Simplifying GWT RPC with Open Source GWT-Tools RPC Service
(GWT 2.4 Version)

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/gwt.html

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Ajax and GWT training, please see courses at http://courses.coreservlets.com/.
Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernates/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
Contact hall@coreservlets.com for details
Topics in This Section

- Quick GWT RPC review
- Motivation
- Advantages/Disadvantages
- Basic steps
- Example
- More examples
- Alternatives

RPC Review: Big Idea

- Write regular methods on server; don’t write servlet methods
  - Methods take arbitrary arguments
    - Not HttpServletRequest and HttpServletResponse
  - Methods return arbitrary results
    - Strings, arrays, lists, custom classes, etc.
- Call methods directly from client; don’t make explicit HTTP requests
  - Call server methods almost as though they were local
  - Pass arbitrary arguments and get arbitrary results
    - Custom form of serialization handles all the parsing
GWT-RPC Development Steps

- **Define main data service interface**
  - Implement RemoteService interface
  - Define regular methods without explicit HTTP
  - Use @RemoteServiceRelativePath to point at servlet
- **Define callback version of data service interface**
  - If main interface is FooService, define FooServiceAsync
- **Make data service servlet**
  - Extend RemoteServiceServlet, implement service interface
  - Supply url-pattern in web.xml that matches relative path
- **Create service proxy**
  - Call GWT.create(YourServiceInterfaceName.class)
- **Define client-side callback objects**
  - With onSuccess and onFailure

Motivation behind RPC Service

- Regular GWT RPC has too many configuration steps *every time* you need to define a new set of methods
- **Hard to integrate central exception handling**
  - E.g., if a particular type of exception occurs, I want to always navigate to some view
  - Without central exception handling, you have to hard code this in *every single* AsyncCallback.onFailure
- **Any changes to existing methods are somewhat cumbersome**
  - Always have to remember to update url-pattern, etc.
- **I want to set it up once and forget it!**
Solution: Command Design Pattern

- Command design pattern fits perfectly to the Request/Response paradigm
  - In the end, this is what GWT RPC is: request/response
- Issue a server request for data like a command
- Server inspects the command type and dispatches it to the appropriate command handler
- Handler executes command and returns a Response

RPC Service: Summary of Features

- Automatic discovery of handlers
- Automatic discovery and autowiring of Spring Framework enabled handlers
  - E.g., allows autowiring of Data Access Objects (DAOs)
- Multiple request commands can share a response
  - E.g., create new item and update item both will return new/updated item data
- Default implementation of AsyncCallback
  - With a way to provide an application-wide exception handler
- Annotation-based configuration
- Only one GWT RPC URL to map
  - Everything goes through that central URL
Configuration Steps

- Download rpc-service-xxx.jar from:
  http://code.google.com/p/gwt-tools/downloads/list
- Place JAR in the WEB-INF/lib directory
  - Depending on the type of Eclipse project you have, you might have to tell Eclipse about the JAR (through project properties – Java Build Path)
- In web.xml, configure RPC Service servlet:
  ```xml
  <servlet>
    <servlet-name>rpc-service-servlet</servlet-name>
    <servlet-class>
      org.tbiq.gwt.tools.rpcservice.server.RpcServiceServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>rpc-service-servlet</servlet-name>
    <url-pattern>/contacts/rpc-service</url-pattern>
  </servlet-mapping>
  ```

Configuration Steps (continued)

- In web.xml, configure auto-discovery of RPC request handlers
  - Two choices: package-based or Spring-based discovery
- Package-based discovery example:
  ```xml
  <listener>
    <listener-class>
      org....discovery.DefaultRpcRequestRegistryInitializationListener
    </listener-class>
  </listener>
  <context-param>
    <param-name>packagesToScan</param-name>
    <param-value>
      com.google.gwt.sample.contacts.server.rpc
    </param-value>
  </context-param>
  ```

Look for RPC request handlers in this package (or under) and register them.
Configuration Steps (continued)

- In web.xml, configure auto-discovery of RPC request handlers
  - Two choices: package-based or Spring-based discovery
- Spring-based discovery example:

```xml
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:/applicationContext.xml</param-value>
</context-param>

<listener>
    <listener-class>
        org...discovery.SpringRpcRequestRegistryInitializationListener
    </listener-class>
</listener>
```

Regular Spring Framework configuration for web modules.

