Google Web Toolkit

Jaroslav Dytrych

Faculty of Information Technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

dytrych@fit.vutbr.cz

25 November 2020

Contents

- Introduction
- Development modes
- User interfaces
- RPC
- Internationalization
- Bookmarkable history
- JSNI (JavaScript Native Interface)

Introduction

- GWT is an open source Java development framework for creating AJAX applications.
  - GWT takes Java code written against a special API and converts it into browser-runnable AJAX code.
  - Subset of Java SE features is available on the client side
    - double arithmetic, emulated long,
    - no multi-threading,
    - subset of JRE libraries.
- Optimized JavaScript
  - generates "permutations" for each browser/locale,
  - Java SE libraries emulated (different implementations for different browsers).
- Development mode ("hosted"), develop/debug applications in the Java environment.
- All is typically on one "Host page".
- Programming model similar to desktop applications.
  - Widgets, Events
  - It is not possible to access the disc of the client.
  - All dependencies must be explicitly declared (`.gwt.xml`).

GWT Application Structure

- `war/`
  - `Hello.html` .... Host HTML Page
  - `Hello.css` .... static CSS stylesheet
  - `hello/` .... compiled JS
    - `hello.nocache.js` .... bootstrap script
  - `WEB-INF/`
    - `web.xml` .... servlet configuration
    - `classes/` .... server side (servlets)
    - `lib/` .... library dependencies
      - `gwt-servlet.jar` .... GWT-RPC servlet
- `src/`
  - `hello`
    - `Hello.gwt.xml` .... application module definition
    - `*.jpg,...` .... static resources
    - `client/` .... client side source files
      - `Hello.java` .... entry point class source
      - `HelloService.java` ... RPC Service interface
    - `server/`
      - `HelloServiceImpl.java` .. RPC Service impl.

GWT Application Structure – Maven

`src/main/`

- `java/cz/vutbr/fit/gja`
  - `Hello.gwt.xml` .... application module definition
  - `client/` .... client side source files
    - `Hello.java` .... entry point class source
    - `HelloService.java` .... RPC Service interface
  - `server/`
    - `HelloServiceImpl.java` .... RPC Service implementation
- `webapp`
  - `WEB-INF/`
    - `web.xml` .... servlet configuration
  - `Hello.html` .... Host HTML Page
  - `Hello.css` .... static CSS stylesheet

`target` .... build outputs

- `HelloWorld` .... compiled module
  - `HelloWorld` .... compiled JS

`war`

- `WEB-INF` .... compiled Java classes

`pom.xml` .... Maven Project Object Model

Application structure

- Client-side code
  - actual Java code implementing the business logic,
  - translated to JavaScript,
  - sources declared as `<source path="path"/>` in `appName.gwt.xml`.
  - Typically all in one HTML page (Host Page).

- Server-side code
  - optional,
  - backend processing,
  - implemented in servlets,
  - integrated with client-side code.

HTML Host Page

- Compiled GWT modules are stored as JavaScript files.
- The page contains the bootstrap script:
  - ```
    <script language="javascript"
        src="hello/hello.nocache.js"></script>
    ```
- Page may contain „slots“ (but it is not necessary)

```
<html>
  <head>
    <script
    language="javascript"
    src=
     "hello/hello.nocache.js">
    </script>
  </head>
  <body>
    <table align=center>
      <tr>
        <td id="slot1"></td>
        <td id="slot2"></td>
      </tr>
    </table>
  </body>
</html>
```

```
final Button button =
  new Button();
final Label bLabel =
  new Label();

RootPanel.get("slot1").add(button);
RootPanel.get("slot2").add(bLabel);

or

RootPanel.get().add(
  new MyWidget());
```
```

Hello World Entry Point

Hello.java

```
public class Hello implements EntryPoint {
  @Override
  public void onModuleLoad() {
    Button b = new Button("Click me", new ClickHandler() {
      public void onClick(ClickEvent event) {
        Window.alert("Hello, World!");
      }
    });
    RootPanel.get().add(b);
  }
}
```

