In diesem Kapitel

- Schreiben von XML Dateien
  - *Rahmenprogramm*
  - *Attribute*
  - *Validation*
  - *Namensräume*
- Ausgabevarianten
  - *Streams*
  - *Writers*
  - *Codierung*
- Servlets
  - *Grunsätzlicher Aufbau*
- XML-RPC
  - *Client Aufbau*
- SOAP
  - *Client Aufbau*
- Zusammenfassung

3. *XML Ausgabe*

3.1. Einleitung

Die Ausgabe von XML geschieht mithilfe eines `FileOutputStream`. Als Testanwendung definieren wir einen Service, welcher uns die Primzahlen bis zu einer bestimmten eingegebenen Zahl liefern soll. Dazu verwenden wir einen Algorithmus, welcher mit hoher Wahrscheinlichkeit eine Primzahl liefert.

In Java steht Ihnen dazu in der Klasse `BigInteger` die Methode `..isProbablePrime(iZahl)` zur Verfügung. Das folgende Beispielprogramm verwendet diese Methode, um (mit einer Wahrscheinlichkeit von 1-1:1'000'000) eine Primzahl zu liefern.

```
/**
 * <p>Title: Primzahl-Berechnung</p>
.* <p>Copyright: Copyright (c) 2003</p>
 * @author J.M.Joller
 * @version 1.0
 */
import java.math.*;
import java.util.*;

public class PrimZahl {
    public int getPrime(int i) {
        // solange i keine Primzahl ist, addiere 1
        while (!BigInteger.valueOf(i).isProbablePrime(20)) {
          System.out.println("Primzahlsuche : "+BigInteger.valueOf(i));
          i++;
        }
        // wahrscheinlich ist i jetzt Primzahl
        // (mit Wahrscheinlichkeit 1- 1/(2^20) ~= 1 zu einer Million)
        // Die 20 stammt aus dem Aufruf 'isProbablePrime(20)'
        return i;
    }

    public static void main(String[] args) {
      System.out.println("[PrimZahl]main()");
      PrimZahl prime = new PrimZahl();
      int iTest = 2134;
```


© J.M.Joller

```
        System.out.println("Primzahl zu "+iTest+" : "+prime.getPrime(iTest));
    }
}
```

Beispielausgabe:

```
[PrimZahl]main()
Primzahlsuche : 2134
Primzahlsuche : 2135
Primzahlsuche : 2136
Nächste grössere Primzahl zu 2134 : 2137
```

Damit die Primzahlmethode remote benutzt werden kann, wird ein Servlet eingesetzt. Zudem können mehrere Primzahlen (von...bis) abgefragt werden, als Erweiterung des obigen Beispiels.

Das Ergebnis wird als XML Dokument an den Client geliefert.

3.2. Schreiben eines XML Dokuments aus Java

Als erstes variieren wir das Primzahlen Programm und erzeugen als Ausgabe eine XML Datei, welche die Primzahlen als Elemente enthält:

```
<?xml version="1.0" ?>
- <PrimZahlen>
    <primzahl>2137</primzahl>
    <primzahl>2141</primzahl>
    <primzahl>2143</primzahl>
    <primzahl>2153</primzahl>
    <primzahl>2161</primzahl>
    <primzahl>2179</primzahl>
    <primzahl>2203</primzahl>
  </PrimZahlen>
```

Das Programm dazu ist denkbar einfach: wir definieren die Tags als Konstanten und schreiben die Ausgabe in eine Datei. Dazu verwenden wir einen `FileStream`, weil in XML auch binäre Daten möglich wären. Für das Beispiel wäre auch ein `Writer` denkbar.

3.2.1. Attribute

Nun möchten wir zu jedem Integer, zu dem wir eine Primzahl suchen, die gefundene Primzahl ausgeben. Der Integer soll dabei als Attribut des `<primzahl>` Tags ausgegeben werden:

```
<?xml version="1.0" ?>
- <PrimZahlen>
    <primzahl Startzahl="2040">2053</primzahl>
    <primzahl Startzahl="2041">2053</primzahl>
  ..
    <primzahl Startzahl="2078">2081</primzahl>
    <primzahl Startzahl="2079">2081</primzahl>
  </PrimZahlen>
```

Die Programmänderungen können Sie im Programm `PrimZahlXMLAttribute` nachsehen. Wichtig ist, dass Sie sich wieder mit den Klassen aus `java.util` vertraut machen. Sonst müssen Sie selber Red-Black-Trees, Hash-tabellen usw. implementieren.

