Seam Wicket Module

Reference Guide

Clint Popetz

Pete Muir

Igor Vaynberg

Introduction

The goal of Seam Wicket is to provide a fully integrated CDI programming model to the Apache Wicket web framework. Although Apache components (pages, panels, buttons, etc.) are created by direct construction using "new", and therefore are not themselves CDI contextual instances, with seam-wicket they can receive injections of scoped contextual instances via @Inject. In addition, conversation propagation is supported to allow a conversation scope to be tied to a wicket page and propagated across pages.

Installation

The seam-wicket-api.jar should be placed in the web application library folder. If you are using *Maven* [http://maven.apache.org/] as your build tool, you can add the following dependency to your pom.xml file:

```
<dependency>
  <groupId>org.jboss.seam.wicket</groupId>
  <artifactId>seam-wicket-api</artifactId>
  <version>${seam-wicket-version}</version>
</dependency>
```

Tip

Replace ${seam-wicket-version} with the most recent or appropriate version of Seam Wicket.

You must also bootstrap Weld according to your environment. As Wicket is normally used in a servlet (non-JavaEE) environment, this is most easily accomplished using the Weld Servlet integration, described in the *Weld Reference Guide* [http://docs.jboss.org/weld/reference/latest/en-US/html/environments.html].

You must extend org.jboss.seam.wicket.SeamApplication rather than org.apache.wicket.protocol.http.WebApplication. In addition:

- if you override newRequestCycleProcessor to return your own IRequestCycleProcessor subclass, you must instead override getWebRequestCycleProcessorClass() and return the class of your processor, and your processor must extend SeamWebRequestCycleProcessor
- if you override newRequestCycle to return your own RequestCycle subclass, you must make that subclass extend SeamRequestCycle.

If you can't extend SeamApplication, for example if you use an alternate Application superclass for which you do not control the source, you can duplicate the three steps SeamApplication takes, i.e. return a SeamWebRequestCycleProcessor NonContextual instance in newRequestCycleProcessor, return a SeamRequestCycle instance in newRequestCycle, and add a SeamComponentInstantiationListener with addComponentInstantiationListener.

Seam Wicket Features

Seam's integration with Wicket is focused on two tasks: conversation propagation through wicket page metadata and contextual injection of wicket components.

2.1. Injection

Any object that extends org.apache.wicket.Component or one of its subclasses is eligible for injection with CDI beans. This is accomplished by annotating fields of the component with the @javax.inject.Inject annotation:

```
public class MyPage extends WebPage {
    @Inject SomeDependency dependency;

    public MyPage()
    {
        depedency.doSomeWork();
    }
```

Note that since Wicket components must be serializable, any non-transient field of a wicket component must be serializable. In the case of injected dependencies, the injected object itself will be serializable if the scope of the dependency is not @Dependent, because the object injected will be a serializable proxy, as required by the CDI specification. For injections of non-serializable @Dependent objects, the field should be marked transient and the injection should be looked up again in a component-specific attach() override, using the BeanManager API.

Upon startup, Weld will examine your component classes to ensure that the injections you use are resolvable and unambiguous, as per the CDI specification. If any injections fail this check, your application will fail to bootstrap.

The scopes available are similar to those in a JSF application, as descibed in the CDI reference. The container, in an JavaEE environment, or the servlet listeners, in a servlet environment, will set up the application, session, and request scopes. The conversation scope is set up by the SeamWebRequestCycle as outlined in the next two sections.

2.2. Conversation Control

Application conversation control is accomplished as per the CDI specification. By default, like JSF/CDI, each wicket http request (whether ajax or not) has a transient conversation, which is destroyed at the end of the request. A conversation is marked long running by injecting the javax.enterprise.context.Conversation bean and calling its begin method.

```
public class MyPage extends WebPage {
```

```
@Inject Conversation conversation;

public MyPage()
{
   conversation.begin();
   //set up components here
}
```

Similarly, a conversation is ended with the Conversation bean's end() method.

2.3. Conversation Propagation

A transient conversation is created when the first wicket IRequestTarget is set during a request. If the request target is an IPageRequestTarget for a page which has previously marked a conversation as non-transient, or if the "cid" parameter is present in the request, the specified conversation will be activated. If the conversation is missing (i.e. has timed out and been destroyed), SeamRequestCycle.handleMissingConversation() will be invoked. By default this does nothing, and your conversation will be new and transient. You can however override this, for example to throw a PageExpiredException or something similar. Upon the end of a response, SeamRequestCycleProcessor will store the cid of a long running conversation, if one exists, to the current page's metadata map, if there is a current page. The key for the cid in the metadata map is the singleton SeamMetaData.CID. Finally, uppon detach(), the SeamRequestCycle will invalidate and deactive the conversation context.

Note that the above process indicates that after a conversation is marked long-running by a page, requests back to that page (whether ajax or not) will activate that conversation. It also means that new Pages set as RequestTargets, if created directly with setResponsePage(somePageInstance) or with setResponsePage(SomePage.class,pageParameters), will have the conversation propagated to them. This can be avoided by (a) ending the conversation before the call to setResponsePage, (b) using a BookmarkablePageLink rather than directly instantiating the response page, or (c) specifying an empty "cid" parameter in PageParameters when using setResponsePage. (Note that the final case also provides a mechanism for switching conversations: if a cid is specified in PageParameters, it will be used by bookmarkable pages, rather than the current conversation.)