GWT Training

Kamal Govindraj

TenXPerts Technologies

Agenda

- Day 1
  - Introduction to GWT
  - Setting Eclipse for development
  - Getting Started with GWT
  - Widgets and Custom Widgets
  - Applying CSS Styles to widgets
  - Event Management / Event Handlers
  - GWT Designer
  - UI Binder
  - Internationalization
  - Number and Date formatting
  - Resource Bundle / Client Bundle

Agenda

- Day 2
  - Modules
  - Deferred binding
  - Making remote calls – GWT-RPC
  - Integration with Spring
  - Timer / Deferred Commands
  - Logging
  - Testing GWT artifacts

Agenda

- Day 3
  - Setup / Getting started
  - Widgets
  - Layouts
  - Grid
  - Forms
  - Windows

Agenda

- Day 4
  - DataSource + GWT-RPC
  - Form Validation
  - Themes and Styling
  - GWT Integration
- Day 5
  - Working with JSON / XML
  - JSNI / JSO
  - History Management
  - Speed Tracer
  - Optimization
  - Good Practices while developing a large project

Introduction To GWT

Why GWT?

- Developing rich applications in javascript is hard
  - Browser incompatibilities
  - Dynamic programming language
  - In Depth knowledge of CSS, HTML, DOM etc
  - Lack of mature development tools
  - Lack of constructs for organizing large code bases

Why GWT?

- With GWT development is done in Java
  - Type safety – compiler catches many errors
  - Mature development tools – refactoring, auto complete etc..
  - Packages, Modules for organizing large code bases
  - Single language for both client / server side
- Swing like API – hides details of HTML / CSS / DOM

What does GWT provide?

- Developers write code in Java
- GWT compiler turns that into JavaScript
- Generates different version of JavaScript for each browser
- Optimizes generated Javascript
- Provides easy way of Internationalization
- Simplifies AJAX

Features

- Compiler / Tools
  - Takes Java code and compiles it into javascript
- RPC
  - Provides communication between javascript client and Java Server
- Widget Library
  - Components to build complex UI
- Event Handling
  - Add dynamic behavior

Lab – Development Environment Setup

Getting Started With GWT

- Development can be done by using the SDK
  - Provides command line tools to create new project and other artifacts
- Or using Maven plugin
  - Compile, Build and Run GWT apps
- Or by using the Google Eclipse Plugin
  - Provides extensions to create GWT project and other artifacts
  - Comes bundled with SDK can be configured to use other versions

Demo – Create a new GWT app in eclipse

Directory Structure

```
-src
   tenx
      gwt
        HelloWorld.gwt.xml  (module descriptor)
        client              (all client side classes)
           HelloWorld.java    (EntryPoint)
         shared
         server
 -war
       HelloWorld.html        (HostPage)
       HelloWorld.css
       WEB-INF
          lib
```

- Module file
  - .gwt.xml file contains all GWT specific config
  - Defines dependencies on modules, EntryPoint etc
- src - Java source code
  - Client – all UI logic
  - Shared – classes shared by client / server (dtos etc..)
  - Server – Code that runs on the server
- War – web resources (HTML, Images, CSS)

Module descriptor

- Inherit tag – refers to dependencies, other GWT module that are used
- The client classes can only depend on the inherited classes
- The EntryPoint tag refers to the main method that will get invoked

```
<module>
        <inherits name="com.google.gwt.user.User" />
        <source path="client" />
        <entry-point class="tenx.gwt.helloworld.client.HelloWorld" />
</module>
```

Host Page

- Includes reference to the javascript file generated
- Has placeholder for hooking up content

```
<head>
		<script type="text/javascript" language="javascript"
src="helloworld/helloworld.nocache.js"></script>
</head>
<body>
		<div content="main"/>
</body>
```

EntryPoint class

- Starting point of execution, onModuleLoad invoked once host page is loaded
- Gets access to one or more DOM elements and add GWT content

```
public class HelloWorld implements EntryPoint {
        @Override
        public void onModuleLoad() {
                RootPanel.get("content")
                        .add(new Label("Hello World"));
        }
}
```

Running the app.

- Compile - creates the javascript files. The app can then be packaged as regular war and deployed to web container
- To speed up development GWT offers dev/hosted mode support
  - Java code is directly run in browser via an emulator plugin
  - Also runs an embedded web container for server side code

Lab – 01-HelloWorld.