3.2.2. Generieren von gültigen XML Dokumenten

Wenn wir die Gültigkeit unserer XML-Ausgabe überprüfen wollen, können wir einfach noch eine DTD ausgeben. Eine mögliche DTD für unser Beispiel ist:

© J.M.Joller

```
<!DOCTYPE PrimZahlen [
  <!ELEMENT PrimZahlen  (primzahl)*>
  <!ELEMENT primzahl (#PCDATA)>
  <!ATTLIST primzahl Startzahl NMTOKEN #REQUIRED>
```

Falls wir die DTD in einer separaten Datei abspeichern:

```
<?xml version="1.0"?>
<!DOCTYPE PrimZahlen SYSTEM
"C:\Joller\UnterrichtsUnterlagen\EnterpriseComputing\JavaUndXML\Beispiele\Kapitel03\Prim
zahlen.dtd">
<PrimZahlen>
    <primzahl Startzahl="2040">2053</primzahl>
…
    <primzahl Startzahl="2079">2081</primzahl>
</PrimZahlen>
```

Falls wir die DTD intern definieren:

```
<?xml version="1.0"?>
<!DOCTYPE PrimZahlen [
  <!ELEMENT PrimZahlen  (primzahl)*>
  <!ELEMENT primzahl (#PCDATA)>
  <!ATTLIST primzahl Startzahl NMTOKEN #REQUIRED>
]>
<PrimZahlen>
    <primzahl Startzahl="2040">2053</primzahl>

    <primzahl Startzahl="2079">2081</primzahl>
</PrimZahlen>
```

Ich habe zur Sicherheit die Syntax in XML Spy überprüft und von XML Spy eine Variante des DTD generieren lassen. Diese ist eher ungünstig. Sie finden sie unter `PrimAttrInterneDTD.xml` bei den Beispielen.

Das Programm `PrimZahlXMLMitDTD`, welches auch eine DTD generiert, finden Sie ebenfalls bei den Beispielen, zusammen mit der Ausgabe `PrimMitDTD.xml`.

Analog kann man vorgehen, wenn man ein XML Schema einbinden will:

```
<?xml version="1.0"?>
<PrimZahlen xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Joller\UnterrichtsUnterlagen\EnterpriseComputing\Ja
vaUndXML\Beispiele\Kapitel03\Primzahlen.xsd">
    <primzahl Startzahl="2040">2053</primzahl>
…
    <primzahl Startzahl="2079">2081</primzahl>
</PrimZahlen>
```

In diesem Fall muss die Syntax-Spezifikation des XML Dokuments extern in einer separaten Datei abgespeichert sein.

Hier das entsprechende Schema :

