte visual;
- Servlet central faz o controle mediante configuração;
- Ações manipulam classes de lógica de negócio (modelo).



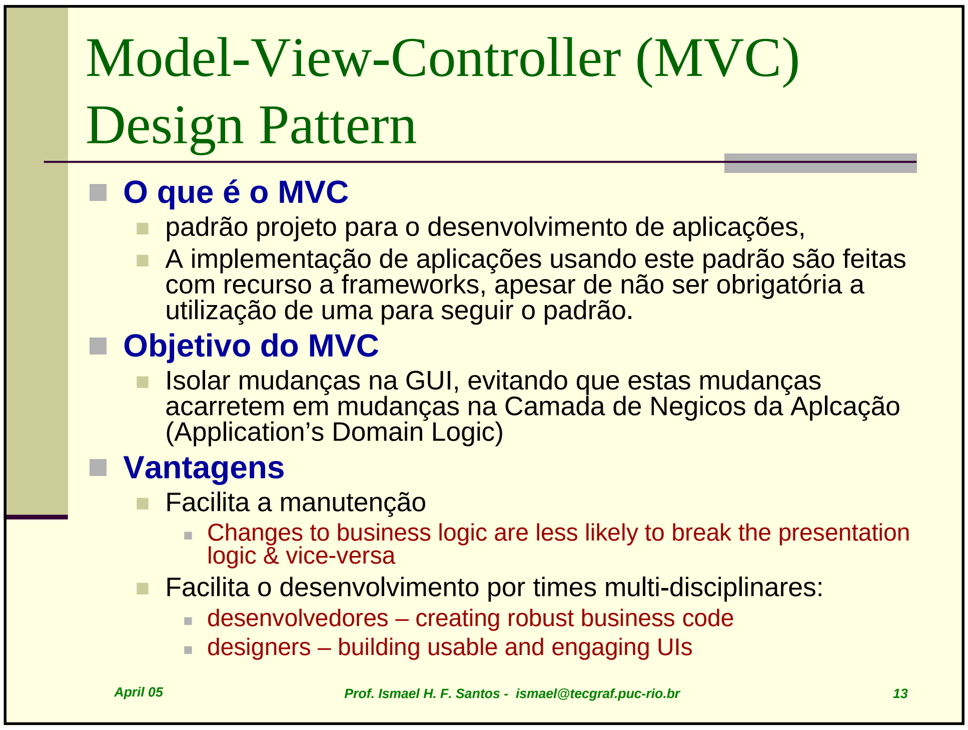
Model-View-Controller (MVC) Design Pattern

- **MVC**
  - Clearly separates business, navigation and presentation logic. It´s a proven mechanism for building a thin, clean web-tier
- **Model**
  - The domain-specific representation of the information on which the application operates.
- **View**
  - Renders the model into a form suitable for interaction, typically a user interface element.
- **Controller**
  - Processes and responds to events, typically user actions, and may invoke changes on the model.



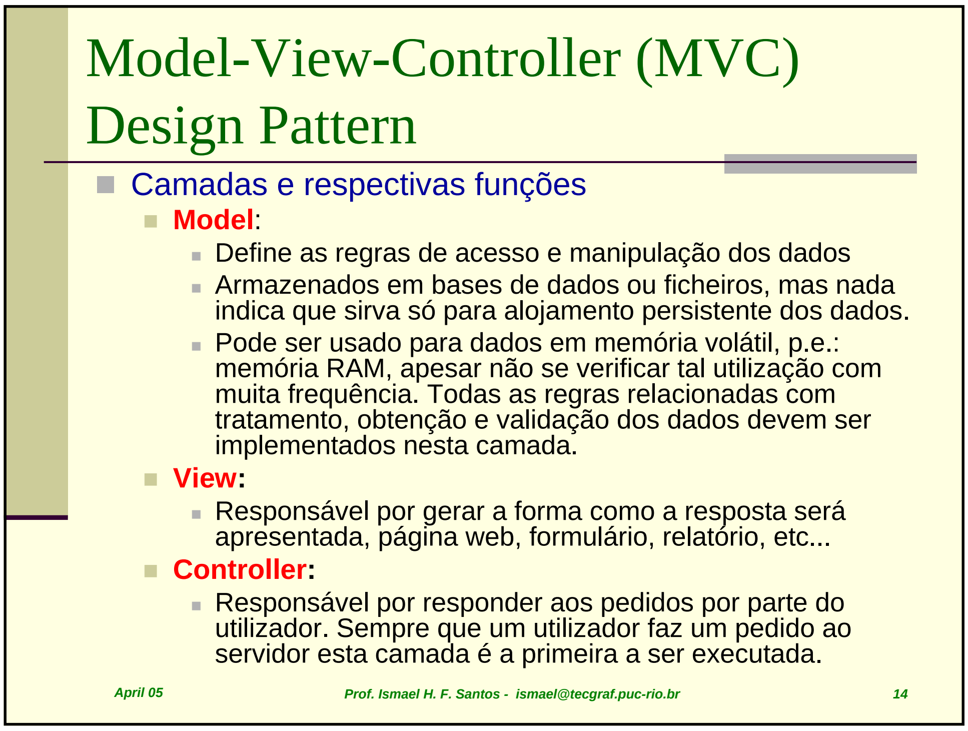
Note: the solid lines indicate a direct association, and the dashed line indicate an indirect association

http://en.wikipedia.org/wiki/Model-view-controller



Model-View-Controller (MVC) Design Pattern

- **O que é o MVC**
  - padrão projeto para o desenvolvimento de aplicações,
  - A implementação de aplicações usando este padrão são feitas com recurso a frameworks, apesar de não ser obrigatória a utilização de uma para seguir o padrão.
- **Objetivo do MVC**
  - Isolar mudanças na GUI, evitando que estas mudanças acarretem em mudanças na Camada de Negicos da Aplcação (Application's Domain Logic)
- **Vantagens**
  - Facilita a manutenção
    - Changes to business logic are less likely to break the presentation logic & vice-versa
  - Facilita o desenvolvimento por times multi-disciplinares:
    - desenvolvedores – creating robust business code
    - designers – building usable and engaging UIs



Model-View-Controller (MVC) Design Pattern

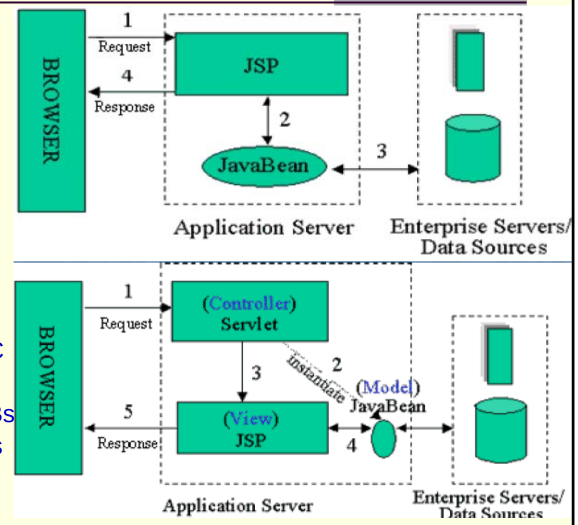
- Camadas e respectivas funções
  - **Model**:
    - Define as regras de acesso e manipulação dos dados
    - Armazenados em bases de dados ou ficheiros, mas nada indica que sirva só para alojamento persistente dos dados.
    - Pode ser usado para dados em memória volátil, p.e.: memória RAM, apesar não se verificar tal utilização com muita frequência. Todas as regras relacionadas com tratamento, obtenção e validação dos dados devem ser implementados nesta camada.
  - **View**:
    - Responsável por gerar a forma como a resposta será apresentada, página web, formulário, relatório, etc...
  - **Controller**:
    - Responsável por responder aos pedidos por parte do utilizador. Sempre que um utilizador faz um pedido ao servidor esta camada é a primeira a ser executada.

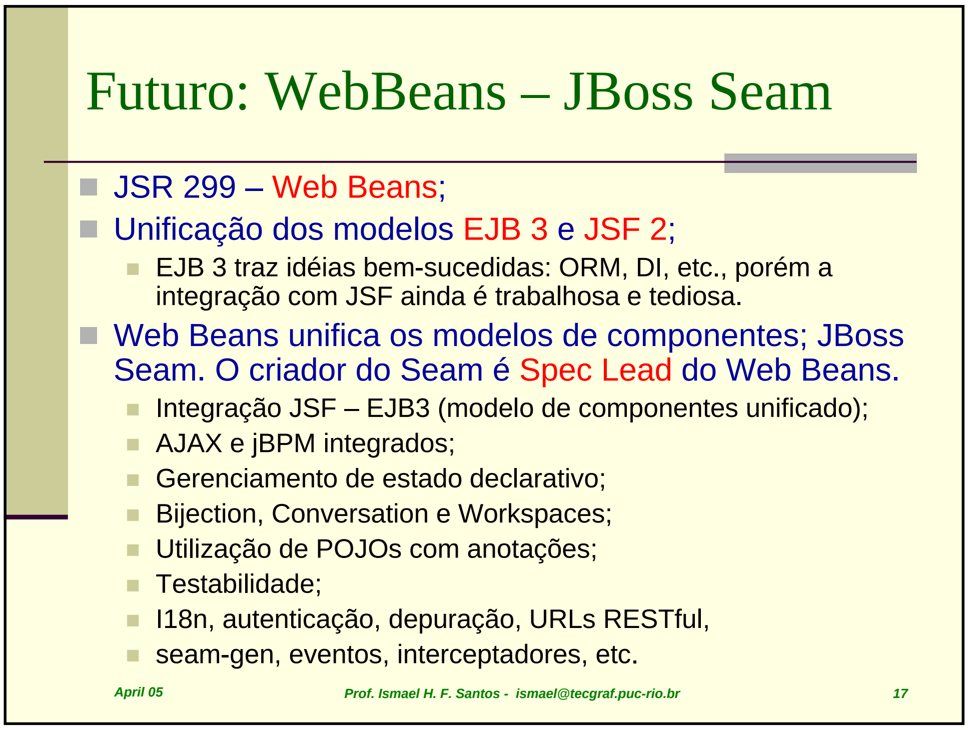
Front Controller (Servlet Controller)

- Dispatcher Servlet - “Front Controller” implementation
  - A single **Front Controller** servlet that dispatches requests to individual Controllers
  - Request routing is completely controlled by the Front Controller
- A lógica do MVC é altamente generalizável;
- Podemos listar mais de 50 frameworks diferentes: Cocoon, Action Framework, Maverick, MyFaces, …., SpringMVC, Struts, Tapestry, WebWork, PHP, RubyOn Raetc

Especificações do J2EE - Arquiteturas de aplicação Web

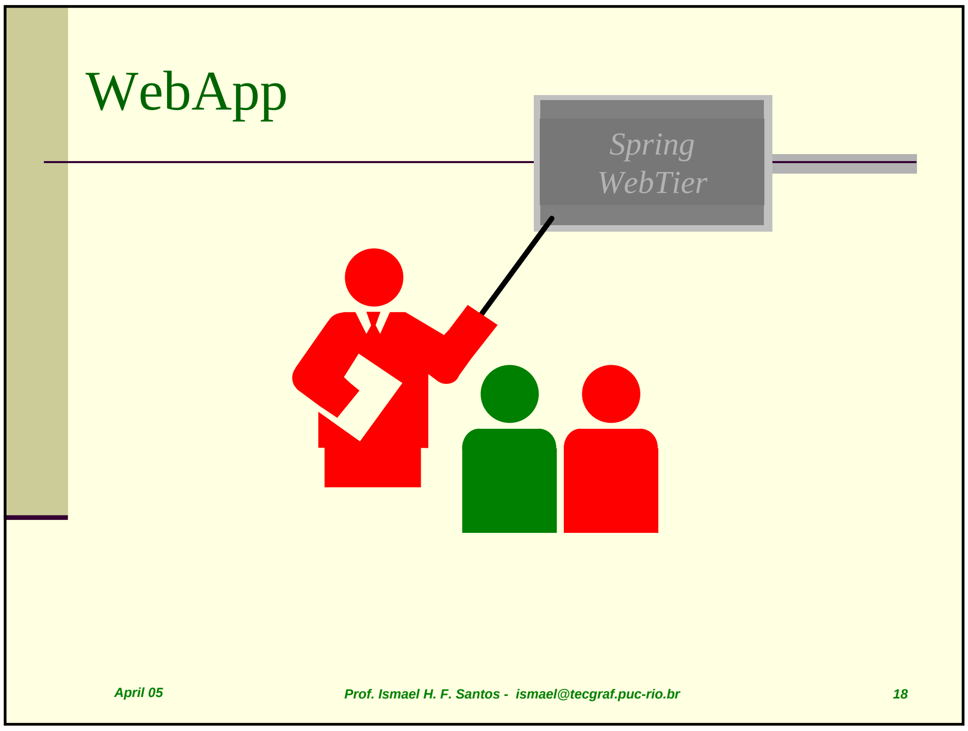
- Model 1
  - Recomendado para projetos pequenos.
  - E/S: Java Server Pages
  - Lógica de negócio: Java Beans e EJBs
- Model 2
  - Recomendada para projetos médios e grandes.
  - Variação do padrão MVC
  - Controller: Servlets
  - Model: JavaBeans e EJBs
  - View: Java Server Pages



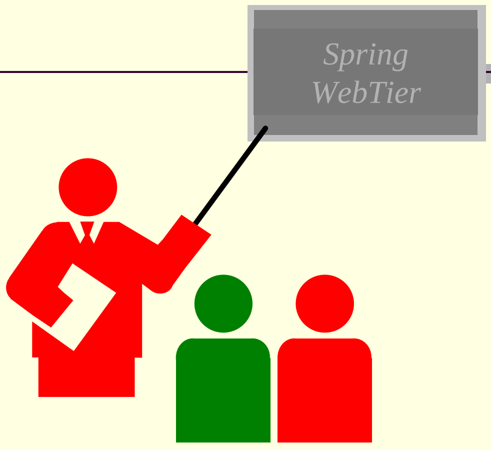


Futuro: WebBeans – JBoss Seam

- JSR 299 – Web Beans;
- Unificação dos modelos EJB 3 e JSF 2;
  - EJB 3 traz idéias bem-sucedidas: ORM, DI, etc., porém a integração com JSF ainda é trabalhosa e tediosa.
- Web Beans unifica os modelos de componentes; JBoss Seam. O criador do Seam é Spec Lead do Web Beans.
  - Integração JSF – EJB3 (modelo de componentes unificado);
  - AJAX e jBPM integrados;
  - Gerenciamento de estado declarativo;
  - Bijection, Conversation e Workspaces;
  - Utilização de POJOs com anotações;
  - Testabilidade;
  - I18n, autenticação, depuração, URLs RESTful,
  - seam-gen, eventos, interceptadores, etc.