Building the UI

UI Elements

- GWT provides a widget library
- Simple Widgets are thin wrappers on top of HTML controls – Button, TextBox, CheckBox etc..
- Panels are specialized widgets that can contain one or more Widgets – HorizontalPanel, VerticalPanel, TabPanel HTMLPanel, FlexTable etc..

Example UI creation logic

```
Button createButton = new Button(“Create”);
Button deleteButton = new Button(“Delete”);
Button editButton = new Button(“Edit”);

HorizontalPanel toolBar = new HorizontalPanel();
toolBar.add(createButton);
toolBar.add(deleteButton);
toolBar.add(editButton);

toolBar.setSpacing(10);
toolBar.setWidth(“100%”);
```

Custom Widgets

- To create new widgets / customize widgets we can extend Widget class
- To create composite widget that combine multiple other widgets extend from the Composite.
- Composite is preferred vs extending from a specific widget class – if the customizations are significant

Styling

- Each Widget type has a set of CSS classes defined
- This can be used to change appearance of all instances of a widget type (.gwt-Label)
- Different themes available out of the box.
- To style individual instances – add css style to widget and specify in CSS file
- Or add the style properties directly

Styling example

```
Button button = new Button(“some button”);
button.addStyleName(“large-button”);

button.getElement().getStyle().setBackGroundColor(“red”)

button.getElement().getStyle().setProperty(“border”,”...”);

// in CSS file
.gwt-Button {
}

.large-button {

}
```

Lab - 02-Widgets

Event Handling

Events

- Events are raised when the user performs an action or by the application code
- Events raised by the browser are Native Events – ClickEvent, BlurEvent etc
- Events raised by GWT widgets or application code are LogicalEvents – SelectionChangedEvent
- Each Event is defined by a class – which exposes data about the event

Event Handlers

- A Handler interface is associated with each event
- To react to an event – provide implementation of the associated Handler
- Handler has onEvent method which is invoked when the event occurs
- Register the Handler with the Widget
- To unregister store the HandlerRegistration and call remove on it

Event Handler example

```
Button refreshButton = new Button();
refreshButton.addClickHandler(new ClickHandler() {
		public void onClick(ClickEvent clickEvent) {
			// Add code to handle event
		}
});
```

Defining New Event

- To define a custom event – implement the Event class and the Handler interface
- Event class should extend GwtEvent
- Handler class should Extend EventHandler (marker interface)
- The Widget can use the HandlerManager to manage registrations

```
interface ItemCreatedHandler extends EventHandler {
            public void onItemCreated(ItemCreatedEvent e);
}


class ItemCreatedEvent
            extends GwtEvent<ItemCreatedHandler> {
  //  event specific data and accessors
   private static Type<ItemCreatedHandler> TYPE
            = new Type<ItemCreatedHandler>();


   public static Type<ItemCreatedHandler> getType() {
     return TYPE;
    }

    protected void dispatch(ItemCreatedHandler handler) {
            handler.onItemCreated(this);
    }
}
```

```
Class MyWidget extends CompositeWidget {
        HandlerManager handlerManager
                        = new HandlerManager(this) ;

        public HandlerRegistration addItemCreatedHandler(
                        ItemCreatedHandler handler) {
                return handlerManager.addHandler(
                        ItemeCreatedEvent.type(),
                        handler
                );
        }

        public void someMethod() {
                handlerManager.fireEvent(
                        new ItemCreatedEvent());
        }
}
```

Lab - 3-events

UI Binder

Problems with programmatic UI

- Constructing UI programmaticly is hard for complex UI
- Not easy to translate input from UI/Graphic designers to code
- Not easy for UI designers to tweak the layout / styling

- UI Binder helps by providing a way to define the UI using XML
- Can directly embed HTML content
- HTML content can have embedded GWT widgets
- Get access to Widgets for programmatic control using @UIField
- Add handlers for widgets using @UIHandler

UI Binder template example

LoginForm.ui.xml

```
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
        xmlns:g="urn:import:com.google.gwt.user.client.ui">
        <g:HTMLPanel>
                User Name : <g:TextBox ui:field=”userName”/>
                Password : <g:TextBox ui:field=”password”/>
        </g:HTMLPanel>
</ui:UiBinder>
```

Associated class

```
class LoginForm extends Composite {

        interface Binder extends
                UIBinder<Widget,LoginForm>{};
        static Binder uiBinder = GWT.create(Binder.class);

        // names should match ui:field attribute in xml
        @UiField TextBox username;
        @UiField TextBox password;

        LoginForm() {
                initWidget(uiBinder.createAndBindUi(this);
        }

}
```

