Tapestry

Code less, deliver more.

Rayland Jeans
What is Apache Tapestry?

- Apache Tapestry is an open-source framework designed to create scalable web applications in Java.
- Tapestry allows developers to create web applications that are a set of pages constructed from components.
- Tapestry is designed specifically to make creating new components easy.
- Simplifies configuration by removing the need for XML and promotes the use of Java annotations and naming conventions.
What is Apache Tapestry?

• Written in Pure Java so pages and components can be written in Java, Groovy or Scala.

• Provides the ability to add new modules using an IoC container.

• Contains built-in support for Ajax and Javascript.

• Provides support for easily unit testing pages and components.
Adaptive API

• A statement made on the Tapestry web site
  •  [http://tapestry.apache.org](http://tapestry.apache.org)

• “In traditional Java frameworks, including Tapestry 4, user code is expected to conform to the framework.
  • You create classes that extend from framework-provided base classes, or implement framework-provided interfaces.
  • This works well until you upgrade to the next release of the framework
  • Interfaces or base classes will have changed and your existing code will need to be changed to match.

• In Tapestry 5, the framework adapts to your code.
  • You have control over the names of the methods, the parameters they take, and the value that is returned.
  • This is driven by annotations, which tell Tapestry under what circumstances your methods are to be invoked.”
Features of Tapestry

• Tapestry 5 has many features. These are the features that will be covered in this presentation.

• Live class reloading.
• Convention over Configuration
• Pages and Components
• Advanced Exception Reporting
• Inversion of Control Container
• Ajax and JavaScript support
Live Class Reloading

• Most Java web application frameworks require you to restart the web server when a change is made to a Java class.

http://xkcd.com/303/
Live Class Reloading

- Tapestry provides automatic reloading of page classes and templates.
- On a change of any class within a controlled package, Tapestry will discard and reload all page instances and the class loader.
- This does not affect data stored in the session.
- This allows developers to make changes while the application is running.
- This also allows developers to focus more on the application being developed and not the web server hosting the application.
Convention over Configuration

• No XML config files
  • Most older Java web frameworks require the use of XML for configuration.
  • Tapestry uses Java annotations for almost all of its configuration.
  • In addition to annotations, Tapestry makes use of naming conventions for configuration, such as:
    • Method names
    • Class names
    • Package names
Configuration

• So if there are no XML configuration files, how do you configure Tapestry?

• Since Tapestry is designed to run in a servlet container like Apache Tomcat or Jetty, you do need to configure the servlet deployment descriptor (web.xml).

• Specific configurations required are:
  • tapestry.app-package
  • Tapestry filter
  • Filter mapping

• This is sort of where configuration stops and convention takes over.
<web-app>
    <display-name>My Tapestry Application</display-name>
    <context-param>
        <param-name>tapestry.app-package</param-name>
        <param-value>com.sample.app</param-value>
    </context-param>
    <filter>
        <filter-name>app</filter-name>
        <filter-class>org.apache.tapestry5.TapestryFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>app</filter-name>
        <url-pattern>/</url-pattern>
    </filter-mapping>
</web-app>
Configuring Tapestry

- **tapestry.app-package** defines the location of your Page files and your Component files.

- Tapestry will use naming conventions to determine where Pages and Components are placed within your application.

- According to the tapestry.app-package setting above, Pages can be found in `com.sample.app.pages` and Components can be found in `com.sample.app.components`
• Application Module Class

• The application module class defines new services, provides overrides of services or makes contributions to service configurations.

• Using naming conventions, Tapestry looks for the application module class under the root package of the application. In this case, Tapestry will look in `com.sample.app.services` for the `AppModule` class.
Pages and Components

- Pages and Components are used to generate the view portion of the application. They replace Servlets and JSPs in traditional Java web apps.
- Pages and Components are Plain Old Java Objects.
  - No super-class to inherit.
    - Most older Java web frameworks require that you inherit from some base super class.
  - No Interfaces to implement.
  - Components for Tapestry instead use annotations to eliminate the need for inheritance, or interfaces.
  - Naming conventions are also used to eliminate the need for any XML configuration.
Pages and Components

- Tapestry does not use servlets or require a base action class to handle requests. Instead, Tapestry uses instances of Page classes and assigns an instance of a page to the thread handling the request.