WebApp

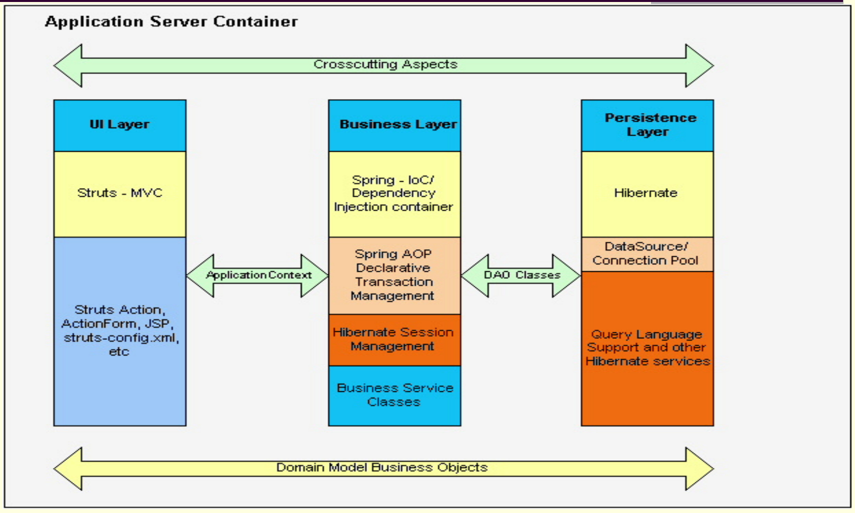


Spring MVC

- Construído sobre o núcleo do Spring.
- Acomoda várias tecnologias de *view*:
  - JSP, Velocity, Tiles, iText, POI etc. JSP, Velocity, Tiles, iText, POI etc.
- Pode ser combinado a outras *web tiers*:
  - Struts, WebWork, Tapestry etc. Struts, WebWork, Tapestry etc.
- Configurável via *strategy interfaces*.

Spring

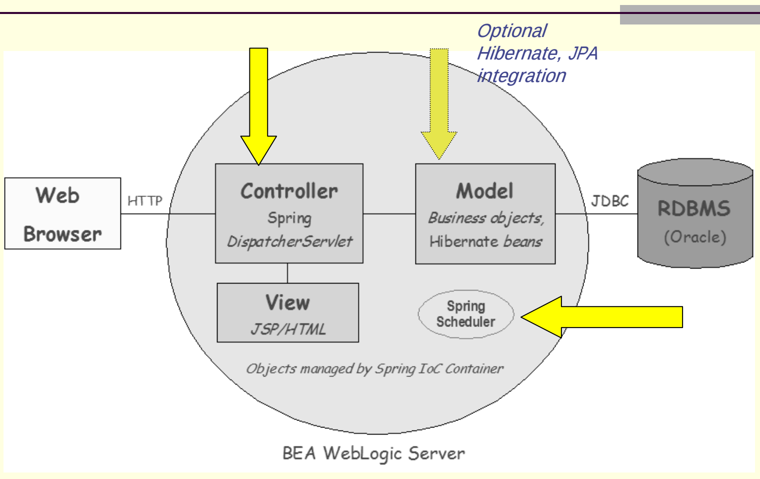
Proposed Web App Layering


Application Server Container
Crosscutting Aspects
UI Layer
Struts - MVC
Struts Action, ActionForm, JSP, struts-config.xml, etc
ApplicationContext
Business Layer
Spring - IoC/ Dependency Injection container
Spring AOP Declarative Transaction Management
Hibernate Session Management
Business Service Classes
DAO Classes
Persistence Layer
Hibernate
DataSource/ Connection Pool
Query Language Support and other Hibernate services
Domain Model Business Objects

Alguns Frameworks MVC

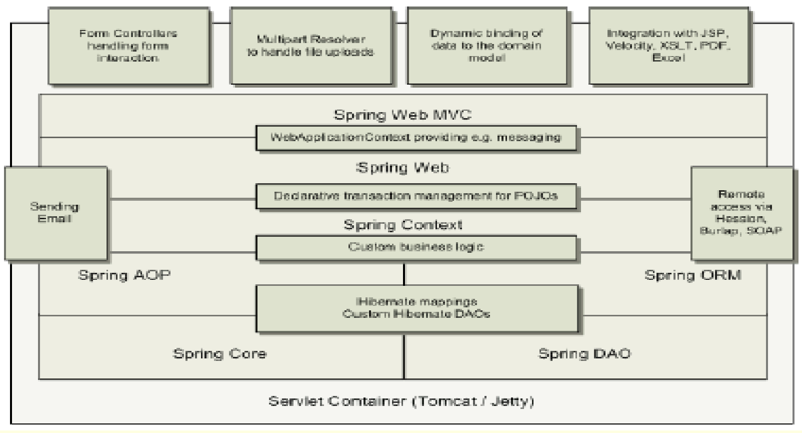
- Struts (Apache)
  - Frameworks e toolkits para aplicações web.
  - Voltado para o desenvolvimento de Model+Controller.
  - Público-alvo: desenvolvedores - http://struts.apache.org/
- Velocity (Apache)
  - *Template engine* para referenciar objetos Java.
  - Voltado para o desenvolvimento da View.
  - Público-alvo: web designers
  - http://jakarta.apache.org/velocity/
- Java Server Faces (Sun)
  - Tecnologia para construção de UIs web para aplicações Java.
  - Voltado para o desenvolvimento da View.
  - Público-alvo: desenvolvedores
  - http://java.sun.com/javaee/javaserverfaces/

Where Spring Framework Fits Into a JEE Architecture



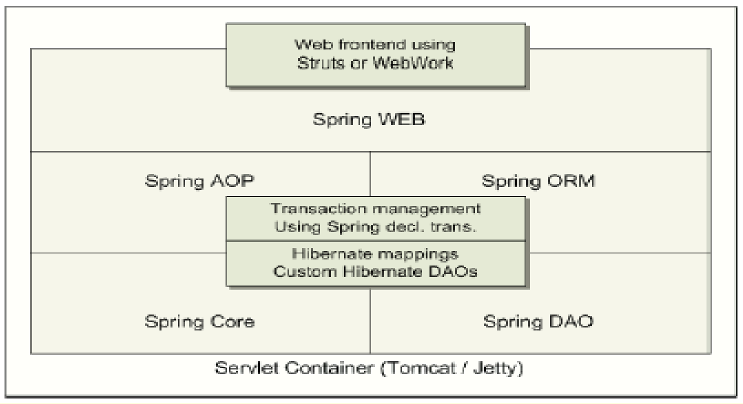
Web App – Scenario 1

Full-fledged Spring web application


Form Controllers handling form interaction
Multipart Resolver to handle file uploads
Dynamic binding of data to the domain model
Integration with JSP, Velocity, XSLT, PDF, Excel
Spring Web MVC
WebApplicationContext providing e.g. messaging
Spring Web
Sending Email
Declarative transaction management for POJOs
Remote access via Hessian, Burlap, SOAP
Spring Context
Custom business logic
Spring AOP
Spring ORM
Hibernate mappings Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat / Jetty)

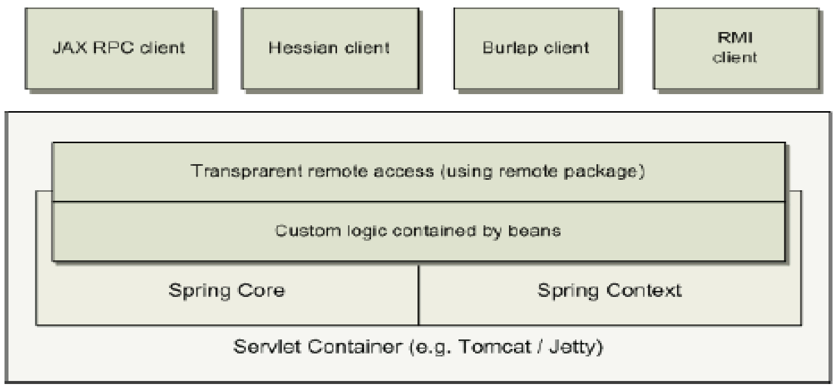
Web App – Scenario 2

Spring middle-tier using a third-party web framework


Web frontend using Struts or WebWork
Spring WEB
Spring AOP
Spring ORM
Transaction management Using Spring decl. trans.
Hibernate mappings Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat / Jetty)

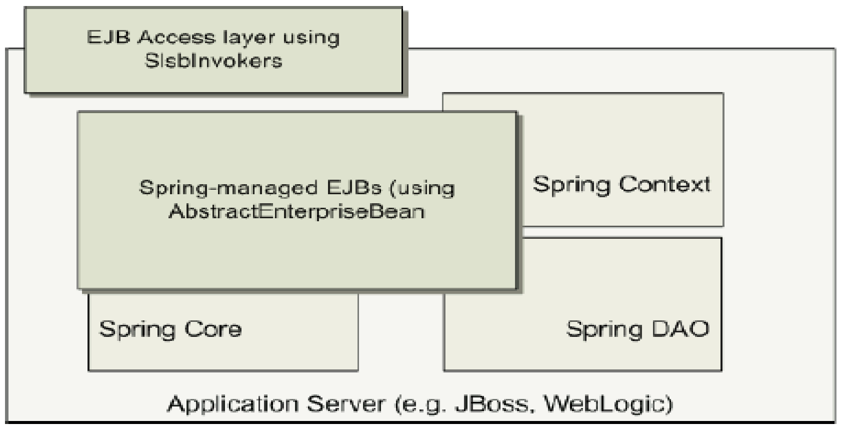
Web App – Scenario 3

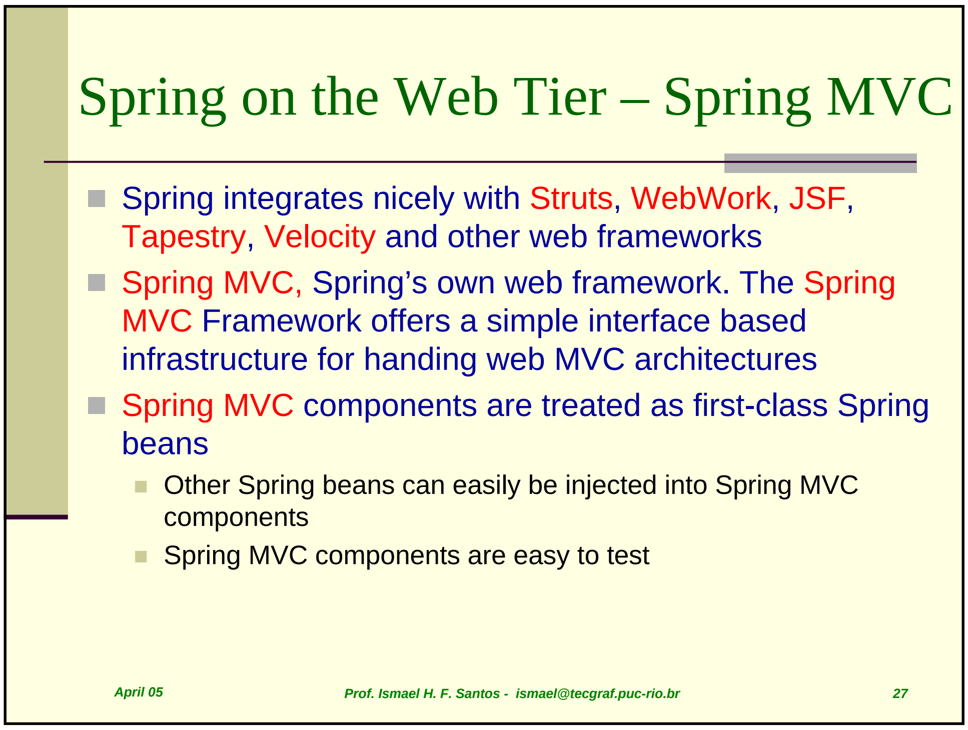
Remoting usage scenario


JAX RPC client
Hessian client
Burlap client
RMI client
Transprarent remote access (using remote package)
Custom logic contained by beans
Spring Core
Spring Context
Servlet Container (e.g. Tomcat / Jetty)

Web App – Scenario 4

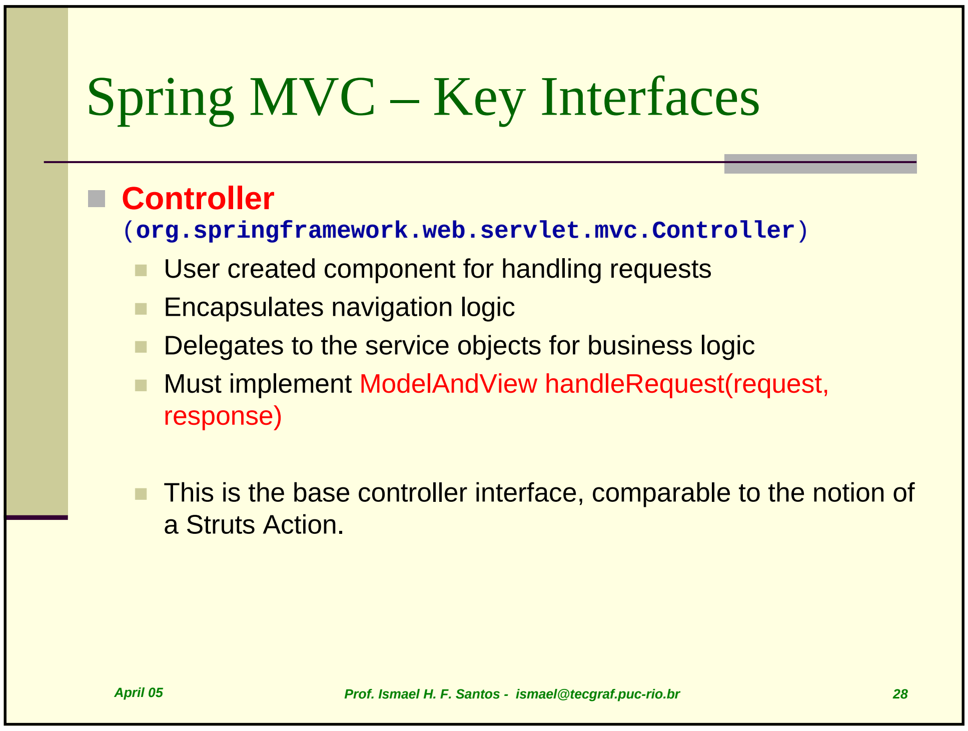
EJBs wrapping existing POJOs


EJB Access layer using SlsbInvokers
Spring-managed EJBs (using AbstractEnterpriseBean
Spring Context
Spring Core
Spring DAO
Application Server (e.g. JBoss, WebLogic)



Spring on the Web Tier – Spring MVC

- Spring integrates nicely with Struts, WebWork, JSF, Tapestry, Velocity and other web frameworks
- Spring MVC, Spring's own web framework. The Spring MVC Framework offers a simple interface based infrastructure for handing web MVC architectures
- Spring MVC components are treated as first-class Spring beans
  - Other Spring beans can easily be injected into Spring MVC components
  - Spring MVC components are easy to test



Spring MVC – Key Interfaces

- **Controller** (`org.springframework.web.servlet.mvc.Controller`)
  - User created component for handling requests
  - Encapsulates navigation logic
  - Delegates to the service objects for business logic
  - Must implement ModelAndView handleRequest(request, response)
  - This is the base controller interface, comparable to the notion of a Struts Action.