Lab - 04-ui-binder

Internationalization

- Applications need to support multiple languages
- All textual content should be externalized to property files
- Based on the locale the value from the appropriate version should be picked up
- As an optimization only the strings for the selected language should be downloaded

- GWT supports two ways of doing this – static, dynamic
- In static approach we define an interface with methods for each property and then define values for each property in a property file
- The property references are resolved at compile time

- We can use Constants or Messages
- Constants are fixed values without any dynamic input – they can be of type String, int, double, String[] or Map
- Messages are values with place holders that are replaced with values at run time

Constants example

```
interface I18NContstants extends Constants {
          String labelUserName();
          String labelPassword();
}

// I18NConcstants.properties
LabelUserName = User Name
LabelPassword = Password

// Usage
I18NConstants i18Contants
          = GWT.create(I18NConstants.class);

Label label = new Label(i18NConstants.labelUserName());
```

Messages Example

```
interface I18NMessages extends Messages {
        String greeting(String name);
}

// I18NMessages.properties
Greeting = Hello {0}, how are you?

// Usage
I18NMessages i18NMessage
        = GWT.create(I18NMessages.class);

Label label = new Label(118NMessage.greeting(“Kamal”));
```

Different locales

- For each supported locale a corresponding property file should be present. I18NMessager_de.properties
- Locale can be selected by passing a query parameter – locale=de
- It will first search for specific locale – if not present will fall back to default locale

Changes to Module xml

```
<inherits name='com.google.gwt.i18n.I18N'/>
<!-- supported locales -->
<extend-property name='locale' values='en_US,de'/>
<!-- fallback locale -->
<set-property-fallback name='locale' value='en_US'/>
```

Number / DateFormat

- Numbers and Date formats vary across languages / regions
- Code need to make sure appropriate format should be used
- GWT provides NumberFormat and DateTimeFormat classes that will help achieve this
- Default formats that adapt to locale available. Or it can be customized by specifying patterns.
- Can't use JDK version – not supported by GWT JRE emulation

Example

```
NumberFormat currencyFormat
          = NumberFormat..getCurrencyFormat();
currencyFormat.format(10000.50);

// $10,000.50 for en_us locale
// 25.000,00 € for de locale

DateTimeFormat timeFormat =
          DateTimeFormat.getFormat(DATE_TIME_SHORT);
timeFormat.format(new Date());
```

Lab – 05- i18n

Deferred Binding

Deferred binding

- Deferred binding is for generating / customizing code based on various parameters
- It is a replacement for Java reflection
- Used extensively by GWT
- Can implement our own – but very rarely required
- Replace class with different impl. Or generate a new class using generator

Advantages

- Reduces download size by customizing generated code based on browser, locale etc
- Generates boiler plate code – e.g., RPC Proxy
- Done at compile time – no run time overhead like reflection

replace-with

- Replaces implementation with another class based on configurable attributes

```
<!-- add to module.xml -->
<replace-with class="PopupImplMozilla">
    <when-type-is class="PopupImpl" />
      <when-property-is name="user.agent" value="gecko"/>
  </replace-with>

<replace-with class="PopupImplIE6">
    <when-type-is class="PopupImpl"/>
    <when-property-is name="user.agent" value="ie6" />
  </replace-with>
```

replace-with sample code

```
// Usage – GWT.create method will return
// instance of the appropriate sub-class
PopupImpl impl = GWT.create(PopupImpl.class);

class PopupImplMozilla extends PopupImpl {
      // override and customize methods
}

class PopupImplIE6  extends PopulImpl {
      // override and customize methods
}
```

Generators

- generate-with is another way of deferred binding
- A generator is invoked during compilation to create a new class
- Generators associated with specific interfaces
- Generator is invoked when GWT.create is called with an interface that is a sub type of the associated interface

