org.springframework.web.servlet

Class DispatcherServlet

java.lang.Object
  javax.servlet.GenericServlet
    javax.servlet.http.HttpServlet
      org.springframework.web.servlet.HttpServletBean
        org.springframework.web.servlet.FrameworkServlet
          org.springframework.web.servlet.DispatcherServlet

All Implemented Interfaces:

`Serializable`, `Servlet`, `ServletConfig`, `Aware`, `ApplicationContextAware`, `EnvironmentAware`, `EnvironmentCapable`

```
public class DispatcherServlet
extends FrameworkServlet
```

Central dispatcher for HTTP request handlers/controllers, e.g. for web UI controllers or HTTP-based remote service exporters. Dispatches to registered handlers for processing a web request, providing convenient mapping and exception handling facilities.

This servlet is very flexible: It can be used with just about any workflow, with the installation of the appropriate adapter classes. It offers the following functionality that distinguishes it from other request-driven web MVC frameworks:

- It is based around a JavaBeans configuration mechanism.
- It can use any `HandlerMapping` implementation - pre-built or provided as part of an application - to control the routing of requests to handler objects. Default is `BeanNameUrlHandlerMapping` and `DefaultAnnotationHandlerMapping`. HandlerMapping objects can be defined as beans in the servlet's application context, implementing the HandlerMapping interface, overriding the default HandlerMapping if present. HandlerMappings can be given any bean name (they are tested by type).
- It can use any `HandlerAdapter`; this allows for using any handler interface. Default adapters are `HttpRequestHandlerAdapter`, `SimpleControllerHandlerAdapter`, for Spring's `HttpRequestHandler` and `Controller` interfaces, respectively. A default `AnnotationMethodHandlerAdapter` will be registered as well. HandlerAdapter objects can be added as beans in the application context, overriding the default HandlerAdapters. Like HandlerMappings, HandlerAdapters can be given any bean name (they are tested by type).
- The dispatcher's exception resolution strategy can be specified via a `HandlerExceptionResolver`, for example mapping certain exceptions to error pages. Default are `AnnotationMethodHandlerExceptionResolver`, `ResponseStatusExceptionResolver`, and `DefaultHandlerExceptionResolver`. These HandlerExceptionResolvers can be overridden through the application context. HandlerExceptionResolver can be given any bean name (they are tested by type).
- Its view resolution strategy can be specified via a `ViewResolver` implementation, resolving symbolic view names into View objects. Default is `InternalResourceViewResolver`. ViewResolver objects can be added as beans in the application context, overriding the default ViewResolver. ViewResolvers can be given any bean name (they are tested by type).
- If a `View` or view name is not supplied by the user, then the configured `RequestToViewNameTranslator` will translate the current request into a view name. The corresponding bean name is "viewNameTranslator"; the default is `DefaultRequestToViewNameTranslator`.
- The dispatcher's strategy for resolving multipart requests is determined by a `MultipartResolver` implementation. Implementations for Apache Commons FileUpload and Servlet 3 are included; the typical choice is `CommonsMultipartResolver`. The MultipartResolver bean name is "multipartResolver"; default is none.
- Its locale resolution strategy is determined by a `LocaleResolver`. Out-of-the-box implementations work via HTTP accept header, cookie, or session. The LocaleResolver bean name is "localeResolver"; default is `AcceptHeaderLocaleResolver`.
- Its theme resolution strategy is determined by a `ThemeResolver`. Implementations for a fixed theme and for cookie and session storage are included. The ThemeResolver bean name is "themeResolver"; default is `FixedThemeResolver`.

NOTE: The `@RequestMapping` annotation will only be processed if a corresponding `HandlerMapping` (for type-level annotations) and/or `HandlerAdapter` (for method-level annotations) is present in the dispatcher. This is the case by default. However, if you are defining custom `HandlerMappings` or `HandlerAdapters`, then you need to make sure that a corresponding custom `DefaultAnnotationHandlerMapping` and/or `AnnotationMethodHandlerAdapter` is defined as well - provided that you intend to use `@RequestMapping`.