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v5 U (http://www.xmlspy.com)-->
```


© J.M.Joller

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="PrimZahlen">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="primzahl" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="primzahl">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="Startzahl" type="xs:NMTOKEN" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3.2.3. Namensräume - Namespaces

Die Mathematiker haben ihr eigenes XML Universum definiert: MathML. Der entsprechende Namensraum ist `http://www.w3.org/1998/Math/MathML` und als Präfix wird `mathml` verwendet. Die Definition von MathML finden Sie unter `http://www.w3.org/Math/`
Auf ein Java Beispiel dazu habe ich verzichtet. Aber interessant ist der Ansatz!

3.3. Output Streams, Writers und Encodings

In den obigen Beispielen haben wir jeweils bereits die ausgabe auf den Bildschrim und in eine Datei geleitet. Wichtig ist, dass ein Stream, nicht ein Writer verwendet wurde, wegen der Codierung. Auf die einzelnen Code Tabellen von ISO und Unicode und deren Umsetzung in Java müssen wir hier sicher nicht eingehen!

© J.M.Joller

3.4. Servlets

Wir haben in Kapitel 2 einige Beispiele erstellt und jeweils gleich den Server Teil mitgeliefert. Der Grund ist der, dass Sie zum testen Ihrer Clients zwingend einen Server besitzen müssen. Mit XML-RPC ist dies sehr einfach, da gleich ein kleiner standalone Server mitgeliefert wird. Bei SOAP (und Axis) müssen wir Servlets einsetzen. Dazu verwende ich Jakarta Tomcat. Eine Kurzanleitung zur Installation und dem Testen von Jakarta Tomcat finden Sie auf dem Web / Server.

Die einfachsten Server lesen eine Nachricht und schreiben diese in System.out. Komplexere Server (und Clients) analysieren die Nachricht mit XML Parsern. Dazu kommen wir später (Sie finden Beispiele dazu bereits in Kapitel 2).

Bei Servlets werden die http Befehle in Form von Methoden implementiert:

- `GET` wird durch `doGet()`
- `POST` durch `doPost()`

Für unser Beispiel mit den Primzahlen sind zwei Parameter definiert:

- die Startzahl
- die Variable `debug`, welche die Ausgabe in der Methode `getPrime()` regelt.

Beispiel:

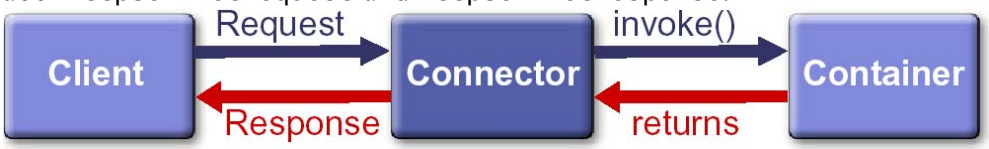
```
http://localhost:8080/JavaXML3/servlet/javaxml.PrimzahlServlet?startZahl=9495&&debug=true
```

Die Eingabeparameter des Servlets (`startZahl`, `debug`) kann man mit der Methode `getParameter(String)` abfragen:

```
String strStart = request.getParameter("startZahl");
```

Daneben könnten auch die Methoden `getParameterValues()` oder `getParameterNames()` eingesetzt werden.

Die Kommunikation mit dem Web Server (und somit dem Web) geschieht mithilfe von `Writer` und `Reader`, die vom Servlet Container dem Programmierer zur Verfügung gestellt werden, über `HttpServletRequest` und `HttpServletResponse`.



Beispiel:

```
package javaxml;

import java.io.IOException;
import java.io.PrintWriter;
import java.math.BigInteger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author jjoller;
 * @version 1.0
```


© J.M.Joller

```
 *
 * Einfaches Servlet, welches ein Eingabe einer Integer Zahl die nächst
mögliche Primzahl liefert.
 * Created on 08.08.2003
 *
 * Beispiel:
 *
http://localhost:8080/JavaXML3/servlet/javaxml.PrimzahlServlet?startZahl=34
5&&debug=true
 * oder
 *
http://localhost:8080/JavaXML3/servlet/javaxml.PrimzahlServlet?startZahl=54
86
 */
public class PrimzahlServlet extends HttpServlet {

      /**
       * GET Befehl
       * @param : Request und Response
       */
      public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
            System.out.println("[PrimzahlServlet]doGet()");
            /**
             * Lesen der Eingabezahl
             */
            int iStart = 10;
            boolean bDebug = false;
            String strStart = request.getParameter("startZahl");
            String strDebug = "false";
            strDebug = request.getParameter("debug");
            if (strDebug == null)
                  strDebug = "false";
            try {
                  iStart = Integer.parseInt(strStart);
            } catch (Exception e) {
                  // NumberFormat oder NullPointerException
                  // somit wird iStart Standardwert verwendet
            }
            System.out.println("[PrimzahlServlet.doGet()]Eingabe \t" +
                                                            iStart);
            System.out.println("[PrimzahlServlet.doGet()]        \t" +
                                                            strDebug);
            bDebug = strDebug.equalsIgnoreCase("true");