RPC Service listener must be configured after the Spring Framework listener in web.xml.

Using: Basic Steps

- Create response class (in .client package)
  - Implements RpcResponse
- Create request class (in .client package)
  - Implements RpcRequest<BlahResponse>
  - Make sure request/response classes have zero arg constructor
    - Interfaces RpcRequest, RpcResponse already Serializable
- Create handler class (outside .client pkg)
  - Implements RpcRequestHandler
  - Implement its execute method
- Setup and make the RPC call from client
Example: (retrieve contact)  
GetContactResponse.java

```java
public class GetContactResponse implements RpcResponse {
    private Contact contact;\----------------- Contact returned from server.

    public GetContactResponse(){}\----------------- Required Zero-arg constructor.

    public GetContactResponse(Contact contact) {
        this.contact = contact;
    }

    public Contact getContact() {
        return contact;
    }

    public void setContact(Contact contact) {
        this.contact = contact;
    }
}
```

Example: (retrieve contact)  
GetContactRequest.java

```java
public class GetContactRequest implements RpcRequest<GetContactResponse> {
    private String contactId;

    public GetContactRequest() {}

    public GetContactRequest(String contactId) {
        this.contactId = contactId;
    }

    public String getContactId() {
        return contactId;
    }

    public void setContactId(String contactId) {
        this.contactId = contactId;
    }
}
```
DefaultRpcAsyncCallback callback = new DefaultRpcAsyncCallback(
    new DefaultApplicationExceptionHandler())
{
    @Override
    protected void handleResponse(RpcResponse response)
    {
        // Cast response to GetContactResponse
        // (because of GWT compiler generics bug)
        GetContactResponse getContactResponse =
            (GetContactResponse) response;
        contact = getContactResponse.getContact();

        // Display retrieved contact
        someTextWidget.setValue(contact.getFirstName());
        someTextWidget.setValue(contact.getLastName());
        someTextWidget.setValue(contact.getEmailAddress());
    }
};

rpcService.execute(new GetContactRequest(id), callback);

---

@RpcHandler
public class GetContactRequestHandler implements
    RpcRequestHandler<GetContactRequest, GetContactResponse> {

    public GetContactResponse execute(GetContactRequest rpcRequest,
        ServletExecutionContext context)
        throws RpcServiceException {
        // Retrieve contact with the given ID
        ContactsStore store = ContactsStore.getContactsStore();
        String contactId = rpcRequest.getContactId() + "";
        Contact contact = store.getContact(contactId);

        // Wrap contact into an RPC response object and return response
        GetContactResponse rpcResponse =
            new GetContactResponse(contact);
        return rpcResponse;
    }

    public Class<GetContactRequest> getCompatibleRpcRequestType() {
        return GetContactRequest.class;
    }
}
**Fully Working Examples**

- Check out all the source code with 3 example apps at:
  http://code.google.com/p/gwt-tools/source/checkout
- Three example apps:
  - Original Contacts app Google provide as an example, but mavenized
  - Same as above plus using place-service
    - History handler framework
      - Use only if stuck with GWT < 2.1
    - Same as above plus using RPC Service with package autodiscovery of RPC Handlers
    - Same as above plus using RPC Service Spring-enabled autodiscovery of RPC Handlers

**Alternatives**

- **gwt-dispatch**
  - Widely used
  - Nice developer forum
- **GWT RequestFactory**
  - Much more advanced but not as straightforward to use
  - Has many more features like automatically creating EntityProxy beans to reuse EJB 3.0 Entity beans directly in GWT client code
- **Google for other open source command-pattern request/response GWT RPC frameworks**
Wrap-Up

Questions?

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.