JSON-RPC
Sending Complex Data to and from the Server

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

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Ajax & GWT training, see training 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
- Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF 1.x & 2.0, Ajax, GWT, custom mix of topics
- Ajax courses can concentrate on one library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
- Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details
Topics in This Section

• **Using the jsabsorb JSON-RPC implementation**
  – Calling server methods synchronously
    • Very useful for testing, but not for real projects
  – Calling server methods asynchronously
    • For real projects
  – Passing and returning strings and primitives
  – Returning JSON objects

• **Other JSON-RPC utilities**
RPC and JSON

- **Idea**
  - Let client code act as though it is calling a method on the server (not accessing a URL)

- **Advantages**
  - Simpler client syntax
    - No explicit use of XMLHttpRequest object
  - Simpler server syntax
    - Regular methods, not server doGet methods
  - Can pass and return data directly
    - Instead of indirectly sending strings in (via request params) and printing to generate output data

- **Disadvantages**
  - Requires extra code on both client and server
  - Ties server code to Java (instead of to arbitrary URL)

JSON-RPC

- **JSON-RPC**
  - Standard for remote calls with JSON data
  - http://json-rpc.org/

- **jabsorb**
  - Popular JSON-RPC implementation
    - “JavaScript to Java Object Request Broker”
  - Simple to use; limited power
  - Installation and setup poorly documented

- **Other JSON-RPC implementations**
  - http://json-rpc.org/wiki/implementations

- **Alternative RPC-to-Java toolkits**
  - Google Web Toolkit (covered later in this tutorial)
  - Direct Web Remoting (http://directwebremoting.org/)
Installation and Setup

- **Download jabsorb**
  - [http://jabsorb.org/Download](http://jabsorb.org/Download)
    - Get “minimal” version to get jabsorb-1.x.jar and jsonrpc.js
    - Get “src” version to get slf4j-xxxx.jar (two files)
- **Installation**
  - Server
    - Put three JAR files in WEB-INF/lib
      - jabsorb-1.x.jar, two JAR files for slf4j
    - Edit web.xml for JSON-RPC mapping (see later slide)
  - Client
    - Load jsonrpc.js into page
- **Documentation**
  - [http://jabsorb.org/Tutorial](http://jabsorb.org/Tutorial)

Example Eclipse Project

- jabsorb
  - Deployment Descriptor: json-rpc
  - Java Resources: src
  - JavaScript Support
  - build
  - WebContent
    - css
    - META-INF
    - scripts
      - ajax-utils.js
      - jsonrpc.js
      - json-rpc-examples.js
  - WEB-INF
    - lib
      - jabsorb-1.3.jar
      - slf4j-api-1.4.2.jar
      - slf4j-jdk14-1.4.2.jar
    - web.xml
    - index.html

Note: you can also download the json-rpc-blank Eclipse project from [http://courses.coreservlets.com/Course-Materials/ajax.html#JSON-RPC](http://courses.coreservlets.com/Course-Materials/ajax.html#JSON-RPC). This is an Eclipse project with all the necessary pieces for jabsorb, but nothing else (no Java code, no app-specific JavaScript). You can use that project as a starting point for your own jabsorb projects.
Server-Side Configuration

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Servlet Mapping

<servlet>
  <servlet-name>org.jabsorb.JSONRPCServlet</servlet-name>
  <servlet-class>org.jabsorb.JSONRPCServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>org.jabsorb.JSONRPCServlet</servlet-name>
  <url-pattern>/JSON-RPC</url-pattern>
</servlet-mapping>

• Note
  - There is also a web.xml entry that tells JSON-RPC to gzip the response whenever possible. This improves performance in real life, but should be omitted in testing so that you can use Firebug to trace the return data. Online docs show how to enable gzipping.
Registering Handler Object on Server

• **Idea**
  - You instantiate a server-side object and register it (give a name). E.g., give it the name objName.
    • Use init of servlet with load-on-startup or context listener
  - For synchronous class, client code calls
    rpcClient.objName.methodName(args)
    • Very useful for interactive testing; do *not* use for final app.
  - For asynchronous calls, client code calls
    rpcClient.objName.methodName(callbackFunction, args)

• **Alternatives**
  - If you have static methods only, you can register class
  - If methods have state, you can make a user-specific handler object that is stored in the user’s session
    • There is good documentation on this at jabsorb.org

Sample Handler Object

```java
public class JsonRpcTester {
    public double getRandomNumber() {
        return(Math.random());
    }

    public double getRandomNumber(double range) {
        return(range * Math.random());
    }

    public City getCity(String cityName) {
        return(CityUtils.getCity(cityName));
    }
}
```