Modules

- Modules (libraries) are similar to packages.
- Configuration bundle (`.gwt.xml`)
  - Inherited modules (used libraries)
  - `User` module
    - interface `EntryPoint`
    - `void onModuleLoad()`
  - Many inherited modules increase compilation time significantly.
    - files are copied to the module output.
    - Access to the resources: `GWT.getModuleBaseURL() + "foo.png"`
  - Deferred binding rules
    - different implementations for different browsers/locale/...
    - `GWT.create(Foo.class)`

Hello World Module

Hello.gwt.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='hello'>
  <inherits name='com.google.gwt.user.User'/>
  <inherits
    name='com.google.gwt.user.theme.standard.Standard'/>
  <entry-point class='hello.client.Hello'/>
  <source path='client'/>
  <source path='shared'/>
  <replace-with
      class="com.google.gwt.user.client.ui.impl.PopupImplMozilla">
    <when-type-is
        class="com.google.gwt.user.client.ui.impl.PopupImpl"/>
    <any>
      <when-property-is name="user.agent" value="gecko"/>
      <when-property-is name="user.agent"
          value="gecko1_8"/>
    </any>
  </replace-with>

// in the implementation...
private static final PopupImpl impl =
    GWT.create(PopupImpl.class);
```

GWT standard modules

Logical name	Contents
com.google.gwt.user.User	Core GWT functionality
com.google.gwt.http.HTTP	Low-level HTTP communication library
com.google.gwt.json.JSON	JSON creation and parsing
com.google.gwt.junit.JUnit	JUnit testing framework integration
com.google.gwt.xml.XML	XML document creation and parsing

Logical name	Contents
com.google.gwt.user.theme.chrome.Chrome	Chrome theme
com.google.gwt.user.theme.dark.Dark	Dark theme
com.google.gwt.user.theme.standard.Standard	Standard theme

Deployment

- Create WAR file
  - Go to the `war` directory in the project structure.
  - Select all files and folders.
  - Zip it.
  - Rename archive to `.war`.
- Deploy WAR
  - Place `.war` to the application server directory for applications.
  - Appropriate directory should be created from the `.war`.
- Run application
  - Enter `http://localhost:8080/<app name>` to your browser.

Deployment in NetBeans

- NetBeans 7.x and 8.x
  - Install plugin GWT4NB `http://plugins.netbeans.org/plugin/70885/gwt4nb-for-nb8-1` (rename the file if necessary – `.nbm`).
  - Run it as other web application projects.
- Apache NetBeans 12.0+
  - Run it as other web application projects.

Development mode

- Run as Java, does not compile to JavaScript.
- Single JVM session
  - single debugger for both server and client side code.
- Browser plugin
  - controls the browser from the remote code server,
  - causes the code server to recompile on refresh.

Development mode

IDE

Your static resources

Your servlets

Your WAR

Your client-side source

HTTP Server

Code Server

(1)

(2)

(3)

(4)

(5)

Developer Plugin

Browser

Typical development mode session (a single JVM session)

port 9997 (Jetty)

Development mode

- Development mode is deprecated.
- Only (really) old browsers are supported.
- From GWT 2.7, Dev Mode launches Super Dev Mode automatically.
    - Just start Dev Mode and reload the page.
    - It will recompile automatically when necessary.

Super development mode

- Super development mode replaces Development Mode
  - works better in modern browsers,
  - do not need browser plugin.
- Super development mode allows GWT developers to quickly recompile their code and see the results in a browser.
- At startup, Development Mode overwrites the GWT application's `nocache.js` files with a stub files that automatically recompiles the GWT application if necessary.
- The GWT application itself is loaded from a separate web server running on a different port (9876 by default).
- Started by
  - `ant devmode`
  - `mvn gwt:codeserver`