A web application can define any number of DispatcherServlets. Each servlet will operate in its own namespace, loading its own application context with mappings, handlers, etc. Only the root application context as loaded by `ContextLoaderListener`, if any, will be shared.

As of Spring 3.1, `DispatcherServlet` may now be injected with a web application context, rather than creating its own internally. This is useful in Servlet 3.0+ environments, which support programmatic registration of servlet instances. See the `DispatcherServlet(WebApplicationContext)` javadoc for details.

Author:

`Rod Johnson, Juergen Hoeller, Rob Harrop, Chris Beams, Rossen Stoyanchev`

See Also:

`HttpRequestHandler`, `Controller`, `ContextLoaderListener`, `Serialized Form`

Field Summary

Fields

Modifier and Type	Field and Description

static String	**EXCEPTION_ATTRIBUTE** Name of request attribute that exposes an Exception resolved with an **HandlerExceptionResolver** but where no view was rendered (e.g.
static String	**FLASH_MAP_MANAGER_ATTRIBUTE** Name of request attribute that holds the **FlashMapManager**.
static String	**FLASH_MAP_MANAGER_BEAN_NAME** Well-known name for the FlashMapManager object in the bean factory for this namespace.
static String	**HANDLER_ADAPTER_BEAN_NAME** Well-known name for the HandlerAdapter object in the bean factory for this namespace.
static String	**HANDLER_EXCEPTION_RESOLVER_BEAN_NAME** Well-known name for the HandlerExceptionResolver object in the bean factory for this namespace.
static String	**HANDLER_MAPPING_BEAN_NAME** Well-known name for the HandlerMapping object in the bean factory for this namespace.
static String	**INPUT_FLASH_MAP_ATTRIBUTE** Name of request attribute that holds a read-only `Map<String,?>` with "input" flash attributes saved by a previous request, if any.
static String	**LOCALE_RESOLVER_ATTRIBUTE** Request attribute to hold the current LocaleResolver, retrievable by views.
static String	**LOCALE_RESOLVER_BEAN_NAME** Well-known name for the LocaleResolver object in the bean factory for this namespace.
static String	**MULTIPART_RESOLVER_BEAN_NAME** Well-known name for the MultipartResolver object in the bean factory for this namespace.
static String	**OUTPUT_FLASH_MAP_ATTRIBUTE** Name of request attribute that holds the "output" **FlashMap** with attributes to save for a subsequent request.
static String	**PAGE_NOT_FOUND_LOG_CATEGORY** Log category to use when no mapped handler is found for a request.
protected static Log	**pageNotFoundLogger** Additional logger to use when no mapped handler is found for a request.
static String	**REQUEST_TO_VIEW_NAME_TRANSLATOR_BEAN_NAME** Well-known name for the RequestToViewNameTranslator object in the bean factory for this namespace.
static String	**THEME_RESOLVER_ATTRIBUTE** Request attribute to hold the current ThemeResolver, retrievable by views.
static String	**THEME_RESOLVER_BEAN_NAME** Well-known name for the ThemeResolver object in the bean factory for this namespace.
static String	**THEME_SOURCE_ATTRIBUTE** Request attribute to hold the current ThemeSource, retrievable by views.
static String	**VIEW_RESOLVER_BEAN_NAME** Well-known name for the ViewResolver object in the bean factory for this namespace.
static String	**WEB_APPLICATION_CONTEXT_ATTRIBUTE** Request attribute to hold the current web application context.

Fields inherited from class org.springframework.web.servlet.FrameworkServlet

DEFAULT_CONTEXT_CLASS, DEFAULT_NAMESPACE_SUFFIX, SERVLET_CONTEXT_PREFIX

Fields inherited from class org.springframework.web.servlet.HttpServletBean

logger

Constructor Summary

Constructors

Constructor and Description
DispatcherServlet()

Create a new `DispatcherServlet` that will create its own internal web application context based on defaults and values provided through servlet init-params.