generate-with definition

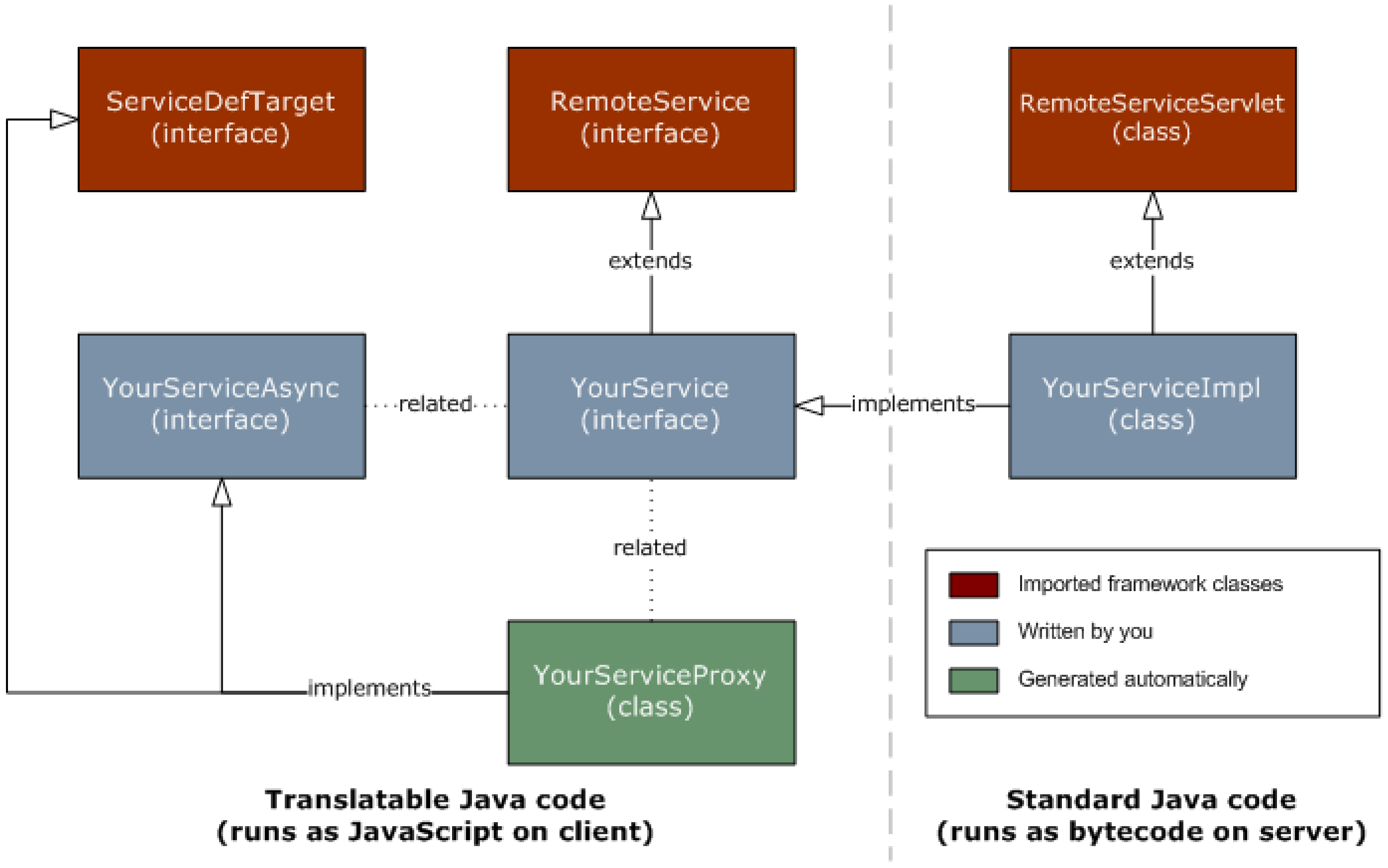
```
<!-- generator definition for RPC proxy creation -->
<generate-with
     class="com....ServiceInterfaceProxyGenerator">
   <when-type-assignable
     class="com....rpc.RemoteService" />
 </generate-with>
```

Demo – Review generated code

GWT-RPC – communicating with server

- An easy way of communicating with the server
- provide interface on the client side – extends RemoteService
- Provide implementation – a servlet on the server side – extends RemoteServlet
- Use GWT.create to create a client side proxy
- Serializes input javascript objects on client, deserializes into Java objects on server

- Client calls are Aynchronous – need to provide a Callback. Method returns immediately – doesn't wait for response
- Once response is received – the methods (onSuccess, onFailure) of the callback are invoked
- Built on top of AJAX support provided by javascript
- Javascript is single threaded – can't block for response (synchronous) – will hang UI



ServiceDefTarget (interface)
RemoteService (interface)
RemoteServiceServlet (class)
extends
extends
YourServiceAsync (interface)
related
YourService (interface)
implements
YourServiceImpl (class)
related
Imported framework classes
Written by you
Generated automatically
implements
YourServiceProxy (class)
Translatable Java code (runs as JavaScript on client)
Standard Java code (runs as bytecode on server)

