-
-
-
-
-
-
-
-
-

XML-RPC

Programmieren von Web Services

Kreieren von Web Applikation Gateways

(Apache) XML-RPC

Ziele

- Sie kennen XML-RPC
  - Datentypen
  - Meldungsaufbau
- Sie können XML-RPC einsetzen
  - Clientseitig
  - Serverseitig

Motivation

- In Enterprise Application Integration (EAI) setzt man primär auf die klassischen Technologien:
  - *synchrone* Kommunikation mittels CORBA
  - *asynchrone* Kommunikation mittels Messaging

Motivation

- Zitat:
  - "…XML-RPC is XML over HTTP, and a great way to develop Web Services. But there's actually more going on here – there's a philosophy to XML-RPC that's different from other software projects. The philosophy is choice, and from choice comes power, and interestingly, a disclaimer of power…."

    Dave Winer, Userland Software , 2001

Motivation

- Der zentrale Ansatz in dieser Richtung (einfache, lose Kopplung) sind die *Web Services*.
  - „leichte" Web Services:
    - XML-RPC
  - komplexere Systeme:
    - SOAP plus
    - Web Service Description Language (WSDL) plus
    - Lookup Dienste Universal Description, Discovery, and Integration(UDDI)

(Apache) XML-RPC - Einleitung

Einleitung

- Was ist XML-RPC?
- Warum XML-RPC?
- Quick Tour durch das Minenfeld

Einleitung

- Problem:
  - die gewünschte Applikation ist nicht in Ihrer gewünschten *Programmiersprache* geschrieben
  - die *Betriebsysteme* auf den unterschiedlichen Rechnern sind inkompatibel.
  - *Datentypen* und *Methodenaufrufe* unterscheiden sich in den verschiedenen Systemen

Einleitung

- Web Services
  - sind ein Set von Tools, mit denen man verteilte Applikationen bauen kann, welche auf einer bestehenden Wen Infrastruktur aufbauen.
  - Diese Applikationen setzen den Web als eine Art „Transport Layer" ein, *bieten aber keinerlei Benutzerinterface über den Browser an*.
  - Der Einsatz der Web Technologie kann zu deutlich tieferen Kosten für die Projektrealisierung führen und zudem die Wiederverwendung bestehender Web Werkzeuge erlauben.

Einleitung

- XML-RPC
  - ist eines der einfachsten und fehlertolerantesten Web Service Konzepte.
  - Damit wird es sehr einfach, Prozeduraufrufe auf entfernten Rechnern durchzuführen.
  - XML-RPC verwendet die Infrastruktur, welche ursprünglich für die Kommunikation zwischen Web Applikationen entwickelt wurde: XML und HTTP
    - XML stellt sozusagen das Vokabular für die Beschreibung der Remote Procedure Calls zur Verfügung.
    - HTTP stellt die Transportmechanismen (speziell POST) zur Verfügung.

(Apache) XML-RPC - Einleitung

Einleitung

- Sie können aus einer Vielzahl Sprachen eine auswählen:
  - Java
  - Python
  - PHP
  - Perl
  - C/C++
- Es wurde sogar schon eine XML-RPC Variante in JavaScript realisiert.

Einleitung : Wie funktioniert XML-RPC?

- Remote Procedure Calls (RPC)
  - sind eine alte Technologie, die auf Sun und Apollo (heute HP) und DEC Workstations implementiert wurde. Die ersten Entwürfe wurden um 1980 publiziert
  - RPC formalisierte die gängigen Ideen, definierte auch eine RPC IDL Sprache.
  - RPC passte in das in den 90er Jahren vorherrschende prozedurale Programmierparadigma.

Einleitung : Wie funktioniert XML-RPC?

- Betrachten wir ein schematisches Beispiel:
  - Sie möchten den Impuls eines Gegenstandes berechnen (Physik: Impuls = Masse * Geschwindigkeit).
  - Sie kennen Masse und Geschwindigkeit.

```
masse=getMasse(ObjektID);
geschwindigkeit=getGeschwindigkeit(ObjectID);

impuls = masse * geschwindigkeit;
```

Einleitung : Wie funktioniert XML-RPC?

- das Programm muss wissen, *auf welchem System* die Prozedur verfügbar ist
- das Programm muss wissen, *in welchem Format* die Daten an das entfernte System übermittelt werden können oder müssen und wie die Daten wieder empfangen werden.
  - Dazu müssen die Daten verpackt,
  - übermittelt und
  - wieder entpackt werden.
  - Das Ganze geschieht zweimal, sofern Sie Daten zurück erhalten.