`DispatcherServlet(WebApplicationContext webApplicationContext)`
Create a new `DispatcherServlet` with the given web application context.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
protected `LocaleContext`	**`buildLocaleContext(HttpServletRequest request)`** Build a LocaleContext for the given request, exposing the request's primary locale as current locale.
protected `HttpServletRequest`	**`checkMultipart(HttpServletRequest request)`** Convert the request into a multipart request, and make multipart resolver available.
protected void	**`cleanupMultipart(HttpServletRequest request)`** Clean up any resources used by the given multipart request (if any).
protected `Object`	**`createDefaultStrategy(ApplicationContext context, Class<?> clazz)`** Create a default strategy.
protected void	**`doDispatch(HttpServletRequest request, HttpServletResponse response)`** Process the actual dispatching to the handler.
protected void	**`doService(HttpServletRequest request, HttpServletResponse response)`** Exposes the DispatcherServlet-specific request attributes and delegates to **`doDispatch(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)`** for the actual dispatching.
protected <T> `List<T>`	**`getDefaultStrategies(ApplicationContext context, Class<T> strategyInterface)`** Create a List of default strategy objects for the given strategy interface.
protected <T> T	**`getDefaultStrategy(ApplicationContext context, Class<T> strategyInterface)`** Return the default strategy object for the given strategy interface.
protected `String`	**`getDefaultViewName(HttpServletRequest request)`** Translate the supplied request into a default view name.
protected `HandlerExecutionChain`	**`getHandler(HttpServletRequest request)`** Return the HandlerExecutionChain for this request.
protected `HandlerAdapter`	**`getHandlerAdapter(Object handler)`** Return the HandlerAdapter for this handler object.
`MultipartResolver`	**`getMultipartResolver()`** Obtain this servlet's MultipartResolver, if any.
`ThemeSource`	**`getThemeSource()`** Return this servlet's ThemeSource, if any; else return `null`.
protected void	**`initStrategies(ApplicationContext context)`** Initialize the strategy objects that this servlet uses.
protected void	**`noHandlerFound(HttpServletRequest request, HttpServletResponse response)`** No handler found -> set appropriate HTTP response status.
protected void	**`onRefresh(ApplicationContext context)`** This implementation calls **`initStrategies(org.springframework.context.ApplicationContext)`**.
protected `ModelAndView`	**`processHandlerException(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)`** Determine an error ModelAndView via the registered HandlerExceptionResolvers.
protected void	**`render(ModelAndView mv, HttpServletRequest request, HttpServletResponse response)`** Render the given ModelAndView.
protected `View`	**`resolveViewName(String viewName, Map<String,Object> model, Locale locale, HttpServletRequest request)`** Resolve the given view name into a View object (to be rendered).
void	**`setCleanupAfterInclude(boolean cleanupAfterInclude)`** Set whether to perform cleanup of request attributes after an include request, that is, whether to reset the original state of all request attributes after the DispatcherServlet has processed within an include request.

void	**setDetectAllHandlerAdapters**(boolean detectAllHandlerAdapters) Set whether to detect all HandlerAdapter beans in this servlet's context.
void	**setDetectAllHandlerExceptionResolvers**(boolean detectAllHandlerExceptionResolvers) Set whether to detect all HandlerExceptionResolver beans in this servlet's context.
void	**setDetectAllHandlerMappings**(boolean detectAllHandlerMappings) Set whether to detect all HandlerMapping beans in this servlet's context.
void	**setDetectAllViewResolvers**(boolean detectAllViewResolvers) Set whether to detect all ViewResolver beans in this servlet's context.
void	**setThrowExceptionIfNoHandlerFound**(boolean throwExceptionIfNoHandlerFound) Set whether to throw a NoHandlerFoundException when no Handler was found for this request.