Spring MVC – Key Interfaces

- **View** (`org.springframework.web.servlet.mvc.View`)
  - Responsible for rendering output
  - Must implement `void render(model, request, response)`
  - *This is the MVC view for a web interaction. Implementations are responsible for rendering content, and exposing the model.*
- **Model**
  - To complete the MVC trio, note that the model is typically handled as a `java.util.Map` which is returned with the view
  - the values of the model are available, for example in a JSP, using a `<jsp:useBean/>` where the **id** corresponds to the key value in the Map

Spring MVC – Key Interfaces

- **ModelAndView**
  - Created by the Controller
  - Stores the Model data
  - Associates a View to the request
    - Can be a physical View implementation or a logical View name
- **ViewResolver**
  - Used to map logical View names to actual View
  - implementations
- **HandlerMapping**
  - Strategy interface used by DispatcherServlet for mapping incoming requests to individual Controllers

MVC and Dependency Injection

- All MVC components are configured in the Spring ApplicationContext
- As such, all MVC components can be configured using Dependency Injection
- Example:

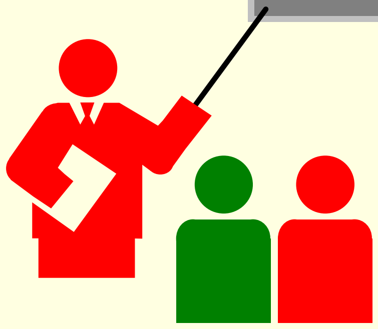
```
<bean id="springCheersController"
   class="com....web.SpringCheersController">
   <property name="methodNameResolver“
                                ref=“springCheersMethodResolver"/>
   <property name="service" ref="service"/>
</bean>
```

Spring on the Web Tier: Integration with Other Frameworks

- Spring integrates nicely with other web frameworks with two methodologies:
  - Look up Spring beans within Controllers/Actions via the convenience static method:
    - WebApplicationContextUtils.
    getWebApplicationContext(servletContext).getBean(“beanName”)
  - Configure the Controllers/Actions for the web framework in a Spring BeanFactory and then use Spring provided proxies in the actual web framework configuration
    - When available, this methodology is preferred
    - This approach lets you design your Controllers/Actions with dependency injection and makes your Controller/Actions more testable

WebApp

SpringMVC application



Aplicação Básica

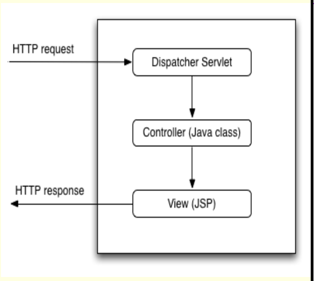
- Web.xml Entry
  - Standard J2EE web container servlet definition
  - Establishes which and where listeners act on requests
- Bean Definitions
  - *${servlet-name}*-servlet.xml
    - Beans unique to the the web context [created for the configured Servlet dispatcher]
  - ApplicationContext.xml
    - Beans common to all contexts

Aplicação Básica - Web.xml

- ${webapp}\WEB-INF\web.xml

Standard Servlet

```
web.xml:
<web-app>
…
  <servlet>
    <servlet-name>springmvc</servlet-name>
     <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>
  …
  <servlet-mapping>
    <servlet-name> springmvc</servlet-name>
    <url-pattern>/hello/world/*</url-pattern>
  </servlet-mapping>
  …
</web-app>
```



Standard URL filtering

Aplicação Básica - Web.xml

- ${webapp}\WEB-INF\web.xml

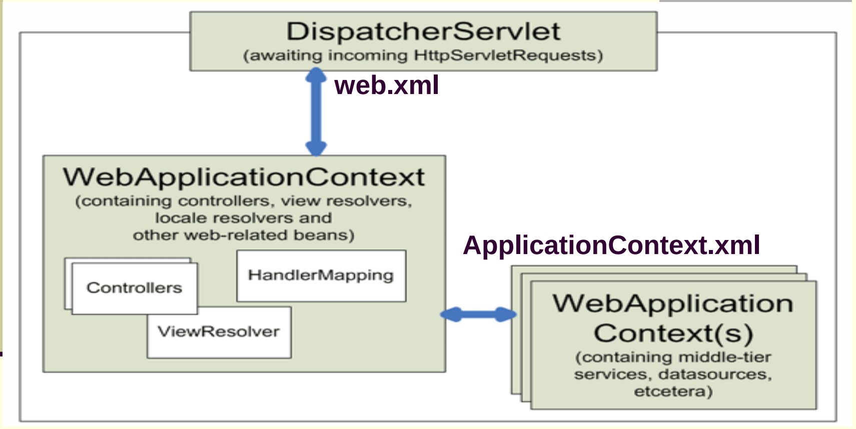
```
web.xml:
<web-app>
  …
  <servlet>
    <servlet-name> springmvc</servlet-name>
     <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
     </servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>
  …
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.html</url-pattern>
  </servlet-mapping>
  …
</web-app>
```

1.Servlet que despacha requisições para as implementações dos Controles registrados
2.Cada servlet tem o seu proprio contexto (default definido em "{servlet-name}-servlet.xml")

Uma webapp pode ter qq numero desses servlets

3. redireciona todo .html para servelt de dispacho

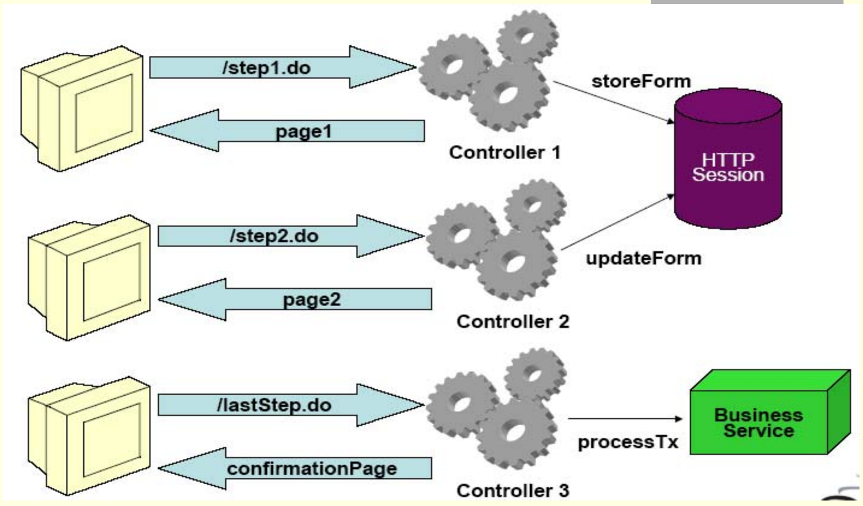
Aplicação Básica - Context Hierarchy


DispatcherServlet
(awaiting incoming HttpServletRequests)
web.xml
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
ApplicationContext.xml
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)

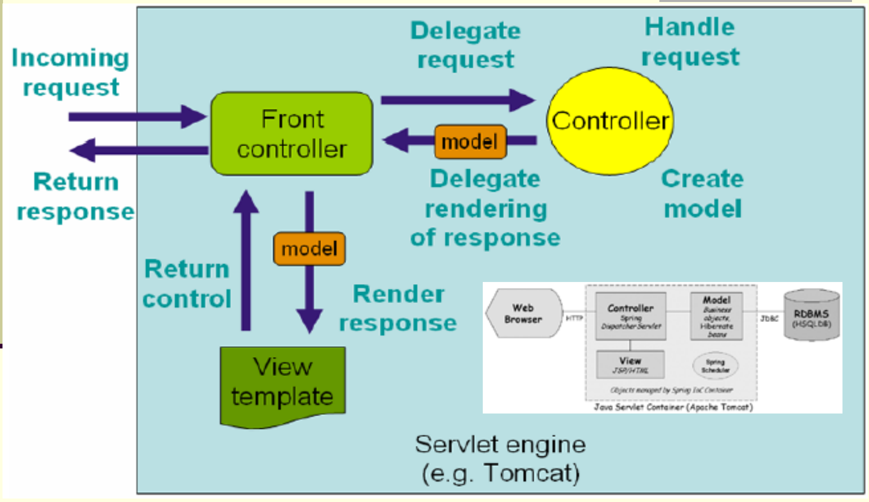
- *${servlet-name}*-servlet.xml

From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

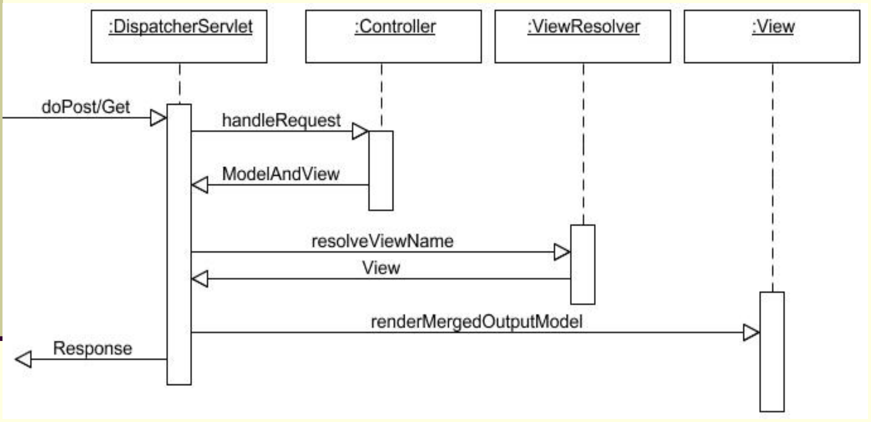
Spring-MVC - Controllers



Spring-MVC - DispatcherServlet



Spring MVC – Execução (Run-time)



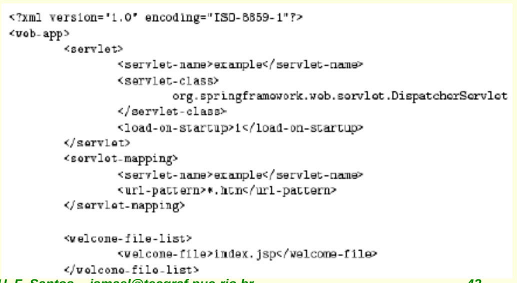
http://www.theserverside.com/tt/articles/article.tss?l=AjaxandSpring

Spring MVC Flow

- DispatcherServlet
  - recebe requisições domeio externo (do navegador, por exemplo) e comanda o fluxo de tarefas no Spring MVC;
- HandlerMapping
  - dada uma requisição em URL, este componente irá retornar o Controller que está associado a ela;
- Controller
  - realiza comuniação entre o MVC do Spring com a camada de negócio. Retorna um objeto ModelAndView;
- ModelAndView
  - armazena os dadosretornados pela camada de negócio para serem exibidos. Além disso, contêm um nome lógico de uma determinada View

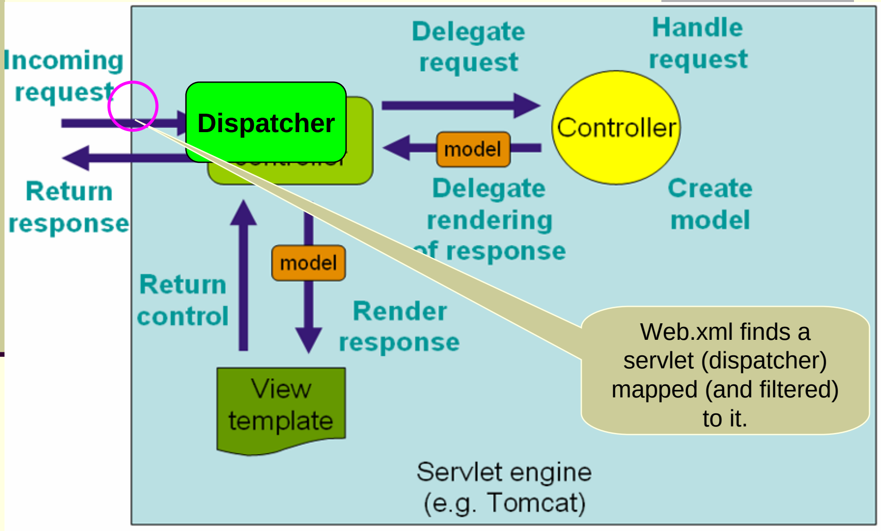
Spring MVC Flow

- View
  - contêm informações de renderização para que o usuário possa ver o que solicitou;
- ViewResolver
  - a partir do nome lógico contido no objeto ModelAndView, este componente determina a View que será exibida.
- DispatcherServlet
  - Delega operações para outros componentes;
  - web.xml -->



```
<?xml version='1.0' encoding="ISO-8859-1"?>
<web-app>
        <servlet>
                <servlet-name>example</servlet-name>
                <servlet-class>
                        org.springframework.web.servlet.DispatcherServlet
                </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>example</servlet-name>
                <url-pattern>*.htm</url-pattern>
        </servlet-mapping>

        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
```

Spring MVC Flow – Pre Dispatcher

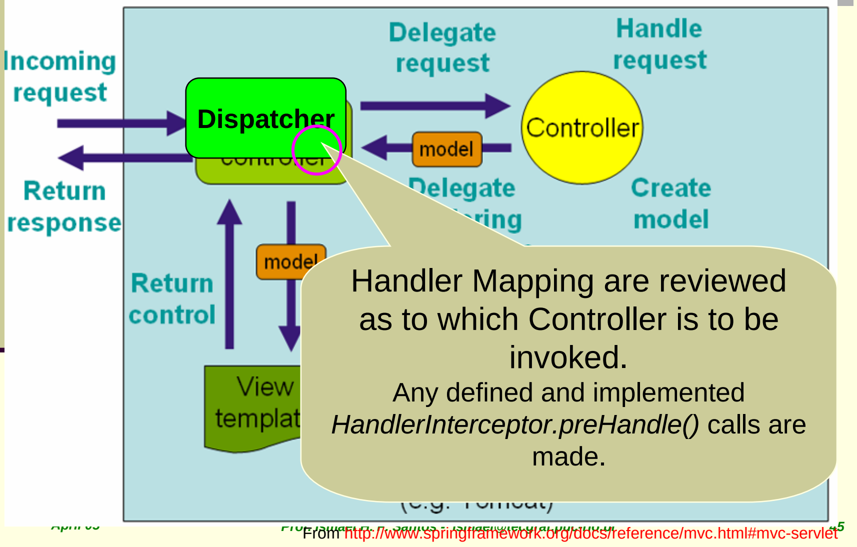


From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

Spring MVC Flow – Pre Dispatcher

```
16:19:22:408 - INFO - HttpServletBean - Initializing servlet 'dispatcher'
16:19:22:408 - INFO - FrameworkServlet - FrameworkServlet 'dispatcher': initialization started
16:19:22:423 - INFO - XmlBeanDefinitionReader - Loading XML bean definitions from ServletContext resource [/WEB-INF/report-
servlet.xml]
16:19:22:455 - INFO - AbstractRefreshableApplicationContext - Bean factory for application context [WebApplicationContext for
namespace 'report-servlet']: org.springframework.beans.factory.support.DefaultListableBeanFactory defining beans
[productStatusController,ProductStatusAsXMLView,urlMapping,viewResolver,messageSource]; parent:
org.springframework.beans.factory.support.DefaultListableBeanFactory defining beans [productManager,product1,product2,product3];
root of BeanFactory hierarchy
16:19:22:455 - INFO - AbstractApplicationContext - 5 beans defined in application context [WebApplicationContext for namespace
'report-servlet']
16:19:22:455 - INFO - AbstractApplicationContext - Using MessageSource
[org.springframework.context.support.ResourceBundleMessageSource: basenames=[spring2webexamples.bus.report.web.messages]]
16:19:22:455 - INFO - AbstractApplicationContext - Unable to locate ApplicationEventMulticaster with name
'applicationEventMulticaster': using default [org.springframework.context.event.SimpleApplicationEventMulticaster@45b199]
16:19:22:455 - INFO - UiApplicationContextUtils - Unable to locate ThemeSource with name 'themeSource': using default
[org.springframework.ui.context.support.DelegatingThemeSource@18ec669]
16:19:22:470 - INFO - DefaultListableBeanFactory - Pre-instantiating singletons in factory
… productStatusController,ProductStatusAsXMLView
16:19:22:517 - INFO - FrameworkServlet - Using context class [org.springframework.web.context.support.XmlWebApplicationContext]
for servlet 'report'
16:19:22:517 - INFO - DispatcherServlet - Unable to locate MultipartResolver with name 'multipartResolver': no multipart request
handling provided
16:19:22:517 - INFO - DispatcherServlet - Unable to locate LocaleResolver with name 'localeResolver': using default
[org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver@ec459a]
16:19:22:517 - INFO - DispatcherServlet - Unable to locate ThemeResolver with name 'themeResolver': using default
[org.springframework.web.servlet.theme.FixedThemeResolver@1cd9466]
16:19:22:517 - INFO - DispatcherServlet - No HandlerAdapters found in servlet 'dispatcher': using default
16:19:22:517 - INFO - DispatcherServlet - Unable to locate RequestToViewNameTranslator with name 'viewNameTranslator': using
default [org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator@6a765]
16:19:22:517 - INFO - FrameworkServlet - FrameworkServlet 'dispatcher': initialization completed in 109 ms
16:19:22:517 - INFO - HttpServletBean - Servlet 'dispatcher' configured successfully
```

