Java XML-RPC

What?

- XML-based Remote Procedure Call (RPC) protocol
- Encode following as XML files
  - Method invocation information (such as method name, parameter list etc.)
  - Reply from the method
- Exchange these messages between applications using the HyperText Transfer Protocol (HTTP).

Advantages

- XML is platform independent
  - Enables heterogeneous applications to interoperate
- HTTP as a transport protocol
  - Makes it relatively easier to integrate XML-RPC with the existing web-enabled applications
- In fact, Dave Winer, the creator of XML-RPC described it as “RPC over HTTP via XML”:

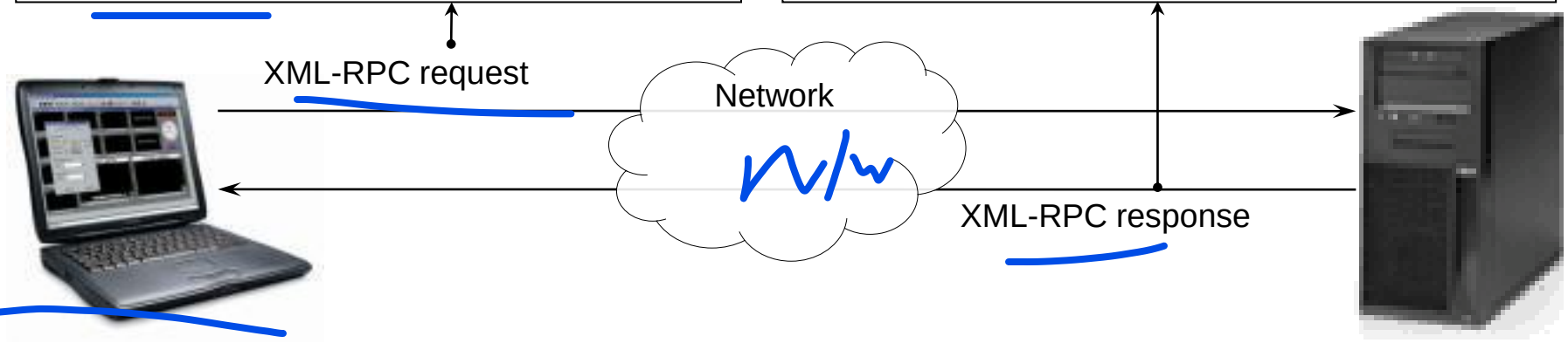
Basic Idea

```
POST /xmlrpc/RPC HTTP/1.0
Content-Type: text/xml
Content-Length: 154

<?xml version="1.0"?>
<methodCall>
  <methodName>fact</methodName>
  <params>
    <param>
      <value><int>5</int></value>
    </param>
  </params>
</methodCall>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 142

<?xml version="1.0">
<methodResponse>
  <params>
    <param>
      <value><i4>120</i4></value>
    </param>
  </params>
</methodResponse>
```



Timing diagram

Client	Server
1. Builds an XML document specifying method name and its parameters.	
2. Sends an HTTP POST request message whose payload (body) is the XML document.	
	3. Receives the message and uses Content-Length header to read the XML document.
	4. Parses the XML document, extracts the method name and parameters for the method to be invoked.
	5. Looks for the specified method and, if found, calls it with the specified parameters.
	6. Builds an XML document including the return value if the method call is successful. If an error occurs during method call, XML document contains error information
	7. Sends an HTTP response message whose payload (body) is this XML document.
8. Receives the message and uses Content-Length header to read the XML document..	
9. Parses the XML document and reports the result or error information to the user.	

Time

Data types

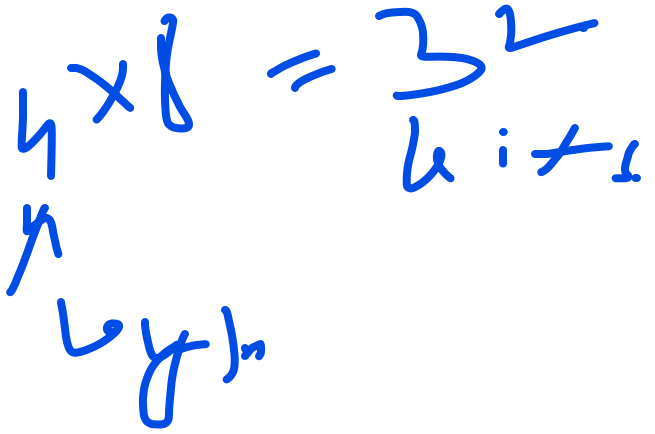
- A method is invoked with zero or more data of some types
- A data item in an XML-RPC request or response is embedded within a `<value>...</value>` element.
- XML-RPC specification defines six *basic data types* and two *compound data types*.
- Only data of these types can be passed to or returned from the remote method.

Basic types

- Six basic types for
  - Integer,
  - Floating point number,
  - Boolean,
  - String,
  - Date-time
  - Binary data.
- These types (except string) are always embedded in **<value>** elements.
- Only strings may be embedded in a **<value>** element directly omitting the **<string>** element.

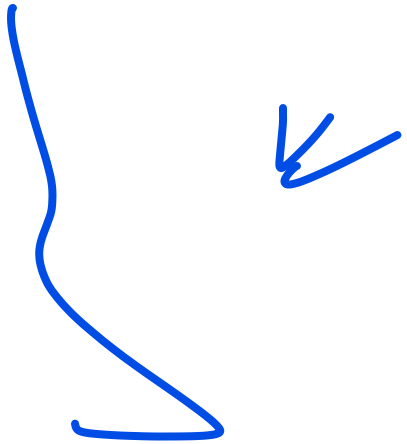
Integers

- It is represented by the tag **<int>** or **<i4>**, where **i4** stands for **4-byte integer**.
- Ranges from −2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31}-1$).
- Examples:
  - **<i4>23</i4>**
  - **<i4>-7</i4>**
  - **<i4>+3</i4>**
  - **<int>3</int>**
  - **<int>-2</int>**
  - **<int>+4</int>**



Floating point numbers

- The type is indicated by the tag **<double>**.
- Refers to 64-bit double precision signed floating point numbers ranging from ~4.94×10^{-324} to ~1.79×10^{308}.
- Examples:
  - **<double>3.42</double>**
  - **<double>-2.13</double>**
  - **<double>-0.13</double>**
  - **<double>+1.22</double>**



Booleans

- It is represented by the tag **<boolean>**.
- This type represents only values 1 or 0 which corresponds to Boolean true and false respectively.
- All possible examples:
  - **<boolean>1</boolean>**
  - **<boolean>0</boolean>**