- Pages and components are ordinary objects, complete with instances variables.
  - With traditional Java web apps that use Servlets, a single instance is created to handle all incoming requests. This means that the Servlet will usually have to be stateless, and instance variables are often of no use.
  - The statelessness requires the use of HttpServletRequest objects to store data per-request and HttpSession objects to store data between requests.
  - Instead of servlets, Tapestry uses a page pool, reserving page instances to particular threads.
  - Pages instance variables are purged and returned back to their default value at the end of the request.
Pages and Components

- Pages are stored in a page pool based on keys.
  - Keys are a combination of the page name and the locale used for that page. For example, the start page used for the “en” would be keyed off of “start” and “en”.
- The number of instances of a page is configurable.
  - Configurations include defining a soft limit and a hard limit of pages to be instantiated.
- When a page is accessed, Tapestry will check to see if the soft limit has been reached. If it has, then Tapestry will wait for a short period for a page instance to become available before trying to instantiate a new instance.
- If the hard limit is reached, then Tapestry will throw an exception, rather than create a new instance.
- Limits are per-page per-locale. So there could be 20 instances of page “start” for locale “en” and 20 instances for locale “fr”.
Pages and Components

- Component classes are the classes associated with a Page. Even though a Page is also a Component, a Page will usually contain one or more Components.

- Each component class will usually have a corresponding component template.
  - Component templates contain markup to a page.
    - However, components do not require a component template to generate markup. In this case, the class would be required to generate the required markup for the request.

- There are a few constraints on component classes:
  - The classes must be public.
  - The classes must be in the correct package.
  - The class must have a default no-argument constructor.
Pages and Components

- What's the difference between a page and a component?
- A page is simply a component that acts as the root component for a page's component tree.
- A page usually consists of a Java class, a page template and a sometimes a collection of components.
- A component consists of just a Java class and a component template.
  - A component can also consist of several other components.
- A page must exist in the pages package:
  - com.example.pages.Index.java
- A component must exist in the components package
  - com.example.components.IndexComponent.java
Pages and Components

A basic Page example

Page Class

```java
package com.example.pages;

import org.apache.tapestry5.annotations.InjectComponent;
import org.apache.tapestry5.annotations.Persist;
import org.apache.tapestry5.annotations.Property;
import org.apache.tapestry5.corelib.components.Zone;

public class Index {
    @Property
    @Persist
    private int clickCount;

    @InjectComponent
    private Zone counterZone;

    Object onActionFromClicker() {
        clickCount++;
        return counterZone.getBody();
    }
}
```

Page Template

```html
<div>
    <t:zone t:id="counterZone" id="counterZone">
        <p>You have clicked the link ${clickCount} times.</p>
    </t:zone>

    <p>
        <t:actionlink t:id="clicker" zone="counterZone">
            increment the count
        </t:actionlink>
    </p>
</div>
```
Pages and Components

- So what is the benefit of Pages and Components vs. using Servlets?
  - No configuration files required. Tapestry uses naming conventions to determine where your pages and components are and when to instantiate them.
  - Pages are just Java objects. No need to inherit a base class or override any super class methods.
  - Easy access to many of Tapestry’s built-in features
    - Ajax support
    - IoC container
    - Data persistence
    - Live class reloading
Advanced Exception Reporting

• With Tapestry there are no cryptic exceptions to interpret.

• Tapestry provides as much information about an exception that was found at runtime.
  • The information given about an exception is more than just a stack trace.

• Exception messages include:
  • What was Tapestry doing?
  • Why it was doing it?
  • What went wrong?
  • Where was the problem found?

• Tapestry also tries to suggest available alternatives.
Advanced Exception Reporting

• What went wrong and what was Tapestry doing?
• Exception messages give plenty of information to be used for debugging.