Einleitung : Wie funktioniert XML-RPC?

- RPC liefert dem Entwickler vereinfacht gesagt also die Möglichkeit ein API zu definieren, gegen das über das Netzwerk kommuniziert werden kann.
- Das API kann sehr einfach, oder aber sehr komplex sein, je nach Anwendung und beispielsweise der Netzwerkarchitektur.

Einleitung : Wie funktioniert XML-RPC?

- Aufgabe:
  - Definieren Sie ein einfaches API

- Elemente eines Netzwerk APIs
  (Ralph Davis: Network Programming)
  - **`Init()`**
  - **`Shutdown()`**

  - **`Call()`**
  - **`Listen()`**
  - **`Hangup()`**
  - **`Send()`**
  - **`Receive()`**

(Apache) XML-RPC - Einleitung

Einleitung : Wie funktioniert XML-RPC?

- XML und Web
  - XML als das formalere HTML
- Web Protokolle und Web Infrastruktur
  - HTTP
- Eine neue Art von Web
  - Nicht Seiten und Informationen sondern
  - Services und Methoden

Einleitung : Wie funktioniert XML-RPC?

- XML-RPC hebt ab:
  - XML-RPC ist ein ausgezeichnetes Hilfsmittel, um unterschiedliche Verbindungen zwischen Rechnern zu definieren.
  - Falls Sie unterschiedliche Rechnerumgebungen integrieren müssen,
    - Unterschiedliche Betriebsysteme
    - Unterschiedliche Programmiersprachen

Einleitung : Wie funktioniert XML-RPC?

- Das Minenfeld
  - Rechnerkommunikation über das Web
    - HTTP ist nicht sehr effizient
    - XML-RPC ist keine Web Seite
    - Sicherheitsfragen : Firewalls und all das

(Apache) XML-RPC – Das Protokoll

Das Protokoll - Wie können wir vorhandenen Dienste nutzen?

- das Client Programm führt einen Prozeduraufruf aus.
  Dies geschieht durch den XML-RPC Client. Dabei muss die Methode, Parameter und der Targetserver angegeben werden.
- der XML-RPC Client nimmt den Methodennamen und die Parameter und packt diese in XML ein.
  Der Client setzt dann einen http POST Request ab, sendet also die Anfrage an den XML-RPC Server.

- ein http Server auf der Target URL empfängt den http POST Request und leitet die Anfrage (in XML) an den XML-RPC Listener weiter.
- der XML Listener parst die XML Nachricht , um den Methodennamen und die Parameter zu bestimmen und dann die passende Methode aufzurufen.
- die Methode liefert eine Antwort an den XML-RPC Prozess. Dieser verpackt die Antwort in XML.
- der Web Server sendet die Antwort als XML an den POST Client zurück.
- jetzt ist wieder der Client dran: entpacken und synchrone Verarbeitung

XML-RPC- Das Protokoll

- Da HTTP als Protokoll eingesetzt wird, ist die Kommunikation sowohl *zustandslos*, als auch *synchron*
- *synchron*
  - Einem XML-RPC Request folgt immer eine XML-RPC Antwort: die Antwort ist synchronisiert mit der Anfrage, die beiden gehören zusammen. Dies geschieht im Rahmen eines einzigen http-Verbindung, der Anfrage und der Antwort darauf.

- *zustandslos*
  - http ist von Grund auf ein zustandsloses Protokoll.
  - Dies bedeutet, dass der Kontext einer Transaktion zwischen zwei POST Anfragen nicht gespeichert wird.
  - XML-RPC erbt diese Eigenschaft.
    - XML-RPC verfügt intern auch über keinerlei Mechanismen, um mehrere Anfragen zu verknüpfen.
    - Der Vorteil ist die Schlankheit des XML-RPC Systems.

