Tapestry5HowToVisualizeComponentHierarchy

Sometimes you need to visualize the component hierarchy just inside your pages. For example when you want a new person to get familiar with a big Tapestry application.

See "Where did all the components come from?" and "TAP5-742: Add optional component tracing comments to rendered output" for discussion.

Here is a temporary hackish solution that can help you until TAP-742 is implemented. It automatically puts ShowComponentName mixin into every component from "myapp.components" package.

It does not look well around floated divs, but for that case you can inspect elements in FireBug.

So, let's create the mixin that shows parent component name:

```
package myapp.mixins;

import org.apache.tapestry5.ComponentResources;
import org.apache.tapestry5.MarkupWriter;
import org.apache.tapestry5.ioc.annotations.Inject;
import org.apache.tapestry5.runtime.Component;

public class ShowComponentName {

	@Inject
	private ComponentResources resources;

	private String getComponentName() {
		Class< ? extends Component > containerClass = resources.getContainer().getClass();
		String componentName = containerClass.getSimpleName().toLowerCase();

		// return null for your top level components here
		if ( componentName.equals( "layout" ) ) {
			return null;
		} else {
			return containerClass.getSimpleName();
		}
	}

	void beginRender( MarkupWriter writer ) {
		String name = getComponentName();
		if ( name != null ) {
			writer.element( "span", "style", "background-color:yellow" );
			writer.writeRaw( " " + name + "( " );
			writer.end();
		}
	}

	void afterRender( MarkupWriter writer ) {
		String name = getComponentName();
		if ( name != null ) {
			writer.element( "span", "style", "background-color:yellow" );
			writer.writeRaw( " " + name + ") " );
			writer.end();
		}
	}

}
```

Note comments in getComponentName() function.

Now let's create a ShowComponentWorker that puts this mixin into every component of your choice:

```
package myapp.services;

import org.apache.tapestry5.model.MutableComponentModel;
import org.apache.tapestry5.services.ClassTransformation;
import org.apache.tapestry5.services.ComponentClassResolver;
import org.apache.tapestry5.services.ComponentClassTransformWorker;

public class ShowComponentWorker implements ComponentClassTransformWorker {
        private final ComponentClassResolver resolver;

        public ShowComponentWorker( final ComponentClassResolver resolver ) {
                this.resolver = resolver;
        }

        public void transform( ClassTransformation transformation, MutableComponentModel model ) {
                String componentClassName = model.getComponentClassName();

                // Place your general rules to filter components here. Place often changing rules to the mixin
- thus you can enjoy the live reloading
                if ( componentClassName.startsWith( "myapp.components." ) ) {
                        if ( model.getMeta( "transformed" ) != null )
                                return;
                        model.setMeta( "transformed", "transformed" );

                        String mixinClassName = resolver.resolveMixinTypeToClassName( "ShowComponentName" );

                        model.addMixinClassName( mixinClassName );
                }

        }
}
```

Note comments in transform().

Now let's activate this worker in AppModule.java:

```
...
        public static void contributeComponentClassTransformWorker(
                        OrderedConfiguration< ComponentClassTransformWorker > configuration,
                        ComponentClassResolver resolver ) {
                configuration.add( "ShowComponentName", new ShowComponentWorker( resolver ), "before:Mixin" );
        }

...
```

P.S. Feel free to refine this.