org.springframework.web.servlet

Interface View

All Known Subinterfaces:

SmartView

All Known Implementing Classes:

AbstractAtomFeedView, AbstractExcelView, AbstractFeedView, AbstractJackson2View, AbstractJasperReportsSingleFormatView, AbstractJasperReportsView, AbstractJExcelView, AbstractPdfStamperView, AbstractPdfView, AbstractRssFeedView, AbstractTemplateView, AbstractUrlBasedView, AbstractView, AbstractXlsView, AbstractXlsxStreamingView, AbstractXlsxView, ConfigurableJasperReportsView, FreeMarkerView, GroovyMarkupView, InternalResourceView, JasperReportsCsvView, JasperReportsHtmlView, JasperReportsMultiFormatView, JasperReportsPdfView, JasperReportsXlsView, JasperReportsXlsxView, JstlView, MappingJackson2JsonView, MappingJackson2XmlView, MarshallingView, RedirectView, ScriptTemplateView, TilesView, TilesView, VelocityLayoutView, VelocityToolboxView, VelocityView, XsltView

---

```
public interface View
```

MVC View for a web interaction. Implementations are responsible for rendering content, and exposing the model. A single view exposes multiple model attributes.

This class and the MVC approach associated with it is discussed in Chapter 12 of Expert One-On-One J2EE Design and Development by Rod Johnson (Wrox, 2002).

View implementations may differ widely. An obvious implementation would be JSP-based. Other implementations might be XSLT-based, or use an HTML generation library. This interface is designed to avoid restricting the range of possible implementations.

Views should be beans. They are likely to be instantiated as beans by a ViewResolver. As this interface is stateless, view implementations should be thread-safe.

Author:

Rod Johnson, Arjen Poutsma

See Also:

AbstractView, InternalResourceView

Field Summary

Fields

Modifier and Type	Field and Description
static String	**PATH_VARIABLES** Name of the **HttpServletRequest** attribute that contains a Map with path variables.
static String	**RESPONSE_STATUS_ATTRIBUTE** Name of the **HttpServletRequest** attribute that contains the response status code.
static String	**SELECTED_CONTENT_TYPE** The **MediaType** selected during content negotiation, which may be more specific than the one the View is configured with.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
String	**getContentType**() Return the content type of the view, if predetermined.
void	**render**(Map<String,?> model, HttpServletRequest request, HttpServletResponse response) Render the view given the specified model.

Field Detail

RESPONSE_STATUS_ATTRIBUTE

```
static final String RESPONSE_STATUS_ATTRIBUTE
```

Name of the HttpServletRequest attribute that contains the response status code.

Note: This attribute is not required to be supported by all View implementations.

PATH_VARIABLES

```
static final String PATH_VARIABLES
```

Name of the HttpServletRequest attribute that contains a Map with path variables. The map consists of String-based URI template variable names as keys and their corresponding Object-based values -- extracted from segments of the URL and type converted.

Note: This attribute is not required to be supported by all View implementations.

SELECTED_CONTENT_TYPE

```
static final String SELECTED_CONTENT_TYPE
```

The MediaType selected during content negotiation, which may be more specific than the one the View is configured with. For example: "application/vnd.example-v1+xml" vs "application/*+xml".

Method Detail

getContentType

```
String getContentType()
```

Return the content type of the view, if predetermined.

Can be used to check the content type upfront, before the actual rendering process.

Returns:

the content type String (optionally including a character set), or null if not predetermined.

render

```
void render(Map<String,?> model,
            HttpServletRequest request,
            HttpServletResponse response)
     throws Exception
```

Render the view given the specified model.

The first step will be preparing the request: In the JSP case, this would mean setting model objects as request attributes. The second step will be the actual rendering of the view, for example including the JSP via a RequestDispatcher.

Parameters:

model - Map with name Strings as keys and corresponding model objects as values (Map can also be null in case of empty model)

request - current HTTP request

response - HTTP response we are building

Throws:

Exception - if rendering failed

What is a View and what's the idea behind supporting different types of View?

View is the base interface for different view technologies (XML, PDF, XLS etc.) Model data is passed to the view and rendered for providing different appearance. By passing the model data from controller to the view, a view resolver is required, which also has different types, that is different resolvers for different view types.