(Apache) XML-RPC – Das Protokoll

XML-RPC Datentypen

- XML-PRC definiert bestimmte Basisdatentypen als mögliche Programmdaten.
  - Einfache Datentypen
    - Beispielsweise sind Integer und Strings
  - Zusammengesetzte Datentypen
    - Structures und Arrays

- Daten werden in XML-RPC in Anfragen oder als Rückgabe in einen XML Tag geklammert:
  - **`<value>...</value>`**

XML-RPC Datentypen

- Einfache Datentypen
  - **Integer**
  - XML-RPC kennt genau einen Datentyp für Integer Werter. Dieser repräsentiert ganzzahlige Werte als 32-Bit vorzeichenbehaftete ganze Zahlen im Bereich –2'147'483'648 (-2**31) bis 2'147'483'647 (2**31 -1). Die Beschreibung in XML sieht folgendermassen aus:
  - `<value><i4>`*n*`</i4></value>`
    oder
  - `<value><int>`*n*`</int></value>`
  - **Beispiel:**
  - **`<value><int>42</int></value>`**
    `<value><i4>-32764</i4></value>`
    `<value><int>0</int></value>`
    `<value><int>+52</int></value>`

XML-RPC Datentypen

- Einfache Datentypen
  - **Fliesskomma / Floating Point Zahlen**
  - Die XML-RPC Spezifikation legt den Bereich gültiger Fliesskommazahlen fest. Die Fliesskommanzahlen werden mit 64 Bits dargestellt. Der Datenbereich ist grob –10**323.3 bis 10**308.3. In XML-RPC wird eine Float Zahl folgendermassen dargestellt:

  - **Schema**:
    <value><double>f</double></value>

  - **Beispiele:**
  - **<value><double>2.0</double></value>**
    <value><double>-0.32653 </double></value>
    <value><double>+4.123</double>
    </value>

XML-RPC Datentypen

- Einfache Datentypen
  - **Boole'sche Werte**
  - XML-RPC definiert Boole'sche Werte als logische Grössen. Der Wertebereich ist 0 und 1. Der Wert 1 entspricht dem Wert true, der Wert 0 entspricht dem Wert false. Als Bezeichner wird <boolean> verwendet:
  - <value><boolean>b</boolean></value>

  - **Beispiel:**

```
<value><boolean>0</boolean></value>
<value><boolean>1</boolean></value>
```

XML-RPC Datentypen

- Einfache Datentypen
  - **String / Zeichenketten**
  - String Werte können in XML-RPC auf zwei Arten dargestellt werden:
  - als Standardwert von `<value>: <value>...</value>`

    Beispiel: `<value>Der Grosse Böse Wolf</value>`

  - explizit als String Variable mit dem `<string>` Tag

    Beispiel:
    `<value><string>Meine Tante in Kanada</string></value>`

XML-RPC Datentypen

- Einfache Datentypen
  - **Datum / Zeitangaben - dateTime**
  - Das Datum und die Zeitangaben werden in XML-RPC als `dateTime.iso8601` codiert.
    - Damit kann man absolute Zeitangaben machen. ISO 8601 gestattet viele unterschiedliche Formate. XML-RPC hat daraus ein bestimmtes Format, eine Vorlage ausgewählt:

  - `<dateTime.iso8601>CCYYMMDDTHH:MM:SS</dateTime.iso8601>`

  - **Beispiel:**
  - **`<value><dateTime.iso8601>19030223T00:30:00</dateTime.iso8601>...`**
    (23 Feb 1903 0 Uhr 30)

XML-RPC Datentypen

- Einfache Datentypen
  - **binary**
    - XML-RPC verwendet dafür base-64 Encoding, wie es auch beispielsweise im Internet email eingesetzt wird.
  - Als Tags verwendet man `<base64>...</base64>`.
  - **Beispiel:**
    - **`<value>`**
      `<base64>SGVsbG8sbG8sIFdvcmokIQ=</base64>`
      `</value>`

XML-RPC Datentypen

- Zusammengesetzte Datentypen
  - Arrays
  - struct

XML-RPC Datentypen

- Zusammengesetzte Datentypen
  - **Arrays** (auch mehrdimensional / verschachtelt)

  - **Beispiel:**

```
<value>
  <array>
       <data>
              <value>Irgend ein XML-RPC Wert</value>
                    ...
              <value>... und noch ein Wert</value>
       </data>
  </array>
 </value>
```