            System.out.println(
            "[PrimzahlServlet.doGet()]generieren der Antwort in XML");
            /**
             * MIME Type
             */
            response.setContentType("text/xml; charset=UTF-8");
            PrintWriter pw = response.getWriter();

            /**
             * XML Header
             */
            pw.println("<?xml version=\"1.0\"?>");
            /**
             * Formattierung der Ausgabe mit CSS (muss ich noch erstellen!)
             */
            //pw.print  ("<?xml-stylesheet ");
```

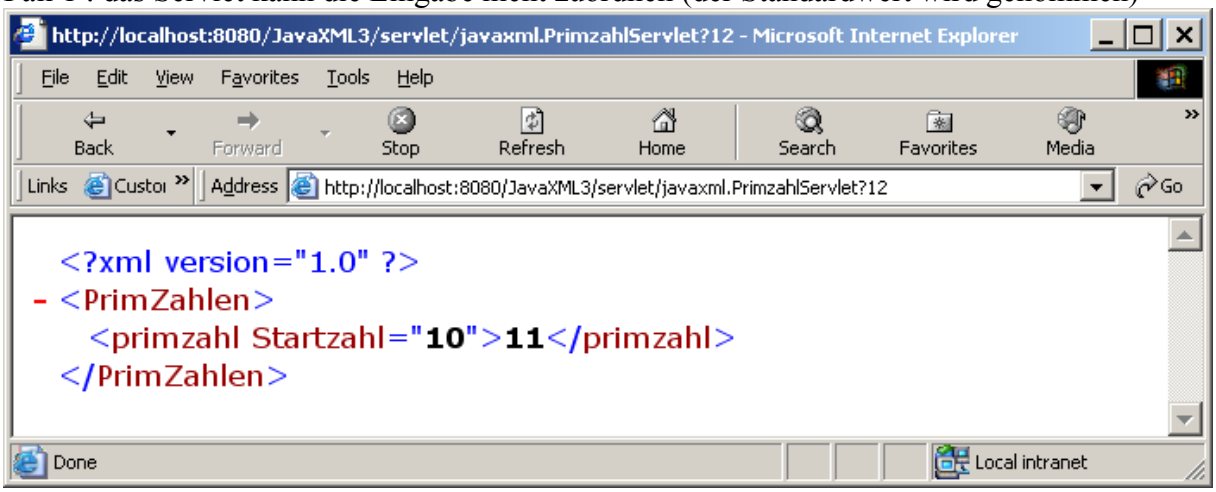

© J.M.Joller

```
            //pw.println("type='text/css' href='styles/primzahlen.css'?>");

            /**
             * Bestimmen der Primzahl zu iStart ohne Debug Info-Ausgabe
             */
            int iPrime = this.getPrime(iStart, bDebug);
            pw.println("<PrimZahlen>");

            /**
             * Ausgabe der Primzahl
             */
            pw.print("  <primzahl Startzahl=\"" + iStart + "\">");
            pw.print(iPrime);

            /**
             * End-Tags
             */
            pw.println("</primzahl>");
            pw.println("</PrimZahlen>");
            pw.close();
      }

      /**
       * doPost sendet die Antwort an den Client bei POST Anfragen
       * Hier ruft die Methode einfach doGet() auf.
       */
      public void doPost(
            HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

            System.out.println("[PrimzahlServlet]doGet()");
            this.doGet(request, response);

      }
      public int getPrime(int i, boolean debug) {
            System.out.println(
                  "[PrimzahlServlet]getPrime(" + i + "," + debug + ")");
            // solange i keine Primzahl ist, addiere 1
            while (!BigInteger.valueOf(i).isProbablePrime(20)) {
                  if (debug)
                        System.out.println("Primzahlsuche : " +
                                                BigInteger.valueOf(i));
                  i++;
            }
            // wahrscheinlich ist i jetzt Primzahl
            // (mit Wahrscheinlichkeit 1- 1/(2^20) ~= 1 zu einer Million)
            return i;
      }

