HowToRunTapestry5Websphere

Tapestry on Websphere

Running Tapestry on Websphere is not so easy (actually running any modern web framework)! I'm talking about the "real" Websphere - not the Websphere Community Edition (CE). This guide applies to Webshpere 6.1 (don't know if it works with 6.0 or 7)

The main problem with Websphere is that it does not support the standards defined in the Java Servlet Specification. Therefore some workarounds have to be applied to run Tapestry.

Fix Packs, Filter Compatibility

Pleas read (and follow) Tapestry 5 Deployment Notes: WebSphere.

Filter initializing - Multiple Registries

Websphere calls the init-method on Servlet filters every time a client hits the matching URL and any of the init-methods haven't finished yet (according to Servlet Spec the init-method must only be called once). The Tapestry filter initializes the IoC registry in "init" - so, if many clients (or system management agents) hit your page on redeploy, you'll have a bunch of Tapestry IoC registries running.

You will notice this by finding something like this in your logs:

```
[WebContainer : 6] ERROR () org.apache.tapestry5.ioc.internal.SerializationSupport - Setting a new service
proxy provider when there's already an existing provider. This may indicate that you have multiple IoC
Registries.
```

When you have this problem there are two ways out:

1. avoid clients from sending requests to your app before it has finished initializing (e.g. by showing a "site down" page on your Apache) 2. or change Tapestry filter to a listener and a filter. The listener is only called once on startup - so everything's ok.

Here is the code for option 2:

TapestryListener.java

```
package org.apache.tapestry5.mine;


import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.tapestry5.internal.ServletContextSymbolProvider;
import org.apache.tapestry5.internal.TapestryAppInitializer;
import org.apache.tapestry5.ioc.Registry;
import org.apache.tapestry5.ioc.def.ModuleDef;
import org.apache.tapestry5.ioc.services.SymbolProvider;
import org.apache.tapestry5.services.HttpServletRequestHandler;
import org.apache.tapestry5.services.ServletApplicationInitializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The TapestryListener is responsible for initializing Tapestry.
 * <p/>
 * The application is configured via context-level init parameters.
 * <p/>
 * <dl>
 * <dt>tapestry.app-package</dt>
```

```
 * <dd>The application package (used to search for pages, components, etc.)</dd>
 * <dt>tapestry.application-name</dt>
 * <dd>The name of the application (used for </dd>
 * </dl>
 */
public class TapestryListener implements ServletContextListener
{
    private final Logger logger = LoggerFactory.getLogger(TapestryListener.class);

    private ServletContext context;

    private Registry registry;

    private HttpServletRequestHandler handler;

    /**
     * Key under which that Tapestry IoC {@link org.apache.tapestry5.ioc.Registry} is stored in the
ServletContext. This
     * allows other code, beyond Tapestry, to obtain the Registry and, from it, any Tapestry services. Such
code should
     * be careful about invoking {@link org.apache.tapestry5.ioc.Registry#cleanupThread()} appopriately.
     */
    public static final String REGISTRY_CONTEXT_NAME = "org.apache.tapestry5.application-registry";

    /**
     * Initializes the application using the {@link TapestryAppInitializer}. The application name is the
capitalization of
     * the context parameter "org.apache.tapestry5.application-name".
     */
    public void contextInitialized(ServletContextEvent sce)
    {
        context = sce.getServletContext();

        String appName = sce.getServletContext().getInitParameter("org.apache.tapestry5.application-name");
        if(appName == null)
                appName = "app";

        SymbolProvider provider = new ServletContextSymbolProvider(context);

        TapestryAppInitializer appInitializer = new TapestryAppInitializer(logger, provider, appName,
"servlet");

        appInitializer.addModules(provideExtraModuleDefs(context));

        registry = appInitializer.createRegistry();

        context.setAttribute(REGISTRY_CONTEXT_NAME, registry);

        ServletApplicationInitializer ai = registry.getService("ServletApplicationInitializer",
                                                               ServletApplicationInitializer.class);

        ai.initializeApplication(context);

        registry.performRegistryStartup();

        handler = registry.getService("HttpServletRequestHandler", HttpServletRequestHandler.class);

        init(registry);

        appInitializer.announceStartup();
    }

    protected final ServletContext getServletContext()
    {
        return context;
    }

    /**
     * Invoked from {@link #init(FilterConfig)} after the Registry has been created, to allow any additional
     * initialization to occur. This implementation does nothing, and my be overriden in subclasses.
     *
```

```
     * @param registry from which services may be extracted
     */
    protected void init(Registry registry)
    {
            // empty for override
    }

    /**
     * Overridden in subclasses to provide additional module definitions beyond those normally located. This
     * implementation returns an empty array.
     */
    protected ModuleDef[] provideExtraModuleDefs(ServletContext context)
    {
        return new ModuleDef[0];
    }

    public final void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException
    {
        try
        {
            boolean handled = handler.service((HttpServletRequest) request, (HttpServletResponse) response);

            if (!handled) chain.doFilter(request, response);
        }
        finally
        {
            registry.cleanupThread();
        }
    }

    /**
     * Shuts down and discards the registry.  Invokes {@link #destroy(org.apache.tapestry5.ioc.Registry)} to
allow
     * subclasses to peform any shutdown logic, then shuts down the registry, and removes it from the
ServletContext.
     */
    public void contextDestroyed(ServletContextEvent sce)
    {
        destroy(registry);

        registry.shutdown();

        getServletContext().removeAttribute(REGISTRY_CONTEXT_NAME);

        registry = null;
        context = null;
        handler = null;
    }

    /**
     * Invoked from {@link #destroy()} to allow subclasses to add additional shutdown logic to the filter. The
Registry
     * will be shutdown after this call. This implementation does nothing, and may be overridden in subclasses.
     *
     * @param registry
     */
    protected void destroy(Registry registry)
    {
            // empty for override
    }
}
```

TapestryFilter.java

```
package org.apache.tapestry5.mine;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.tapestry5.ioc.Registry;
import org.apache.tapestry5.services.HttpServletRequestHandler;

/**
 * The TapestryFilter is responsible for intercepting all requests into the web application. It identifies the
requests
 * that are relevant to Tapestry, and lets the servlet container handle the rest.
 */
public class TapestryFilter implements Filter
{
    private HttpServletRequestHandler handler;

    private Registry registry;

    /**
     * Initializes the filter
     */
    public final void init(FilterConfig filterConfig) throws ServletException
    {
            registry = (Registry) filterConfig.getServletContext().getAttribute(TapestryListener.
REGISTRY_CONTEXT_NAME);

            if(registry == null)
            {
                    throw new ServletException("No Tapestry registry found. Please check that TapestryListener
is added as <listener>");
            }

        handler = registry.getService("HttpServletRequestHandler", HttpServletRequestHandler.class);
    }

    public final void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException
    {
        try
        {
            boolean handled = handler.service((HttpServletRequest) request, (HttpServletResponse) response);

            if (!handled) chain.doFilter(request, response);
        }
        finally
        {
            registry.cleanupThread();
        }
    }

    public final void destroy()
    {
        registry = null;
        handler = null;
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

        <display-name>My Tapestry 5 Test Application</display-name>

        <context-param>
                <param-name>tapestry.app-package</param-name>
                <param-value>fill in my-app-package</param-value>
        </context-param>

        <filter>
                <filter-name>app</filter-name>
                <filter-class>org.apache.tapestry5.mine.TapestryFilter</filter-class>
        </filter>

        <filter-mapping>
                <filter-name>app</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

        <listener>
                <listener-class>org.apache.tapestry5.mine.TapestryListener</listener-class>
        </listener>

</web-app>
```