Strings

- Refers to the sequence of ASCII characters
- A string value is represented using **<string>** tag
- Examples:
  - `<string>RPC</string>`
  - `<value>Remote Procedure Call</value>`
  - `<string>XML &amp; RPC</string>`
  - `<string>4 &gt; 3</string>`

Date and Time

- Both date and time are indicated using **<dateTime.iso8601>** element.
- CCYYMMDDTHH:MM:SS format
- Examples:
  - **<dateTime.iso8601>**20130902T06:49:21**</dateTime.iso8601>**
    - "2nd September, 2013" and time "06:49:21"
  - **<dateTime.iso8601>**20141015T15:45:00**</dateTime.iso8601>**
    - "15th October, 2014" and time "15:45:00"

Binary

- Control characters (having ASCII code lower than that of space character (32)) are not allowed in XML
- Arbitrary binary data cannot be transported
- For this reason, XML-RPC specification introduced a type for raw binary data. This is indicated using `<base64>` tag.
- A value of this type is a base64 encoded value from binary data. The encoding technique is described in RFC 2045

Binary contd.

- Encoding of the string "Hi!":
  - `<base64>SGkh</bas`
- Encoding of the string “Hello World!”
  - `<base64>SGVsbG8gV29ybGQh</base64>`

Compond types

- Basic types represent simple values and are sufficient for simple and less complex applications.
- However, complex applications may require more complex types to represent compound structure such as array, struct etc
- XML-RPC provides two types array and struct for compound values.

Array

- An array is a sequence of data items
- Unlike traditional array, XML-RPC allows array elements to be of different types
- Also no provision for indexing array elements
- General syntax:

```
<array>
  <data>
    <value>An XML-RPC value</value>
    <value>An XML-RPC value</value>
    …
    <value>An XML-RPC value</value>
  </data>
</array>
```

Array contd.

- An array of 3 integers

```
<array>
  <data>
    <value><int>2</int></value>
    <value><int>3</int></value>
    <value><int>4</int></value>
  </data>
</array>
```

Array contd.

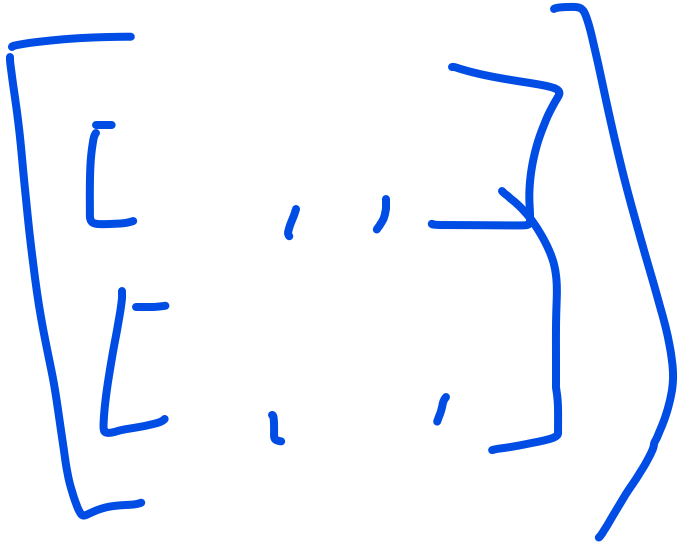
- Elements need not be of the same type.
- So, the following XML-RPC array is valid.

```
<array>
  <data>
    <value><string>B. S. Roy</string></value>
    <value><int>35</int></value>
    <value><double>48.34</double></value>
  </data>
</array>
```

Array contd.

- The array elements need not always be basic type.
- Example of 2X3 array

```
<array>
  <data>
    <value>
      <array>
        <data>
          <value><int>1</int></value>
          <value><int>2</int></value>
          <value><int>3</int></value>
        </data>
      </array>
    </value>
    <value>
      <array>
        <data>
          <value><int>4</int></value>
          <value><int>5</int></value>
          <value><int>6</int></value>
        </data>
      </array>
    </value>
  </data>
</array>
```



Struct

- A sequence of name-value pair
- Each pair is said to be a member of struct and is represented by **<member>** element.
- The **<name>** of a member is an ASCII string and the **<value>** is any valid XML-RPC value including array or struct.

Struct contd.

- Syntax:

```
<struct>
  <member>
    <name>name1</name>
    <value>value1</value>
  </member>
  <member>
    <name>name2</name>
    <value>value2</value>
  </member>
  ...
  <member>
    <name>nameN</name>
    <value>valueN</value>
  </member>
</struct>
```

Struct contd.

- Example(information of a person):

```
<struct>
  <member>
    <name>firstName</name>
    <value><string>Banhishikha</string></value>
  </member>
  <member>
    <name>lastName</name>
    <value><string>Roy</string></value>
  </member>
  <member>
    <name>age</name>
    <value><int>35</int></value>
  </member>
</struct>
```

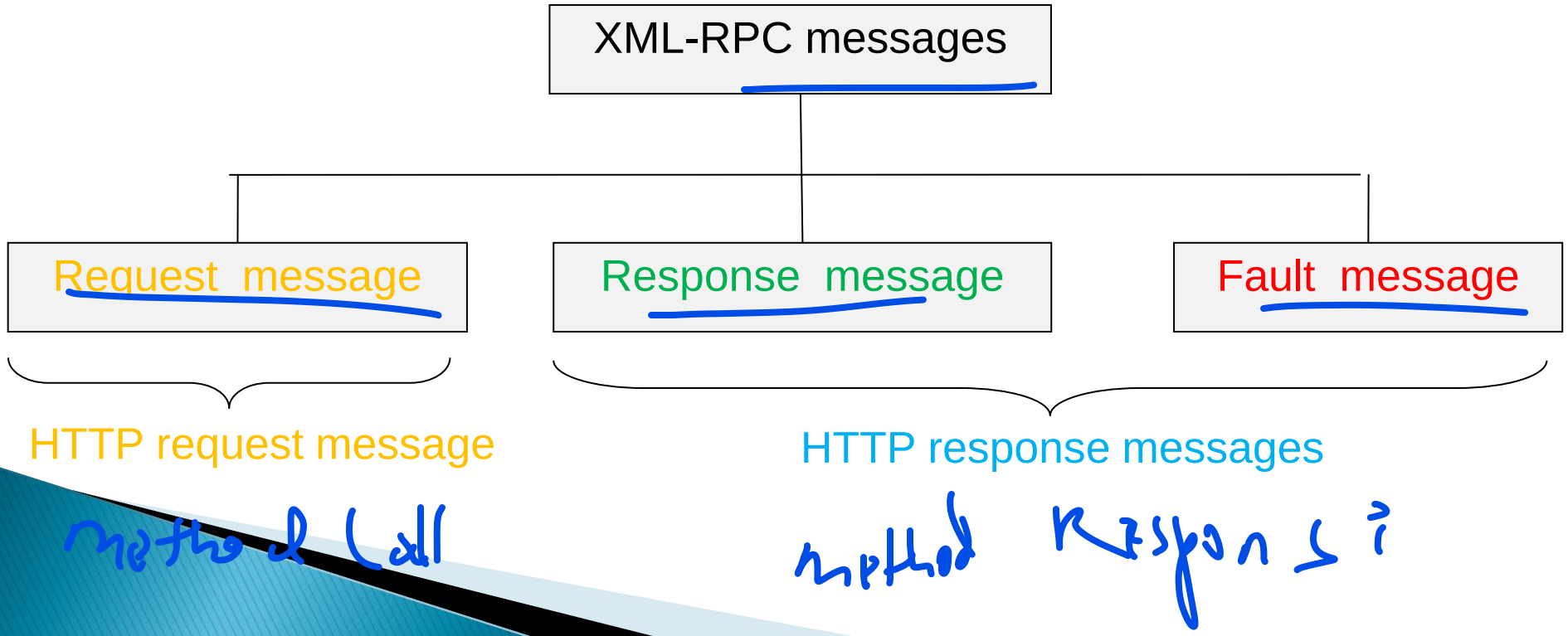
Struct contd.

