For Tapestry Users

true

Comparing Wicket & Tapestry

solidTable of contents1
I haven't used Tapestry in a while and don't remember much... so somebody please continue the comparison...

Comparing Wicket and Tapestry

Date: 4 November 2005<p>
Author: David Leangenhttp://www.leangen.net

Overview

Both Wicket and Tapestry are component-based web application framework written in Java. Although the design approaches are very similar, some of the implementation details are quite different.

Moving to Wicket from Tapestry is double-edged: on the one hand, Tapestry users are already familiar with the component-driven approach, so should have no problem understanding Wicket concepts. On the other hand, however, the actual coding can be very different.

This article is aimed at Tapestry users moving over to Wicket. The goal is to give such users a running start by showing some of the key implementation details in Wicket that differ from Tapestry.

Note: I only started writing this article recently, so the contents are not yet written in any logical order.

About the Author

This article was originally written by David Leangen[Since he himself moved to Wicket from Tapestry, he wanted to share his experiences with others in the same situation.

The Wicket Way

This is a quote taken from the mailing list:

Although they are both component oriented frameworks, they have
completely different approaches. In wicket the focus is on java
code not on the template. Wicket templates are simple and limited
(purposefully) whereas tapestry allows for a lot more customization
from within the template. Wicket's approach makes sure all your
logic is kept in code. That is the wicket way.

– Igor Vaynberg

And:

Some of the main differences:

- Wicket is not a managed framework. That means that you - the programmer - are in charge of component creation yourself. You do this with javacode opposed to doing it declarative.

Pro: gives you flexibility/ you wont be limited
by what the framework builders thought up/ you
don't have to learn how the framework manages
and how the declaring language (xml) works.

Con: harder to integrate with other frameworks
sometimes/ wicket internals sometimes difficult.

- (like Igor said) Wicket purposefuly does not support scripting like features in your markup.

Pro: everything stays very clean and it is easier to guess
how things should be done.

Con: harder for people that are used to a 'php/jsp way of
doing things' and you need programmers that at least
understand the basics of OO Java programming.

Pro: clarity and cleaneness. Con: some things are more
work with Wicket and you have to keep your java
component tree in sync with the markup nesting.

- With Wicket every component is truly stateful. Every
  property you define is part of it's state, and there
  is a flexible undo mechanism you can use to support
  any advanced backbutton support you might want.
  No need for a rewind mechanism. Furthermore, wicket
  component can be nested and can take part of any
  collaboration you want in the same fashion you could
  do in e.g. Swing.

Pro: flexibility and very easy to do complex things
if you know your Java.

Con: sometimes easy to end up with unoptimized spaghetti
like code if you take too many short cuts (much
like you could have with Swing).

- Creating custom components with Wicket is super easy.
  Just extend from an existing one (or from base class
  WebComponent or WebMarkupContainer), make it available
  in your classpath and your done. There's no extra
  configuration (libraries) and magical strings (ids)
  involved. For advanced component initialization, you
  can use IInitializer.
    - Eelco Hillenius

Coding Components

The main differences with Wicket when coding components are:

- there is no XML page file
- you cannot pass in parameters via HTML

Example: create a menu component

In this example, we will explore how to create a menu component for a website. The goal is to output HTML code such as the following:

```
<div id="menu">
<p class="menu-item">Item 1</p>
<p class="menu-item"><a href="item2.html">Item 2</a></p>
<p class="menu-item"><a href="item3.html">Item 3</a></p>
<p class="menu-item"><a href="item4.html">Item 4</a></p>
</div>
```

Item 1 does not contain a link since it is the "active" page. The same would occur for any one of the other links when the page corresponding to that link is "active".

Note that in most cases, you would not even need to think about this since Wicket's <wicket:link> component takes care of this for you. However, we will continue to use this as our example.

First, the Tapestry Way

To be continued...

Now, the Wicket Way

To be continued...

I18n

First, the Tapestry Way

(I never really got around to working with i18n in Tapestry... could somebody write a brief explanation here?)

Now, the Wicket Way

Internationalisation (i18n) is easy in Wicket, since it is supported out of the box.

All you need to do is create the localised page for each Page in your application. For instance:

Hello.java
Hello.html
Hello_fr.html
Hello_ja.html
...

The application will auto-detect the default language setting on the client machine and serve that locale by default. If the locale is unknown, it will default to that provided in the file with no locale specification (so in our example, "Hello.html").

You can switch locales by simply adding a link to your page, such as:

In Hello.java: add(new Link("toJapaneseLink") { public void onClick() { getSession().setLocale(Locale.JAPANESE); } });
In Hello*.html

<a href="#" wicket:id="toJapaneseLink">Japanese.]</a>

See the "pub" example for a very simple, but concrete implementation.