

# **Client/Server Communication in GWT**

**John A. Tamplin**

**[jat@google.com](mailto:jat@google.com)**

**Software Engineer, Google**



# GWT Client/Server Communication

- Protocols
  - GWT RPC, SOAP/ASP.net, XMLRPC, JSON/JSON-RPC/JSONP, REST
- Communication Methods
  - Server->Client, Forms, IFrames
- Best practices
  - Stateless, failure handling, data models
- Discussion



# Protocols

Some possible choices:

- GWT RPC
- JSON/JSON-RPC/JSONP
- SOAP
- XMLRPC
- REST

*Almost anything is possible...*



# GWT RPC

## Benefits:

- Built-in support in GWT
- Type-safe (including exceptions)
- Serialize entire object graphs (even cyclic!)
- Share code between client and server
- Asymmetric protocol for speed

## Recommendation

- GWT RPC is the best choice if you are writing server code in Java.



## GWT RPC Example: Service Interfaces

```

public interface LookupService extends
    RemoteService {
    Person lookupPerson(String name);
}
  
```

// GWT 1.5 syntax below

```

public interface LookupServiceAsync {
    void lookupPerson(String name,
        AsyncCallback<Person> callback);
}
  
```



# GWT RPC Example: Client Call

```

LookupServiceAsync lookup =
    GWT.create(LookupService.class);
// set entry point URL
lookup.lookupPerson(search, new
    AsyncCallback<Person>() {
    public void onSuccess(Person person) {
        detailPanel.showPerson(person);
    }
    public void onFailure(Throwable caught) {
        ...
    }
}
}
  
```



## GWT RPC Example: Server Code

```
public class LookupServiceImpl extends
    RemoteServiceServlet implements
    LookupService {
    public Person lookupPerson(String name)
    {
        // find the person
        return person;
    }
}
```



## GWT RPC: Limitations

- Asynchronous only (true for all AJAX)
- Client must be compiled with types that may be sent by the server
- Shared code must be translatable
- Serialization issues
- See Rob Jellinghaus's talk for more details on GWT RPC



# JSON/JSON-RPC/JSONP

- Basic idea is to send data in a form that is directly parsable by Javascript
- GWT makes this as easy to do as directly in Javascript
- Since Javascript is directly executing the JSON response, don't use JSONP with a server you don't trust



# JSON/etc – Client Request Code

```

RequestBuilder rb = new
  RequestBuilder(RequestBuilder.POST, url);
rb.setHeader(...) // etc
rb.sendRequest(postData, new RequestCallback() {
  void onResponseReceived(Request req,
    Response resp) {
    MyObject obj = getJSON(resp.getText());
    updateName(obj.name());
  }
  void onError(Request req, Throwable e) ...
}
  
```



# JSON/etc – Client Parsing Code

```

public class MyObject extends
    JavaScriptObject {
    public final native String name() /*- {
        return this.name;
    }-*/;
}

public native MyObject
    getJSON(String text) /*- {
    // validate text if needed (RFC4627)
    return eval('(' + text + ')');
} -*/;

```



# JSON-RPC

- JSON-RPC is just JSON with a convention for passing arguments and return values for RPC
- Same code, you just have to follow those conventions
- Can support server->client RPC
- Still subject to same-origin problem
- Can do notify-only messages



# JSON-RPC Sample Interaction

```
C->S: { "method": "lookupPerson", "params": [
  "*Smith", ["id", "name"] ], "id": 1 }
```

---

```
S->C: { "result": [ { "id": 27, "name": "John
  Smith" }, { "id": 29, "name": "Joe Smith" } ],
  "error": null, "id": 1 }
```

```
S->C: { "method": "warnUser", "params": [
  "The sale ends in 5 minutes" ], "id": 27 }
```

---

```
C->S: { "result": 0, "error": null, "id": 27 }
```



# JSONP

- JSONP allows a way around the same-origin limitation
- Pass args in `<script>` URL, including a callback function to be executed
- Response is executable JS code that calls the callback function with the result of the call
- GData APIs support JSONP