- Example(information of a method call):

```
<struct>
   <member>
      <name>methodName</name>
      <value><string>sqrt</string></value>
   </member>
   <member>
      <name>y</name>
      <value><int>4</int></value>
   </member>
</struct>
```

XMP-RPC messages

- Method call and response are described as HTTP messages having body written in XML
- Three types of messages: *request message*, *response message* and *fault message*.



Request Message

- An HTTP POST request message whose body is an XML document.
- XML message body encodes the method call syntax
- root element `<methodCall>` that contains `<methodName>` and `<params>` element which contains a list of `<param>`.
- The value of the argument is specified by the `<value>` element.

Request Message contd.

- **Example:**

```
<?xml version="1.0"?>
<methodCall>
   <methodName>fact</methodName>
   <params>
      <param>
         <value><int>5</int></value>
      </param>
   </params>
</methodCall>
```

Request Message contd.

- HTTP request line and header look something like this:

```
POST /xmlrpc/RPC HTTP/1.0
User-Agent: Apache XML-RPC
Host: client.host.name
Content-Type: text/xml
Content-Length: length of XML body in bytes
```

Request Message contd.

```
POST /xmlrpc/RPC HTTP/1.0
```

Request line

```
Content-Type: text/xml
Content-Length: 154
```

Headers

Blank line

```
<?xml version="1.0"?>
<methodCall>
   <methodName>fact</methodName>
   <params>
      <param>
         <value><int>5</int></value>
      </param>
   </params>
</methodCall>
```

Body

Request Message contd.

- The interpretation of the content of **<methodName>** element depends entirely on the server.
- E.g. may be the exact name of the method or method name with object name as a prefix etc.
- Apache XML-RPC Java library expects this string as the following format:
- **objectName.methodName**

Request Message contd.

- The **<params>** tag may be absent if a method does not take any parameter.

```
<?xml version="1.0"?>
<methodCall>
   <methodName>sayHello</methodName>
</methodCall>
```

Response Message

- An HTTP response message whose body is an XML document
- Sent by the XML-RPC server as a result of method call
- In HTTP response line, the server should always return **'200 OK'** unless an HTTP error occurs
- Header section should contain two headers `Content-Type` and `Content-Length`.
- `Content-Type` should be `text/xml`
- `Content-Length` is the **correct** number of bytes in the XML message body.

Response Message cond.

- Root element is <methodResponse> that contains a single <params> tag, which contains a single <param> tag.
- Example:

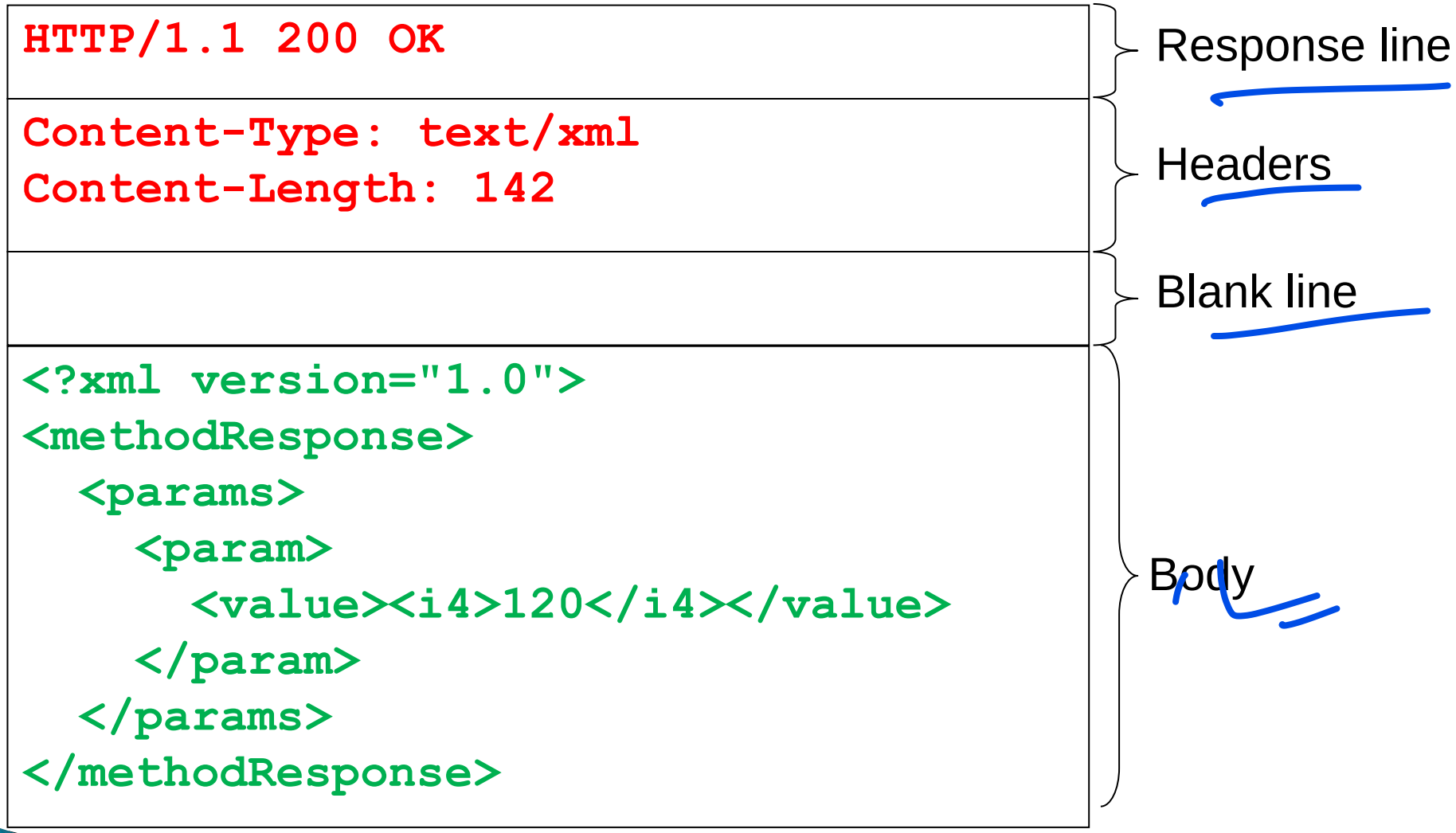
```
<?xml version="1.0">
  <methodResponse>
    <params>
      <param>
        <value><i4>120</i4></value>
      </param>
    </params>
</methodResponse>
```

