GWT Presentation

Igor Polevoy

What is GWT?

- Google Web Toolkit
- Set of APIs
- Compiler
- No runtime – this is a browser

What is GWT…really?

- Java:
  - *.java -> javac -> bytecode .class
  - Runtime: JVM
- GWT:
  - *.java -> gwtc -> JavaScript + HTML
  - Runtime: Browser

Why Java to JavaScript?

- IDE support
- Huge development community (millions)
- Statically typed language – compiler can catch a lot of runtime errors (common in JS)
- Java is OO, JS is not
- JUnit – test your client code!

Benefits (besides obvious)

- Develop Rich Web Apps, utilize client CPUs
- GWT is ultimate Ajax
- Reuse tools, knowledge
- Build new generation of distributed apps
- GWT transmits a tiny fraction of data compared to traditional applications (should really be compared to real contenders – Java Applets/Flex/Silverlight)

Ever used Swing?

- GWT is not Swing, but it is like Swing
- There is an extensive library of “standard” UI components: buttons, labels, panels, trees, tables, stc.
- Programming model is a lot like Swing:

```
Panel p = new Panel();
p.add(new Button());
```

Hosted and Browser modes

- Hosted – bytecode execution
  - Advantage: use your favorite Java debugger!
  - Incremental compilation – just refresh a page and see the change
- Browser – “production” mode

Efficiency

- GWT compiler produces cross-browser compatible JS
- Generated JS is optimized to be as fast as it can be
- Result:
  - GWT apps are fast

GWT Pieces

- Compiler
- Hosted Browser plugin
- JRE emulation library (not all JDK supported, but getting better)
- GWT UI (and other) library

Disadvantages

- No SEO (example from MMH)
- RPC hard to debug
- If there is a need for a very specific look and feel, stock components might not do (generate specific HTML)
- Application is stateful, meaning that it is difficult to get to a screen that requires a lot of state for development/testing
- Not all of JDK supported (enough for large projects though)
- JSON is not as natural as in JS, parsing is clunky

GWT Programming Model

```
public void HelloExample() {
Button b = new Button(“Say Hello");
b.addClickHandler(new ClickHandler() {
   public void onClick(Widget sender) {
            Window.alert(“got clicked: ” + sender);
   }
   });
}
```

Ways to communicate with server

- GWT Way: RPC (this is text based – see in Firebug)
- RequestBuilder
  - Free form text
  - JSON
  - XML

GWT RPC – design by contract

Define Java interface

```
public interface EmailService extends RemoteService{
  List<MailMessage> readMailBox(String name) throws
   IllegalArgumentException; }
```

Define Asynch interface

```
public interface EmailServiceAsync {
  void readMailBox(String name, AsyncCallback<List<MailMessage>> async);
}
```

Provide server implementation

```
public class MailServiceImpl extends RemoteServiceServlet implements
   EmailService {
public List<MailMessage> readMailBox(String mailBox) {} }
```

Provide handler implementation

```
class AcceptMessagesAsync implements AsyncCallback<List<MailMessage>>
        public void onFailure(Throwable throwable) {..}
        public void onSuccess(List<MailMessage> messages) {..}
```

Non-RPC Server Communication

- This will completely abstract away AJAX in a platform-independent way:

```
RequestBuilder builder = new
  RequestBuilder(RequestBuilder.GET,url);

builder.sendRequest(null, new
  RequestCallback() {
  public void onError(Request request,
  Throwable exception) {...}

public void onResponseReceived(Request
  request, Response response) {...}
});
```

Non-RPC options

- JSON, XML, Free form text
- Advantages:
  - Transparent, super easy to debug with browser, curl, wget
  - Will support existing services
  - Services can support multiple types of client, GWT being one
  - Easy to use different versions of services (URL versioning)
- Disadvantages:
  - No type checking, runtime errors
  - needs extensive test coverage
  - No code sharing between server and client (could be a good thing:))
  - Supposedly less efficient

XML Parsing

```
Document doc = XMLParser.parse(xmlResponse);
Node job = doc.getElementsByTagName(“job”).item(0);
String from = ((Element)job).getAttribute(“type");
```

Bottom line: this is doable, but somewhat ugly. Cannot use XMLParser on the server side, meaning hard to write a unit test for this code.

This style of development is called DOM and is considered pretty heavy handed in Java.

XML Generation

```
Document doc = XMLParser.createDocument();
Element el = doc.createElement(“job”);
el.setAtribute(“type”, “batch”);
El.appendChild(el1);
```

..