• **Note**
  - This is normal object with normal methods, not a servlet
Registering Handler Object on Server: Servlet Context Listener
	package coreservlets;

import javax.servlet.*;
import org.jabsorb.);

public class JsonRpcInitializer
    implements ServletContextListener {
    public void contextInitialized(ServletContextEvent event) {
        JsonRpcTester rpcTester = new JsonRpcTester();
        JSONRPCBridge globalBridge =
            JSONRPCBridge.getGlobalBridge();
        globalBridge.registerObject("rpcTester", rpcTester);
    }

    public void contextDestroyed(ServletContextEvent event) {
    }
}

Registering Handler Object on Server: web.xml Entry

<!-- Run JsonRpcInitializer when app starts up. -->

<listener>
    <listener-class>
        coreservlets.JsonRpcInitializer
    </listener-class>
</listener>
Client-Side Configuration

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Registering RPC Client in Browser

- **Idea**
  - Client loads jsonrpc.js in page
  - Client does
    - `rpcClient = new JSONRpcClient("address");`
      - Address is from the URL pattern that you set in web.xml (e.g., /JSON-RPC)
  - For synchronous calls
    - `rpcClient.serverObjName.methodName(args)`
      - Never deploy real apps with synchronous calls, but this is tremendously useful for interactive testing in Firebug. You can directly type in remote method calls and make sure everything works before hooking it up to callback handler for the real app.
  - For asynchronous calls
    - `rpcClient.serverObjName.methodName(callback, args)`
Registering RPC Client in Browser: Example

```javascript
var rpcClient;

window.onload = function() {
    rpcClient = new JSONRpcClient("JSON-RPC");
}
```

This is relative URL. This example assumes that page is in top level of Web application. If it was in subfolder, you would use ../JSON-RPC. If you used this from pages scattered in various subfolders at different levels, you could do /appName/JSON-RPC.
Synchronous RPC Calls

- **Idea**
  - Make normal function call
    - `var result = rpcClient.serverObj.methodOfServerObj(…);`
- **Arguments to method and return value**
  - You can pass numbers, strings, arrays, or objects.
  - You do not need to escape Strings you pass to server
  - Beans that are returned from server are automatically JSONified as in previous lecture
  - Passing objects to server is trickier (but not done often). Server-side code should expect JSONObject.
- **Important: use asynchronous calls**
  - Synchronous calls are only for practice and testing of the server-side code.
  - *Always use asynchronous calls in real life,* otherwise JavaScript will be frozen until server returns.

---

Synchronous Calls: Example of Interactive Testing

First, deploy app and start server. Bring up a page that loads both jabsorb.js and a JavaScript file that does basic client-side setup as just described. Then bring up Firebug and interactively try out remote method calls. To add a new remote method, just add a method to the class in Eclipse and save file. Then R-click on server and choose “Restart”. Then go to HTML page and hit reload button. Then go back to Firebug and try the new method.
Synchronous Call: Example (JavaScript)

```javascript
function showRandomNumber1(resultRegion) {
    var randomNumber =
        rpcClient.rpcTester.getRandomNumber();
    htmlInsert(resultRegion,
               "Number is " + randomNumber);
}
```

Synchronous Call: Example (HTML)

```html
<script src='./scripts/ajax-utils.js'
type='text/javascript'></script>
<script src='./scripts/json-rpc-examples.js'
type='text/javascript'></script>
<script src='./scripts/jsonrpc.js'
type='text/javascript'></script>
...
<fieldset>
    <legend>Synchronous Server Call</legend>
    <form action="#">
        <input type="button" value="Show Random Number"
               onclick='showRandomNumber1("ran-num-result-1")'/>
    </form>
    <div id="ran-num-result-1" class="ajaxResult"></div>
</fieldset>
```
Synchronous Call: Example (Result)

Using JSON-RPC

Synchronous Server Call

Show Random Number

Done

Using JSON-RPC

Synchronous Server Call

Show Random Number

Number is 0.5695268881097731

Done

Making Asynchronous Remote Calls

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Asynchronous RPC Calls

• Idea
  – When calling remote function, pass JavaScript callback function as the first argument to the call
  – The callback function should take two arguments
    • The real data that will come from the server
    • An exception
  – Callback will be invoked on readyState 4.
    • No freezing browser until response come in
    • The exception will be undefined if everything was ok

• Arguments and return values
  – Same as in previous example. Numbers and strings passed directly. Objects have type conversion rules.
    • You do not escape strings. Done automatically.