1. Right click on project and select `Properties`.
2. In `Frameworks`, *Google Web Toolkit* must be present (if it is not, add it, confirm the window and open it again).
3. In `Actions` Add Custom
   - Action Name: `Hosted mode`
   - Execute Goals: `gwt:debug`
   - Set Properties – Add property `runTarget=GwtButton.html` (according to your host page)
4. Clean and build the project.
5. Run the project.
6. Right click on the project and select `Run Maven` – `Hosted mode`.
7. In the main menu select `Debug` – `Attach Debugger`.
8. Change the port to 8000 and confirm.
9. Wait a moment.
10. Launch the browser or copy (using button `Copy to clipboard`) and paste the address to it.
11. The application will recompile and will be ready for debugging on port 8888.

1. In the menu of the browser (described for Google Chrome but very similar for Firefox) select `Tools` – `Developer Tools`.
2. The tool should detect Source map.
3. In the tab `Sources` search for appropriate Java source.
4. Debug it in the browser.
5. Code server on 9876 and Jetty od 8888 are stopped by closing of the debugger window.

Super dev mode in NetBeans 12 1/2

1. Right click on project and select `Properties`.
2. In `Actions` Add Custom
   - Action Name: `Hosted mode`
   - Execute Goals: `gwt:debug`
   - Set Properties – Add property `runTarget=GwtButton.html` (according to your host page)
3. Clean and build the project.
4. Run the project.
5. Right click on the project and select `Run Maven` – `Hosted mode`.
6. In the main menu select `Debug` – `Attach Debugger`.
7. Change the Connector to `SocketAttach` and port to 8000 and confirm.
8. Wait a moment.
9. Launch the browser or copy (using button `Copy to clipboard`) and paste the address to it.
10. The application will recompile and will be ready for debugging on port 8888.

1. In the menu of the browser (described for Google Chrome but very similar for Firefox) select `Tools` – `Developer Tools`.
2. The tool should detect Source map.
3. In the tab `Sources` search for appropriate Java source.
4. Debug it in the browser.
5. Code server on 9876 and Jetty od 8888 are stopped by closing of the debugger window.

Problems

- Something can be occupating some of used ports (8080, 8000, 9876, 8888) and the startup procedure will fail (in this case check the ports and free them).
- After switch from development to production mode it may be necessary to clean and build the project to update bootstrap file (`nocache`).
- Bootstrap file (`nocache`) can be cached by the browser and it is not reloaded with the page (e.g. after switch from development to production mode) – if you think that you have the file cached, display the source of the page. Then display the appropriate JavaScript file and reload it.

Production mode

- Compiles Java to JavaScript
  - `com.google.gwt.dev.Compiler`
  - bootstrap file `hello.nocache.js`
  - application files <md5>`.cache.html`
    - permutations
    - "perfect caching" – application filenames will always change if your codebase changes (MD5), so your clients can safely cache these resources and don't need to refetch the GWT application files each time they visit your site. The resource that should never be completely cached is the bootstrap script.

- widgets
  - rendered as dynamically-created HTML,
  - abstracts cross-browser incompatibilites
    - recent versions of Chrome, Firefox, Internet Explorer, Safari, Opera,
  - styled with CSS.
  - All standard HTML controls have corresponding GWT Swing-like classes
    - native HTML controls where possible.
  - GWT defines many extended components that are combination of HTML elements
    `http://code.google.com/webtoolkit/doc/latest/RefWidgetGallery.html`

- By default each element has a classname
  - `gwt-<ClassName>` (like `gwt-Button`, `gwt-TextBox`)
- Each element can have assigned a specific ID