}
```

Zur Zeit verwende ich Eclipse mit dem Tomcat Plugin von Sysdeo. Dies gestattet den Start von Tomcat inklusive HotReplace (Debug) in Eclipse. Insgesamt ist die Programmierung etwas schwieriger als mit JBuilder (man muss etwas mehr im aktiven Gedächtnis haben). Aber Eclipse benötigt scheinbar weniger Ressourcen.

© J.M.Joller

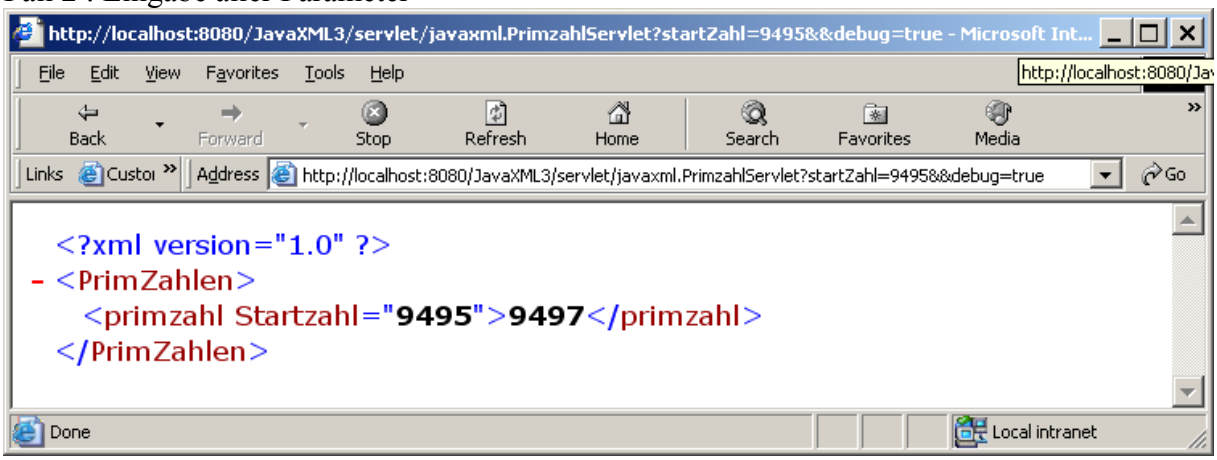
Den Aufruf des Servlets finden Sie als Kommentar im Servlet. Hier einige mögliche Ausgabe:
Fall 1 : das Servlet kann die Eingabe nicht zuordnen (der Standardwert wird genommen)



Und hier die Ausgabe im Server (vom Servlet):

```
[PrimzahlServlet]doGet()
[PrimzahlServlet.doGet()]Eingabe    10
[PrimzahlServlet.doGet()]           false
[PrimzahlServlet.doGet()]generieren der Antwort in XML
[PrimzahlServlet]getPrime(10,false)
```

Fall 2 : Eingabe aller Parameter




```
[PrimzahlServlet]doGet()
[PrimzahlServlet.doGet()]Eingabe    9495
[PrimzahlServlet.doGet()]           true
[PrimzahlServlet.doGet()]generieren der Antwort in XML
[PrimzahlServlet]getPrime(9495,true)
Primzahlsuche : 9495
Primzahlsuche : 9496
```


© J.M.Joller

3.5. Ein XML-RPC Client

Die Implementierung von Apache XML Client und Server haben wir bereits in Kapitel 2 kurz besprochen. Als Anwendungsbeispiel der dort vorgestellten Programme plus dem Apache Tomcat Tunnel betrachten wir als beispiel den Primzahl-Server. Zu beachten sind die Parameter Datentypen (`int` wird zu `Integer`,...).

Beispiel:
(`SendToURL`, `XMLRPCPrimeServer`, und die Apache Tunnel Applikation
`javaw org.apache.soap.util.net.TcpTunnelGui 5555 localhost 8080`)
Client Anfrage

```
<?xml version="1.0"?>
<methodCall>
   <methodName>primeServer.getPrime</methodName>
   <params>
      <param>
         <value>
            <int>2345</int>
         </value>
      </param>
      <param>
         <value>
            <boolean>true</boolean>
         </value>
      </param>
   </params>
</methodCall>
```

Server Antwort

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
   <params>
      <param>
         <value>
            <int>2347</int>
         </value>
      </param>
   </params>
</methodResponse>
```

Dieses Beispiel verwendet die Klasse java.net.HttpURLConnection. Diese verwendet intern POST. Im Beispiel `XMLRPCPrimeClient` ist XML nicht mehr sichtbar, ausser im Tunnel.