An unexpected application exception has occurred.

```
org.apache.tapestry5.ioc.internal.OperationException
Exception assembling root component of page index: Could not convert 'headingLevel' into a component parameter binding: Exception generating conduit for expression 'headingLevel': Class com.example.web.pages.Index does not contain a property (or public field) named 'headingLevel'.

trace:
- Constructing instance of page class com.example.web.pages.Index
- Assembling root component for page Index

java.lang.RuntimeException
Exception assembling root component of page index: Could not convert 'headingLevel' into a component parameter binding: Exception generating conduit for expression 'headingLevel': Class com.example.web.pages.Index does not contain a property (or public field) named 'headingLevel'.

org.apache.tapestry5.ioc.internal.util.TapestryException
Could not convert 'headingLevel' into a component parameter binding: Exception generating conduit for expression 'headingLevel': Class com.example.web.pages.Index does not contain a property (or public field) named 'headingLevel'.
```
Advanced Exception Reporting

- Where was the problem found?
  - Exception messages include the code and line number where the error was found.

```html
location:
classpath:com/example/web/pages/index.tml, line 16
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
</ol>

```
Advanced Exception Reporting

- Suggests available alternatives.
  - Exception messages include suggestions of values that could be used to fix the exception.
  - In this case, the expression used was `headingLevel` but the exception lists `headerLevel` as a possible alternative.
Tapestry Inversion of Control Container

- Tapestry contains an IoC package created with the developer in mind.
  - Designed to be easy to use and understand.
  - Does not require verbose XML configuration files.
    - This can be a bit confusing if you’re used to IoC containers such as older versions of Spring.
  - Exists specifically to address the need to add functionality while balancing the need to test and maintain existing code.
  - The IoC container provides a way to add new services to the application and make those services easier to test.
  - Provides a way to convert large, complicated blocks into small testable pieces.
Tapestry Inversion of Control Container

- Tapestry IoC Container is made up of a Registry that provides services to modules within a Tapestry application.
- The Registry contains services from the built-in IoC modules and services from the web framework module.

IoC Registry

<table>
<thead>
<tr>
<th>Tapestry IoC Module</th>
<th>Tapestry Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Factory</td>
<td>ApplicationGlobals</td>
</tr>
<tr>
<td>Property Access</td>
<td>Request</td>
</tr>
<tr>
<td>Type Coercer</td>
<td>Cookies</td>
</tr>
<tr>
<td>Symbol Sources</td>
<td>ApplicationStateManager</td>
</tr>
</tbody>
</table>
Tapestry Inversion of Control Container

• Tapestry services are lazy.
  • They are not fully instantiated until they are needed.
  • A service is actually a proxy. The first time a method on the proxy is invoked, the service is instantiated.
    • Tapestry refers to this as the service being realized.
• The IoC container is also how developers would add new services to a Tapestry application.
  • All that is needed is the application define the new service to Tapestry to make it available using the AppModule class.
Tapestry Inversion of Control Container

• Example Application Module class.

```java
package com.sample.app.services;

import org.apache.tapestry5.ioc.OrderedConfiguration;
import org.apache.tapestry5.ioc.ServiceBinder;
import org.apache.tapestry5.ioc.annotations.Local;
import org.apache.tapestry5.services.RequestFilter;

/**
 * Sample application module
 */
public class SampleModule {
    public static void bind(ServiceBinder binder) {
        binder.bind(MySampleDispatcher.class);
    }

    /**
     * Contribute a Sample service to Tapestry's pipeline as a RequestFilter.
     *
     * @param configuration
     *     Incoming configuration that allows the method provide
     *     contributed values to the service's configuration.
     * @param dispatcher
     *     An instance of MySampleDispatcher.
     */
    public static void contributeRequestHandler(
            OrderedConfiguration<RequestFilter> configuration,
            @Local RequestFilter dispatcher) {
        configuration.add("MySampleService", dispatcher);
    }
}
```
Tapestry Inversion of Control Container

- Tapestry IoC promotes coding to an interface over coding to an implementation.
- Tapestry promotes IoC techniques that lead to applications that are:
  - More testable
  - More robust
  - More scalable
  - Easier to maintain
  - Easier to extend
- The separation between interface and implementation allows developers to work on the same code base, lowering the risk of interference and conflict.
Ajax and JavaScript support