Methods inherited from class org.springframework.web.servlet.FrameworkServlet

applyInitializers, buildRequestAttributes, configureAndRefreshWebApplicationContext, createWebApplicationContext, createWebApplicationContext, destroy, doDelete, doGet, doOptions, doPost, doPut, doTrace, findWebApplicationContext, getContextAttribute, getContextClass, getContextConfigLocation, getContextId, getNamespace, getServletContextAttributeName, getUsernameForRequest, getWebApplicationContext, initFrameworkServlet, initServletBean, initWebApplicationContext, onApplicationEvent, postProcessWebApplicationContext, processRequest, refresh, service, setApplicationContext, setContextAttribute, setContextClass, setContextConfigLocation, setContextId, setContextInitializerClasses, setContextInitializers, setDispatchOptionsRequest, setDispatchTraceRequest, setNamespace, setPublishContext, setPublishEvents, setThreadContextInheritable

Methods inherited from class org.springframework.web.servlet.HttpServletBean

addRequiredProperty, createEnvironment, getEnvironment, getServletContext, getServletName, init, initBeanWrapper, setEnvironment

Methods inherited from class javax.servlet.http.HttpServlet

doHead, getLastModified, service

Methods inherited from class javax.servlet.GenericServlet

getInitParameter, getInitParameterNames, getServletConfig, getServletInfo, init, log, log

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

MULTIPART_RESOLVER_BEAN_NAME

```
public static final String MULTIPART_RESOLVER_BEAN_NAME
```

Well-known name for the MultipartResolver object in the bean factory for this namespace.

See Also:
Constant Field Values

LOCALE_RESOLVER_BEAN_NAME

```
public static final String LOCALE_RESOLVER_BEAN_NAME
```

Well-known name for the LocaleResolver object in the bean factory for this namespace.

See Also:
Constant Field Values

THEME_RESOLVER_BEAN_NAME

```
public static final String THEME_RESOLVER_BEAN_NAME
```

Well-known name for the ThemeResolver object in the bean factory for this namespace.

See Also:
Constant Field Values

HANDLER_MAPPING_BEAN_NAME

```
public static final String HANDLER_MAPPING_BEAN_NAME
```

Well-known name for the HandlerMapping object in the bean factory for this namespace. Only used when "detectAllHandlerMappings" is turned off.

See Also:

`setDetectAllHandlerMappings(boolean)`, `Constant Field Values`

HANDLER_ADAPTER_BEAN_NAME

```
public static final String HANDLER_ADAPTER_BEAN_NAME
```

Well-known name for the HandlerAdapter object in the bean factory for this namespace. Only used when "detectAllHandlerAdapters" is turned off.

See Also:

`setDetectAllHandlerAdapters(boolean)`, `Constant Field Values`

HANDLER_EXCEPTION_RESOLVER_BEAN_NAME

```
public static final String HANDLER_EXCEPTION_RESOLVER_BEAN_NAME
```

Well-known name for the HandlerExceptionResolver object in the bean factory for this namespace. Only used when "detectAllHandlerExceptionResolvers" is turned off.

See Also:

`setDetectAllHandlerExceptionResolvers(boolean)`, `Constant Field Values`

REQUEST_TO_VIEW_NAME_TRANSLATOR_BEAN_NAME

```
public static final String REQUEST_TO_VIEW_NAME_TRANSLATOR_BEAN_NAME
```

Well-known name for the RequestToViewNameTranslator object in the bean factory for this namespace.

See Also:

`Constant Field Values`

VIEW_RESOLVER_BEAN_NAME

```
public static final String VIEW_RESOLVER_BEAN_NAME
```

Well-known name for the ViewResolver object in the bean factory for this namespace. Only used when "detectAllViewResolvers" is turned off.

See Also:

`setDetectAllViewResolvers(boolean)`, `Constant Field Values`

FLASH_MAP_MANAGER_BEAN_NAME

```
public static final String FLASH_MAP_MANAGER_BEAN_NAME
```

Well-known name for the FlashMapManager object in the bean factory for this namespace.

See Also:

`Constant Field Values`

WEB_APPLICATION_CONTEXT_ATTRIBUTE

```
public static final String WEB_APPLICATION_CONTEXT_ATTRIBUTE
```

Request attribute to hold the current web application context. Otherwise only the global web app context is obtainable by tags etc.

