How JSF State Management works

{scrollbar}

How JSF State Management Works

Overview

StateManager manages state, but you, as an end-user, have to give it hints about what state needs to be kept. That's either via implementing **Serializable** (and correct use of transient is part of implementing **Serializable**) or by implementing **StateHolder**.

Data is stored in two different ways in JSF: in scoped beans or in the component tree. The scope of scoped beans are hopefully self-explanatory. The state of the components themselves are stored in the response, and then restored when a request arrives. This effectively gives data stored by the components a "page" scope since they exist so long as the page doesn't change. Note that components store value bindings and method bindings (#{} – el expressions) as string literals, so the backing beans they point to are not stored in the component tree at page scope.

Tomahawk has a *SaveState* component named <t:saveState> that allows you to store data (including entire backing beans) as part of the component tree, effectively making the page-scoped (or more since you can preserve such values across pages. In JSF 2.0, a new scope called @ViewScope was added to provide the same functionality as t:saveState. You still need to properly implement **Serializable** or **StateHolder** on your backing beans.

Backtracking

Note that **StateManager** is pluggable, so it's possible to implement your own alternatives to the standard state management. Thus you could implement a strategy that allowed backtracking.

Another alternative to handling backtracking is to use client-side state saving and avoid the use of session-scoped data. This also requires resyncing any other persisting backing stores (i.e. databases) when the page is reloaded.

{scrollbar}