Spring MVC Flow – Dispatcher

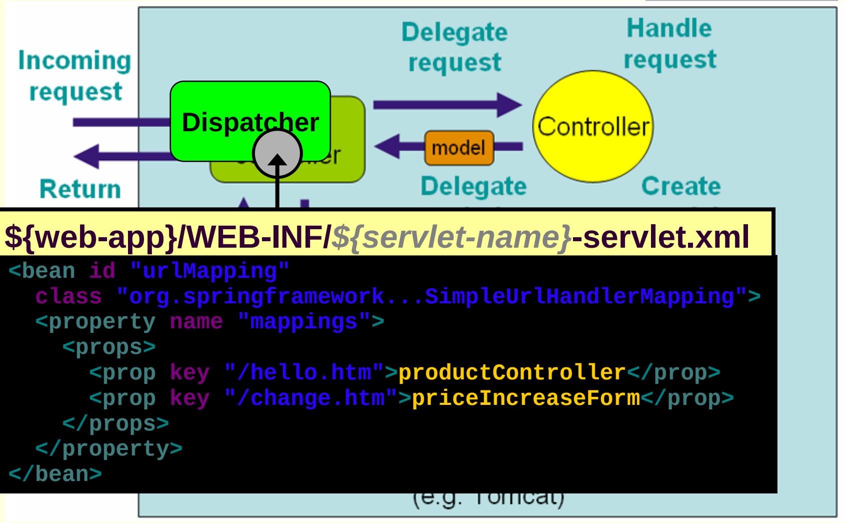


Handler Mapping are reviewed as to which Controller is to be invoked.

Any defined and implemented *HandlerInterceptor.preHandle()* calls are made.

From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

Spring MVC Flow – Dispatcher



${web-app}/WEB-INF/*${servlet-name}*-servlet.xml

```
<bean id "urlMapping"
  class "org.springframework...SimpleUrlHandlerMapping">
  <property name "mappings">
    <props>
      <prop key "/hello.htm">productController</prop>
      <prop key "/change.htm">priceIncreaseForm</prop>
    </props>
  </property>
</bean>
```

Handler Mappings

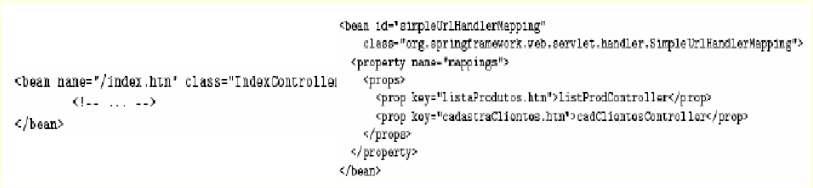
- The process of *Handler Mapping* binds incoming web requests to appropriate handlers that then resolve to a Controller(s) (in most cases)
  - On inbound requests, the DispatcherServlet hands it over to the handler mapping to come up with an appropriate HandlerExecutionChain
- Note: Handler Mappings apply from the base url-pattern specified in the web.xml
- By default, if no handler mapping can be found in the context, the DispatcherServlet creates a BeanNameUrlHandlerMapping for you

"Out of the box" HandlerMappings

- Concrete Implementations
  - BeanNameUrlHandlerMapping
    - maps incoming HTTP requests to names of beans, defined in the web application context
  - SimpleUrlHandlerMapping
    - Map Ant-style path matching *(see org.springframework.util.PathMatcher)* to a Controller
  - CommonsPathMapHandlerMapping
    - use Commons Attributes to determine mapping
  - ControllerClassNameHandlerMapping
- Most often, the out-of-the box implementations are sufficient

“Out of the box” HandlerMappings

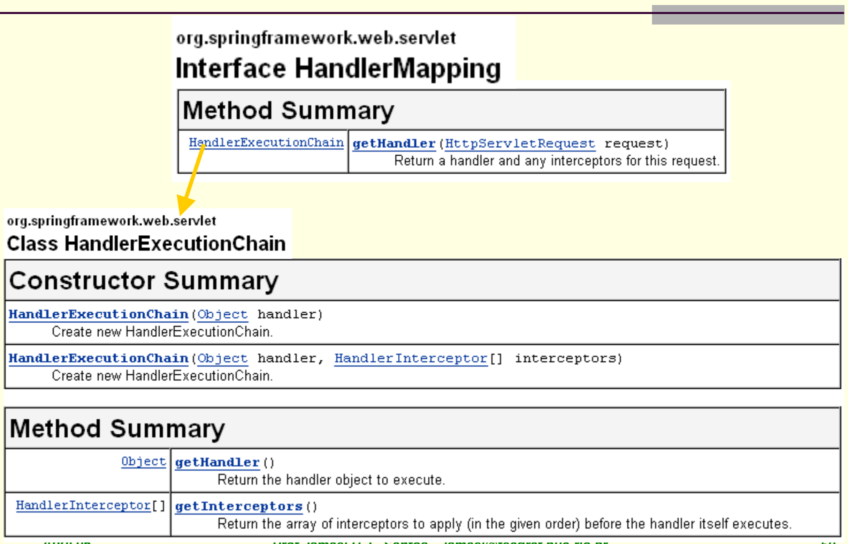
- BeanNameUrlHandlerMapping;
  - procura por controladores cujo nome corresponde à alguma porção de um nome URL de alguma requisição.
- SimpleUrlHandlerMapping;
  - é que é muito mais versátil, pois ele permite que se declare uma lista de associações url-controladores, e não interfere na nomencaltura dos beans de controladores, deixando-os mais claros.



```
<bean name="/index.htm" class="IndexControlle
        <!-- ... -->
</bean>
```

```
<bean id="simpleUrlHandlerMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="listaProdutos.htm">listProdController</prop>
      <prop key="cadastraClientes.htm">cadClientesController</prop>
    </props>
  </property>
</bean>
```

Anecdote: Under the covers Mapping Execution



org.springframework.web.servlet
Interface HandlerMapping

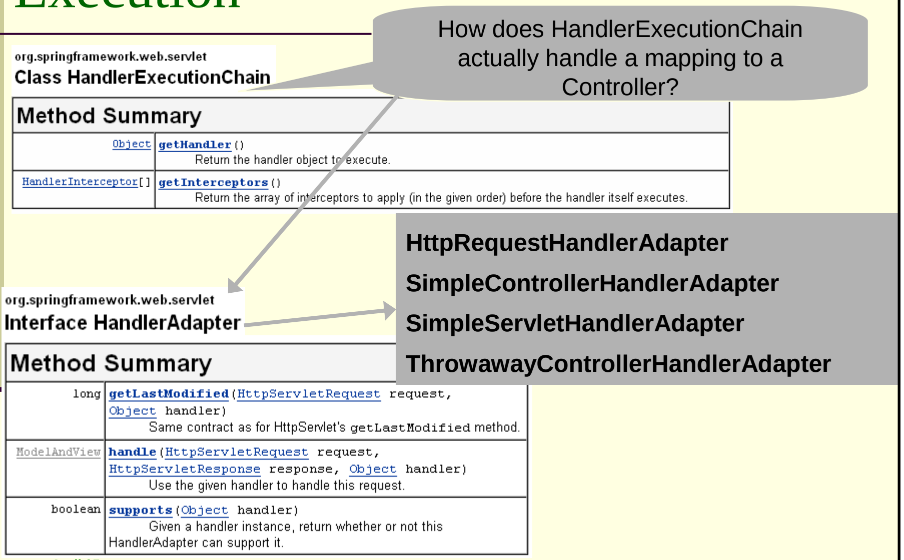
Method Summary	
HandlerExecutionChain	**getHandler**(HttpServletRequest request) Return a handler and any interceptors for this request.

org.springframework.web.servlet
Class HandlerExecutionChain

Constructor Summary
HandlerExecutionChain(Object handler) Create new HandlerExecutionChain.
HandlerExecutionChain(Object handler, HandlerInterceptor[] interceptors) Create new HandlerExecutionChain.

Method Summary	
Object	**getHandler**() Return the handler object to execute.
HandlerInterceptor[]	**getInterceptors**() Return the array of interceptors to apply (in the given order) before the handler itself executes.

Anecdote: Under the covers Mapping Execution



How does HandlerExecutionChain actually handle a mapping to a Controller?

org.springframework.web.servlet
Class HandlerExecutionChain

Method Summary	
Object	**getHandler**() Return the handler object to execute.
HandlerInterceptor[]	**getInterceptors**() Return the array of interceptors to apply (in the given order) before the handler itself executes.

org.springframework.web.servlet
Interface HandlerAdapter

HttpRequestHandlerAdapter
SimpleControllerHandlerAdapter
SimpleServletHandlerAdapter
ThrowawayControllerHandlerAdapter

Method Summary	
long	**getLastModified**(HttpServletRequest request, Object handler) Same contract as for HttpServlet's `getLastModified` method.
ModelAndView	**handle**(HttpServletRequest request, HttpServletResponse response, Object handler) Use the given handler to handle this request.
boolean	**supports**(Object handler) Given a handler instance, return whether or not this HandlerAdapter can support it.

Spring-MVC

- As you can see, all incoming HTTP requests from a web browser are handled by Controllers. A *controller*, as the name indicates, controls the view and model by facilitating data exchange between them.
- The key benefit of this approach is that the model can worry only about the data and has no knowledge of the view.
- The view, on the other hand, has no knowledge of the model and business logic and simply renders the data passed to it (as a web page, in our case)
- The MVC pattern also allows us to change the view without having to change the model

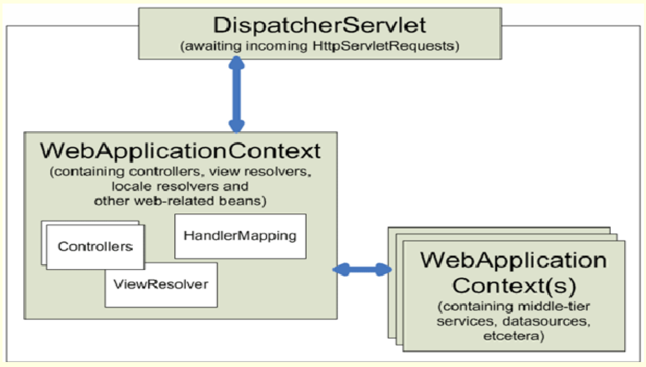
Spring-MVC

- The DispatcherServlet *is* an actual Servlet
  - Requests that you want the DispatcherServlet to handle will have to be mapped using a URL mapping in the same web.xml file.

```
<web-app>

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

    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>*.form</url-pattern>
    </servlet-mapping>

</web-app>
```

Spring-MVC

- Each DispatcherServlet has its own WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext. *file named [servlet-name]-servlet.xml* in the WEB-INF *directory*

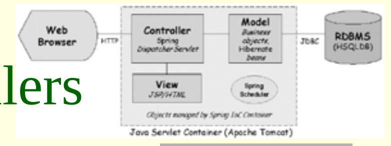

DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)

Spring-MVC

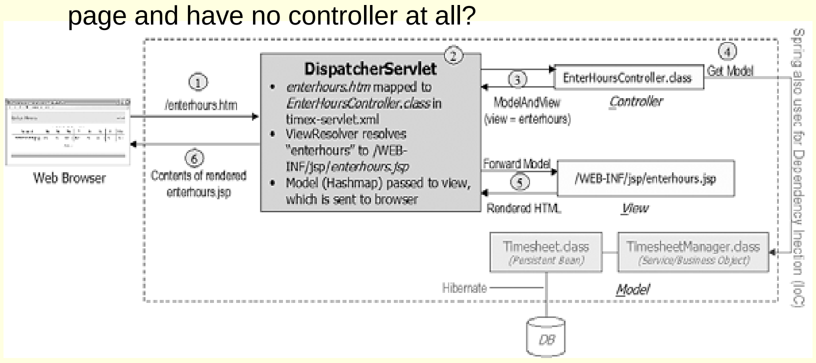
- With the above servlet configuration in place, you will need to have a file called '/WEB-INF/**golfing**-servlet.xml' in your application; this file will contain all of your *Spring Web MVC-specific* components (beans). The exact location of this configuration file can be changed via a servlet initialization parameter (see below for details).

```
<web-app>
    ...
    <servlet>
        <servlet-name>golfing</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>golfing</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

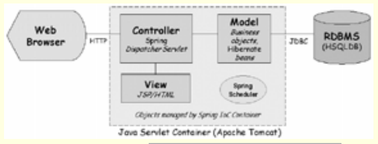
Spring-MVC Controllers


Web Browser
HTTP
Controller
Spring Dispatcher Servlet
Model
Business objects, Hibernate beans
JDBC
RDBMS (HSQLDB)
View
JSP/HTML
Spring Scheduler
Objects managed by Spring IoC Container
Java Servlet Container (Apache Tomcat)

- Spring provides many types of controllers.
  - The controller type depends on the functionality you need. For example, do your screens contain a form? Do you need wizardlike functionality? Do you just want to redirect to a JSP page and have no controller at all?


Web Browser
1
/enterhours.htm
6
Contents of rendered enterhours.jsp
2
DispatcherServlet
• enterhours.htm mapped to EnterHoursController.class in timex-servlet.xml
• ViewResolver resolves "enterhours" to /WEB-INF/jsp/enterhours.jsp
• Model (Hashmap) passed to view, which is sent to browser
3
ModelAndView (view = enterhours)
EnterHoursController.class
Controller
4
Get Model
Forward Model
5
Rendered HTML
/WEB-INF/jsp/enterhours.jsp
View
Timesheet.class (Persistent Bean)
TimesheetManager.class (Service/Business Object)
Hibernate
Model
DB
Spring also used for Dependency Injection (IoC)

Spring-MVC Objects



Web Browser
HTTP
Controller
Spring
Dispatcher Servlet
Model
Business objects, Hibernate beans
JDBC
RDBMS
(HSQLDB)
View
JSP/HTML
Spring Scheduler
Objects managed by Spring IoC Container
Java Servlet Container (Apache Tomcat)

- Model & View
  - Many of the methods in the Controller related subclasses return a org.springframework.web.servlet.ModelAndView object.This object holds the model (as a java.util.Map object) and view name and makes it possible to return both in one return value from a method.
- Command (Form Backing) Object
  - Spring uses the notion of a command object, which is a JavaBean class that gets populated with the data from an HTML form's fields.
  - This same object is also passed to our validators for data validation, and if the validations pass, it is passed to the onSubmit method (in controller related classes) for processing of valid data.

Spring-MVC

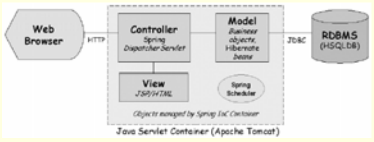
- Validator
  - An optional class that can be invoked for validating form data for a given command (form) controller. This validator class is a concrete class that implements org.springframework.validation.Validator interface.
  - One of the two methods required by this interface is the validate method, which is passed a command object, as mentioned previously, and an Errors object, which is used to return errors.
  - Another notable validation class is org.springframework.validation. ValidationUtils, which provides methods for rejecting empty fields.
- Spring Tag Library (spring-form.tld in spring.jar)
  - The spring bind tag library is simple yet powerful. It is typically used in JSP files via the <spring:bind> tag, which essentially binds HTML form fields to the command object.

Spring-MVC Configuration Concepts

- DispatcherServlet
  - DispatcherServlet (part of the org.springframework.web.servlet package) is the entry point to the world of Spring Web MVC.
- Handler Mappings
  - You can map handlers for incoming HTTP requests in the Spring application context file.These handlers are typically controllers that are mapped to partial or complete URLs of incoming requests
  - The handler mappings can also contain optional interceptors, which are invoked before and after the handler.