See Also:

`RequestContextUtils.findWebApplicationContext(javax.servlet.http.HttpServletRequest, javax.servlet.ServletContext)`

LOCALE_RESOLVER_ATTRIBUTE

```
public static final String LOCALE_RESOLVER_ATTRIBUTE
```

Request attribute to hold the current LocaleResolver, retrievable by views.

See Also:
`RequestContextUtils.getLocaleResolver(javax.servlet.http.HttpServletRequest)`

THEME_RESOLVER_ATTRIBUTE

```
public static final String THEME_RESOLVER_ATTRIBUTE
```

Request attribute to hold the current ThemeResolver, retrievable by views.

See Also:
`RequestContextUtils.getThemeResolver(javax.servlet.http.HttpServletRequest)`

THEME_SOURCE_ATTRIBUTE

```
public static final String THEME_SOURCE_ATTRIBUTE
```

Request attribute to hold the current ThemeSource, retrievable by views.

See Also:
`RequestContextUtils.getThemeSource(javax.servlet.http.HttpServletRequest)`

INPUT_FLASH_MAP_ATTRIBUTE

```
public static final String INPUT_FLASH_MAP_ATTRIBUTE
```

Name of request attribute that holds a read-only `Map<String,?>` with "input" flash attributes saved by a previous request, if any.

See Also:
`RequestContextUtils.getInputFlashMap(HttpServletRequest)`

OUTPUT_FLASH_MAP_ATTRIBUTE

```
public static final String OUTPUT_FLASH_MAP_ATTRIBUTE
```

Name of request attribute that holds the "output" `FlashMap` with attributes to save for a subsequent request.

See Also:
`RequestContextUtils.getOutputFlashMap(HttpServletRequest)`

FLASH_MAP_MANAGER_ATTRIBUTE

```
public static final String FLASH_MAP_MANAGER_ATTRIBUTE
```

Name of request attribute that holds the `FlashMapManager`.

See Also:
`RequestContextUtils.getFlashMapManager(HttpServletRequest)`

EXCEPTION_ATTRIBUTE

```
public static final String EXCEPTION_ATTRIBUTE
```

Name of request attribute that exposes an Exception resolved with an `HandlerExceptionResolver` but where no view was rendered (e.g. setting the status code).

PAGE_NOT_FOUND_LOG_CATEGORY

```
public static final String PAGE_NOT_FOUND_LOG_CATEGORY
```

Log category to use when no mapped handler is found for a request.

See Also:
`Constant Field Values`

pageNotFoundLogger

```
protected static final Log pageNotFoundLogger
```

Additional logger to use when no mapped handler is found for a request.

Constructor Detail

DispatcherServlet

```
public DispatcherServlet()
```

Create a new `DispatcherServlet` that will create its own internal web application context based on defaults and values provided through servlet init-params. Typically used in Servlet 2.5 or earlier environments, where the only option for servlet registration is through `web.xml` which requires the use of a no-arg constructor.

Calling `FrameworkServlet.setContextConfigLocation(java.lang.String)` (init-param 'contextConfigLocation') will dictate which XML files will be loaded by the default XmlWebApplicationContext

Calling `FrameworkServlet.setContextClass(java.lang.Class<?>)` (init-param 'contextClass') overrides the default `XmlWebApplicationContext` and allows for specifying an alternative class, such as `AnnotationConfigWebApplicationContext`.

Calling `FrameworkServlet.setContextInitializerClasses(java.lang.String)` (init-param 'contextInitializerClasses') indicates which `ApplicationContextInitializer` classes should be used to further configure the internal application context prior to refresh().

See Also:

`DispatcherServlet(WebApplicationContext)`

DispatcherServlet

```
public DispatcherServlet(WebApplicationContext webApplicationContext)
```

Create a new `DispatcherServlet` with the given web application context. This constructor is useful in Servlet 3.0+ environments where instance-based registration of servlets is possible through the `ServletContext.addServlet(java.lang.String, java.lang.String)` API.