```
Button b = new Button();
DOM.setElementAttribute(b.getElement(), "id",
"my-button-id")
```

- Styling API
  - `setStyleName(String style)`
    - Clears all of the object's style names and sets it to the given style.
  - `setStyleName(String style, boolean add)`
    - Adds or removes a style name.
  - `addStyleName`
  - `removeStyleName`
  - `String getStyleName`
    - Gets all of the object's style names, as a space-separated list.
  - `setStylePrimaryName`
    - Sets the object's primary style name and updates all dependent style names.
    - Added style names are secondary (this can change it).

Example GwtCssExample

Basic widgets

- Well-defined class hierarchy
- CSS styled
  - More complex behaviour – Secondary styles

UIObject

Widget

Layout Panels,
Basic Widgets,
Form Widgets,

MenuItems,
TreeItem,
MenuItemSeparator

- `HTML`
  - contains arbitrary HTML code,
  - uses `<div>` element,
  - displayed as a block.
- `Image`
  - Clipped mode – viewport is overlaid on top of an image.
  - Unclipped – whole image is visible.
- `Anchor`
  - represents `<a>` element.

Examples GwtHtml, GwtImage

Form Widgets

- `Label`
- `Button`, `PushButton`, `ToggleButton`
- `CheckBox`
- `RadioButton`
- `ListBox`
- `SuggestBox` (autocomplete)
- `TextBox`, `PasswordTextBox`
- `TextArea`, `RichTextArea`
- `FileUpload`
- `Hidden`

Examples GwtLabel, GwtButton, GwtWidgets

Complex widgets

- `Tree`
- `MenuBar`
- `DatePicker`
- `CellTree`
- `CellList`
- `CellTable`
- `CellBrowser`

Examples GwtComplexWidgets, GwtCellTree, GwtCellList, GwtCellTable, GwtCellBrowser

```
Command cmd = new Command() {
    public void execute() {
        Window.alert("You have selected a menu item!");
    }
};

MenuBar fooMenu = new MenuBar(true);  // vertical=true
fooMenu.addItem("do foo", cmd);
fooMenu.addItem("exit", cmd);

MenuBar barMenu = new MenuBar(true);
barMenu.addItem("do bar", cmd);
barMenu.addItem("exit", cmd);
...
MenuBar menu = new MenuBar();
menu.addItem("foo", fooMenu);
menu.addItem("bar", barMenu);

RootPanel.get().add(menu);
```

Layout Panels

- Layout panels contains other widgets.
- `Panel` – abstract base class for all panels
- `FlowPanel` (HTML flow)
- `VerticalPanel`, `HorizontalPanel`
- `HorizontalSplitPanel`, `VerticalSplitPanel`
- `FlexTable` (creates cells on demand)
- `Grid`
- `DeckPanel` (only one child widget visible, used by `TabPanel`)
- `DockPanel` (similar to the `FullPageLayout` in the PrimeFaces)
- `HTMLPanel`
- `TabPanel`
- `SimplePanel` (contains only one widget)
- `ScrollPanel`
- `FocusPanel` (makes its contents focusable)
- `FormPanel` (has ability to catch mouse and keyboard events)
- `PopupPanel`, `DialogBox`
- `Composite` (widget that can wrap another widget, hiding the wrapped widget's methods)

Examples GwtPanels, GwtFlowPanel

Event handling

- Listener interface defined for each event.
- Classes wishing to react must implement given interface.
- All event handlers extends `EventHandler` interface.
- Callback argument is allways of type `Event`

```
public class MyClickHandler implements ClickHandler {
    @Override
    public void onClick(ClickEvent event) {
        Window.alert("Hello World!");
    }
}

