General Questions

General Questions

Contents

How do I get started with Tapestry?

The easiest way to get started is to use Apache Maven to create your initial project; Maven can use an *archetype* (a kind of project template) to create a bare-bones Tapestry application for you. See the Getting Started page for more details.

Even without Maven, Tapestry is quite easy to set up. You just need to download the binaries and setup your build to place them inside your WAR's WEB-INF/lib folder. The rest is just some one-time configuration of the web.xml deployment descriptor.

Why does Tapestry use Prototype (in versions before 5.4)? Why not *insert favorite JavaScript library here*?

An important goal for Tapestry is seamless DHTML and Ajax integration. To serve that goal, it was important that the built in components be capable of Ajax operations, such as dynamically re-rendering parts of the page. Because of that, it made sense to bundle a well-known JavaScript library as part of Tapestry.

At the time (this would be 2006-ish), Prototype and Scriptaculous were well known and well documented, whereas jQuery was just getting started.

The intent has always been to make this aspect of Tapestry pluggable. Tapestry 5.4 includes the option of either Prototype or jQuery, and future versions of Tapestry will likely remove Prototype as an option..

Why does Tapestry have its own Inversion of Control Container? Why not Spring or Guice?

An Inversion of Control Container is *the* key piece of Tapestry's infrastructure. It is absolutely necessary to create software as robust, performant and extensible as Tapestry.

Tapestry IoC includes a number of features that distinguish itself from other containers:

- Configured in code, not XML
- Built-in extension mechanism for services: configurations and contributions
- Built-in aspect oriented programming model (service decorations and advice)
- Easy modularization
- Best-of-breed exception reporting

Because Tapestry is implemented on top of its IoC container, and because the container makes it easy to extend or replace any service inside the container, it is possible to make the small changes to Tapestry needed to customize it to any project's needs.

In addition – and this is critical – Tapestry allows 3rd party libraries to be built that fully participate in the configurability of Tapestry itself. This means that such libraries can be configured the same way Tapestry itself is configured, and such libraries can also configure Tapestry itself. This *distributed configuration* requires an IOC container that fully supports such configurability.

How do I upgrade from Tapestry 4 to Tapestry 5?

There is no existing tool that supports upgrading from Tapestry 4 to Tapestry 5; Tapestry 5 is a complete rewrite.

Many of the basic concepts in Tapestry 4 are still present in Tapestry 5, but refactored, improved, streamlined, and simplified. The basic concept of pages, templates and components are largely the same. Other aspects, such as server-side event handling, is markedly different.

Tapestry 5 is designed so that it can live side-by-side in the same servlet as a Tapestry 4 app, without package namespace conflicts, sharing session data and common resources such as images and CSS. This means that you can gradually migrate a Tapestry 4 app to Tapestry 5 one page (or one portion of the app) at a time.

How do I upgrade from one version of Tapestry 5 to another?

Main Article: How to Upgrade.

A lot of effort goes into making an upgrade from one Tapestry 5 release to another go smoothly. In the general case, it is just a matter of updating the version number in your Maven `build.xml` or Gradle `build.gradle` file and executing the appropriate commands (e.g., `gradle idea` or `mvn eclipse:eclipse`) to bring your local workspace up to date with the latest binaries.

After changing dependencies, you should always perform a clean recompile of your application.

We make every effort to ensure backwards-compatibility. Tapestry is mostly coded in terms of interfaces; those interfaces are stable to a point: interfaces your code is expected to implement are usually completely frozen; interfaces your code is expected to invoke, such as the interfaces to IoC services, are stable, but may have new methods added in a release; existing methods are not changed.

In *rare* cases a choice is necessary between fixing bugs (or adding essential functionality) and maintaining complete backwards compatibility; in those cases, an incompatible change may be introduced. These are always discussed in detail in the Release Notes for the specific release. You should always read the release notes before attempting an upgrade, and always (really, *always*) be prepared to retest your application afterwards.

Note that you should be careful any time you make use of **internal** APIs (you can tell an API is internal by the package name, `org.apache.tapestry5.internal`). Internal APIs may change *at any time*; there's no guarantee of backwards compatibility. Please always check on the documentation, or consult the user mailing list, to see if there's a stable, public alternative. If you do make use of internal APIs, be sure to get a discussion going so that your needs can be met in the future by a stable, public API.

Why are there both Request and HttpServletRequest?

Tapestry's Request interface is *very* close to the standard HttpServletRequest interface. It differs in a few ways, omitting some unneeded methods, and adding a couple of new methods (such as `isXHR()`), as well as changing how some existing methods operate. For example, `getParameterNames()` returns a sorted List of Strings; HttpServletRequest returns an Enumeration, which is a very dated approach.

However, the stronger reason for Request (and the related interfaces Response and Session) is to enable the support for Portlets at some point in the future. By writing code in terms of Tapestry's Request, and not HttpServletRequest, you can be assured that the same code will operate in both Servlet Tapestry and Portlet Tapestry.