Using this constructor indicates that the following properties / init-params will be ignored:

- `FrameworkServlet.setContextClass(Class)` / 'contextClass'
- `FrameworkServlet.setContextConfigLocation(String)` / 'contextConfigLocation'
- `FrameworkServlet.setContextAttribute(String)` / 'contextAttribute'
- `FrameworkServlet.setNamespace(String)` / 'namespace'

The given web application context may or may not yet be refreshed. If it has **not** already been refreshed (the recommended approach), then the following will occur:

- If the given context does not already have a parent, the root application context will be set as the parent.
- If the given context has not already been assigned an id, one will be assigned to it
- `ServletContext` and `ServletConfig` objects will be delegated to the application context
- `FrameworkServlet.postProcessWebApplicationContext(org.springframework.web.context.ConfigurableWebApplicationContext)` will be called
- Any `ApplicationContextInitializer`s specified through the "contextInitializerClasses" init-param or through the `FrameworkServlet.setContextInitializers(org.springframework.context.ApplicationContextInitializer<?>...)` property will be applied.
- `refresh()` will be called if the context implements `ConfigurableApplicationContext`

If the context has already been refreshed, none of the above will occur, under the assumption that the user has performed these actions (or not) per their specific needs.

See `WebApplicationInitializer` for usage examples.

Parameters:

`webApplicationContext - the context to use`

See Also:

`FrameworkServlet.initWebApplicationContext()`, `FrameworkServlet.configureAndRefreshWebApplicationContext(org.springframework.web.context.ConfigurableWebApplicationContext)`, `WebApplicationInitializer`

Method Detail

setDetectAllHandlerMappings

```
public void setDetectAllHandlerMappings(boolean detectAllHandlerMappings)
```

Set whether to detect all HandlerMapping beans in this servlet's context. Otherwise, just a single bean with name "handlerMapping" will be expected.

Default is "true". Turn this off if you want this servlet to use a single HandlerMapping, despite multiple HandlerMapping beans being defined in the context.

setDetectAllHandlerAdapters

```
public void setDetectAllHandlerAdapters(boolean detectAllHandlerAdapters)
```

Set whether to detect all HandlerAdapter beans in this servlet's context. Otherwise, just a single bean with name "handlerAdapter" will be expected.

Default is "true". Turn this off if you want this servlet to use a single HandlerAdapter, despite multiple HandlerAdapter beans being defined in the context.

setDetectAllHandlerExceptionResolvers

```
public void setDetectAllHandlerExceptionResolvers(boolean detectAllHandlerExceptionResolvers)
```

Set whether to detect all HandlerExceptionResolver beans in this servlet's context. Otherwise, just a single bean with name "handlerExceptionResolver" will be expected.

Default is "true". Turn this off if you want this servlet to use a single HandlerExceptionResolver, despite multiple HandlerExceptionResolver beans being defined in the context.

setDetectAllViewResolvers

```
public void setDetectAllViewResolvers(boolean detectAllViewResolvers)
```

Set whether to detect all ViewResolver beans in this servlet's context. Otherwise, just a single bean with name "viewResolver" will be expected.

Default is "true". Turn this off if you want this servlet to use a single ViewResolver, despite multiple ViewResolver beans being defined in the context.

setThrowExceptionIfNoHandlerFound

```
public void setThrowExceptionIfNoHandlerFound(boolean throwExceptionIfNoHandlerFound)
```

Set whether to throw a NoHandlerFoundException when no Handler was found for this request. This exception can then be caught with a HandlerExceptionResolver or an `@ExceptionHandler` controller method.

Note that if `DefaultServletHttpRequestHandler` is used, then requests will always be forwarded to the default servlet and a NoHandlerFoundException would never be thrown in that case.

Default is "false", meaning the DispatcherServlet sends a NOT_FOUND error through the Servlet response.

Since:

`4.0`

setCleanupAfterInclude

```
public void setCleanupAfterInclude(boolean cleanupAfterInclude)
```

Set whether to perform cleanup of request attributes after an include request, that is, whether to reset the original state of all request attributes after the DispatcherServlet has processed within an include request. Otherwise, just the DispatcherServlet's own request attributes will be reset, but not model attributes for JSPs or special attributes set by views (for example, JSTL's).