Button button = new Button("Click Me!");
button.addClickHandler(new MyClickHandler());
```

Example GwtEvents

- Three ways to create custom component
  - Extend `Composite` class – easiest
    - Sencha GXT library
  - Use GWT DOM API
    - complicated, use with caution
    - Vaadin framework
  - Use Javascript, wrap it via JSNI (JavaScript Native Interface)
    - Smart GWT library
- Internal widgets can be still styled via CSS.

Examples GwtCustomWidgets, SmartUpload

- GWT UI Binder is a framework designed to separate functionality and View of User Interface.
- Allows developers to build GWT applications as HTML pages with GWT widgets configured through them.
- Makes easier collaboration with UI designers who are more comfortable with XML, HTML and CSS than with Java source code.
- Provides a declarative way of defining user Interface.
- The UIBinder is similar to what JSP is to Servlets.

Steps for using UI binder

- Create UI Declaration XML File (e. g. `Login.ui.xml`)
- Use `ui:field` for Later Binding

```
<gwt:TextBox ui:field="loginBox" res:styleName="style.box" />
```

- Create Java counterpart of UI XML (e. g. `Login.java`)
- Bind Java UI fields with `UiField` annotation

```
@UiField
TextBox loginBox;
...
@UiHandler("loginBox")
void handleLoginChange(ValueChangeEvent<String> event) {
    ...
```

- Bind Java UI with UI XML with `UiTemplate` annotation

```
@UiTemplate("Login.ui.xml")
interface LoginUiBinder extends UiBinder<Widget, Login> {
}
```

Steps for using UI binder

- Create CSS File (e. g. `Login.css`)

```
.blackText {
    font-family: Arial, Sans-serif;
    color: #000000;
}
```

- Create Java based Resource File for CSS File (e. g. `LoginResources.java`)

```
public interface LoginResources extends ClientBundle {
    public interface MyCss extends CssResource {
        String blackText();
        ...
```

- Attach CSS resource in Java UI Code file

```
@UiField(provided = true)
final LoginResources res;

public Login() {  // constructor
    this.res = GWT.create(LoginResources.class);
    res.style().ensureInjected();
```

Example GwtUI

- RPC (Remote Procedure Call) is the mechanism used by GWT which allows the client code to directly execute the server side methods.
- GWT RPC is servlet based.
- GWT RPC is asynchronous and client is never blocked during the communication.
- Using GWT RPC Java objects can be sent directly between the client and the server.
- Server-side servlet is termed as service.
- Remote procedure call which is calling methods of server side servlets from client side code is referred to as invoking a service.

GWT RPC Components

- A remote service (server-side servlet) runs on the server.
- Client code, which is invoking the service.
- Java data objects which will be passed between client and server.
- GWT client and server both serialize and deserialize the data automatically so developers are not required to serialize/deserialize the objects and data objects can travel over HTTP.

Communication with the Server

- HTTP Client
  - “standard” AJAX
  - HTTP module
    - `<inherits name="com.google.gwt.http.HTTP"/>`

```
String url = "http://www.myserver.com/getData?type=3";
RequestBuilder builder = new RequestBuilder(
    RequestBuilder.GET, URL.encode(url));

Request request = builder.sendRequest(null,
    new RequestCallback() {
  public void onError(Request request,
                      Throwable exception) {
      //  (timeout, SOP violation, etc.)
  }
  public void onResponseReceived(Request request,
                                 Response response) {
    if (200 == response.getStatusCode()) {
      // Process the response in response.getText()
    } else {
      // Handle the error in response.getStatusText()
  }
});
```

GWT RPC workflow

- Java specific protocol
  - GWT's "serializable" is slightly different than standard Java Serialization

ServiceDefTarget (interface)

RemoteService (interface)

RemoteServiceServlet (class)

extends

extends

YourServiceAsync (interface)

related

YourService (interface)

implements

YourServiceImpl (class)

related

implements

YourServiceProxy (class)

Imported framework classes

Written by you

Generated automatically

Translatable Java code (runs as JavaScript on client)

Standard Java code (runs as bytecode on server)

- Create serializable model class.

```
public class Message implements Serializable {
    ...
    private String message;
    public Message(){};
    public void setMessage(String message){
        this.message = message;
    }
    ...
}
```

- Create a Service Interface
  - Define an interface for service on the client side that extends `RemoteService` listing all service methods.
  - Use annotation `@RemoteServiceRelativePath` to map the service with a default path of remote servlet relative to the module base URL.

