Tapestry5Roadmap

WARNING: This is outdated content, kept for historical reference. Current information is available at

http://tapestry.apache.org/introduction.html

High Level (email dump)

I agree that <service> was easier than the HM approach, which is (infinitely) flexible.

Perhaps we just autowire the service instance with properties of matching type from the Infrastructure service. So, if you need a LinkFactory in your service, just define a setter method that takes a LinkFactory parameter.

I made the decision that engine services would use HM because that's a very uncommon operation in Tapestry, as opposed to creating pages and components.

I've been struggling with the following choice:

- Keep backwards compatibility and evolve the code base (give or take)
- Sacrifice backwards compatibility, but create the simpler, less ambiguous (easier to learn) framework people want

I'm prototyping some stuff right now that I'd have to call Tapestry 5. I don't think there's a way to remove inheritance in the Tapestry 4 code base without distorting many interfaces and breaking backwards compatibility.

My current vision is that the 4.1 code base will be about creating new components, including Ajax integration. Most of the innovation is present in the 3.0 -> 4.0 transition. Some new development may be to make certain things easier, such as re-introducing <service> in some way.

I'm starting to focus beyond that, on Tapestry 5. I'll be drawing out the code from Tapestry 4 and providing new code. My vision is:

- Annotations based. JDK 1.5.
- No XML for pages and components. Just HTML and Annotations.
- Concrete, not abstract, page and component classes
- No inheritance imposition
- Transforming class loader driven by annotations (i.e., `@Persist` on the field, not the method)
- Change detection; automatic invalidate and reload of changed objects (including Java classes)
- "Modern" templates: Templates as wellformed XML, using a namespace for Tapestry attributes and elements
- A "preview" mode for pages
- Partial rendering of pages (for Ajax operations)
- Improved management of page state (including dynamic state during the render)
- Vastly simplified API
- Clear deliniation between public/stable and private/internal/unstable APIs

My vision for state changing components like For is to introduce an extra layer of control.

Instead of the For component just updating its instance variable, it should record, with some page-global objects, a Command for making the change.

The Command will be executed during the render.

The interface for Command will probably be `void execute(IComponent component);` the control layer will store and serialize a ComponentAddress; during the subsequent request, it will use the ComponentAddress to reaquire the component and pass it to the Command. The Command impl will downcast and invoke.

In addition, the Commands will be collected and serialized. The direct service will encode these serialized commands as a new query parameter, pagestate.

By deserializing and executing these Commands, the page state can be restored to the render state as it was during the render process, just before triggering the component and its listener.

There are a couple of minor issues ... for instance, a For component will render a series of contradictory state (that is, item1, item2, item3 ...) do we need to store, in the pagestate query parameter, all of those in sequence, or just the most recent one?

This is a solution I hope to implement for 4.1 and certainly for 5.

--I'm not sure how Tapestry 5 will shake out implementation wise (xml/no xml) but taking Tapestry 4 as a starting point I'd like to open a debate about some implementation (mostly lookup details). see [Tapestry5LookupDebate this page]. GeoffLongman