Making Asynchronous Calls: Example (JavaScript)

```javascript
function showRandomNumber2(inputField, resultRegion) {
    var range = parseInt(getValue(inputField));
    if (isNaN(range)) {
        range = 1;
    }
    var callback = function(randomNumber, exception) {
        if(exception) {
            alert(exception.msg);
        } else {
            htmlInsert(resultRegion, "Number is " + randomNumber);
        }
    };
    rpcClient.rpcTester.getRandomNumber(callback, range);
}
```

Note: http://jabsorb.org/Manual says to use "exception.message". However, debugging with Firebug shows that it is "exception.msg" in the current release.
Making Asynchronous Calls: Example (HTML)

```html
<fieldset>
  <legend>Asynchronous Server Call</legend>
  <form action="#">
    <label for="ran-num-range">Range (default 1):</label>
    <input type="text" id="ran-num-range"/><br/>
    <input type="button" value="Show Random Number" onclick='showRandomNumber2("ran-num-range", "ran-num-result-2")'/>
  </form>
  <div id="ran-num-result-2" class="ajaxResult"></div>
</fieldset>
```

Making Asynchronous Calls: Example (Result)
Passing or Returning Complex Data

• Data sent to server
  – Numbers and strings need no special handling
    • Do not escape strings.
    • This is done automatically.
  – Objects can be sent only if server expects JSONObject

• Data returned from server
  – Numbers and strings need no special handling
  – Returned objects are automatically passed to JSONObject constructor
    • Which means you can easily return beans without explicit conversion
Aside: Reading Textfield Values

- Done previously
  ```javascript
  function getValue(id) {
    return(escape(document.getElementById(id).value));
  }
  ```

- Done here
  ```javascript
  function getRawValue(id) {
    return(document.getElementById(id).value);
  }
  ```

Returning Complex Data: Example (JavaScript)

```javascript
function showCity(inputField, resultRegion) {
  var cityName = getRawValue(inputField);
  var callback = function(city, exception) {
    if(exception) {
      alert(exception.msg);
    } else {
      var result;
      if (city) {
        result = "<ul>" +
        "<li>Name: " + city.name + "</li>" +
        "<li>Time: " + city.time + "</li>" +
        "<li>Population: " + city.population + "</li>" +
        "</ul>";
      } else {
        result = "Unknown City";
      }
      htmlInsert(resultRegion, result);
    }
  }; rpcClient.rpcTester.getCity(callback, cityName);
```
Returning Complex Data: Example (Server Code)

```java
package coreservlets;

public class JsonRpcTester {
    public double getRandomNumber() {
        return (Math.random());
    }

    public double getRandomNumber(double range) {
        return (range * Math.random());
    }

    public City getCity(String cityName) {
        return (CityUtils.getCity(cityName));
    }
}
```

Returning Complex Data: Example (HTML)

```html
<fieldset>
    <legend>Returning Complex Data</legend>
    <form action="#" onsubmit="return false;">
        <label for="city-name">City Name:</label>
        <input type="text" id="city-name"/>
        <br/>
        <input type="button" value="Show City"
               onclick='showCity("city-name", "city-result")'/>
    </form>
    <p/>
    <div id="city-result" class="ajaxResult"></div>
</fieldset>
```
Returning Complex Data: Example (Results)

Complex Data: Returning Lists or Arrays

- **Server-side code**
  - Specify return type as `List<Type>` or `Type[]`

- **Client-side code**
  - Get the “list” property (not the direct return value)
  - This is an array (probably of JSONified beans)

- **Example**
  ```javascript
  var callback = function(cityList, exception) {
    if(exception) {
      alert(exception.msg);
    } else {
      doSomethingWith(cityList.list);
    }
  }
  ```

  Even though remote method returns List, the converted List (array) does not go into the top-level variable (cityList). Instead, it gets put into an automatically-created property called “list”. 

  Note: [http://jabsorb.org/Manual](http://jabsorb.org/Manual) says to use “exception.message”. However, debugging with Firebug shows that it is “exception.msg” in the current release.
Update to JsonRpcTester

```java
public class JsonRpcTester {
    public double getRandomNumber() {
        return (Math.random());
    }

    public double getRandomNumber(double range) {
        return (range * Math.random());
    }

    public City getCity(String cityName) {
        return (CityUtils.getCity(cityName));
    }

    public List<City> findCities(String description) {
        return (CityUtils.findCities(description));
    }
}
```

Eclipse process: added this new method to class, hit Control-S to save it, then R-clicked on server and chose "Restart". Then went back to HTML page shown earlier and hit reload button. Then went to Firebug to try this method out (see next page).

Testing the Update

When you do a remote call, the rpcClient automatically evals the network data and passes you the "result" property. Normally, this is the actual return value of the method. But in the case of lists and arrays, this contains a "list" subproperty that is actual return value.
Wrap-up

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Summary

- **Register server handler object**
  - JSONRPCBridge.getGlobalBridge(name, object);
- **Make browser RPC client**
  - rpcClient = new JSONRpcClient("address");
- **Make synchronous call (testing only!)**
  - rpcClient.serverObj.methodName(args)
- **Make asynchronous call**
  - rpcClient.serverObj.methodName(callback, args)
- **Returning bean from server**
  - Return it directly. Converted automatically with JSONObject as shown in previous tutorial section.
Questions?

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.