Default is "true", which is strongly recommended. Views should not rely on request attributes having been set by (dynamic) includes. This allows JSP views rendered by an included controller to use any model attributes, even with the same names as in the main JSP, without causing side effects. Only turn this off for special needs, for example to deliberately allow main JSPs to access attributes from JSP views rendered by an included controller.

onRefresh

```
protected void onRefresh(ApplicationContext context)
```

This implementation calls `initStrategies(org.springframework.context.ApplicationContext)`.

Overrides:

`onRefresh` in class `FrameworkServlet`

Parameters:

`context` - the current WebApplicationContext

See Also:

`FrameworkServlet.refresh()`

initStrategies

```
protected void initStrategies(ApplicationContext context)
```

Initialize the strategy objects that this servlet uses.

May be overridden in subclasses in order to initialize further strategy objects.

getThemeSource

```
public final ThemeSource getThemeSource()
```

Return this servlet's ThemeSource, if any; else return `null`.

Default is to return the WebApplicationContext as ThemeSource, provided that it implements the ThemeSource interface.

Returns:

the ThemeSource, if any

See Also:

`FrameworkServlet.getWebApplicationContext()`

getMultipartResolver

```
public final MultipartResolver getMultipartResolver()
```

Obtain this servlet's MultipartResolver, if any.

Returns:

the MultipartResolver used by this servlet, or null if none (indicating that no multipart support is available)

getDefaultStrategy

```
protected <T> T getDefaultStrategy(ApplicationContext context,
                                   Class<T> strategyInterface)
```

Return the default strategy object for the given strategy interface.

The default implementation delegates to `getDefaultStrategies(org.springframework.context.ApplicationContext, java.lang.Class<T>)`, expecting a single object in the list.

Parameters:

`context` - the current WebApplicationContext

`strategyInterface` - the strategy interface

Returns:

the corresponding strategy object

See Also:

`getDefaultStrategies(org.springframework.context.ApplicationContext, java.lang.Class<T>)`

getDefaultStrategies

```
protected <T> List<T> getDefaultStrategies(ApplicationContext context,
                                           Class<T> strategyInterface)
```

Create a List of default strategy objects for the given strategy interface.

The default implementation uses the "DispatcherServlet.properties" file (in the same package as the DispatcherServlet class) to determine the class names. It instantiates the strategy objects through the context's BeanFactory.

Parameters:

`context` - the current WebApplicationContext

`strategyInterface` - the strategy interface

Returns:

the List of corresponding strategy objects

createDefaultStrategy

```
protected Object createDefaultStrategy(ApplicationContext context,
                                       Class<?> clazz)
```

Create a default strategy.

The default implementation uses AutowireCapableBeanFactory.createBean(java.lang.Class<T>).

Parameters:

context - the current WebApplicationContext

clazz - the strategy implementation class to instantiate

Returns:

the fully configured strategy instance

See Also:

ApplicationContext.getAutowireCapableBeanFactory(), AutowireCapableBeanFactory.createBean(java.lang.Class<T>)

doService

```
protected void doService(HttpServletRequest request,
                         HttpServletResponse response)
                  throws Exception
```

Exposes the DispatcherServlet-specific request attributes and delegates to doDispatch(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) for the actual dispatching.

Specified by:

doService in class FrameworkServlet

Parameters:

request - current HTTP request

response - current HTTP response

Throws:

Exception - in case of any kind of processing failure

See Also:

HttpServlet.doGet(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse), HttpServlet.doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)

doDispatch

```
protected void doDispatch(HttpServletRequest request,
                          HttpServletResponse response)
                   throws Exception
```

Process the actual dispatching to the handler.

The handler will be obtained by applying the servlet's HandlerMappings in order. The HandlerAdapter will be obtained by querying the servlet's installed HandlerAdapters to find the first that supports the handler class.

All HTTP methods are handled by this method. It's up to HandlerAdapters or handlers themselves to decide which methods are acceptable.

Parameters:

request - current HTTP request

response - current HTTP response

Throws:

Exception - in case of any kind of processing failure

buildLocaleContext

```
protected LocaleContext buildLocaleContext(HttpServletRequest request)
```

Build a LocaleContext for the given request, exposing the request's primary locale as current locale.