© J.M.Joller

Beispiel
SentToURL

```
public class SendToURL {

	public static void main(String[] args) {
		System.out.println("[SendToURL]main()");
		try {
			URL url = new URL("http://localhost:5555/");
			URLConnection urlConnection = url.openConnection();
			urlConnection.setDoOutput(true);
			OutputStreamWriter wr = new
			OutputStreamWriter(urlConnection.getOutputStream());

			String xml_rpc_body =   "<?xml version=\"1.0\"?>\n"+
								"<methodCall>\n"+
		"<methodName>primeServer.getPrime</methodName>\n"+
								"<params>\n"+
								"<param>\n"+
						"<value><int>2345</int></value>\n"+
								"</param>\n"+
								"<param>\n"+
				"<value><boolean>true</boolean></value>\n"+
								"</param>\n"+
								"</params>\n"+
								"</methodCall>\n";
			wr.write(xml_rpc_body);
			wr.flush();

			// Get the response
			BufferedReader rd = new BufferedReader(new
			InputStreamReader(urlConnection.getInputStream()));
			String line;
			BufferedWriter bw = new BufferedWriter(new
						FileWriter("XMLRPCPrimzahl.xml"));
			while ((line = rd.readLine()) != null) {
				bw.write(line);
			}
			bw.flush();
			wr.close();
			rd.close();

		} catch(MalformedURLException urlexp) {
			urlexp.printStackTrace();
		} catch(IOException ioe) {
			ioe.printStackTrace();
		}
	}
}
```


© J.M.Joller

Beispiel
XMLRPCPrimeClient

```
package javaxml;

import java.util.Vector;
import org.apache.xmlrpc.XmlRpcClient;
import org.apache.xmlrpc.XmlRpcException;

public class XMLRPCPrimeClient {

	// Server.
	private final static String server_url =
		"http://localhost:5555/";

	public static void main (String [] args) {
		try {
		// Link auf den Server
		XmlRpcClient serverAnbindung = new XmlRpcClient(server_url);

			// Parameterliste für den Aufruf
			Vector params = new Vector();
			System.out.println("[XMLRPCPrimeClient]Parameter");
			Integer oZahl = new Integer("123");
			Boolean oDebug = new Boolean("true");
			params.add(0,oZahl);
			params.add(1,oDebug);

			// Aufruf des Servers
			System.out.println("[XMLRPCPrimeClient]getPrime()");
			Integer oPrime =
			(Integer)serverAnbindung.execute("primeServer.getPrime",
								params);

			// Ergebnis-Ausgabe
			System.out.println("[XMLRPCPrimeClient]Eingabe: " +
								oZahl.toString());
			System.out.println("[XMLRPCPrimeClient]Ausgabe: " +
                                                oPrime.toString());

		} catch (XmlRpcException exception) {
			System.err.println("[XMLRPCPrimeClient]XML-RPC Fault #" +
                                        Integer.toString(exception.code) + ": " +
                                                exception.toString());
			exception.printStackTrace();
		} catch (Exception exception) {
			System.err.println("[XMLRPCPrimeClient]Fehler: " +
                                                exception.toString());
			exception.printStackTrace();
		}
	}
}
```