Service interface

```
@RemoteServletPath(“greetingService.rpc”)
interface GreetingService extends RemoteService {
    String getMessage(String name);
}

// create an Async interface which will be used by
// client code – same package interfacenameAsync

interface GreetingServiceAsync {
    void getMessage(String name,
        AsyncCallback<String> callBack);
}
```

Service Implementation

```
class GreetingServiceImpl extends RemoteServlet
        implements GreetingService {

    String getMessage(String name) {
        return “Hello, ” + name;
    }
}
```

Web.xml

```
<servlet>
    <servlet-name>greetingService</servlet-name>
    <servlet-class>GreetingServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>greetingService</servlet-name>
    <url-pattern>greetingService/greetingServie.rpc
    </url-pattern
</servlet-mapping>
```

Client code

```
GreetingServiceAsync greetingService
    = GWT.create(GreeintService.class);

greetingService.getMessage(“kamal”,
    new AsyncCallback<String> () {
        public void onFailure(Throwable t) {
            // called in case of failure
        }
        public void onSuccess(String result) {
            // result is return value of service
            // call – use it here
        }
    });
```

Requirement

- All input / return type need to be Serializable
- Should have no argument constructor
- Any change to class structure (addition / removal of fields) – both server / client needs to be recompiled

Lab 06-gwt-rpc

Spring Integration

- Remote service implementation have dependencies
- Implementing it as a servlet makes it hard to manage dependency
- We need a way to turn these into regular spring beans
- GWT-SL library provides the glue to achieve this. Extends spring MVC

```
<beans>
      <bean id=”greetingService”
            class=”GreetingServiceImpl”>
            <!-- specify dependencies -->
      </bean>

      <bean class=”org.gwtwidgets....GWTHandler”>
        <property name=”mappings”>
          <map>
            <entry key=”/helloworld/HelloWorld.rpc”
                   Value-ref=”greetingService”/>
          </map>
         </property>
      </bean>
</beans>
```

Lab 07-spring-integration

Misc – Logging, Timer, Deffered Command..

GWT logging support based on JDK util logging API

Inherit logging module and add configuration to module xml

Can be enabled / disabled for various levels at compile time. Logging code will be removed completely if it is not enabled

Various handlers available – devmode, stdout, popup, firebug etc

Logging configuration

```
    <inherits name="com.google.gwt.logging.Logging"/>
    <set-property name="gwt.logging.logLevel" value="INFO"/>
    <set-property name="gwt.logging.enabled" value="TRUE"/>

<!-- enable / disable handler - where log should be writtern-->
   <set-property name="gwt.logging.consoleHandler"
               value="ENABLED"/>
    <set-property name="gwt.logging.developmentModeHandler"
               value="ENABLED" />
    <set-property name="gwt.logging.popupHandler"
               value="DISABLED" />
    <set-property name="gwt.logging.systemHandler"
               value="DISABLED" />
    <set-property name="gwt.logging.firebugHandler"
          value="ENABLED" />
    <set-property name="gwt.logging.simpleRemoteHandler"
          value="DISABLED" />
```

Timer

Used to execute some action after a delay or repeatedly at regular intervals

Example refresh page every 5 seconds or for time outs

```
// Set up time in onLoad method
Timer refresh = new Timer() {
      public void run() {
            // Do somethig
      }
}
refresh.scheduleRepeating(5000);

// Cancel in unLoad method
refresh.cancel();
```

DeferredCommand

- DeferredCommand deprecated – replaced with scheduler
- Scheduling an action to be executed after the current event handler are processed
- Provide an implementation of a ScheduleCommand or RepeatingCommand
- Process asynchronously – don't want to hold up
- Breakup a long running process into smaller steps – so that the UI doesn't hang

Lab – logging, timer and deferred command

SmartGWT - Introduction

What is SmartGWT

- SmartGWT is a widget library
- Has comprehensive set of widgets (ListGrid, Tree, Menu, Editable Grids ..)
- Is based on SmartClient JavaScript framework
- Open source version – most of the client side features
- Pro & Enterprise version provide Server side connectors & some advanced features

Advantages

- Very rich set of widgets – most of the applications needs are met out of the box
- Good looking out of the box – professional styling, looks good without requiring any styling
- Provides some features missing in GWT – data binding in list / grids
- Integration with XML webservices, RESTful webservices

Disadvantages

- Not native GWT – it is just wrappers around SmartClient javascript framework
  - Causes some issues when mixed with GWT widgets
  - Difficult to debug - can't view object state while debugging
- Heavy – couple of MB download first time. Not an issue for Enterprise Apps. Can be tweaked
- Some of the GWT optimizations don't apply
- Many things are done differently – server communication for e.g.,

Approaches

- Use SmartGWT as the main widget library – limited use of GWT widgets
- Or Embedd SmartGWT widgets selectively while developing mainly with GWT widgets
- Depends on the application requirement – richness, performance etc.