```
<bean id="urlMap"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="urlMap">
        <props>
            <prop key="/signin.htm">signInController</prop>
            <prop key="/signout.htm">signOutController</prop>
        </props>
    </property>
</bean>
```

Spring-MVC

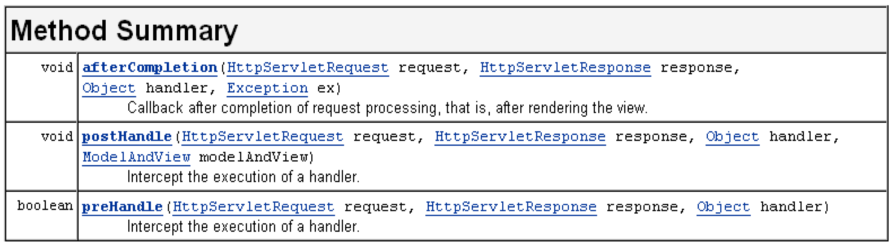

Web Browser
HTTP
Controller
Spring
Dispatcher Servlet
Model
Business objects, Hibernate beans
JDBC
RDBMS
(HSQLDB)
View
JSP/HTML
Spring Scheduler
Objects managed by Spring IoC Container
Java Servlet Container (Apache Tomcat)

- View Resolvers
  - Resolve view names to the actual views (enterhours to enterhours.jsp, for example). InternalResourceViewResolver is a class to resolve view names.
- Installing the Spring Framework on a Servlet Container
  - Spring Framework can be downloaded from http://springframework.org.

Use of Interceptors

- Some HandlerMappings allow you to call an interceptor before the controller
  - Useful for checking for session timeout, adding things to the request/session
  - common services: security, traffic logging, and perhaps front-end common data validation
    - Implement the HandlerInterceptor Interface
    - Simply wire your implemented Interceptors into the HandlerMapping
- Kind of like AOP, but for Controllers

Use of Interceptors

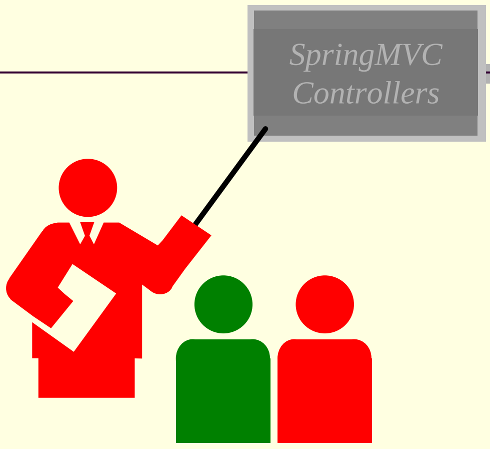


Method Summary

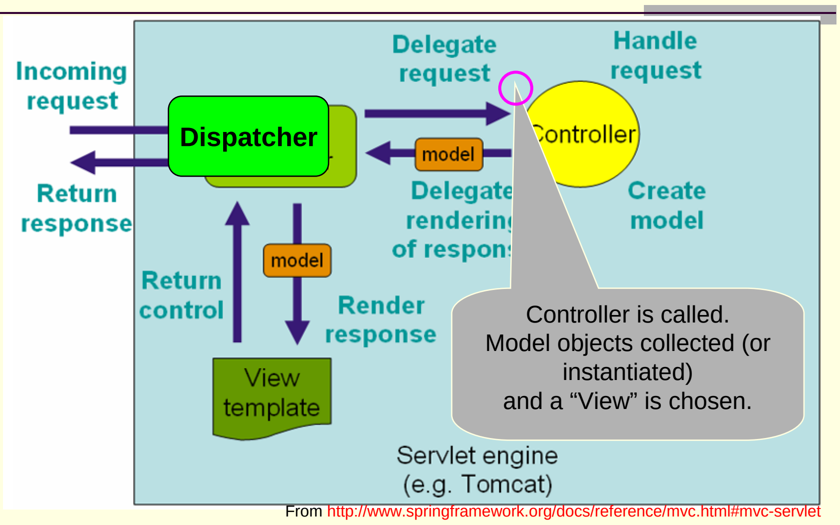
void	**afterCompletion**(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) Callback after completion of request processing, that is, after rendering the view.
void	**postHandle**(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) Intercept the execution of a handler.
boolean	**preHandle**(HttpServletRequest request, HttpServletResponse response, Object handler) Intercept the execution of a handler.

```
<bean id "urlMapping"
  class "org.springframework...SimpleUrlHandlerMapping">
  ...
  <property name "intercepters">
    <list>
      <bean class "...CustomHandlerIntercepter" />
    </list>
  </property>
</bean>
```

WebApp



Spring MVC Flow: Controller



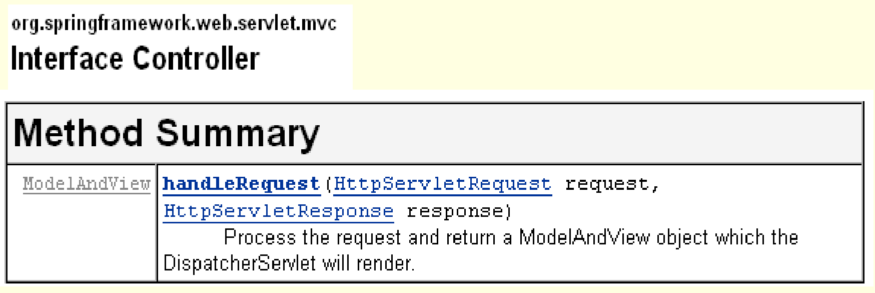
From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

MVC: Give me a "C"

- Controllers…
  - Provide access to the application behavior which is typically defined by a service interface
  - Controllers interpret input and transform said input into a sensible model which will be represented for output by the view
- The Controller in Spring is very abstract, allowing different kinds of controllers for different use cases.
- Out of the box Controllers existing to facilitate working with User driven Forms, "command" models, execute wizard-style logic, and more
- Role your own Controllers:
  - Common data handling, control logic, etc.

http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html#mvc-controller

The Simple Controller Interface

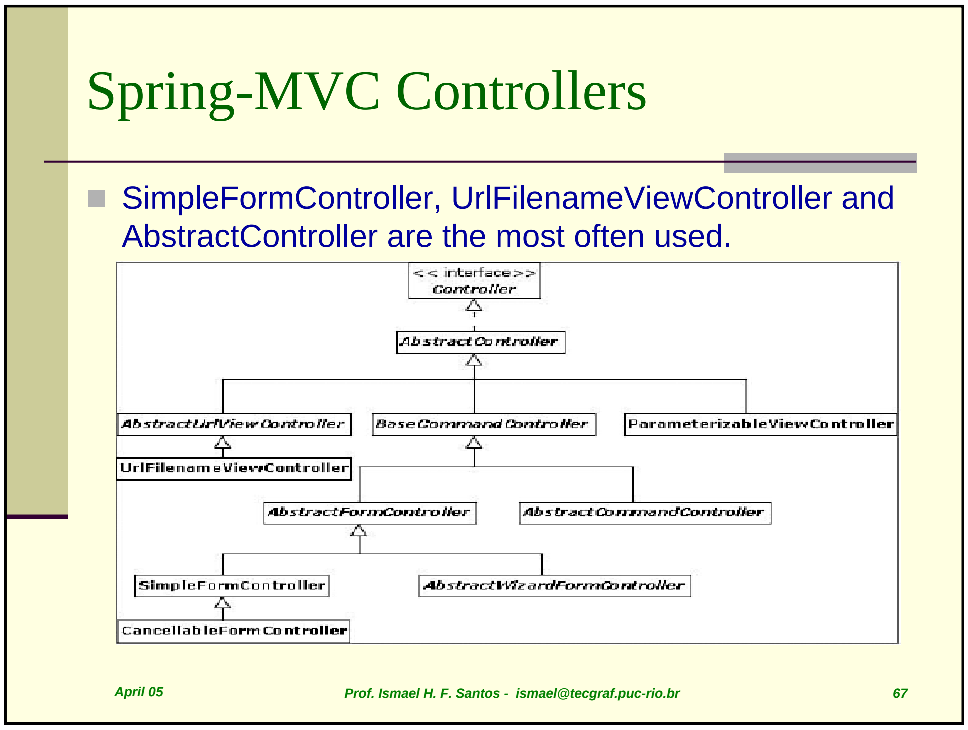


org.springframework.web.servlet.mvc

Interface Controller

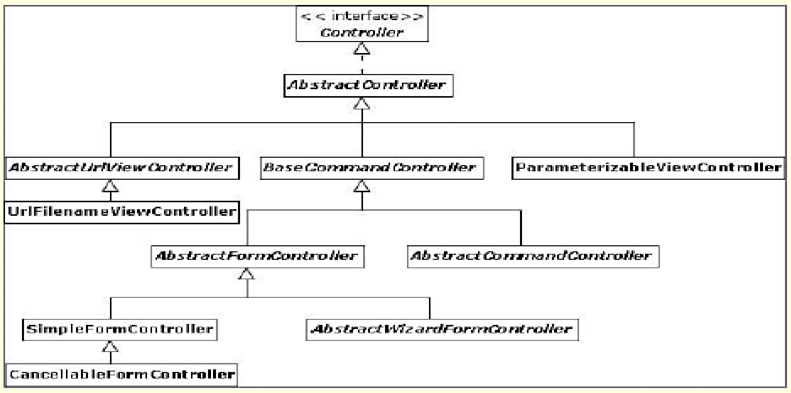
Method Summary	
`ModelAndView`	`handleRequest(HttpServletRequest request, HttpServletResponse response)` Process the request and return a ModelAndView object which the DispatcherServlet will render.

http://static.springframework.org/spring/docs/2.0.x/api/org/springframework/web/servlet/mvc/Controller.html

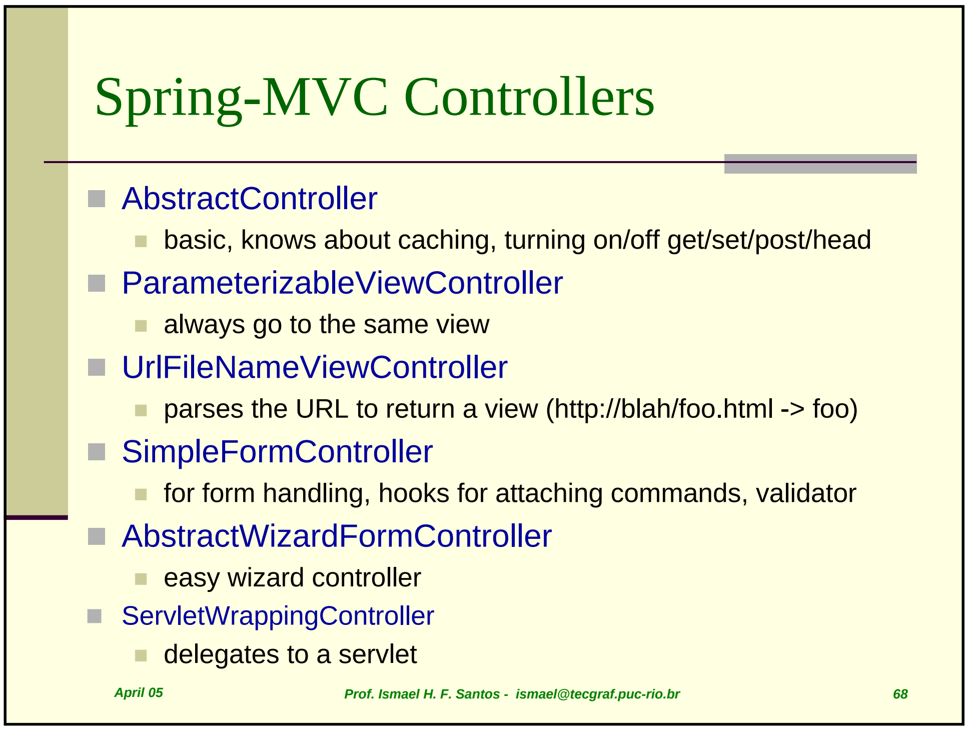


Spring-MVC Controllers

- SimpleFormController, UrlFilenameViewController and AbstractController are the most often used.



<< interface>>
Controller
AbstractController
AbstractUrlViewController
BaseCommandController
ParameterizableViewController
UrlFilenameViewController
AbstractFormController
AbstractCommandController
SimpleFormController
AbstractWizardFormController
CancellableFormController



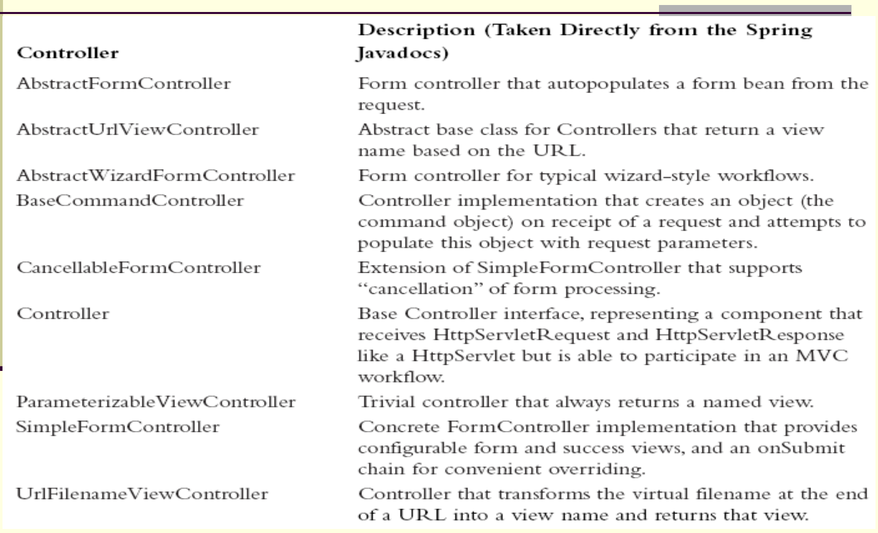
Spring-MVC Controllers

- AbstractController
  - basic, knows about caching, turning on/off get/set/post/head
- ParameterizableViewController
  - always go to the same view
- UrlFileNameViewController
  - parses the URL to return a view (http://blah/foo.html -> foo)
- SimpleFormController
  - for form handling, hooks for attaching commands, validator
- AbstractWizardFormController
  - easy wizard controller
- ServletWrappingController
  - delegates to a servlet

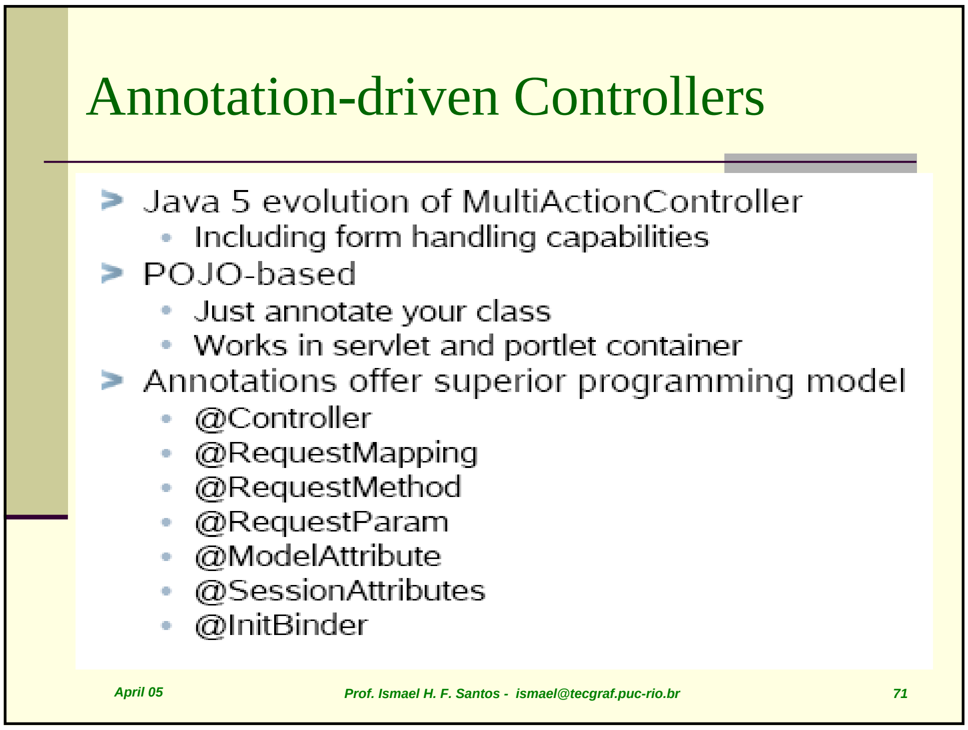
“Out of the box” Controllers

- Infrastructure
  - AbstractController
- Command Controllers
  - AbstractCommandController
  - BaseCommandController
- Form
  - SimpleFormController
  - AbstractFormController
  - AbstractWizardFormController
  - CancellableFormController
- Specialized
  - MultiActionController
  - ServletWrappingController
- Resource
  - ParameterizableViewController
  - ServletForwardingController
  - AbstractUrlViewController
  - UrlFilenameViewController

Spring-MVC Controllers



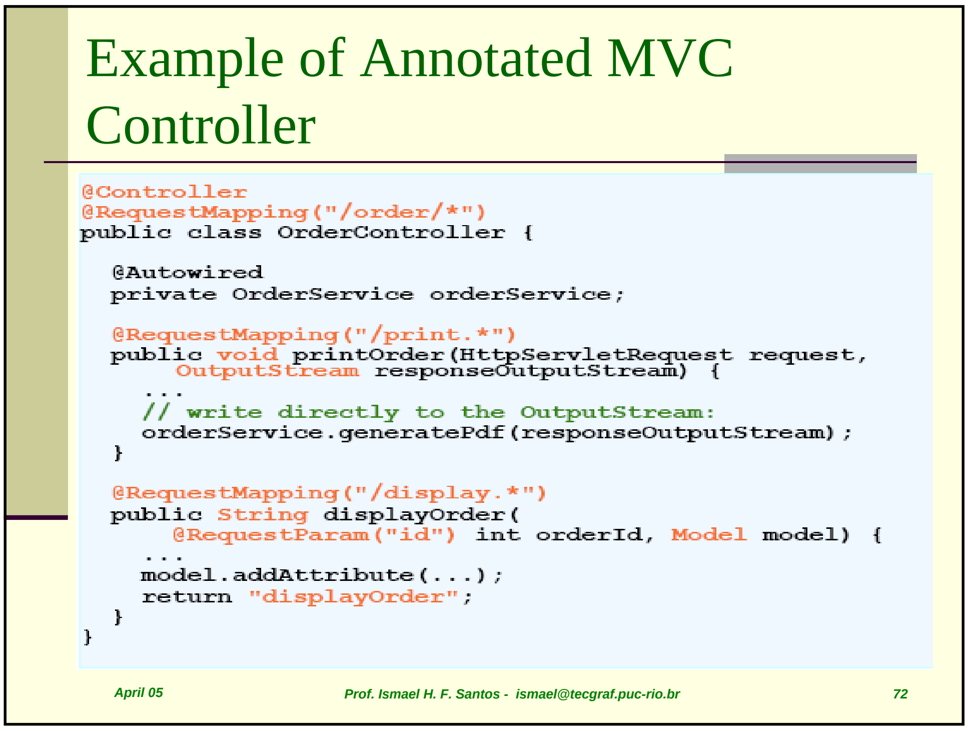
Controller	Description (Taken Directly from the Spring Javadocs)
AbstractFormController	Form controller that autopopulates a form bean from the request.
AbstractUrlViewController	Abstract base class for Controllers that return a view name based on the URL.
AbstractWizardFormController	Form controller for typical wizard-style workflows.
BaseCommandController	Controller implementation that creates an object (the command object) on receipt of a request and attempts to populate this object with request parameters.
CancellableFormController	Extension of SimpleFormController that supports “cancellation” of form processing.
Controller	Base Controller interface, representing a component that receives HttpServletRequest and HttpServletResponse like a HttpServlet but is able to participate in an MVC workflow.
ParameterizableViewController	Trivial controller that always returns a named view.
SimpleFormController	Concrete FormController implementation that provides configurable form and success views, and an onSubmit chain for convenient overriding.
UrlFilenameViewController	Controller that transforms the virtual filename at the end of a URL into a view name and returns that view.



Annotation-driven Controllers

- Java 5 evolution of MultiActionController
  - Including form handling capabilities
- POJO-based
  - Just annotate your class
  - Works in servlet and portlet container
- Annotations offer superior programming model
  - @Controller
  - @RequestMapping
  - @RequestMethod
  - @RequestParam
  - @ModelAttribute
  - @SessionAttributes
  - @InitBinder

April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 71

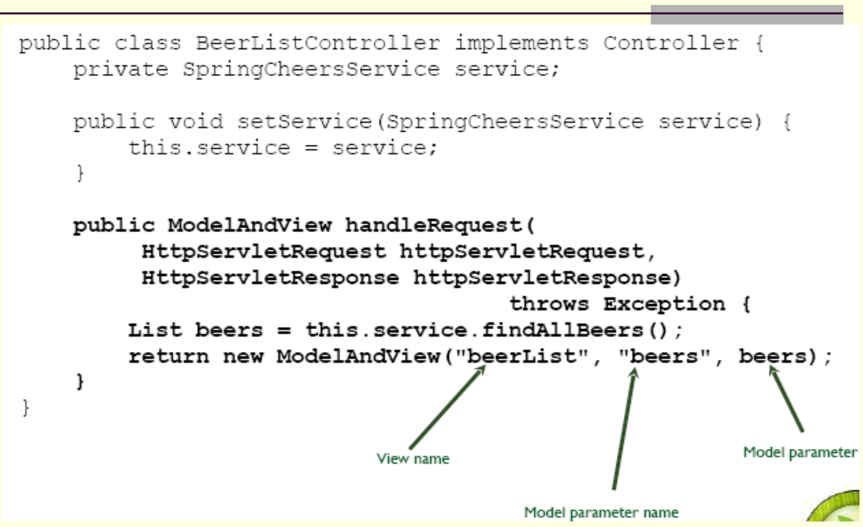


Example of Annotated MVC Controller