© J.M.Joller

Beispiel
XMLRPCPrimeServer

```
public class XMLRPCPrimeServer {
	public int getPrime(int i, boolean debug) {
		System.out.println(
			"[PrimzahlServlet]getPrime(" + i + "," + debug + ")");
		// solange i keine Primzahl ist, addiere 1
		while (!BigInteger.valueOf(i).isProbablePrime(20)) {
			if (debug)
		System.out.println("Primzahlsuche : " + BigInteger.valueOf(i));
			i++;
		}
		// wahrscheinlich ist i jetzt Primzahl
		// (mit Wahrscheinlichkeit 1- 1/(2^20) ~= 1 zu einer Million)
		return i;
	}
	public static void main(String[] args) {
		System.out.println("[XMLRPCPrimeServer]main()");
		try {
			WebServer server = new WebServer(8080);
			server.addHandler("primeServer", new
								XMLRPCPrimeServer());
		} catch (Exception exception) {
			System.err.println("[XMLRPCPrimeServer]" +
								exception.toString());
		}
	}
}
```

Beispiel
Client

```
POST / HTTP/1.1
Content-Length: 224
Content-Type: text/xml
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2
Host: localhost:5555
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>primeServer.getPrime</methodName>
   <params>
      <param>
         <value>
            <int>123</int>
         </value>
      </param>
      <param>
         <value>
            <boolean>1</boolean>
         </value>
      </param>
   </params>
</methodCall>
```

Beispiel

© J.M.Joller

Server

```
HTTP/1.1 200 OK
Server: Apache XML-RPC 1.0
Connection: close
Content-Type: text/xml
Content-Length: 137
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
   <params>
      <param>
         <value>
            <int>127</int>
         </value>
      </param>
   </params>
</methodResponse>
```

3.6. Ein SOAP Client

Bei SOAP kommen noch Header (optional) und Umschlag hinzu. Wenn Sie eine SOAP Message an das Primzahl Servlet senden, erhalten Sie einfach die Standard-Antwort, da SOAP nicht explizit verwendet wird, und somit die Message nicht ausgepackt werden kann.

Wenn Sie auf der Server Seite SOAP analysieren, können Sie beispielsweise mit DOM (siehe spätere Kapitel) von Apache (Xerxes-J) einsetzen.

Beispiel:

SOAP Message (*SOAPPrimeClient* an *SOAPPrimzahlServlet*)

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
<javaxml:MessageHeader xmlns:javaxml="urn:javaxml" SOAP-ENV:mustUnderstand="1"
xmlns:SOAP-ENV="urn:javaxml"><From>From_Client</From><To>To_Client</To>
<MessageId>9999</MessageId></javaxml:MessageHeader>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<methodCall>
   <methodName>primeServer.getPrime</methodName>
   <params>
      <param>
         <value>
            <int>123</int>
         </value>
      </param>
      <param>
         <value>
            <boolean>1</boolean>
         </value>
      </param>
   </params>
</methodCall>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Und hier die Antwort von *SOAPPrimzahlServlet*:

© J.M.Joller

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <PrimZahlen>
         <primzahl Startzahl="123">127</primzahl>
      </PrimZahlen>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Der Client sieht im Wesentlichen wie der XML-RPC Client aus, falls wir die URLConnection Klasse verwenden. Die Klassen SOAPPrimeClient und SOAPPrimzahlServlet sehen wesentlich anders aus, weil darin der DOM Parser eingesetzt wird (und Apache Xerxes).

3.7. Zusammenfassung

Da XML Dokumente Text-Dokumente sind, ist es recht einfach mithilfe der java.net Klassen Clients und Server zu schreiben. Die echte Funktionalität von XML-RPC und SOAP kann man aber nur einsetzen, wenn die entsprechenden Klassen (Parser,...) eingesetzt werden.

In den folgenden Kapiteln werden wir den Einsatz unterschiedlicher Parser kennen lernen.

3. XML AUSGABE 1

3.1. EINLEITUNG 1
3.2. SCHREIBEN EINES XML DOKUMENTS AUS JAVA 2
3.2.1. Attribute 2
3.2.2. Generieren von gültigen XML Dokumenten 2
3.2.3. Namensräume - Namespaces 4
3.3. OUTPUT STREAMS, WRITERS UND ENCODINGS 4
3.4. SERVLETS 5
3.5. EIN XML-RPC CLIENT 9
3.6. EIN SOAP CLIENT 13
3.7. ZUSAMMENFASSUNG 14

© J.M.Joller