Demo – SwartGWT showcase

SmartGWT - widgets

Getting started

- Include the smart-gwt jar in the classpath
- Inherit it into the application modules

```
<!-- includes all the widgets, JS, CSS/Images
 files for the default theme-->
<inherits name=”com.smartgwt.SmartGwt”/>
```

- Provides similar Layout / Panels as GWT – Hlayout, VLayout, Window etc
- ListGrid – sort, filter, scrollable, editable. Support complex form items in editable mode
- DynamicForm – complex form layouts (multi column, grouping etc)
- DataBinding – automatically copy data into controls and back.

Canvas

- Base class for all SmartGWT widgets
- Most of the complex widgets expect their members to be of type Canvas
- WidgetCanvas – provides a bridge to use GWT widgets where Canvas is required

Layouts

- Hlayout / Vlayout – layout widgets side-by-side or one-below-another
- Need to set size – either in percentage terms or pixels
- addMember to add child widgets – (there is confusing addChild as well that doesn't work)

ListGrid

- For creating a scrollable, editable, sortable list
- Supports both client side / server side sorting
- Features (sorting, column resizing) can be turned on / off using setCanXXXX methods
- Setup by passing a list of ListGridItems
- Data is set by creating ListGridRecords

ListGrid example

```
// first parameter name an internal identifier – can't have spaces
// second parameter is title for display
ListGridItem nameItem = new ListGridItem(“name”,”Name”);
ListGridItem address = new ListGridItem(“address”,”Address”);

ListGrid personsGrid = new ListGrid();
personsGrid.setFields(nameItem,address); // var arg method



// For setting data – create array of ListGridRecord
ListGridRecord[] records = new ListGridRecord[2];
// attr name should match – item name
record1.setAttribute(“name”,”Kamal”);
records[0] =record1;

personsGrid.setData(records);
```

Form

- Forms with sophisticated layout facilities
- Supports a wide range of controls (spinner, slider, date – different variants etc)
- Create a DyanmicForm and then configure it with list of FormItems

DynamicForm example

```
DynamiForm personForm = new DynamicForm();
ButtonItem saveButton = new ButtonItem(“save”,”Save”);
// register click handler

personForm.setItems(new TextItem(“name”,”Name”),
          new DateItem(“dob”,”Date of Birth”),
          saveButton);

personForm.setValue(“name”,”Kamal”);
personForm.setValue(“dob”,new Date());
```

Lab – SmartGWT widgets

SmartGWT - DataSource

- DataSource is the key component behind SmartGWT data binding capability
- They can be used to link together multiple components (List + Form – master detail)
- DataSource defines metadata all the fields. It can also be set up to make server calls to fetch data

- While using DataSource with ListGrid , Dynamic form no need to explicitly set up fields
- Field setup only required to customize presentation aspect
- Changed made via UI are save back by invoking appropriate methods on DataSource
- Filtering, Sorting will also trigger calls on datasource

```
class PersonDataSource extends DataSource {
  public PersonDataSource() {
    DataSourceField id = new DataSourceIntegerField(
        “id”, “ID”
      );
     id.setHidden(true);
    DataSourceField name = new DataSourceTextField(
           “name”,”Name”
           );
     setFields(id,name);
  }
}
```

```
ListGrid listGrid = new ListGrid();
listGrid.setDataSource(PersonDataSource.getInstance());

// No need for setting up fields explicity, will be
// picked up from DataSource

// Trigger fetch when list needs to be drawn
listGrid.setAutoFetch(true);

// or explicitly call fetch – for e.g., a search form
listGrid.fetchData();
```

- If a list and Form use the same DataSource – any changes made in one will be synced with the other
- As selection changes in the List the form will reflect the values of that row
- This is useful for Master-Detail use cases
- For forms also the setup is similar to how it was done for ListGrid

Lab – SmartGWT data sources

SmartGWT – RPC DataSource

- Setting records onto grid or datasource directly not optimal – especially for cases involving more that 25 rows
- Need to delegate sorting, filtering, scrolling/pagination to server.
- Fetching all the rows will lead to poor response time as well as high memory usage
- Use datasource with RPC or REST / WS integration