```
@Controller
@RequestMapping("/order/*")
public class OrderController {

  @Autowired
  private OrderService orderService;

  @RequestMapping("/print.*")
  public void printOrder(HttpServletRequest request,
      OutputStream responseOutputStream) {
    ...
    // write directly to the OutputStream:
    orderService.generatePdf(responseOutputStream);
  }

  @RequestMapping("/display.*")
  public String displayOrder(
      @RequestParam("id") int orderId, Model model) {
    ...
    model.addAttribute(...);
    return "displayOrder";
  }
}
```

April 05 Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br 72

Creating a Basic Controller

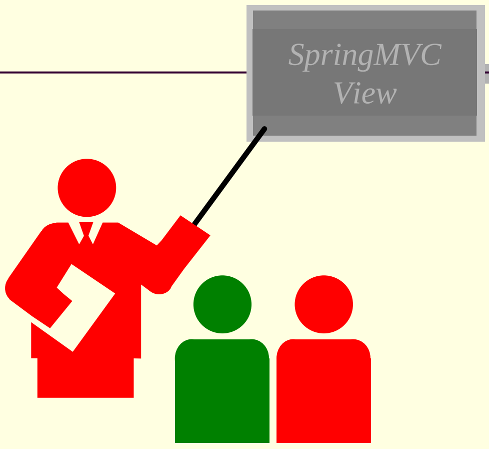


```
public class BeerListController implements Controller {
    private SpringCheersService service;

    public void setService(SpringCheersService service) {
        this.service = service;
    }

    public ModelAndView handleRequest(
         HttpServletRequest httpServletRequest,
         HttpServletResponse httpServletResponse)
                                    throws Exception {
        List beers = this.service.findAllBeers();
        return new ModelAndView("beerList", "beers", beers);
    }
}
```

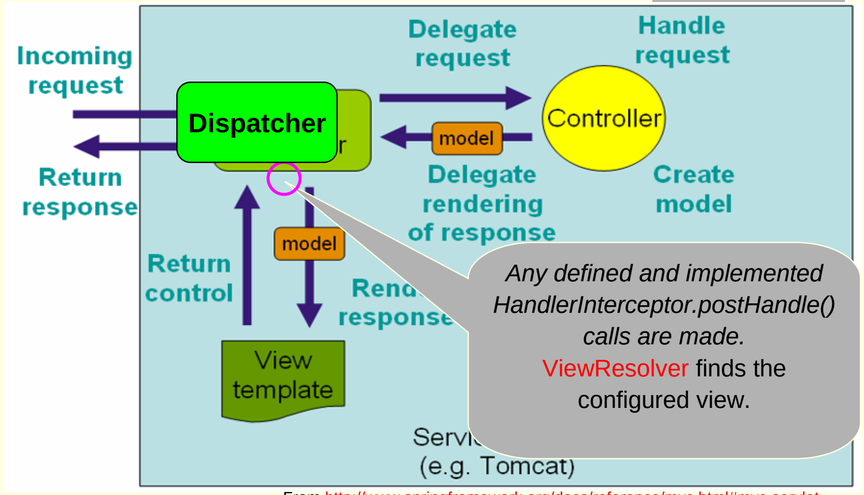
Handling Forms

- Set the Command (just a bean)
- Set a Validator if needed (extend org.springframework.validation.Validator)
- Set destination views (form, success, failure, error)
- By default, uses GET/POST to determine whether it needs to load the form or process it

WebApp



Spring MVC Flow: View Resolving



Any defined and implemented HandlerInterceptor.postHandle() calls are made.
ViewResolver finds the configured view.

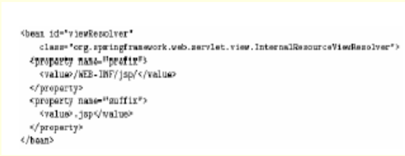
From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

View Resolving

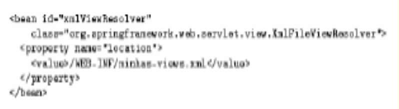
- If the ModelAndView suppplied to the Dispatcher from a Controller requires resolution *(the isReference() method is true)*, the a ViewResolver has the responsibility for finding a view matching the configured names
- View names are mapped to actual view implementations using ViewResolvers
- ViewResolvers are configured in the web-tier ApplicationContext
  - Automatically detected by DispatcherServlet
  - Can configure multiple, ordered ViewResolver

View Resolver

- O ViewResolver obtém o ***View*** a ser usado pelo aplicativo através da referência passada pelo objeto ***ModelAndView***, que por sua vez foi retornado por algum ***Controller***;
- Tipos de ViewResolver
  - InternalResourceViewResolver;
  - BeanNameViewResolver;
  - ResourceBundleViewResolver;
  - XmlViewResolver;



```
<bean id="viewResolver"
   class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix">
    <value>/WEB-INF/jsp/</value>
  </property>
  <property name="suffix">
    <value>.jsp</value>
  </property>
</bean>
```



```
<bean id="xmlViewResolver"
   class="org.springframework.web.servlet.view.XmlFileViewResolver">
  <property name="location">
    <value>/WEB-INF/minhas-views.xml</value>
  </property>
</bean>
```

View Resolving: Implementation Classes

InternalResourceViewResolver	A convenience subclass of UrlBasedViewResolver that supports InternalResourceView (i.e. Servlets and JSPs), and subclasses such as JstlView and TilesView. The view class for all views generated by this resolver can be specified via setViewClass(..). See the Javadocs for the UrlBasedViewResolver class for details.
BeanNameViewResolver	Simple implementation of ViewResolver that interprets a view name as bean name in the current application context, i.e. in the XML file of the executing DispatcherServlet. This resolver can be handy for small applications, keeping all definitions ranging from controllers to views in the same place.
UrlBasedViewResolver	A simple implementation of the ViewResolver interface that effects the direct resolution of symbolic view names to URLs, without an explicit mapping definition. This is appropriate if your symbolic names match the names of your view resources in a straightforward manner, without the need for arbitrary mappings.
ResourceBundleViewResolver	An implementation of ViewResolver that uses bean definitions in a ResourceBundle, specified by the bundle basename. The bundle is typically defined in a properties file, located in the classpath. The default file name is views.properties.
XmlViewResolver	An implementation of ViewResolver that accepts a configuration file written in XML with the same DTD as Spring's XML bean factories. The default configuration file is /WEB-INF/views.xml.
AbstractCachingViewResolver	An abstract view resolver which takes care of caching views. Often views need preparation before they can be used, extending this view resolver provides caching of views.
VelocityViewResolver / FreeMarkerViewResolver	A convenience subclass of UrlBasedViewResolver that supports VelocityView (i.e. Velocity templates) or FreeMarkerView respectively and custom subclasses of them.

From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

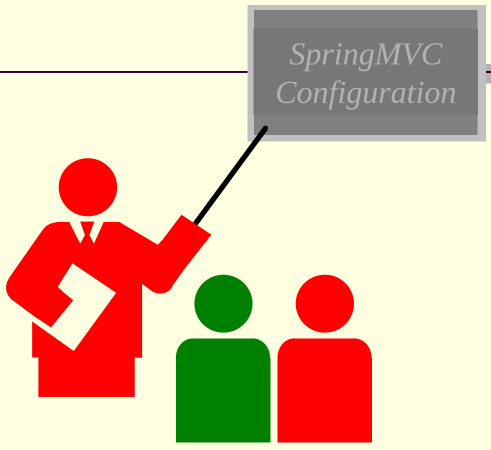
Different Views

- Plenty of Views are packaged with Spring MVC:
  - JstlView
    - map to a JSP page
  - RedirectView
    - Perform an HTTP Redirect
  - TilesView, TilesJstlView
    - integration with tiles
  - VelocityLayoutView, VelocityToolboxView, VelocityView
    - Integration with the Velocity templating tool
  - FreeMarkerView
    - use the FreeMarker templating tool
  - JasperReportsView, JasperReportsMultiFormatView, JasperReportsMultiFormatView, JasperReportsPdfView, JasperReportsXlsView
    - Support for Jasper Reports

Creating a View with JSP and JSTL

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<html>
  <head><title>Beer List</title></head>
  <body>
  <table border="0">
  <c:forEach items="${beers}" var="beer">
    <tr>
      <td><c:out value="${beer.id}"/></td>
      <td><c:out value="${beer.brand}"/></td>
    </tr>
  </c:forEach>
  </table>
  </body>
</html>
```

