Declarative Layouts with GWT 2.0

About

- Joakim Recht
- Software Pilot @ Trifork for 4½ years
- Works with
  - Java (EE), Spring, GWT, REST, WS-(death)*
- Contact: jre@trifork.com

Agenda

- (Very) short introduction to GWT 2.0
- Layouts in GWT < 2.0
- What is a declarative layout?
- Principles for declarative layouts in GWT
- Building a basic layout
- Controlling parts of the layout from code
- Binding events to code

Google Web Toolkit in 5 minutes

- GWT makes it possible to write Java instead of Javascript
- Abstracts browser quirks, HTML rendering, Javascript handling
- You can keep you existing tooling
- Strong type system, testable and maintainable code

GWT Main Concepts

- At least one class must implement EntryPoint
  - This is the class responsible for starting the application
- There must be one host page for each EntryPoint
  - This defines basic HTML and bootstraps GWT
- Each EntryPoint is described in a module XML file (.gwt.xml)

GWT Applications

- GWT applications consists of
  - Widgets, organized in a hierarchy
  - Event handlers, attached to certain widgets
  - Async callbacks, used for issuing requests against a server
  - Resources, such as CSS, images, resource bundles
    - These can be combined into ClientBundles

Developing GWT applications

- GWT applications are compiled into Javascript
  - A somewhat slow process
  - Typically done by CI
- GWT can also be run Development Mode (previously known as Hosted Mode)
  - Uses a special browser plugin
  - Much faster than compiling

Declarative Layouts in GWT

Pre-2.0 Layouts in GWT

- Component-based
- Quite similar to Swing (but without layout managers)
- HTML completely abstracted away
  - No direct control
  - Hard to convert HTML into GWT code
- Still the foundation for GWT

Layout Changes in GWT 2.0

- It's easier to create predictable layouts
- Fewer <table>s, more CSS constraints
- New UiBinder for creating layouts externally

Declarative Layouts

- Typically not code
  - Or at least DSL
  - In a format suitable for other than programmers
- Focuses on describing the layout
- No logic, just structure
- HTML, XAML, XUL

UiBinder

- GWTs implementation of declarative layout
- Makes it possible to define a component structure in XML
- Layouts can either be described as regular HTML or using GWT widgets

UiBinder is not

- A templating engine
  - There are no loops, statements, or anything else
- The solution to all your problems
  - Sometimes, it makes sense to define layouts in code
-

UiBinder Basics

- A UiBinder is encapsulated in a Widget class
  - This is the host/owner class
- The owner class behaves just like any other widget
  - Can be inserted into a component hierarchy
- The owner class uses UiBinder to
  - load the layout
  - manipulate the declared layout
  - bind events to code

UiBinder layouts

- Defined in XML
- Placed on classpath
- Must be called OwnerClass.ui.xml

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'>
  <ui:style>
    .test {
      font-weight: bold;
    }
  </ui:style>
  <div class=”{style.test}”>
    <div>This is the UiBinder</div>
    <div ui:field=”text”>Text here</div>
  </div>
</ui:UiBinder>
```

Using the XML Layout

- Create a new interface which extends UiBinder
  - Takes two type parameters: The root element type in the layout, and the owner class name
- Use GWT.create to create an instance of the binder
- Call createAndBindUi to actually create the layout

Using the XML Layout

```
public class MainLayout extends Widget {

        interface Binder extends UiBinder<DivElement, MainLayout> {}

        private static Binder binder = GWT.create(Binder.class);

        public MainLayout() {
                setElement(binder.createAndBindUi(this));
        }

}
```

- GWT.create does all the magic
- createAndBindUi instantiates the layout
- setElement sets the contents of this widget

Binding fields to code

- Often, you want to manipulate parts of the layout from code
  - Setting texts, populating lists from DB data, adding handlers
- Using UiBinder, this is done using ui:field in XML and @UiField in code

Binding fields to code

.ui.xml:

```
<ui:UiBinder>
 ...
  <div ui:field=”text”>Text here</div>
 ...
</ui:UiBinder>
```

Code:

```
public class MainLayout extends Widget {
  ...
  @UiField DivElement text;

  public MainLayout() {
    setElement(binder.createAndBindUi(this));
    text.setInnerText("testing");
  }
}
```

- Simply add @UiField properties to code
- Property name should match the field attribute name
- The property cannot be private
- The property will exist only after createAndBindUi

Layouts using components

- GWT supports two types of layouts:
  - HTML based layouts
    - <ui:UiBinder> simply contains raw (X)HTML
  - Widget based layouts
    - <ui:UiBinder> contains a description of real GWT widgets
- Widgets can be used in HTML layouts
  - However, you have to prepare for it
- Some HTML can be used in widget layouts

GWT Widget Layout

- Builds on real GWT widgets
- Describe the widget layout using XML
  - Any GWT widget can be used, both the built-in and any custom widgets
- Wire events using annotations

GWT Widget Layout