- RPC support not provided as part of SmartGWT bundle
- SmartGWT provides some lower level API that can be used to build such capability on top of DataSource
- There are 3rd party implementations available – which have been widely used. Work for common cases – issues when using editable gids

- Whenever an action to fetch / update / delete / add is performed the transformRequest method is called
- Passed DSRequest parameter which has requestId, sort desc, filter criteria, start / endRow , operation type etc
- Create a DSResponse and call processReponse with requestId
- Based on operation type invoke different rpc methods

Data Source methods

- fetchData call on the ds or associated data bound control – results in FETCH operation
- addData – results in an ADD
- updateData – results in an UPDATE
- removeData – results in REMOVE

GWTRPCDataSource

- An implementation is available in smartgwt-extensions module. Abstract class with 4 methods for fetch, update, add and remove

```
public class PersonDataSource
        extends GWTRpcDataSource {

        void executeFetch (String requestId,
                DSRequest request,
                DSResponse response)
        // executeAdd, executeUpdate
        // and executeRemove
}
```

Fetch

```
int start = dsRequest.startRow();
Int endRow = dsRequest.endRow();

serviceAsync.fetchPersons(start,endRow,AsyncCallBack.. {
        ….
        public void onFailure(Throwable t) {
                dsResponse.setStatus (
                        RPCResponse.STATUS_FAILURE);
                processResponse(requestId,dsResponse);
        }
        public void onSuccess(PaginationResult r) {
                dsResponse.setTotalRows(..);
                dsResponse.setData(..);
                processResponse(requestId,dsResponse);
        }
});
```

Lab – SmartGWT RPC Data Source

SmartGWT - Validation

Simplifying validation

- Provides support for attaching validators with DataSourceFields, ListGridFields or FormItems
- Default type validation based on type specified while creating field
- setRequired property to specify mandatory fields
- Add other validators – IntegerDataRange,RegularExpressionVali dator etc..

When it is triggered?

- Validation can be triggered explicitly
- Or happens implicitly when save operation is invoked
- Or can be setup to be performed on value change event
- Server calls also can return errors – in specific format which can then be easily displayed on the form

Example

```
DataSourceField ageField = new DataSourceIntegerField();
ageField.setRequired(true);
IntegerRangeValidator ageRangeValidator = new …
ageRangeValidator.setMin(18);
ageRangeValidator.setMax(100);

ageField.setValidators(ageRangeValidator);

// The above will catch type conversion errors,
// missing value error and invalid values
```

Lab SmartGWT Form Validation

SmartGWT Themes

Changing UI look & Feel

- Themes are collection of UI artifacts (CSS, Images etc)
- Comes bundled with 3 themes
- More can be added by using smart-gwt-skins.jar
- Can also be customized – typically using the closest matching one as the base version

Using a different theme

- By default configured with Enterprise theme
- To use other theme move to SmartGWTNoTheme module and inherit the theme specific module after it

```
<!-- <inherits name=”com.smartgwt.SmartGWT>-->
<inherits name=”com.smartgwt.SmartGWTNoTheme/>
<inherits name=”com.smartclient.theme.grpahite.Graphite”/>
```

Creating new theme

- To create new theme – extract the files under one of themes folder to your src tree
- Rename the package and also the module files
- Modify the path in the module.gwt.xml
- And public/<...>/load_skin.js and change

  isc.Page.setSkinDir variable

Chaning styles in new theme

- Change skin_style.css file or the images under images folder.

Lab – SmartGWT themes

JSNI

JSNI

- We might need to access other javascript code form within GWT managed code
- For using some utilities (encode, encrypt etc) which are not supported in GWT
- Use some features of JavaScript library – like SmartGWT using smartclient
- Expose some GWT implemented functionality to other javascript
- JSNI feature of GWT allows for two way communication – GWT code to javascript and vice versa

- Leverages keywords and conventions used by JNI (for accessing native code from Java)
- Define a method with native keyword – body enclosed in /*-{ }-*/
- Body is javascript code, can access java variable / functions by using a pattern

```
[instance-expr.]@class-name::method-name
          (param-signature) (arguments)

this.@tenx.gwt.JavaScriptHelper::forma(D)(x)
```

Parameter signature

- Based on JNI conventions

Type Signature	Java Type
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L fully-qualified-class ;	fully-qualified-class
[type	type[]
(arg-types) ret-type	method type

To access javascript from GWT

```
<script>
        function sayHello(name) {
                alert(“Hello, “ +name)
        }
</script>

// GWT code
public static native sayHello(String name) /*-{
        $wnd.alert(“Hello “ + name);
-*/}
```

To access GWT from javascript

```
public native void publish(String input) /*-{
        $wnd.encode
                = this.@tenx.gwt.Helper::encode(input)
        }
}-*/

OnModuleLoad() {
        publish();
}

// Java script code
<script ….>encode("xyz");</script>
```

Labl JSNI

History Management

History mgmt. in AJAX apps.