JSON Processing

- There are two ways to process JSON in GWT:
  - Ugly
  - Elegant

JSON Ugly processing

Parsing:

```
JSONValue jsonValue = JSONParser.parse(responseText);
```

Classes:

```
JSONArray
JSONObject
JSONString
JSONValue
...
```

Code example:

```
==================
 if ((jsonArray = jsonValue.isArray()) != null) {
      for (int i = 0; i < jsonArray.size(); ++i) {
        TreeItem child = treeItem.addItem(getChildText("["
            + Integer.toString(i) + "]"));
        addChildren(child, jsonArray.get(i));
      }
    } else if ((jsonObject = jsonValue.isObject()) != null) {
      Set<String> keys = jsonObject.keySet();
      for (String key : keys) {
        TreeItem child = treeItem.addItem(getChildText(key));
        addChildren(child, jsonObject.get(key));
      }
    } else if ((jsonString = jsonValue.isString()) != null) {
```

JSNI – JNI for browser

```
// Java method declaration...
native String alert(String name) /*-{
   Window.alert(name);
}-*/;
```

JSON Elegant – use JSNI and overlay types

- Overlay type is a Java class type that represents a JSON object, extends `JavaScriptObject`.
- Example:

```
static class Customer extends JavaScriptObject {
protected Customer(){}//protected const.
public final native String getFirstName()
/*-{ return this.first_name; }-*/; //native method
...
}
```

Building UI

- Swing-like environment, but no Swing
- Panels, buttons, text, selects, radio, etc – many widgets

AbsolutePanel	DIV	ComplexPanel	Position widgets absolutely
CellPanel	TABLE	ComplexPanel	[Abstract] Subclass your own cell panels
ComplexPanel	-	Panel	[Abstract] Subclass panels with more than one widget
DeckPanel	DIV	ComplexPanel	Display widgets - one visible at a time
DisclosurePanel	TABLE	Composite	Show/hide a details pane
DockPanel	TABLE	CellPanel	Add widgets N, S, E, W around a central cell
FlowPanel	DIV	ComplexPanel	Add widgets as if to a normal DIV
FocusPanel	DIV	SimplePanel	Add focus to non-focusable widgets
FormPanel	DIV	SimplePanel	Submit a form to a server
Frame	IFRAME	-	Add an IFRAME to the application
HorizontalPanel	TABLE	CellPanel	Add a chain of cells horizontally
HorizontalSplitPanel	DIV	SplitPanel	Move the border between two cells
HTMLPanel	DIV		Add HTML, then access the elements that have IDs
Panel	-	Widget	[Abstract] Base for all panels
ScrollPanel	DIV	SimplePanel	Stack child widgets, displaying contents of only one
StackPanel	TABLE	SimplePanel	Add a vertical chain of widgets
SimplePanel	DIV	Panel	[Abstract] One-widget panel
TabPanel	TABLE	VerticalPanel	Add virtual card-index dividers
VerticalPanel	TABLE	CellPanel	Add a chain of cells vertically
VerticalSplitPanel	DIV	SplitPanel	Move the border between two cells

Panels – layout widgets

- AbsolutePanel Position widgets absolutely
- DeckPanel: Display widgets - one visible at a time
- DisclosurePanel : Show/hide a details pane
- DockPanel: Add widgets N, S, E, W around a central cell
- FlowPanel: Add widgets as if to a normal DIV
- FocusPanel: Add focus to non-focusable widgets
- FormPanel: Submit a form to a server
- HorizontalPanel: Add a chain of cells horizontally
- **HorizontalSplitPanel:** Move the border between two cells
- HTMLPanel: Add HTML, then access the elements that have IDs
- StackPanel: Add a vertical chain of widgets
- TabPanel: Add virtual card-index dividers
- VerticalPanel: Add a chain of cells vertically
- VerticalSplitPanel: Move the border between two cells

Widgets

- Button
- CheckBox
- FlexTable – grows as necessary
- Grid - fixed-size table
- HTML - contains raw HTML
- Hyperlink
- Image
- Label A one-line text box
- ListBox
- MenuBar
- MenuItem
- PopupPanel
- PushButton
- RadioButton
- RichTextArea
- SuggestBox
- TabBar use to make tab panels
- TextArea
- TextBox
- ToggleButton
- Tree
- …

Handling Events

```
ToggleButton button = new ToggleButton("read");
button.addClickHandler(new ClickHandler() {
     public void onClick(ClickEvent clickEvent) {
                ...
            }
});
```

Applying styles

Excerpt from: standard.css

```
.gwt-Button {
  margin: 0;
  padding: 3px 5px;
  text-decoration: none;
  font-size: small;
  cursor: pointer;
  cursor: hand;
  background: url("images/hborder.png") repeat-x 0px -27px;
  border: 1px outset #ccc;
}
.gwt-Button:active {
  border: 1px inset #ccc;
}
.gwt-Button:hover {
  border-color: #9cf #69e #69e #7af;
}
.gwt-Button[disabled] {
  cursor: default;
  color: #888;
}
.gwt-Button[disabled]:hover {
  border: 1px outset #ccc;
}
```

Setting element IDs

Set ID:

```
Button b = new Button();
DOM.setElementAttribute(b.getElement(), "id",
   “search_button")
```

Reference in CSS:

```
 #search_button { font-size: 100%; }
```

Would be nice to have:

```
b.setId(“search_button”);
```

Many ways to write GWT

- Entire rich app in one component
- Sprinkle GWT widgets of the same application on different locations on the same page
- Add multiple independent GWT applications to the same page

Communicating with DOM

- GET/SET element attributes
- Find elements by ID
- Insert before/after
- Scroll element into view
- GET/SET styles
- And many, others

External projects

- GWT-Ext – UI widgets
- GWT-SL – Publish Spring beans as RPC services to GWT client
- GWTiger
- GWT Mosaic

Questions?