Response Message cond.

- Response line and header look something like this:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 142
```

Response Message cond.



Fault Message

- If server faces some problem processing the request, it notifies the client with a special message called ***fault message***.
- Fault message is also a response message except that it has a different format
- `<methodResponse>` tag contains `<fault>` tag, which designates the message as a fault message.

Imp

Fault Message contd.

- The <fault> contains a <value>, which is a <struct> containing two members.
- first member is <faultCode>, whose value is an <int>, which indicates the error code.
- The second member is <faultString>, whose value is a <string> which gives an explanation about the error occurred.

Fault Message Example

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 365

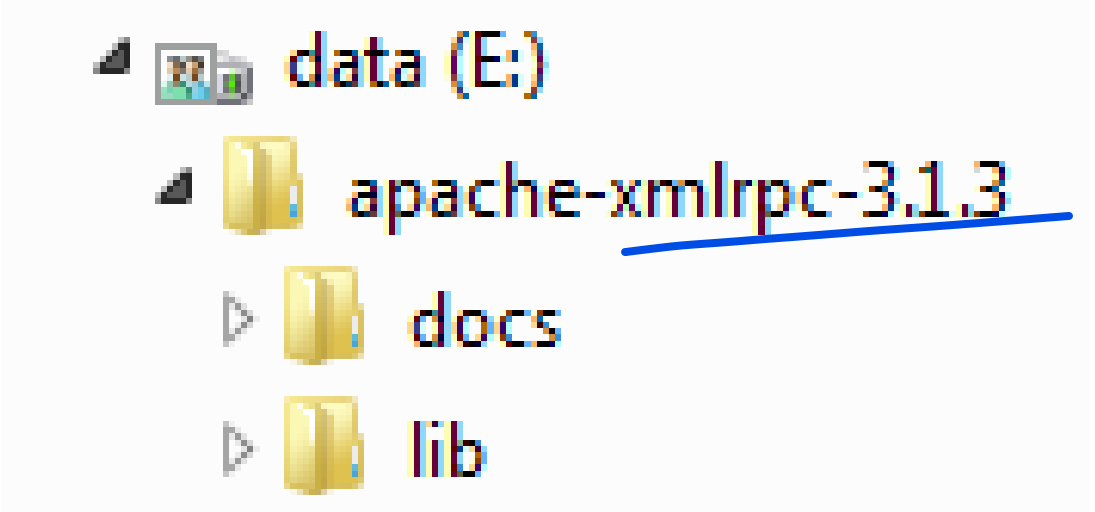
<?xml version="1.0"?>
  <methodResponse>
    <fault>
      <value>
        <struct>
          <member>
            <name>faultCode</name>
            <value><int>0</int></value>
          </member>
          <member>
            <name>faultString</name>
            <value>No such handler: fact1</value>
          </member>
        </struct>
      </value>
    </fault>
</methodResponse>
```

Apache XML-RPC

- Popular Java implementation of XML-RPC.
- Provides interfaces and classes to create XML-RPC client and server quickly
- Provides a light-weight web server to build up XML-RPC on systems that have no web server installed in them
- A light-weight servlet engine for efficiently servicing clients even under extremely high load

Installing Apache XML-RPC

- To use Apache XML-RPC Java library, download the necessary JAR files from http://ws.apache.org/xmlrpc/download.html.
- We downloaded the zip file **`apache-xmlrpc-3.1.3-bin.zip`** and unzipped it in E: drive. It creates the following directory structure:



Installing … contd.

- The docs directory contains documentation
- the lib directory contains XML-RPC Java library as 5 (five) JAR (Java ARchive) files as follows:
  - commons-logging-1.1.jar
  - ws-commons-util-1.0.2.jar
  - xmlrpc-client-3.1.3.jar
  - xmlrpc-common-3.1.3.jar
  - xmlrpc-server-3.1.3.jar
- Include all .jar files in **`CLASSPATH`** or
- Specify them during compilation and execution

XML-RPC vs. Java data types

Basic data		Apache extension	
XML Tag Name	**Java Type**	**XML Tag Name**	**Java Type**
i4, or int	Integer	ex:nil	None
boolean	Boolean	ex:i1	byte
string	String	ex:float	float
double	Double	ex:i8	long
dateTime.iso8601	java.util.Date	ex:dom	org.w3c.dom.Node
base64	byte[]	ex:i2	short
struct	java.util.Map	ex:serializable	java.io.Serializable
array	Object[]	ex:bigdecimal	BigDecimal
		ex:biginteger	BigInteger
		ex:dateTime	java.util.Calendar

Example applicaton

- Server
- The class for factorial object

```
public class FactImpl {
  public int fact(int n) {
    System.out.println("Received : "+n);
    int prod = 1;
    for(int i = 2; i <= n; i++)
      prod *= i;
    System.out.println("Sent : "+prod);
    return prod;
  }
}
```

Server cond.

- Minimal HTTP server that can handle only XML-RPC messages
  - WebServer webServer = new WebServer(6789);
- Not capable of calling a method on an object. It is provided by XmlRpcServer.
  - XmlRpcServer rpcServer = webServer.getXmlRpcServer();
- It maintains a list of named objects on which methods may be invoked
- Specify it as a map

Server cond.