```
@RemoteServiceRelativePath("message")
public interface MessageService extends RemoteService {
    Message getMessage(String input);
}
```

- Create an Async Service Interface
  - Define an asynchronous interface to the service on the client side which will be used in the GWT client code.
  - Async Service Interface can be generated automatically – in the Maven `<goal>generateAsync</goal>`

```
public interface MessageServiceAsync {
    void getMessage(String input,
                    AsyncCallback<Message> callback);
}
```

- Create a Service Implementation Servlet class.
- Implement the interface at the server side and that class should extend `RemoteServiceServlet` class.

```
public class MessageServiceImpl
        extends RemoteServiceServlet
        implements MessageService {
    ...
    public Message getMessage(String input){
        String messageString = "Hello " + input + "!";
        Message message = new Message();
        message.setMessage(messageString);
        return message;
    }
}
```

- Update `web.xml` to include the servlet declaration.

```
<web-app>
    ...
    <servlet>
        <servlet-name>messageServiceImpl</servlet-name>
        <servlet-class>
          com.tutorialspoint.server.MessageServiceImpl
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>messageServiceImpl</servlet-name>
        <url-pattern>/helloworld/message</url-pattern>
    </servlet-mapping>
</web-app>
```

- Make the remote procedure call in the application code.
  - Create the service proxy class.
    - `MessageServiceAsync messageService = GWT.create(MessageService.class);`
  - Create the `AsyncCallback` handler to handle RPC callback in which server returns the Message back to the client.

GWT RPC – finished

```
class MessageCallBack implements AsyncCallback<Message> {
  @Override
  public void onFailure(Throwable caught){
    Window.alert("Unable to obtain server response: " +
        caught.getMessage());
  }
  @Override
  public void onSuccess(Message result){
    Window.alert(result.getMessage());
  }
}
public class HelloWorld implements EntryPoint {
  private MessageServiceAsync messageService = GWT.create( ...
  ...
  public void onModuleLoad(){
  ...
    buttonMessage.addClickHandler(new ClickHandler(){
      @Override
      public void onClick(ClickEvent event){
        messageService.getMessage(txtName.getValue(),
          new MessageCallBack());
      }});
  ...
  }
}
```

Example GwtRpc

GWT Internationalization

- Three ways to internationalize GWT Application
  - Static String Internationalization
    - Little overhead in runtime
    - Constants and parametrized strings
  - Dynamic String Internationalization
    - More flexible
    - For integrating with existing "localized" server
  - Localizable interface
    - The most powerful
    - Custom localizable types

Static String Internationalization

- Create properties files
  - `HelloWorld.properties` (default language)
  - `HelloWorld_cs.properties` (czech version)
- Add i18n module to the Module Descriptor XML File

```
<extend-property name="locale" values="cs"/>
```

- Create interface equivalent to the properties file.
- Use `Message` interface in the UI component.

Internationalization properties

- `HelloWorld.properties`

```
enterName=Enter your name
clickMe=ClickMe
applicationTitle=Application Internationalization Demonstration
greeting=Hello{0}
```

- `HelloWorld_cs.properties`
  - Unicode !!!