The default implementation uses the dispatcher's LocaleResolver to obtain the current locale, which might change during a request.

Overrides:

buildLocaleContext in class FrameworkServlet

Parameters:

request - current HTTP request

Returns:

the corresponding LocaleContext

See Also:

LocaleContextHolder.setLocaleContext(org.springframework.context.i18n.LocaleContext)

checkMultipart

```
protected HttpServletRequest checkMultipart(HttpServletRequest request)
                                     throws MultipartException
```

Convert the request into a multipart request, and make multipart resolver available.

If no multipart resolver is set, simply use the existing request.

Parameters:

request - current HTTP request

Returns:

the processed request (multipart wrapper if necessary)

Throws:

MultipartException

See Also:

MultipartResolver.resolveMultipart(javax.servlet.http.HttpServletRequest)

cleanupMultipart

```
protected void cleanupMultipart(HttpServletRequest request)
```

Clean up any resources used by the given multipart request (if any).

Parameters:

request - current HTTP request

See Also:

MultipartResolver.cleanupMultipart(org.springframework.web.multipart.MultipartHttpServletRequest)

getHandler

```
protected HandlerExecutionChain getHandler(HttpServletRequest request)
                                    throws Exception
```

Return the HandlerExecutionChain for this request.

Tries all handler mappings in order.

Parameters:

request - current HTTP request

Returns:

the HandlerExecutionChain, or null if no handler could be found

Throws:

Exception

noHandlerFound

```
protected void noHandlerFound(HttpServletRequest request,
                              HttpServletResponse response)
                       throws Exception
```

No handler found -> set appropriate HTTP response status.

Parameters:

request - current HTTP request

response - current HTTP response

Throws:

Exception - if preparing the response failed

getHandlerAdapter

```
protected HandlerAdapter getHandlerAdapter(Object handler)
                                    throws ServletException
```

Return the HandlerAdapter for this handler object.

Parameters:

handler - the handler object to find an adapter for

Throws:

ServletException - if no HandlerAdapter can be found for the handler. This is a fatal error.

processHandlerException

```
protected ModelAndView processHandlerException(HttpServletRequest request,
                                               HttpServletResponse response,
                                               Object handler,
                                               Exception ex)
                                        throws Exception
```

Determine an error ModelAndView via the registered HandlerExceptionResolvers.

Parameters:

request - current HTTP request

response - current HTTP response

handler - the executed handler, or null if none chosen at the time of the exception (for example, if multipart resolution failed)

ex - the exception that got thrown during handler execution

Returns:

a corresponding ModelAndView to forward to

Throws:

Exception - if no error ModelAndView found

render

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

Render the given ModelAndView.

This is the last stage in handling a request. It may involve resolving the view by name.

Parameters:

mv - the ModelAndView to render

request - current HTTP servlet request

response - current HTTP servlet response

Throws:

ServletException - if view is missing or cannot be resolved

Exception - if there's a problem rendering the view

getDefaultViewName

```
protected String getDefaultViewName(HttpServletRequest request)
                             throws Exception
```

Translate the supplied request into a default view name.

Parameters:

request - current HTTP servlet request

Returns:

the view name (or null if no default found)

Throws:

Exception - if view name translation failed

resolveViewName

```
protected View resolveViewName(String viewName,
                               Map<String,Object> model,
                               Locale locale,
                               HttpServletRequest request)
                        throws Exception
```

Resolve the given view name into a View object (to be rendered).

The default implementations asks all ViewResolvers of this dispatcher. Can be overridden for custom resolution strategies, potentially based on specific model attributes or request parameters.

Parameters:

viewName - the name of the view to resolve

model - the model to be passed to the view

locale - the current locale

request - current HTTP servlet request

Returns:

the View object, or null if none found

Throws:

Exception - if the view cannot be resolved (typically in case of problems creating an actual View object)

See Also:

ViewResolver.resolveViewName(java.lang.String, java.util.Locale)