- Create the map
  - PropertyHandlerMapping mapping = new PropertyHandlerMapping();
- Populate the map
  - mapping.addHandler("Factorial", FactImpl.class);
- Specify it to XMP-RPC server
  - rpcServer.setHandlerMapping(mapping);
- Start the Web server
  - webServer.start();

Client

- We create an XmlRpcClient object

```
XmlRpcClient client = new XmlRpcClient();
```

- Specify information about the XML-RPC server

```
XmlRpcClientConfigImpl config = new
 XmlRpcClientConfigImpl();

config.setServerURL(new
 URL("http://"+args[0]+":6789/"));

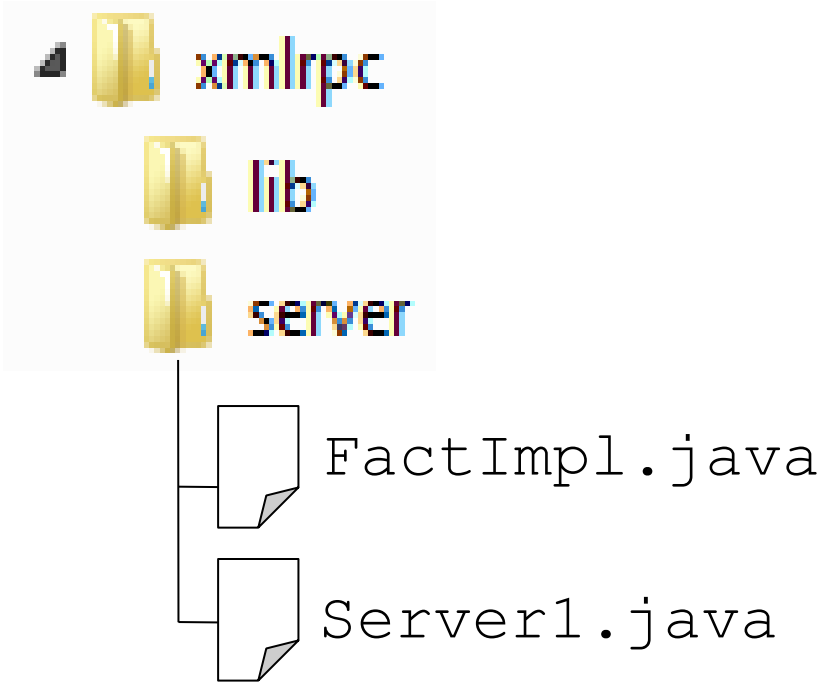
client.setConfig(config);
```

Client contd.

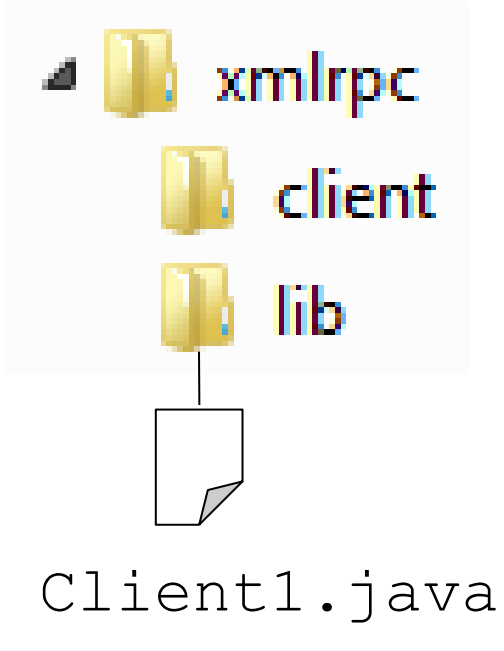
- Prepare parameters
  - int n = 6;
  - Object[] params = new Object[]{ new Integer(n)};
- Invoke the method fact() on the server object
  - Integer result = (Integer)client.execute("Factorial.fact", params);

Running the application

- Directory structure
- Server

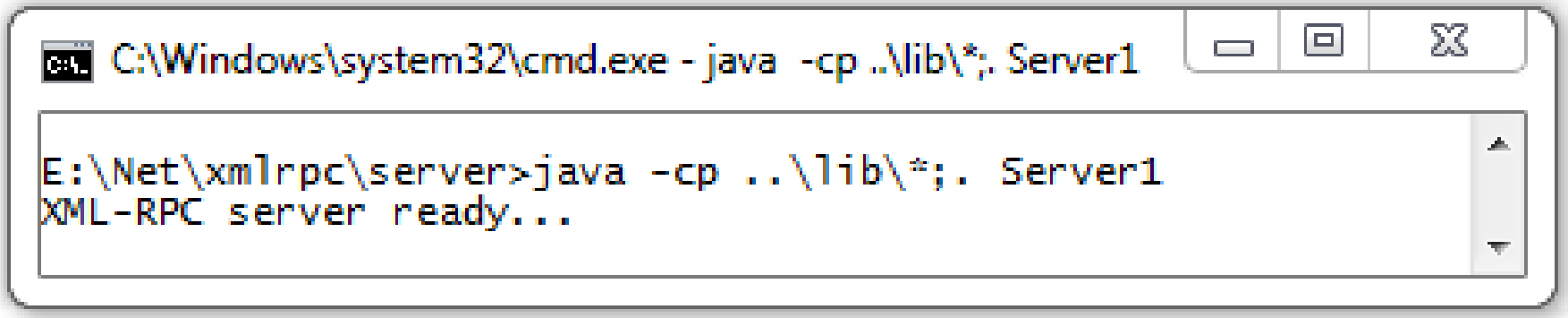


client



Starting Server

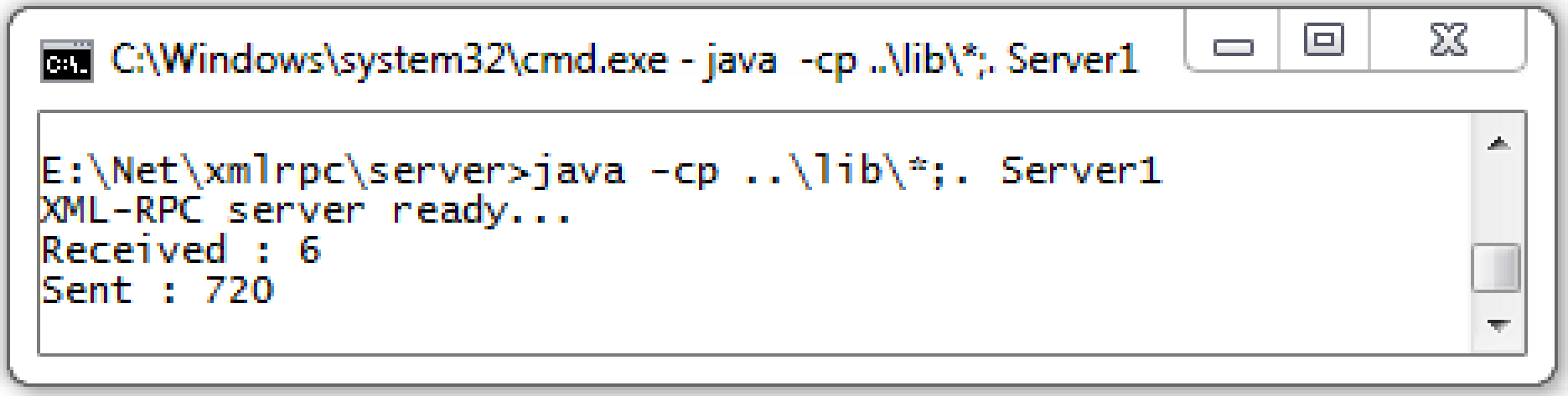
- Go to the server directory
- Use following command to compile files
  - **`javac -cp ..\lib\*;. FactImpl.java Server1.java`**
- Run the server
  - **`java -cp ..\lib\*;. Server1`**

