Internet Services
XML-RPC
Homework 1

Outline

- XML-RPC
- Basic structure
- Using XML-RPC
- Homework 1

XML-RPC

- Remote Procedure Call using XML as the encoding and HTTP as the transport
  - Simple
  - Least common denominator
- Easy to port to any programming environment
  - Java, Perl, Python, C, Fortran …

Purposes XML-RPC

- Glue disparate programming environments
- Offer services

XML-RPC overview

- Data model
- Request structure
- Reply Structure

Basic Data Types

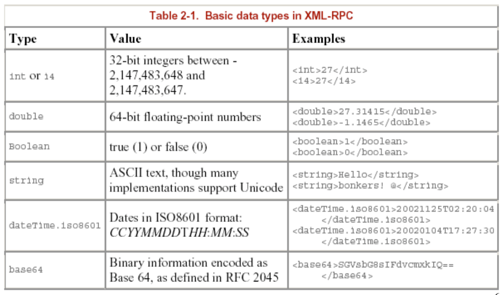

Table 2-1. Basic data types in XML-RPC

Type	Value	Examples
`int` or `i4`	32-bit integers between -2,147,483,648 and 2,147,483,647.	`<int>27</int>` `<i4>27</i4>`
`double`	64-bit floating-point numbers	`<double>27.31415</double>` `<double>-1.1465</double>`
`Boolean`	true (1) or false (0)	`<boolean>1</boolean>` `<boolean>0</boolean>`
`string`	ASCII text, though many implementations support Unicode	`<string>Hello</string>` `<string>bonkers! @</string>`
`dateTime.iso8601`	Dates in ISO8601 format: *CCYYMMDD*T*HH*:*MM*:*SS*	`<dateTime.iso8601>20021125T02:20:04 </dateTime.iso8601>` `<dateTime.iso8601>20020104T17:27:30 </dateTime.iso8601>`
`base64`	Binary information encoded as Base 64, as defined in RFC 2045	`<base64>SGVsbG8sIFdvcmxkIQ== </base64>`

Complex Types

- Arrays
  - Ordered list of values
  - Values can have any type
  - Can mix different types in the array
- Structs
  - Collections of name/value pairs
  - Like perl associative arrays, Java maps

Array Example

```
<value>
    <array>
          <data>
                    <value><string>This</string></value>
                    <value><string>is </string></value>
                    <value><string>an </string></value>
                    <value><string>array.</string></value>
          </data>
    </array>
</value>
```

Mixed Type Arrays

```
<value>
    <array>
        <data>
            <value><boolean>1</boolean></value>
            <value><string>Chaotic collection, eh?</string></value>
            <value><int>-91</int></value>
            <value><double>42.14159265</double></value>
        </data>
    </array>
</value>
```

9

Struct Example

```
<value>
    <struct>
        <member>
            <name>givenName</name>
            <value><string>Joseph</string></value>
        </member>
        <member>
            <name>familyName</name>
            <value><string>DiNardo</string></value>
        </member>
    </struct>
</value>
```

10

XML Request Example

```
<?xml version="1.0"?>
    <methodCall>
        <methodName>circleArea</methodName>
        <params>
        <param>
            <value><double>2.41</double></value>
        </param>
        </params>
    </methodCall>
```

11

XML-RPC Responses

- Returns a single parameter
  - Keeps with programming language traditions
  - Can contain a complex type
- Error encoded in the reponse
  - Not in the HTTP error codes

12

Resulting HTTP Request

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 128.6.4.4
Content-Type: text/xml
Content-Length: 169
<?xml version="1.0"?>
  <methodCall>
      <methodName>circleArea</methodName>
      <params>
          <param>
          <value><double>2.41</double></value>
          </param>
      </params>
  </methodCall>
```

13

Example Response

```
HTTP/1.1 200 OK
Date: Sat, 06 Oct 2001 23:20:04 GMT
Server: Apache.1.3.12 (Unix)
Connection: close
Content-Type: text/xml
Content-Length: 124

<?xml version="1.0"?>
  <methodResponse>
      <params>
            <param>

  <value><double>18.24668429131</double></value>
            </param>
      </params>
  </methodResponse>
```

14

Example Server

```
package com.ecerami.xmlrpc;
import java.io. IOException;
import org.apache.xmlrpc.WebServer;
import org.apache.xmlrpc.XmlRpc;
public class AreaHandler {
            public double circleArea(double radius) {
            double value=(radius*radius*Math.PI);
            return value;
    }
}
public class AreaServer {
    public static void main(String[] args) {
            if (args.length < 1) {
                        System.out.println("Usage: java AreaServer [port]");
                        System.exit(-1);
            }
            try {
                        startServer(args);
            } catch (IOException e) {
                        System.out.println("Could not start server: " +
            e.getMessage( ));
}
```

15

Example Server,cont

```
    public static void startServer(String[] args) throws IOException {
              // Start the server, using built-in version
          System.out.println("Attempting to start XML-RPC Server...");
          WebServer server = new WebServer(Integer.parseInt(args[0]));
           System.out.println("Started successfully.");

     // Register our handler class as area
          server.addHandler("area", new AreaHandler( ));
          System.out.println("Registered AreaHandler class to area.");
          System.out.println("Now accepting requests. (^C to stop.)");
    }
}
```

16

Example Client

- (See emacs file)

17

Summary

- Very simple representation and transport
  - Request/reponse, 5 basic types, XML and HTTP as transport
- 0th-order way to implement web-services
  - Is it sufficient?
  - A 90/10 solution? Maybe a 99%/1% solution
  - SOAP has more momentum …

18

Homework 1

- Go to google’s API site
- Download the packages and follow the instructions
- Implement a client “Detector.java”
- Takes as input a text file as the 2nd argument, key as the first arg.
- Break file into chunks of 10-word phrases
- Search Google for each phrase, up to 500 phrases maximum (5000 words)

19

Homework 1

- Output:
- Total number of unique phrases
- List of 25 <count, URL> pairs
  - URL’s matching all phrases, reverse sorted by frequency, out of top 10 URL’s per phrase
  - E.g., if URL 1 matched phrases 1,4,5,6,100,101, would get a count of 6.
  - Or, the message “No matches found”

20