```
enterName=Napi\u0161te sv\u0117 jm\u0117no
clickMe=Klikn\u011Bte sem
applicationTitle=Demonstrace internacionalizace aplikace
greeting=Ahoj{0}
```

- Translated application available at `http://127.0.0.1:8888/HelloWorld.html?gwt.codesvr=127.0.0.1:9997&locale=cs`

```
public interface HelloWorldMessages extends Messages{
    @DefaultMessage("Enter your name")
    String enterName();
    @DefaultMessage("Click Me")
    String clickMe();
    @DefaultMessage("Application Internalization
        Demonstration")
    String applicationTitle();
    @DefaultMessage("Hello {0}")
    String greeting(String name);
}
```

Example GwtInternationalization

Bookmarkable History

- "AJAX" applications breaks browser "back" button.
- GWT History mechanism
  - History tokens
    - `http://www.example.com/helloworld/Hello.html#page1`
  - History
    - get the current URL fragment (`page1`)
      `String getToken()`
    - add new history token to the history stack
      `newItem(String token)`
    - Event of history change
      `ValueChangeEvent`
  - Link to another state of the running application.
    - When clicked, it will create a new history frame without reloading the page.

    `Hyperlink(String text, String token)`
  - Reconstruct all the application state from the token.

- Enable History support

```
<iframe src = "javascript:''"
        id = "__gwt_historyFrame"
        style = "width:0;height:0;border:0">
</iframe>
```

- Add token to the history

```
index = 0;
History.newItem("pageIndex" + index);
```

- Retrieve token from the history

```
History.addValueChangeHandler(
    new ValueChangeHandler<String>(){
  @Override
  public void onValueChange(ValueChangeEvent<String>event){
    String historyToken = event.getValue();
    ...
```

Example GwtHistory

Delayed logic

- Timer
  - `schedule(int delayMillis)`
  - `scheduleRepeating(int periodMillis)`

```
Timer timer = new Timer() {
    public void run() {
        Window.alert ("Timer expired!");
    }
};
timer.schedule(2000);
```

- RepeatingCommand
  - ```
    Scheduler scheduler = Scheduler.get();
    scheduler.scheduleIncremental(repCmd, delayMs);
    ```
    - The next invocation of the command will be scheduled for `delayMs` milliseconds after the last invocation completes.
  - `Scheduler.RepeatingCommand`
    - command to be executed
    - for breaking long running tasks into chunks
    - ```
      boolean execute()  // Returns true if the
                         // RepeatingCommand should be
                         // invoked again.
      ```

Drawing in GWT

- `Canvas`
  - drawing area
  - `getContext2d()`
- `Context2D`
  - no `onDraw` as in Swing
  - we are drawing directly (e.g. on button click)
  - `Arc`
  - `DrawImage`
  - `FillRect`
  - `FillText`
  - `StrokeRect`
  - `StrokeText`
  - `Scale`
  - `Rotate`
  - `Translate`

Example Kufr

JavaScript Native Interface (JSNI)

- JSNI allows Java and JavaScript interaction.
- JSNI methods are declared `native`.
- Special syntax
  - Begins with `/*-{`
  - End with `}-*/;`
- JSNI syntax is Java-to-JavaScript compiler directive.
- Special variables
  - `$doc` – refers to document
  - `$wnd` – refers to browser's window

JSNI examples

- Parameter passing is simple

```
public static native void alert(String msg) /*-{
    $wnd.alert(msg);
}-*/;
public static native int inv(int num) /*-{
    return -num;
}-*/
public static void printInv(int num){
    System.out.println(inv(num));
}
```

- Type mismatch

```
public static native int badExample() /*-{
    return "Not A Number";
}-*/;
public void onClick () {
    try {
        int myValue = badExample();
        GWT.log("Got value " + myValue, null);
    } catch (Exception e) {
        GWT.log("JSNI method badExample() threw an
            exception:", e);
    }
}
```

- It is possible to manipulate Java objects in JavaScript code
- Invoking Java methods from JavaScript

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

  - **instance-expr** – necessary when calling instance method (must be absent when calling static one) (mostly `this`)
  - **Class-name** – fully qualified, contains path in package
  - **param-signature** – uses JNI type signatures
  - **arguments** – passed variables
- JNI (Java Native Interface)
  - `D` – double
  - `I` – int
  - `C` – char
  - `L` – follows fully qualified name (`Ljava/lang/String;`)
  - ...