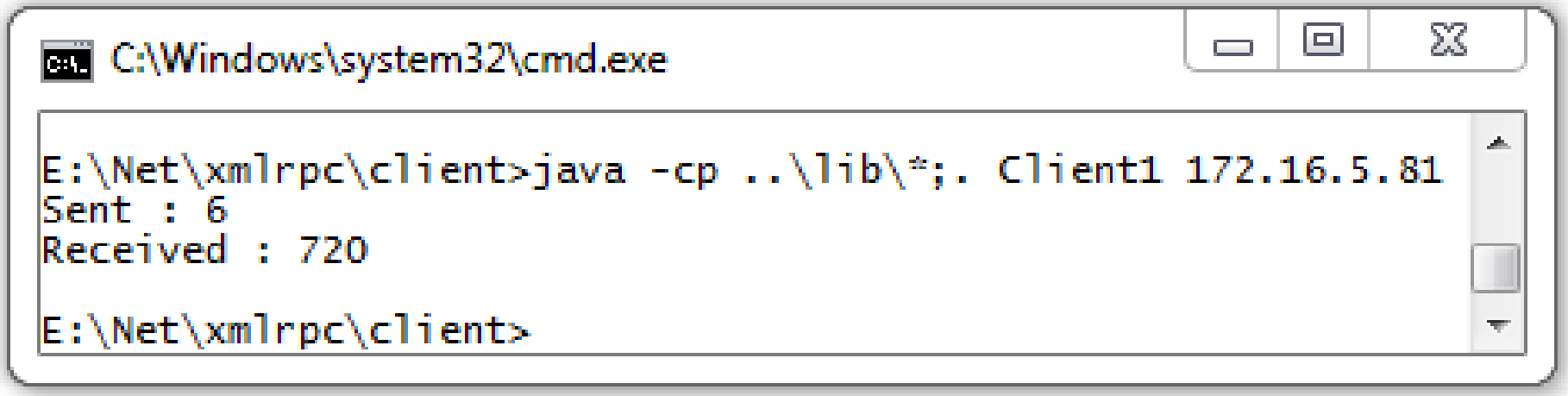


Starting client

- Go to the client directory
- Use following command to compile files
  - `javac -cp ..\lib\*;. Client1.java`
- Assume that server is running on host having IP `172.16.5.81`
- Run the client
  - `java -cp ..\lib\*;. Client1 172.16.5.81`

Result


C:\Windows\system32\cmd.exe - java -cp ..\lib*;. Server1
E:\Net\xmlrpc\server>java -cp ..\lib*;. Server1
XML-RPC server ready...
Received : 6
Sent : 720


C:\Windows\system32\cmd.exe
E:\Net\xmlrpc\client>java -cp ..\lib*;. Client1 172.16.5.81
Sent : 6
Received : 720
E:\Net\xmlrpc\client>

Dynamic Proxies

- Client used **`execute()`** method to invoke a method
- Does not use traditional method call syntax(?)
  - **`int result = obj.fact(6);`**
- Apache XML-RPC provides dynamic proxy allows us to invoke remote methods as above
- Needs an interface to be written

```
//Fact.java
public interface Fact {
  public int fact(int n);
}
```

Writing Server

- Implement the method
  - `public class FactImpl implements Fact {`
- Add an object
  - `mapping.addHandler(Fact.class.getName(), FactImpl.class);`
- **`Fact.class.getName()`** [returns “Fact”] is important as client uses interface name
- Alternatively
  - `mapping.addHandler("Fact", FactImpl.class);`

Writing Client

- Import
  - `import org.apache.xmlrpc.client.util.ClientFactory;`
- Use followig code

```
ClientFactory factory = new ClientFactory(client);
Fact dProxy = (Fact) factory.newInstance(Fact.class);

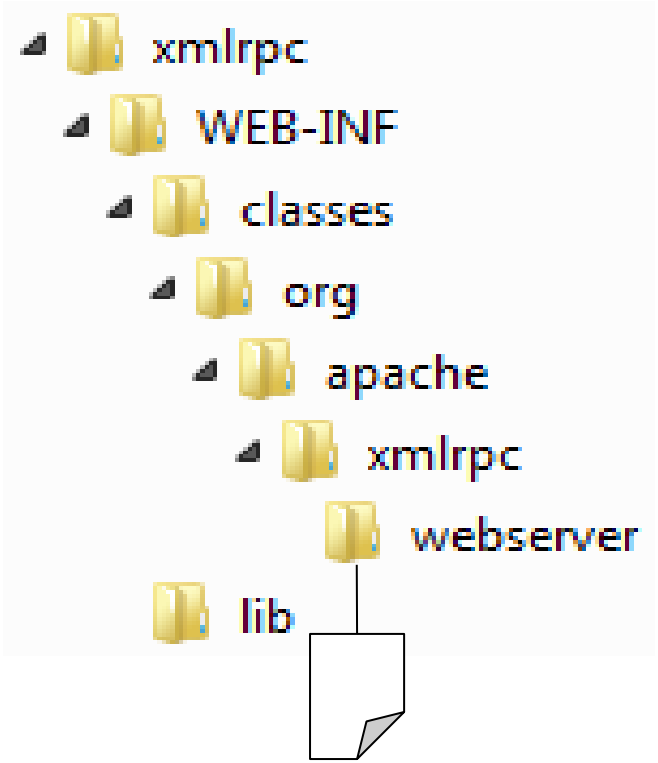

int result = dProxy.fact(n);
```

Using XmlRpcServlet

- **`WebServer`** class is light-weight and not a full-blown web server
- Fortunately, apache XML-RPC provides **`XmlRpcServlet`**
- Its instance of can be installed to any web server
- This way we can add RPC functionality to any web server
- We shall use Tomcat

Configuring tomcat

- Create following directory structure under **webapps**
- Put all XML-RPC .jar files in lib directory
- Place **FactImpl.class** in classes directory
- Make following entry in **XmlRpcServlet.properties** file
- **Factorial=FactImpl**



Configuring tomcat contd.