# JSONP Example

```

String jsonpURL =
    "http://www.google.com/base/feeds
    /snippets?bq=digital+camera&alt=
    json-in-script&callback=myCallback";

jsonpRequest(jsonpURL, new Callback() {
    void callback(MyResp resp) {
        ... use resp as before ...
    }
});
  
```



# JSONP Example – Request Code

```

public native void jsonpRequest(String url,
    Callback cb) /*- {
    window.myCallback = function(resp) {
        cb.@Callback::callback(LMyResp;)(resp);
    }
    var script =
        document.createElement("script");
    script.url = url;
    document.body.appendChild(script);
} -*/
  
```



## **XMLRPC – Predecessor to SOAP**

- Same basic idea as JSON-RPC, except use XML instead of JS code for requests and responses
- Much simpler protocol than SOAP



# XMLRPC – Typical Request Structure

```

<?xml version="1.0"?>
<methodCall>
  <methodName>myMethod</methodName>
  <params>
    <param>
      <value><i4>integer</i4></value>
      <value><string>str</string></value>
    </param>
  </params>
</methodCall>
  
```



# XMLRPC – Sample Response

```

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>John
        Doe</string></value>
    </param>
  </params>
</methodResponse>
  
```



# XMLRPC – Example Parsing Code

*... request code as before ...*

```

void onResponseReceived(Request req,
    Response resp) {
    Document d = XMLParser.parse(resp.getText());
    NodeList params =
        d.getElementByTagName("param");
    // you will want more checking than this :)
    String name = params.item(0).getFirstChild()
        .getFirstChild().getValue();
    ...
}
  
```



# SOAP

- Larger, more complicated form of XMLRPC
- WSDL is an XML document describing the types and services provided
- Tools exist for many languages to build client code given WSDL; a generator could be written for GWT to do the same but hasn't been done



# SOAP Request/Response Outline

```

<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC=
    "http://schemas.xmlsoap.org/soap/encoding/"
  ...more namespace definitions...>
  <SOAP-ENV:Header>
    .... typically security related data...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body id="...">
    ... xml data for request or response...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
  
```



# Representational State Transfer

- Typical web interface – objects have URI and the objects are retrieved via GET, updated via POST, etc.
- RequestBuilder is perfect for this application
- Objects can be cached using normal web caching behavior



## Other Communication Methods

- Server to Client (Comet/Hanging gets)  
A way to get server to client requests, some issues to consider.
- Forms – necessary for file upload, or interact with legacy CGI server code
- IFrames – alternative to XHR, with some issues



## Comet/Hanging Gets

- The client keeps a request active and the server responds when it has something to send
- Various ways of doing it, with different implications
- Uses one of only two active connections allowed, and can cause excessive server load



# Server to Client Methods

- Server waits to respond to a GET until it has data, client starts another one (can be done with script tags for cross-domain RPC)
- Infinite frame with script tags (UI issues)
- Multipart responses (Firefox only)
- Streaming XHR response (Firefox/Webkit only)
- HTML5 Events (Opera only)
- Client-side polling



# Forms

- Frequently the only way to interact with legacy CGI-style server code
- The only way to upload files from the client to the server
- See `FormPanel` and `FileUpload` in `com.google.gwt.user.client.ui`



# IFrames

- Can pass information to the server by manipulating the URL
- Can retrieve responses from the IFrame's content or the server can write script tags to be executed
- History and browser compatibility issues to consider



## Best Practices

- Use stateless servers – better failure handling, better scalability
- Consider possible failure modes, keep the user's needs in mind
- Minimize data sent to the client
  - Reduces bandwidth requirements
  - Reduces memory pressure in browser



# Stateless Servers

- Obviously any real app will need to maintain state, such as in a database
- We are talking about keeping conversational state in the browser
- Any server can answer any request, so just add more servers behind a load balancer, no user state lost if a server dies



## Minimize Data in the Client

- Reduces bandwidth requirements
- Reduces updates required to keep client data in sync
- Client runs faster if it has less data
  - If you have a table of 10,000 rows, you don't want to actually create HTML with 10,000 rows but instead page through the rows asking for data as needed





# Questions and Answers