WebApp



Configuring DispatcherServlet

```
<servlet>
  <servlet-name>springcheers</servlet-name>
  <servlet-class>
      o.s.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>springcheers</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Configuring ContextLoaderListener

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

<listener>
   <listener-class>
       o.s.web.context.ContextLoaderListener
   </listener-class>
</listener>
```

```
<bean id="beerListController"
   class="com.springcheers.web.BeerListController">
     <property name="service" ref="service"/>
</bean>
```

Mapping URLs to Controllers

- Mapping request (URLs) to `Controller`
- Controlled by implementations of the `HandlerMapping` interface
- Useful out-of-the-box implementations
  - `BeanNameUrlHandlerMapping`
    - Uses the `Controller` bean name as the URL mapping
  - `SimpleUrlHandlerMapping`
    - Define a set of URL pattern to bean mappings
  - Most out of the box implementations support Ant-style path matching

Configure a HandlerMapping

```
<bean id="urlMapping"
   class="o.s.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/list.htm">springCheersController</prop>
      <prop key="/view.htm">springCheersController</prop>
      <prop key="/edit.htm">customerForm</prop>
      <prop key="/create.htm">customerForm</prop>
      <prop key="/beer/list.htm">beerListController</prop>
    </props>
  </property>
</bean>
```

Configuring the ViewResolver

```
<bean id="viewResolver"
   class="o.s.w.servlet.view.InternalResourceViewResolver">
     <property name="prefix" value="/WEB-INF/jsp/"/>
     <property name="suffix" value=".jsp"/>
</bean>
```

Spring MVC Flow: View Routing

${web-app}/WEB-INF/*${servlet-name}*-servlet.xml

```
<beans>
  <!-- View Resolvers:
       Note: view resolvers are searched in ORDER -->
  <bean id "viewResolverJstl"
    class "org.springframework...InternalResourceViewResolver">
    <property name "viewClass">
      <value>org.springframework.web.servlet.view.JstlView</value>
    </property>
    <property name "prefix">
      <value>/WEB-INF/jsp/</value>
    </property>
    <property name "suffix">
      <value>.jsp</value>
    </property>
  </bean>

  <bean id "viewResolverBeanName"
    class "org.springframework...BeanNameViewResolver" />
</beans>
```

Anecdote: View Routing Search Order

1

```
...
// Did the handler return a view to render?
if (mv != null && !mv.wasCleared()) {
   render(mv, processedRequest, response);
}
else {
   //... assume HandlerAdapter completed
}
...
```

–DispatchController.doDispatch()

–DispatchController

```
protected void render(ModelAndView mv, HttpServletRequest request,
                      HttpServletResponse response) throws Exception {

  // Determine locale for request and apply it to the response.
  Locale locale = this.localeResolver.resolveLocale(request);
  response.setLocale(locale);

  View view = null;

  // Do we need view name translation?
  if (!mv.hasView()) {
    mv.setViewName(getDefaultViewName(request));
  }

  if (mv.isReference()) {
    // We need to resolve the view name.
    view = resolveViewName(mv.getViewName(), mv.getModelInternal(),
    [illegible]
    if (view == null) {
      throw new ServletException("Could not resolve view... ");
    }
  }
  else {
    // No need to lookup: the ModelAndView object contains the
    // actual View object.
    view = mv.getView();
    if (view == null) {
      throw new ServletException "ModelAndView ... neither contains a view
name nor a View object in Servlet... "
    }
  }

  //debug msgs
  view.render(mv.getModelInternal(), request, response);
```

2

```
protected View resolveViewName(String viewName,
                               Map model,
                               Locale locale,
                               HttpServletRequest request)
                               throws Exception {
  for (Iterator it = this.viewResolvers.iterator();
       it.hasNext();) {
    ViewResolver viewResolver =
       (ViewResolver) it.next();
    View view = viewResolver.resolveViewName(viewName,
                                             locale);
    if (view != null) {
      return view;
    }
  }
  return null;
}
```

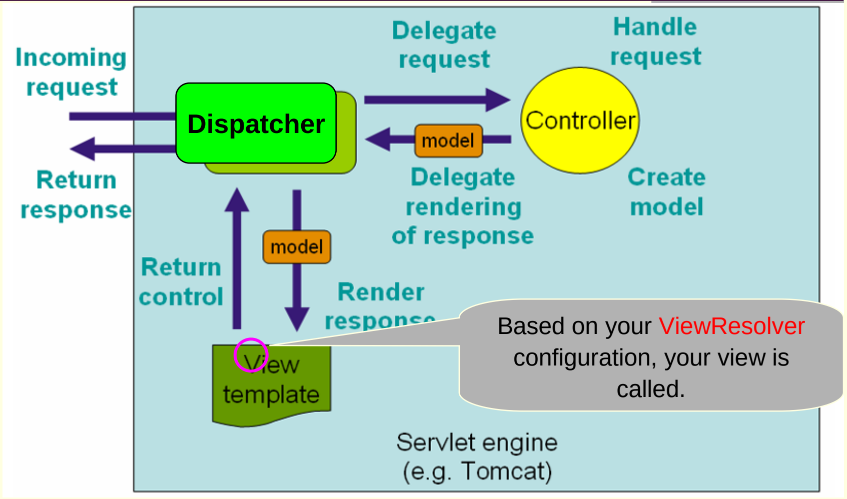
3

–DispatchController

Anecdote: View Routing Search Order

- From Spring 2 Reference Document 13.5.2:
  - "Chaining ViewResolvers: the contract of a view resolver mentions that a view resolver can return null to indicate the view could not be found. Not all view resolvers do this however!
  - … Check the Javadoc for the view resolver to see if you're dealing with a view resolver that does not report non-existing views."

Spring MVC Flow: View



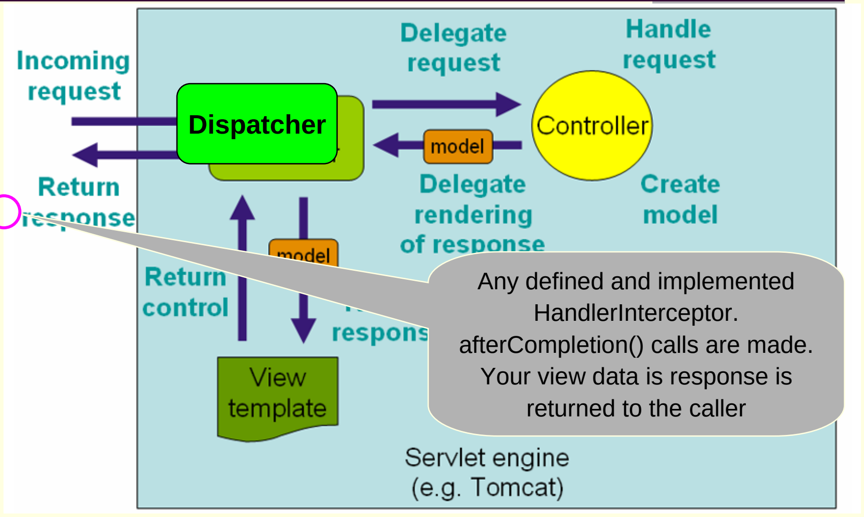
Based on your ViewResolver configuration, your view is called.

From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

Spring MVC Flow: View

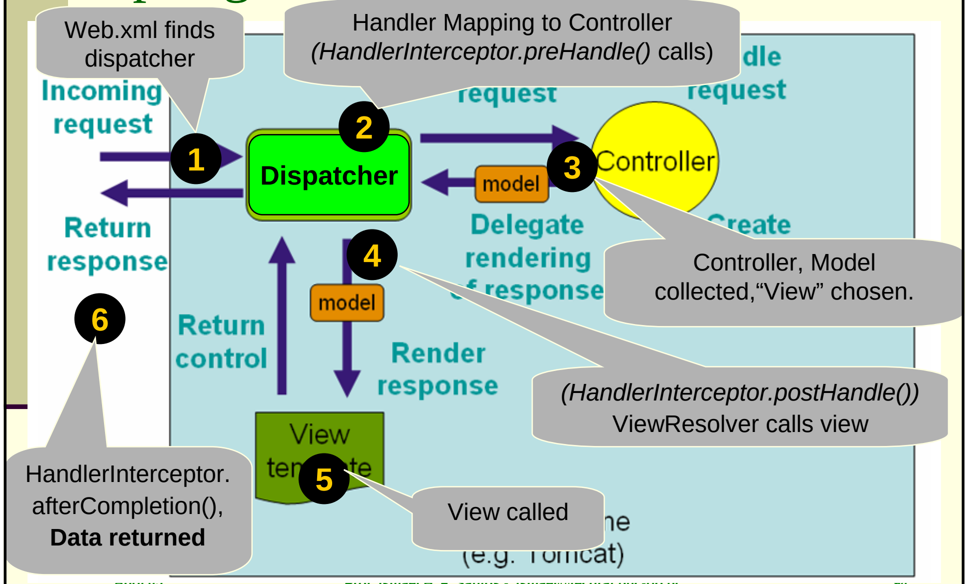
Technology	Typical View Implementations
JSP & JSTL	…web.servlet.view.InternalResourceView …web.servlet.view.JstlView
Tiles (Struts)	…web.servlet.view.tiles.TilesConfigurer
Velocity & FreeMarker	…web.servlet.view.velocity.VelocityViewResolver …web.servlet.view.freemarker.FreeMarkerConfigurer
XSLT	Inherit …view.xslt.AbstractXsltView
Documents (PDF/Excel)	Inherit …view.document.AbstractPdfView Inherit …view.document.AbstractJExcelView
JasperReports	JasperReportsCsvView, JasperReportsHtmlView, JasperReportsPdfView, JasperReportsXlsView, JasperReportsMultiFormatView
Custom Implementations	

Spring MVC Flow: View



Any defined and implemented HandlerInterceptor. afterCompletion() calls are made. Your view data is response is returned to the caller

Spring MVC Run-time Flow



Web.xml finds dispatcher

Handler Mapping to Controller (*HandlerInterceptor.preHandle()* calls)

Controller, Model collected,"View" chosen.

(*HandlerInterceptor.postHandle()*) ViewResolver calls view

View called

HandlerInterceptor. afterCompletion(), **Data returned**

From http://www.springframework.org/docs/reference/mvc.html#mvc-servlet

JSTL Introduction: What is it?

- What is JSTL?
  - JavaServer Pages Standard Tag Library, JSR 52.
  - JSTL provides an effective way to embed logic within a JSP page without using embedded Java code directly.
- Goal
  - Provide the tags to the web page authors, so that they can easily access and manipulate the application data without using the scriptlets (java code).

http://www.roseindia.net/jstl/jstl.shtml

JSTL Introduction: What is it?

- Other facts
  - JSP tags are xml like tags, which can be used by non-programmers (page authors or designers) without knowledge of Java programming.
  - Because JSTL tags are xml, they fit nicely into an XHTML strategy allowing web pages have greater tool vender support (for validation and dynamic data insertion) among other beneficial attributes (such as schema support).
  - Bean driven (for data setting/access)

http://www.roseindia.net/jstl/jstl.shtml

JSTL Introduction: Concepts

- Tag Library
  - Xml compliant structures performing JSP features without knowledge of Java programming.
  - Concepts such as flow control (if, for each) and more…
  - Tags are extendable
- Expression Language
  - Allows former java Scriptlet logic to be simplified into expressions
  - Expression functions are extendable

JSTL Introduction: Before / After

- old

```
<% For (Enumeration e = cart.getProducts();
       e.hasMoreElements();) {
    Product thisProduct = (Product)e.nextElement();
%>
<br />
<%=thisProduct.getDescription()%>
<% } %>
```

- new

```
<c:forEach var="thisProduct" items="${cart.products}">
    <br />
    <p>${thisProduct.description}</p>
</c:forEach>
```

JSTL Introduction: EL Expressions

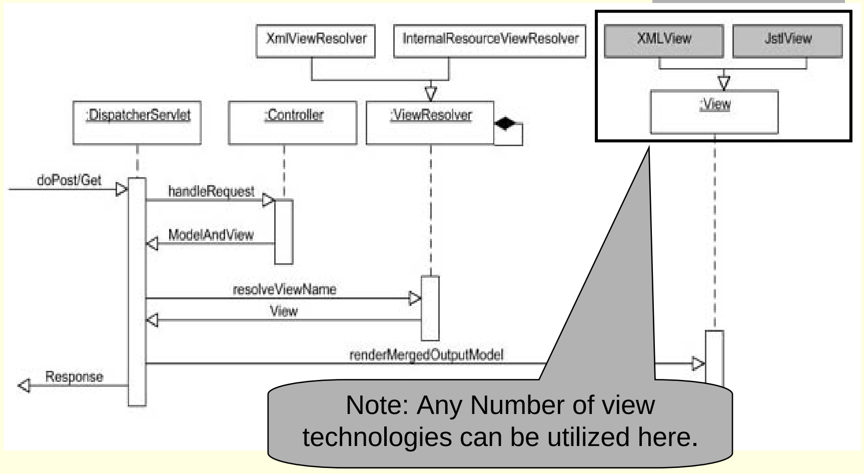
EL Expression	Result
${1 > (4/2)}	false
${4.0 >= 3}	true
${100.0 == 100}	true
${(10*10) ne 100}	false
${'a' < 'b'}	true
${'hip' gt 'hit'}	false
${4 > 3}	true
${1.2E4 + 1.4}	12001.4
${3 div 4}	0.75
${10 mod 4}	2
${empty param.Add}	True if the request parameter named Add is null or an empty string
${pageContext.request.contextPath}	The context path
${sessionScope.cart.numberOfItems}	The value of the numberOfItems property of the session-scoped attribute named cart
${param['mycom.productId']}	The value of the request parameter named mycom.productId
${header["host"]}	The host
${departments[deptName]}	The value of the entry named deptName in the departments map
${requestScope['javax.servlet.forward.servlet_path']}	The value of the request-scoped attribute named javax.servlet.forward.servlet_path

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html#wp77083

JSTL Introduction: Other Resources

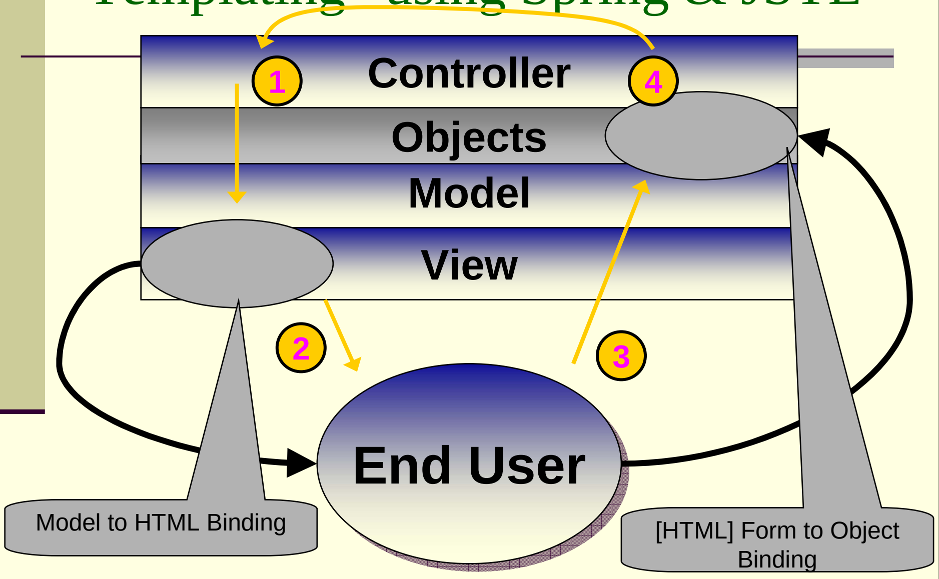
- http://www.sitepoint.com/print/java-standard-tag-library
- http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSTL4.html
- http://en.wikipedia.org/wiki/JSTL
- http://www.roseindia.net/jstl/jstl.shtml
- … lots more! *(one word: Google)*