```
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
        xmlns:g="urn:import:com.google.gwt.user.client.ui">

        <g:VerticalPanel>
                <g:Label text="Why not try clicking the button?"></g:Label>
                <g:Button ui:field="button" text="Click me"/>
        </g:VerticalPanel>

</ui:UiBinder>
```

- Namespaces match package names
- Can import own widget packages
- XML attributes correspond to Java properties

Binding events to code

- Using @UiField, it is possible to add event handlers in code
- Traditional approach:
  - Lots of anonymous inner classes
  - Quite verbose
- UiBinder makes it possible to bind owner class methods to events using annotations

Binding events

```
public class WidgetLayout extends Composite {
  ...
  @UiField Button button;

  public WidgetLayout() {
    initWidget(uiBinder.createAndBindUi(this));
  }

  @UiHandler("button")
  void onClick(ClickEvent e) {
    Window.alert("Hello!");
  }
}
```

Binding events

- @UiHandler binds an event on the specified ui field to the method
- Can have multiple values
- Method must take exactly one parameter
- Parameter type depends on event type
  - A subclass of DomEvent
  - The widget event handler must match getAssociatedType()

Mixing HTML and Widgets

- HTML and Widgets can be mixed
- HTML can be embedded in some widgets
  - Those implementing HasHTML
  - Must be wrapped in g:HTML
- Widgets can be embedded in HTML
  - HTML must be wrapped in g:HTMLPanel
  - Placeholder widgets must be added to HTML

Adding HTML to Widgets

```
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
        xmlns:g="urn:import:com.google.gwt.user.client.ui">

        <g:VerticalPanel>
                <g:Label text="Why not try clicking the button?"></g:Label>
                <g:Button ui:field="button" text="Click me"/>

                <g:HTML>
                        <div>Here is some html</div>
                </g:HTML>
        </g:VerticalPanel>

</ui:UiBinder>
```

Adding Widgets to HTML

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
xmlns:g='urn:import:com.google.gwt.user.client.ui'>
  <g:HTMLPanel>
    <div>
      <div>This is the UiBinder</div>
      <div ui:field=”text”>Text here</div>
      <g:SimplePanel ui:field=”content” />
    </div>
  </g:HTMLPanel>
</ui:UiBinder>
```

- HTML must be wrapped in HTMLPanel
  - No other HTML changes necessary
- Add placeholder panel where widgets can be added
  - Bind the field using @UiField

Styling and CSS

- CSS styles can be declared and used
  - Will be checked by compiler
- Declare styles inline or externally in CSS files
- CSS will be bundled and obfuscated

CSS Styles in layouts

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'>
  <ui:style>
    .test {
      font-weight: bold;
    }
  </ui:style>
  <ui:style src=”styles.css” />

  <div class=”{style.test}”>
    <div class=”{style.header}”>This is the UiBinder</div>
    <div ui:field=”text”>Text here</div>
  </div>
</ui:UiBinder>
```

- Styles are declared using normal CSS syntax
- Styles are referred to using {}
- All other style references are not checked

Other UiBinder features

- Using widgets which require constructor arguments
- Multiple layouts for same owner class
- Internationalization

Widgets with constructor args

- GWT cannot instantiate when no default constructor
- Solution: Use a factory method
- Annotate a method with @UiFactory
  - Method name is not important
  - Any method parameters must be specified in XML

Multiple layouts for same owner

- By default, layouts must be defined in *.ui.xml
- Can be overridden using @UiTemplate
  - Specifies the layout either using a relative name or a FQN
  - Is placed in the layout interface
  - Also means that multiple interfaces can be defined in the same class, using different layouts

Other news in GWT 2.0

- New Development Mode
  - Replaces the old Hosted Mode
  - Use any browser for testing
  - Almost no turnaround time
  - Use existing tools (like Firebug) for debugging
- Code Splitting
  - Let GWT split Javascript into modules which are downloaded on demand
- Compile Reports
- ImageBundle abstracted into ClientBundle
  - Static files, CSS, images
- Standard themes

More info

- http://code.google.com/webtoolkit/doc/latest/DevGuideUiBinder.html
- http://code.google.com/webtoolkit/overview.html