- Constructor
  - `new TopLevel()` becomes `@pkg.TopLevel::new()()`
  - `new StaticInner()` becomes `@pkg.TopLevel.StaticInner::new()()`
  - `someTopLevelInstance.new InstanceInner(123)` becomes `@pkg.TopLevel.InstanceInner::new(Lpkg/TopLevel;I)(someTopLevelInstance, 123)`
    - Enclosing instance is the first parameter.
- Accessing Java fields from JavaScript
  - `[instance-expr.]@class-name::field-name`

JSNI calls

- Calling Java methods from handwritten JavaScript

```
package mypackage;

public MyUtilityClass
{
  public static int computeLoanInterest(int amt,
                                        float interestRate,
                                        int term) { ... }
  public static native void exportStaticMethod() /*-{
    $wnd.computeLoanInterest =
      $entry(@mypackage.MyUtilityClass::computeLoanInterest(IFI));
  }-*/;
}
```

- Notice that the reference to the exported method has been wrapped in a call to the `$entry` function. This implicitly-defined function ensures that the Java-derived method is executed with the uncaught exception handler installed and pumps a number of other utility services. The `$entry` function is reentrant-safe and should be used anywhere that GWT-derived JavaScript may be called into from a non-GWT context.

Passing Java values into JavaScript

Outcoming Java type	How it appears to JavaScript code
String	`var s = "my string"`
boolean	`var b = true`
long	disallowed to pass into JavaScript (in Java it can be emulated)
numeric primitives	`var x = 10`
JavaScriptObject	JavaScriptObject that must have originated from JavaScript code, typically as the return value of some other JSNI method
Java Array	can only be passed back to java code
any other object	accessible via special syntax (with `L`)

Incoming Java type	What must be passed
String	JavaScript string, as `return "abc";`
boolean	JavaScript boolean value, as in `return true;`
long	disallowed
numeric primitives	JavaScript numeric value, as in `return 10;`
JavaScriptObject	some native JavaScript object, as in `return document.createElement("div")`
any other object (including arrays)	must originate from Java code

JSNI exceptions

- An exception can be thrown during the execution of either normal Java code or the JavaScript code within a JSNI method.
- One can catch `JavaScriptException`
  - contains only class name and description.
- Exceptions originating from Java code retain its identity.
  - When Java calls JavaScript and JavaScript calls Java, where an exception occurs

Further examples (1/2)

```
public class JSNIExample {
  String myInstanceField;
  static int myStaticField;
  void instanceFoo(String s) {
    // use s
  }
  static void staticFoo(String s) {
    // use s
  }
```

Further examples (2/2)

```
public native void bar(JSNIExample x, String s) /*-{
    // Call instance method instanceFoo() on this
    this.@com.google.gwt.examples.JSNIExample::instanceFoo
        (Ljava/lang/String;)(s);
    // Call instance method instanceFoo() on x
    x.@com.google.gwt.examples.JSNIExample::instanceFoo
        (Ljava/lang/String;)(s);
    // Call static method staticFoo()
    @com.google.gwt.examples.JSNIExample::staticFoo
        (Ljava/lang/String;)(s);
    // Read instance field on this
    var val =
        this.@com.google.gwt.examples.JSNIExample::myInstanceField;
    // Write instance field on x
    x.@com.google.gwt.examples.JSNIExample::myInstanceField =
        val + " and stuff";
    // Read static field (no qualifier)
    @com.google.gwt.examples.JSNIExample::myStaticField = val +
        " and stuff";
  }-*/;
}
```

Resources

- http://www.gwtproject.org/
- http://www.gwtproject.org/javadoc/latest/
- http://www.gwtproject.org/doc/latest/tutorial/
- http://www.gwtproject.org/doc/latest/DevGuide.html
- https://gwt-maven-plugin.github.io/gwt-maven-plugin/
- http://www.tutorialspoint.com/gwt/

Thank you for your attention!