ISMLA Session 8 - GWT

Björn Rudzewitz

Tübingen University

December 11, 2017

Plan

1. Motivation
2. History Management
3. RPC

Motivation

- last time: building a client–side frontend for a job application website
- but: nothing happens when using the submit button
- topics today:
  - adding functions to the frontend/event handling
  - communicating with the server

Event Handling

- bringing function to a user interface
- i.e. when a user interacts, the interface reacts
- event-driven programming paradigm
- flow of the program determined by user actions and handlers reacting to events

Event Handling

- listeners are attached to widgets to handle events
- one widget can fire different events
- different listeners for different types of events
- Example: attaching a click and a key press handler to a text box

Event Handling

- type of events fired depends on the widget
- e.g. key press events only make sense for input fields where the user can type
- certain event types common across widgets, e.g. click handlers

Event Handling

- event handling technically realized by attaching event handler classes to widgets
- event handlers: Java classes implementing GWT interfaces overriding specific methods
- possibility to add event handlers as *anonymous inner class* or as separate class (for reusability)
- any non-final/non-global variable used in event handler needs to be passed via constructor

Event Handling

Example:

```
private class TextBoxKeyPressHandler implements
KeyPressHandler {
        @Override
        public void onKeyPress(KeyPressEvent event) {
                Window.alert("Key Press Event detected");
        }
}
...

TextBox tb = new TextBox();
tb.addKeyPressHandler(new TextBoxKeyPressHandler());
RootPanel.get().add(tb);
```

Event Handling

Passing arguments to event handlers

1. Create a event handler class implmenting the corresponding GWT class (e.g. `ClickHandler`)
2. add instance fields/variables in this class
3. add a constructor with arguments to set the class' instance fields
4. then in the methods (e.g. `onClick`) the value of the instance fields can be used

Event Handling

Demonstration

- see project GWTEventHandlingDemo for a demonstration/minimal example

History Management

- application technically is one page $\rightarrow$ using back button kicks user out of application
- desired: navigating between different states of the application via back button
- adding #hash parts ("history token") ot URL doesn't trigger change in web page by browser
- allows users to bookmark applications in a specific state
- attach a `ValueChangeHandler` to static `History` class

History Management

- entry point class should implement `ValueChangeHandler<String>`)
- then attach `History.addValueChangeHandler(this)`
- in the `onValueChange` method handle changes (not at first loading of page)
- add new history tokens via `History.newItem("sometoken");`

⇒ see project GWTHistoryDemo for an example of GWT history

History Management

- potential security flaw: history tokens for applications with a login mechanism
- if a user bokmarked a page after the login screen and opens it (e.g. via bookmark), the application should not directly forward there but rather show login screen and store history token for later
- further potential security problem: user entering URL with history token of a page he might not have reached normally

RPC

- RPC: Remote Procedure Call
- handles interaction between server and client code
- asynchronous programming:
  - send request to server
  - server reponds at some time, handle results (or exception) when they come back
  - UI remains responsive while waiting for the response

RPC

RPCs require (de)serialization:

- objects send to and from the server need to be serializable
- primitive types and their wrappers are serializable by default
- further serializable types: enumeration, string, dates, throwables, arrays, ArrayList, HashMap, HashSet, Stack, ...
- complex classes can be made serializable:
  - implement Java Serializable or GWT IsSerializable
  - make instance fields also serializable
  - default constructor provided

RPC

- RPCs work via proxies
- proxy: client-side interface providing the same method signatures like the server implementation
- when proxy methods are called it's necessary to provide a AsyncCallback object prametrized with the return type
- in the AsyncCallback method the server's response is handled

RPC

Setting up a RPC[1]

Client-side classes:

```
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {

public interface GreetingServiceAsync
```

Server-side class:

```
public class GreetingServiceImpl extends RemoteServiceServlet
implements GreetingService
```

[1] see auto-generated code in example project

RPC

web.xml

```
<!-- Servlets -->
<servlet>
  <servlet-name>greetServlet</servlet-name>
  <servlet-class>de.ws1718.ismla.server.GreetingServiceImpl</s
</servlet>

<servlet-mapping>
  <servlet-name>greetServlet</servlet-name>
  <url-pattern>/gwthistorydemo/greet</url-pattern>
</servlet-mapping>
```

RPC

Adding functions

1. Write a public method in the server-side class
2. Add the `@Override` annotation to this method
3. Follow the steps suggested by the plugin to create client–side proxy methods
4. on the client–side, add the proxy:

```
private final GreetingServiceAsync greetingService =
GWT.create(GreetingService.class);
```

RPC

Callbacks

- RPC calls require to provide a callback with the return type of the server method
- every callback overrides two methods:
  1. `onSuccess`: provides the return value of server call
  2. `onFailure`: provides a throwable (exception) thrown on the server
- like event handlers, callbacks can be implemented as anonymous inner or separate classes

RPC

Callbacks

```
private final GreetingServiceAsync greetingService =
GWT.create(GreetingService.class);

 ...

greetingService.greetServer(textToServer,
new AsyncCallback<String>() {
        public void onFailure(Throwable caught) {
                Window.alert("Server error: "
                + caught.toString());
        }
        public void onSuccess(String result) {
                // display message sent by the server
                Window.alert("RPC success. Result: " + result);
        }
});
```

RPC

- RPC example: web app tokenizing text by a user
- wokflowuser:
  1. user inputs text
  2. text is sent to the server
  3. server processes the text (tokenization)[2]
  4. server sends processing results back to the client

[2] more elaborate processing chain thinkable in real project

- using libraries on the server requires making them available there
- libraries should be put under WEB-INF/lib/
- resources can not be read as files, but instead from the servlet context and realtive to the WAR directory

Example:

```
InputStream stream =
getServletContext().getResourceAsStream(
"/WEB-INF/lib/my-model.bin");
```

RPC

Deomnstration

- see project *GWTRPCDemo*

Exercises

see handout

References

GWT books used: [Tacy et al., 2013], [Cooper and Collins, 2008], [Kereki, 2010]

excellent tutorial from which part of the structure of the presentation has been inspired: https://www.tutorialspoint.com/gwt/index.htm

Robert Cooper and Charlie Collins. *GWT in Practice*. Manning Publications Co., 2008.

Federico Kereki. *Essential GWT: building for the web with Google Web toolkit 2*. Pearson Education, 2010.

Adam Tacy, Robert Hanson, Jason Essington, and Anna Tokke. *GWT in Action*. Manning Publications Co., 2013.