- Managing browser history is very critical
- Things like back / forward button, refresh, bookmarking won't work if it is not handled explicitly
- While using AJAX the base url can't be changed – another segment can be appended followed by # key
- Every significant action the app should make sure that the # tag associated with url changes

History mgmt. In AJAX apps.

- Browser will track the changes
- On pressing back button it will set the url with the previous #tag
- Application needs to process this and initialize the state accordingly

Demo

GWT History support

- Makes it simple
- Register a ValueChangeHandler
- OnValueChange will get called when the browser url changes, the string following hash tag is part of the event (token)
- The code can examine token to update UI
- To add new entry History.newItem
- Token can be constructed with base part identifying main UI and parameters

History Handling example

```
History.addValueChangeHandler(new ValueChangehandler(){
        public void onValueChange(ValueChangeEvent e) {
                String token = e.getValue();
                If (token.startWith(“accounts”)) {
                        doShowAccountsList();
                } else if (token.startsWith(“accDetail”)) {
                        doShowAccountDetail();
                }
                else {
                        doShowHome();
                }
        }
});

To generate new entry
History.newItem(“accDetail:accId=1001”);
```

Lab – History Management

XML / JSON services

Why we need XML / JSON

- To integrate with legacy apps which already expose an XML or JSON API
- If the server side implementation is not java
- If mutiple types of clients need to be supported by server (GWT web UI, Android, iPhone etc..)

RequestBuilder

- GWT provides RequestBuilder component as a wrapper on top of XMLHttpRequest (the basis of most AJAX implementations)
- Can use it to make HTTP calls
- set headers, request body for POST requests
- Callback returns Response – with headers, body text..

```
RequestBuilder rb = new RequestBuilder(GET,
          GWT.getModuleBaseUrl() + “userService”);

rb.setCallBack(new RequestCallback() {
          public void onResponseReceived(request,response) {
                    response.text(); // has the data, need to convert
                                             // to java objs
          }
          public void onError(request,throwable) {
          }
});

rb.send();
```

Handling json response

- Can be manually copied into model objects – after parsing it into JSONObjects
- Can use Overlay types to get the data as Java Objects.
- Overlay types are value objects that extend JavaScript object and have the get / set methods as native

```
// data
[{name:'Kamal',address:'Bangalore'},
{name:'Vineeth',address:'Bangalore)]'

// overlay type
Class Person extends JavaScriptObject {
        protected Person() {
        }
        public native String getName() /*-{
                return this.name;
        }-*/}
}
```

Conversion code

```
JSArray[Perfon] asPersonArray(String input) /*-{
        return eval(input);
}-*/
```

XML support

- There is also some minimal support for consuming XML
- DoM parser that parses and extracts the Dom (nodes, attributes and child nodes)

```
document doc = XMLParser.pare(input);
document.getElementByTagName(“person”);
```

Lab – consuming XML / JSON services

MVP

- Large complex / apps – code becomes spaghetti if not organized correctly
- Mixing of logic with UI makes it difficult to write automated tests
- GWT provides option to write tests that test the view as well – that is very slow and not usable

MVP

- MVP – Model View Presenter is a pattern for structuring GWT code
- View should only have presentation logic – like layout, styling etc
- Presenter is the active component – has all the logic (history, RPC services...)

Lab – MVP + Testing

Code Splitting

- Most applications have multiple logic parts (accounts, credit cards, customer service ...)
- Users tend to use a small subset of those parts
- For larger applications download speed becomes an issues
- Need to break up the javascript into smaller parts based on logical grouping

- We indicate the starting points of those grouping and GWT takes care of the rest of the work
- Code split point indicated by wrapping the call in a runAsync call
- GWT compiler analyses the code dependencies from all the points and generate multiple smaller js files

- GWT by way of compile-report provides info to decide on split points and also element non required dependency
- To generate compile report run with -compileReport
- Provides detailed stats – current split points, size of initial download and subsequent download for each split point
- Using MVP + Events + History help by untangling the code – makes it easier to break into independent groups

- If download time is important – need to take care of it right from the beginning
- Adopted approaches that don't introduce tight coupling between different classes

Lab – code splitting

Speed Tracer

Thank You