- Add following entries to the **WEB-INF\web.xml** file.

```
<servlet>
   <servlet-name>XmlRpcServlet</servlet-name>
   <servlet-
  class>org.apache.xmlrpc.webserver.XmlRpcServlet</se
  rvlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>XmlRpcServlet</servlet-name>
   <url-pattern>/servlet</url-pattern>
</servlet-mapping>
```

Running the application

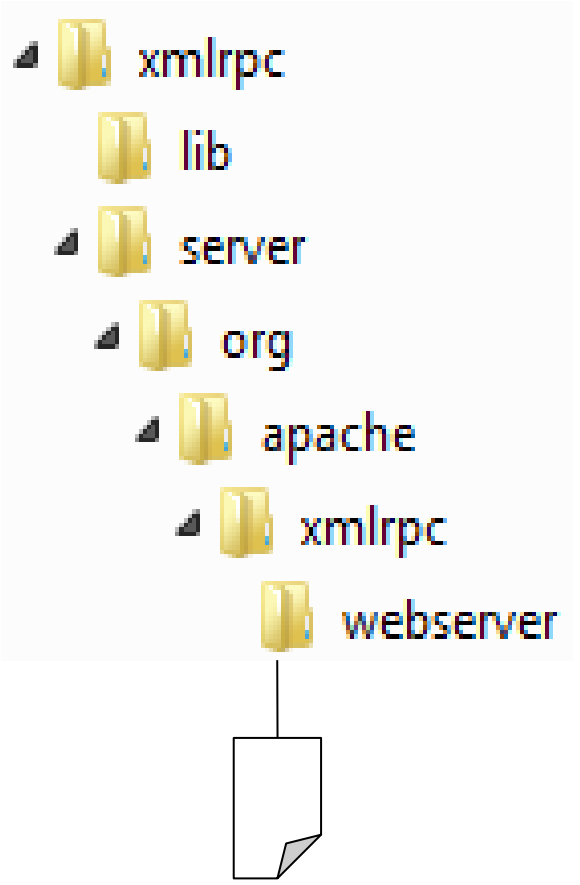
- Start the tomcat as usual
- The URL of the servlet will be
  - `http://"+args[0]+":8080/xmlrpc/servlet`
- The IP address is provided as a command line argument
- Specify this URL in the client code
- Start the client as usual