- In Tapestry, JavaScript is referred to as a first-class concept where sophisticated support is provided right out of the box.
  - In production mode, Tapestry will take advantage of browser caching and automatically minify JavaScript libraries.
- Tapestry comes with Prototype and Scriptaculous. Another version of tapestry, hosted on Github comes with JQuery and JQueryUI.
- Provides an @Import annotation to add additional JavaScript libraries from within your Java code.
  - CSS can also be imported using this annotation.
  - You can still use the <script> tags, but Tapestry prefers using the annotations.
- Tapestry also provides support for Ajax using built-in components and component mixins.
  - A Component mixin is a way to add additional functionality to a built-in component.
Ajax and JavaScript support

- Tapestry provides Ajax support through an approach known as Zones.
- A Zone allows a way for Tapestry to perform partial page updates.
  - A Zone typically is used to update a `<div>` element with a page.
  - In most cases, a Zone is a wrapper for dynamic content.
- A server side event handler is responsible for returning the content to be rendered.
- A Zone update is usually triggered by an ActionLink, an EventLink or by a Form.
- A Zone allows developers a way to implement Ajax updates without being required to write any JavaScript.
- This speeds up development of simple page updates.
Ajax and JavaScript support

A Zone example

```java
package com.example.pages;

import org.apache.tapestry5.annotations.InjectComponent;
import org.apache.tapestry5.annotations.Persist;
import org.apache.tapestry5.annotations.Property;
import org.apache.tapestry5.corelib.components.Zone;

public class Index {
    @Property
    @Persist
    private int clickCount;

    @InjectComponent
    private Zone counterZone;

    Object onActionFromClicker() {
        clickCount++;
        return counterZone.getBody();
    }
}
```

```html
<div>
    <t:zone t:id="counterZone" id="counterZone">
        <p>You have clicked the link ${clickCount} times.</p>
    </t:zone>
    <p>
        <t:actionlink t:id="clicker" zone="counterZone">
            increment the count
        </t:actionlink>
    </p>
</div>
```
The disadvantages of using Tapestry

• So after a basic introduction to Tapestry, what are the drawbacks?
  • As mentioned on the Tapestry website, there is not a lot of learning to work with Tapestry, instead there is a lot of unlearning.
  • You would have to try to put aside everything you’ve learned about writing Java web applications using Servlets and JSP’s.
  • If you are migrating from an existing framework, such as Struts, there may need to be some redesign in the code to migrate existing code to work in Tapestry.
  • The current version of Tapestry ships with the Prototype Javascript library. This could make it difficult to work with other libraries such as JQuery.
  • If your URL’s require a specific format, Tapestry may get in the way and make it difficult to generate the URL content required.
The advantages of using Tapestry

• Tapestry does a lot of the heavy lifting when it comes to developing a Java web application.
  • Tapestry was created with developer in mind.
• Easy integration of Spring and Hibernate.
  • Tapestry comes with built-in modules that make integrating Spring and Hibernate easy.
• Tapestry supports developing web applications across a team of developers by allowing developers to write clean Java code for pages and components.
  • No need to write Servlets and large XML configuration files.
• Tapestry templates are easily viewable using WYSIWYG editor.
Improved Developer Productivity

• Java web developers that do not spend most of their time developing web GUIs often spend most of their GUI development time trying to figure out or re-learn how the framework works.

• This can be challenging, especially when several weeks or months have passed since the last web development task.

• Configuration is usually what makes most web development tasks difficult and time consuming.

• With annotations and naming conventions, most of the time consumed by configuration is removed, which leaves more time to write code.

• Tapestry allows developers to focus on the tasks required for the business and not get distracted by the framework.
Overview of Tapestry

• Tapestry emphasizes convention over configuration.
• Designed with the developer in mind.
  • Focusing on improved productivity.
• Takes a different approach to creating Java web applications than other Java web frameworks.
• Provides the capability to create scalable applications using the built-in Inversion of Control container.
• Contains built-in support for Ajax and JavaScript.
More information on Tapestry

• There are plenty of features not discussed in this presentation such as:
  • Forms and Beans
  • Internationalization
  • Logging, Debugging and Testing
  • Module loading and more

• Descriptions of these features can be found on the Tapestry 5 website.
More information on Tapestry

http://tapestry.apache.org
More information on Tapestry

http://tapestry5-jquery.com/