Spring TagLib: Using JSTL



http://www.theserverside.com/tt/articles/article.tss?l=AjaxandSpring

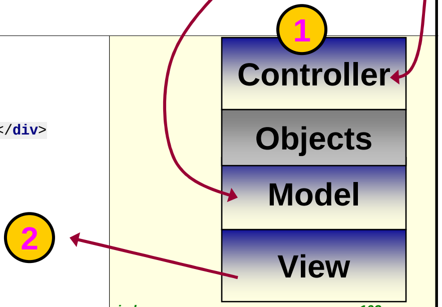
“Templating” using Spring & JSTL



Model to HTML Binding

```
...
public class ProductController implements Controller {
  private final static String DEFAULT_VIEW = "hello";
  public ModelAndView handleRequest(HttpServletRequest request,
                                    HttpServletResponse response)
      throws ServletException, IOException {
      String now = (new java.util.Date()).toString();
      ...
      logger.info("returning hello view with " + now);
      Map<String, Object> myModel = new HashMap<String, Object>();
      myModel.put(NOW_KEY, now);
      myModel.put(PRODUCTS_MGR_KEY,
                  getProductManager().getProducts());
      return new ModelAndView(DEFAULT_VIEW, "model", myModel);
  }
}
...
```

```
<html xmlns="http://www.w3.org/1999/xhtml" ...>
...
<div id="products">
<table>
  <tr class="TableHeadingColor">
    <td class="TableHeadingColor">
      <fmt:message key="productHeading" />
    </td>
    <td><fmt:message key="priceHeading" /></td>
  </tr>
  <c:set var="count" value="0"></c:set>
  <c:forEach items="${model.products}" var="prod">
    <tr>
      <td>
        <div id="productCount-${count}">${prod.description}</div>
      </td>
      <td>
        <div id="priceCount-${count}">$ ${prod.price}</div>
      </td>
    </tr>
    <c:set var="count" value="${count + 1}"></c:set>
  </c:forEach>
</table>
</div>
...
</body>
</html>
```



String Externalization in JSTL

- I18N functionality in JSTL is provided by the “fmt” tag
  - Locale and resource bundles

<fmt:setlocale>	<fmt:message>
<fmt:bundle>	<fmt:param>
<fmt:setBundle>	<fmt:requestEncoding>

  - Formatting for numbers, dates, and currency

<fmt:timeZone>	<fmt:parseNumber>
<fmt:setTimezone>	<fmt:formatDate>
<fmt:formatNumber>	<fmt:parseDate>

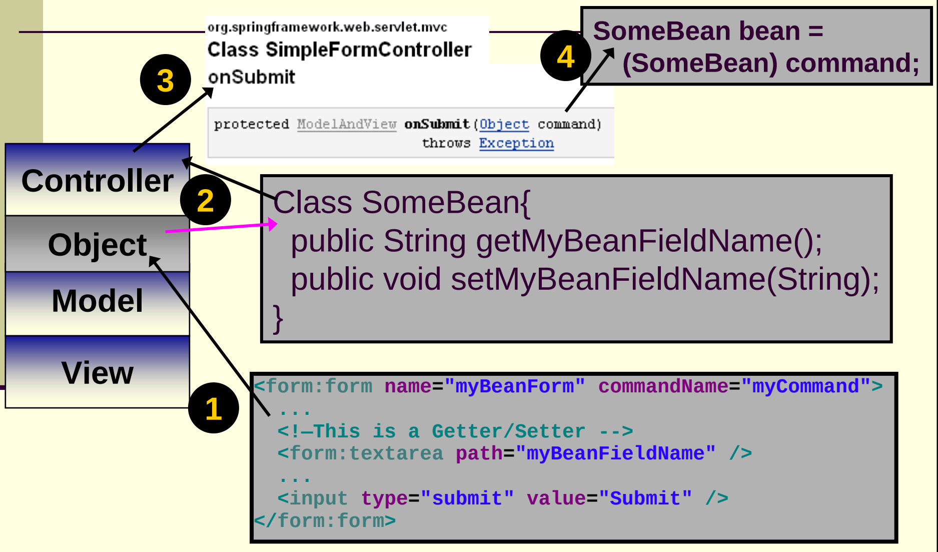
http://www.javaranch.com/newsletter/200309/AnIntroductionToJstl.html

String Externalization in JSTL

- Spring enhances these I18N support tags with it’s own <Spring:message> tag
  - MessageSource integrated with Spring context
    - Define where your messages are stored
  - Works with the locale support that comes with Spring
    - Spring gives more flexibility in how the locale for messages is resolved… fmt:message tag uses request.getLocale() to establish the locale for messages

http://forum.springframework.org/archive/index.php/t-10454.html

Form to Object Binding



org.springframework.web.servlet.mvc
Class SimpleFormController
onSubmit

```
protected ModelAndView onSubmit(Object command)
                throws Exception
```

```
SomeBean bean =
  (SomeBean) command;
```

```
Class SomeBean{
  public String getMyBeanFieldName();
  public void setMyBeanFieldName(String);
}
```

```
<form:form name="myBeanForm" commandName="myCommand">
  ...
  <!—This is a Getter/Setter -->
  <form:textarea path="myBeanFieldName" />
  ...
  <input type="submit" value="Submit" />
</form:form>
```

Spring TagLib

- Starting in Spring 2, a standard JSTL Tag Library was added to help simplify form-based web pages
  - form
    - input
    - checkbox
    - radiobox
    - password
    - select
    - option
    - options
    - textarea
    - hidden
    - errors

Spring TagLib: Binding Forms

Sample JSP

```
<form:form name="regexpTestForm" commandName="regexpTest">
  <table width="600" bgcolor="f8f8ff" border="0" cellspacing="0"
         cellpadding="5">
    <tr>
      <td align="left" width="20">Regular Expression:</td>
      <td align="left" width="580"><form:textarea path="regexp"
         rows="4" cols="100" title="${msgregexpTitle}" /></td>
      <td><form:errors path="regexp" /></td>
    </tr>
    <tr>
      <td align="left" width="20">Sample Data:</td
      <td align="left" width="580"><form:textarea
         rows="10" cols="100" title="test me" /><
      <td><form:errors path="testData" /></td>
    </tr>
    <tr>
      <td colspan="2" align="center" width="100">
        <div><input type="submit" align="middle"
                    value="Test Regexp and Sample
      </td>
    </tr>
  </table>
</form:form>
```

```
<!-- Controllers -->
<bean id="regexpTesterForm"
  class="...my*FormControllerImp">
  ...
  <property name="commandName">
    <value>regexpTest</value>
  </property>
  <property name="commandClass">
    <value>MyBean</value>
  </property>
  <property name="formView">
    <value>regexpTester</value>
  </property>
  <property name="successView">
    <value>regexpTester</value>
  </property>
  ...
</bean>
```

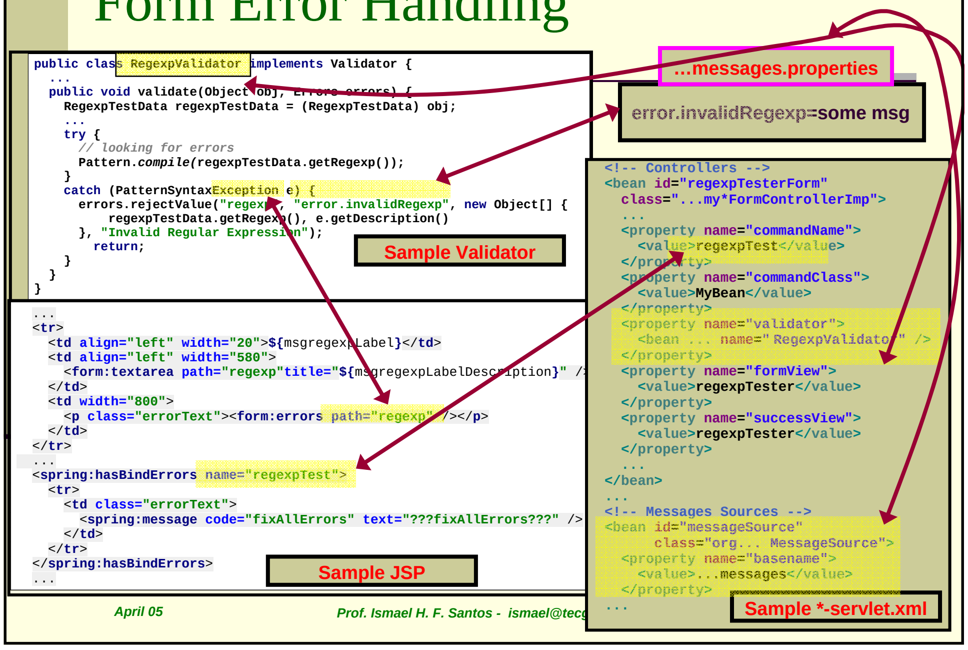
Sample *-servlet.xml

http://forum.springframework.org/show

Form Error Handling

- Built-in workflow for form error management
  - Servers side Validator(s)
    - Identifiers errors at a “field” level
    - Leverages Message keys for Locale support
  - JSTL tags
    - <spring:hasBindErrors name=“commandObjName">
    - <form:errors path=“myBeanFieldName" />

Form Error Handling



```java
public class RegexpValidator implements Validator {
  ...
  public void validate(Object obj, Errors errors) {
    RegexpTestData regexpTestData = (RegexpTestData) obj;
    ...
    try {
      // looking for errors
      Pattern.compile(regexpTestData.getRegexp());
    }
    catch (PatternSyntaxException e) {
      errors.rejectValue("regexp", "error.invalidRegexp", new Object[] {
          regexpTestData.getRegexp(), e.getDescription()
      }, "Invalid Regular Expression");
        return;
    }
  }
}
```

```
error.invalidRegexp=some msg
```

```jsp
...
<tr>
  <td align="left" width="20">${msgregexpLabel}</td>
  <td align="left" width="580">
    <form:textarea path="regexp"title="${msgregexpLabelDescription}" />
  </td>
  <td width="800">
    <p class="errorText"><form:errors path="regexp" /></p>
  </td>
</tr>
...
<spring:hasBindErrors name="regexpTest">
  <tr>
    <td class="errorText">
      <spring:message code="fixAllErrors" text="???fixAllErrors???" />
    </td>
  </tr>
</spring:hasBindErrors>
...
```

```xml
<!-- Controllers -->
<bean id="regexpTesterForm"
  class="...my*FormControllerImp">
  ...
  <property name="commandName">
    <value>regexpTest</value>
  </property>
  <property name="commandClass">
    <value>MyBean</value>
  </property>
  <property name="validator">
    <bean ... name=" RegexpValidator" />
  </property>
  <property name="formView">
    <value>regexpTester</value>
  </property>
  <property name="successView">
    <value>regexpTester</value>
  </property>
  ...
</bean>
...
<!-- Messages Sources -->
<bean id="messageSource"
      class="org... MessageSource">
  <property name="basename">
    <value>...messages</value>
  </property>
...
```

Themes and Localization

- Themes
  - a collection of static resources affecting the visual style of the application, typically style sheets and images.
- Localization
  - DispatcherServlet enables you to automatically resolve messages using the client's locale via the LocaleResolver

http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html#mvc-localeresolver
http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html#mvc-themeresolver

Theme and Locale: Resolver and Intercepter

Class	Description
Fixed*Resolver	Selects a fixed theme/locale, set using the "default[Theme|Locale]Name" property.
Session*Resolver	The theme/locale is maintained in the users HTTP session. It only needs to be set once for each session, but is not persisted between sessions.
Cookie*Resolver	The selected theme/locale is stored in a cookie on the user-agent's machine.

```
<bean id="themeChangeInterceptor"
        class="org...ThemeChangeInterceptor" >
 <property name="paramName“name=“theme“ >
</bean>
```

```
<bean id="localeChangeIntercepter"
        class="org... LocaleChangeInterceptor" >
 <property name="paramName“name=“locale“ >
</bean>
```

http://.../test.htm?theme=mello-yellow&locale=en_US

Theme Management Configuration

${web-app}/WEB-INF/*${servlet-name}*-servlet.xml

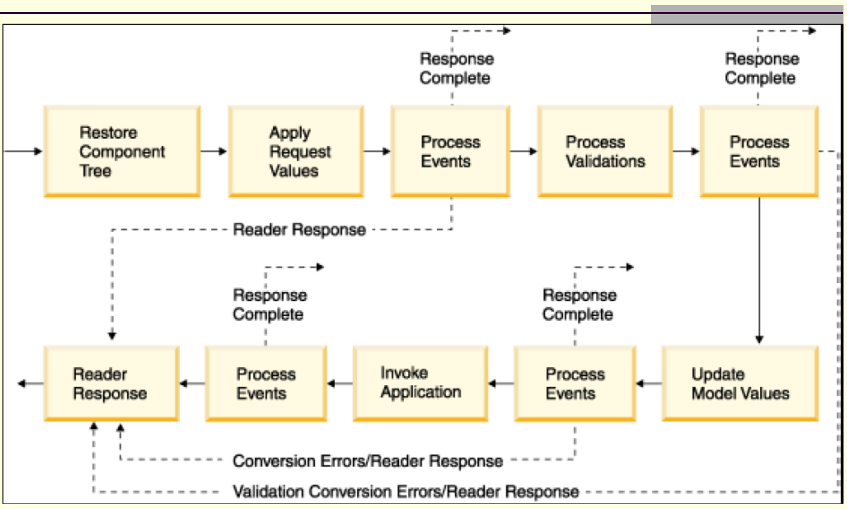
```
…
<!-- Handler Mappings -->
 <bean id "urlMapping"
   class "org.springframework...SimpleUrlHandlerMapping">
   <property name "mappings">
     <props>
       <prop key "regexpTester.htm">regexpTesterForm</prop>
     </props>
   </property>
   <property name "interceptors">
     <list>
       <ref local "themeChangeInterceptor" />
     </list>
   </property>
 </bean>
 ...
 <!-- Theme definitions -->
 <bean id "themeResolver"
   class "org.springframework...theme.SessionThemeResolver">
   <property name "defaultThemeName" value "mello-yellow" />
 </bean>
 <bean id "themeChangeInterceptor"
   class "org.springframework...theme.ThemeChangeInterceptor" />

 </bean>
…
```

FIM

Java Server Faces

- JSR 127 – padrão oficial (27/05/2004);
  - Várias implementações;
  - Garantia de continuidade.
- Similar aos frameworks MVC;
- Foco no desenvolvedor:
  - Projetado para ser utilizado por IDEs;
  - Componentes UI extensíveis;
  - Tratamento de eventos (como no Swing!);
  - Suporte à navegação simples.

Ciclo de Vida de uma aplicação JSF


Response Complete
Response Complete
Restore Component Tree
Apply Request Values
Process Events
Process Validations
Process Events
Reader Response
Response Complete
Response Complete
Reader Response
Process Events
Invoke Application
Process Events
Update Model Values
Conversion Errors/Reader Response
Validation Conversion Errors/Reader Response