ServletWebServer

- If a **`WebServer`** is really required, Apache recommends to use its enhanced version **`ServletWebServer`**.
- It enables the programmers to bypass complicated tasks involved in setting up a full-blown web server and servlet engine.
- Use of the **`ServletWebServer`** is very simple
- Simple create a XmlRpcServlet as follows:
  - ```
    XmlRpcServlet servlet = new XmlRpcServlet();
    ```
    ```

ServletWebServer

- Create the following directory structure
- It always reads the object mapping information from a file named **`XmlRpcServlet.properties`**,
- Must be stored in the directory **`org/apache/xmlrpc/server/webserver/`**, relative to the servlet's directory.



ServletWebServer

- following entry in this file
  - `Factorial=FactImpl`
- A ServletWebServer may then be created as follows:
  - `ServletWebServer webServer = new ServletWebServer(servlet, 6789);`
- Start the web server
  - `webServer.start();`
- Run Server and client as before

Introspection

- To invoke a remote method, a client must know method's signature
- How will a client know it?
- XML-RPC provide a facility called ***introspection*** that enables a client to dynamically learn from a server specifically
  - What method names do you offer?
  - What is the signature of the given method name?
  - Can you provide me help on the given method name?

Introspection

- A client may make these queries to the server by invoking special methods
  - system.listMethods
  - system.methodSignature
  - system.methodHelp
- Server must configure itself to provide such info as follow:
  - `import org.apache.xmlrpc.metadata.*;`
  - `XmlRpcSystemImpl.addSystemHandler(mapping);`

Client

- To examine the returned information, we have developed a simple client
  - sends a simple XML-RPC request message using socket
  - prints a XML-RPC response message.
- IntroClient.java

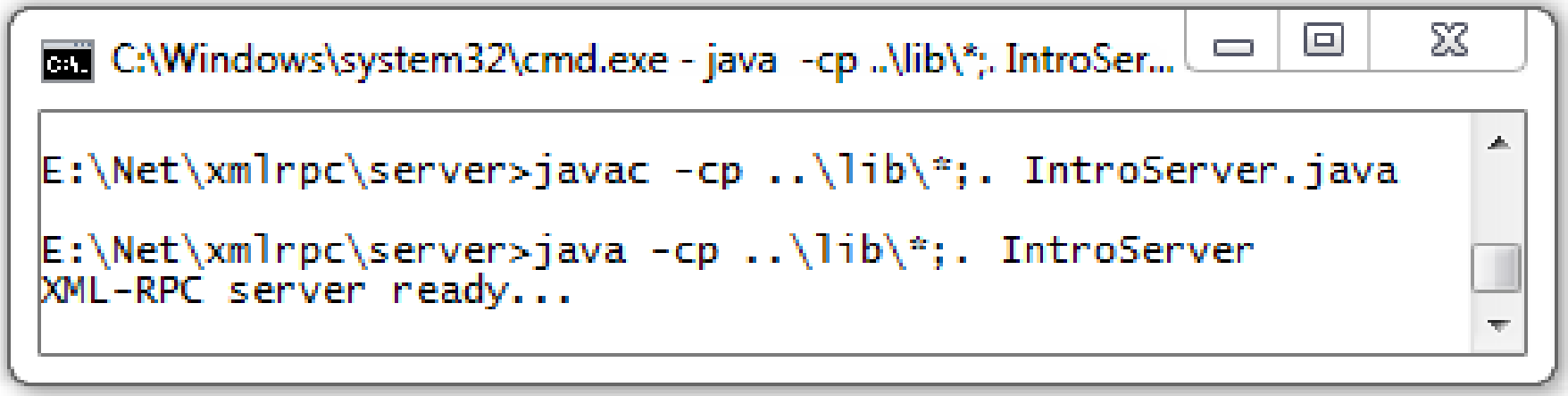
Client

- list.xml

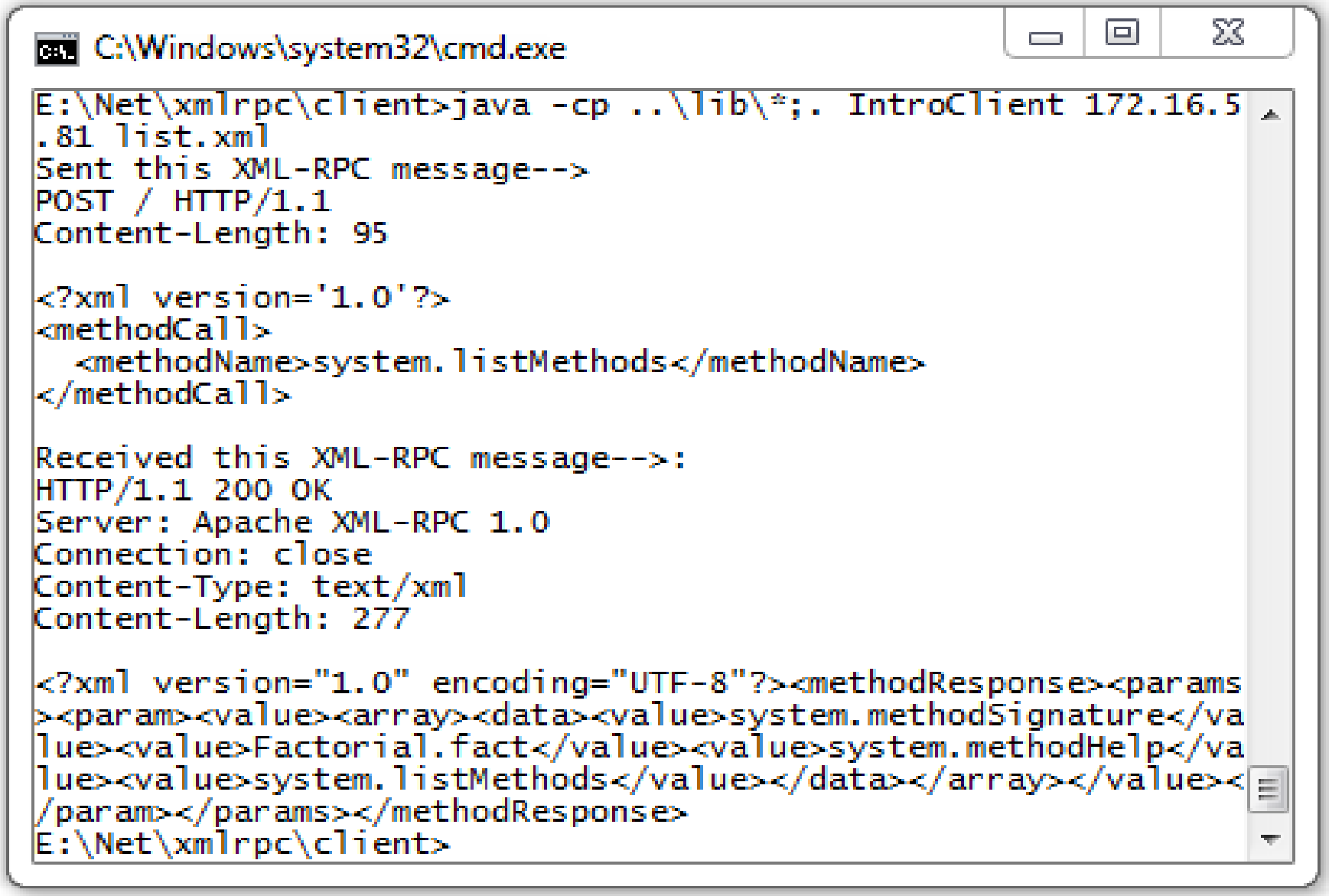
```
<?xml version='1.0'?>
<methodCall>
  <methodName>system.listMethods</methodName>
</methodCall>
```

- Run the client with this XML file
  - **java -cp ..\lib*;. IntroClient 172.16.5.81 list.xml**
- sd

Server


C:\Windows\system32\cmd.exe - java -cp ..\lib*;. IntroSer...
E:\Net\xmlrpc\server>javac -cp ..\lib*;. IntroServer.java
E:\Net\xmlrpc\server>java -cp ..\lib*;. IntroServer
XML-RPC server ready...

Client


C:\Windows\system32\cmd.exe
E:\Net\xmlrpc\client>java -cp ..\lib*;. IntroClient 172.16.5
.81 list.xml
Sent this XML-RPC message-->
POST / HTTP/1.1
Content-Length: 95
<?xml version='1.0'?>
<methodCall>
<methodName>system.listMethods</methodName>
</methodCall>
Received this XML-RPC message-->:
HTTP/1.1 200 OK
Server: Apache XML-RPC 1.0
Connection: close
Content-Type: text/xml
Content-Length: 277
<?xml version="1.0" encoding="UTF-8"?><methodResponse><params
><param><value><array><data><value>system.methodSignature</va
lue><value>Factorial.fact</value><value>system.methodHelp</va
lue><value>system.listMethods</value></data></array></value><
/param></params></methodResponse>
E:\Net\xmlrpc\client>

Server returns

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value>system.methodSignature</value>
            <value>Factorial.fact</value>
            <value>system.methodHelp</value>
            <value>system.listMethods</value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodResponse>
```

XML file

```
<?xml version='1.0'?>
<methodCall>
  <methodName>system.methodSignature</methodName>
  <params>
    <param>                    \
      <value><string>Factorial.fact</string></value>
    </param>
  </params>
</methodCall>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
<params>
  <param>
    <value>
      <array>                  <!—array of signatures -->
        <data>
          <value>    <!—one signature found -->
            <array>
       <data>
                 <value>int</value>   <!—return type -->
            <value>int</value>        <!—first argument -->
              </data>
               </array>
          </value>
        </data>
      </array>
    </value>
  </param>
</params>
</methodResponse>
```

XML file

```
<?xml version='1.0'?>
<methodCall>
  <methodName>system.methodHelp</methodName>
  <params>
    <param>
      <value><string>Factorial.fact</string></value>
    </param>
  </params>
</methodCall>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
<params>
  <param>
    <value>Invokes the method
 FactImpl.fact(int).</value>
  </param>
</params>
</methodResponse>
```

Limitations of XML-RPC

- Choice of data types is limited.
- No facility to represent NaN for floating point number.
- Only ASCII characters are allowed in strings.
- There is no provision for passing objects.
- No way to check types of array values
- No facility to check if a struct has duplicate names.
- Little or no provision for security
- The XML-RPC specification is now frozen