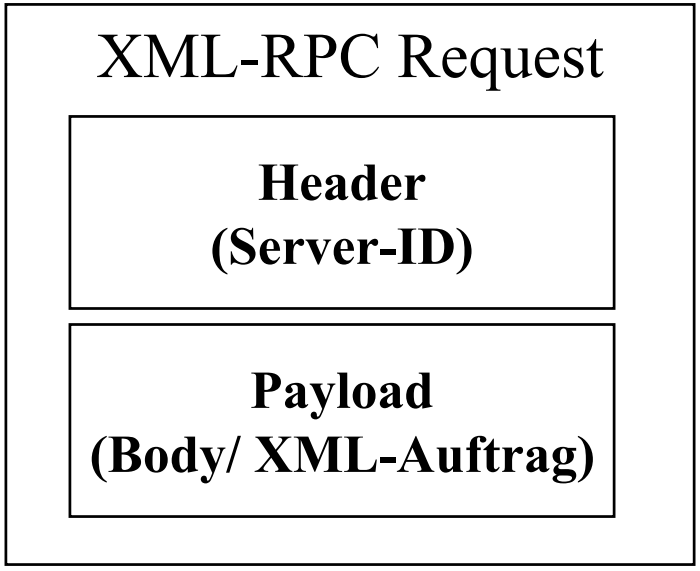
XML-RPC Datentypen

- Zusammengesetzte Datentypen
  - **struct**
  - **Beispiel:**

```
<value>
  <struct>
        <member>
              <name>Joller</name>
              <value>Jurist</value>
        </member>
        …
        <member>
              <name>Peterhans</name>
              <value>Chirurg</value>
        </member>
  </struct>
</value>
```

XML-RPC Request Format

- Grundsätzlich besteht ein Request aus zwei Teilen:
  - dem HTTP *Header*, mit dem der Server identifiziert wird.
  - XML *Payload*: dem XML Auftrag, der die Informationen enthält, die für den Methodenaufruf benötigt werden.



(Apache) XML-RPC – Das Protokoll

XML-RPC Request Format - Payload

- **<methodCall>...</methodCall>**

- **Beispiel**
  - **<?xml version=„1.0“?>**
    <methodCall>
    <methodName>print</methodName>
    <params><param>
    <value>Hallo Du da</value>
    </param></params>
    </methodCall>

XML-RPC Request Format – Payload

- Parameter
  - **<params>** enthält kein oder mehrere **<param>**
  - **<params>** *muss* vorhanden sein, auch ohne **<param>**

- Methoden
  - Alphanummerisch plus _, :, . / (**math.sin, math.multiply,...**)

Beispiel

```
<?xml version=„1.0“?>
<methodCall>
  <methodName>print</methodName>
  <params>
        <param>
              <value>Hallo Du da</value>
        </param>
  </params>
</methodCall>
```

XML-RPC Request Format – HTTP Headers

- POST
- Host
- Content-Type / Length
- User-Agent

- HTTP Header
  - **POST /meinservice.php http/1.0**
    User Agent: PHP XMLRPC 1.0
    Host: xmlrpc.meinpc.com
    Content-Type: text/xml
    Content-Length: 216

XML-RPC Request Format

```
POST /rpchandler http/1.0
User-Agent: meinXMLRPC/1.0
Host: xmlrpc.meinpc.org
Content-Type: text/xml
Content-Length: 165

<?xml version="1.0"?>
<methodCall>

<methodName>wieHeisstDerBuergermeisterVonWesel
   </methodName>
        <params>
                <param>
                        <value>nil</value>
                </param>
        </params>
    </methodCall>
```

XML-RPC Response Format

- Mögliche Antworten
  1. die korrekte Antwort auf den Methodenaufruf
  2. eine Fehlermeldung

XML-RPC Response Format

- Die Antwort wird in XML Tags eingekleidet:
  <methodResponse> ... </methodResponse>.

- Hier ein hypothetisches Beispiel für eine Abfrage der Hauptstadt von England:

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

XML-RPC Response Format

- Falls keine Parameter zurück gegeben werden, wie im Falle eines **`void`** Wertes einer Java Methode, dann haben Sie folgende Möglichkeiten:

  1. verwenden Sie eine *Boole'sche Variable*, um den Erfolg des Aufrufes zu dokumentieren.
  2. Geben Sie *irgend einen Wert* zurück, aber dokumentieren Sie genau, was dieser zu bedeuten hat oder was nicht.
  3. Verwenden Sie den **`nil`** *Wert*. Dieser ist allerdings erst vorgeschlagen, also noch nicht Teil der Spezifikation.

XML-RPC Response Format

```
<?xml version="1.0"?>
<methodResponse>
    <fault>
        <value>
        <struct>
            <member>
            <name>faultCode</name>
            <value><int>3</int></value>
            </member>
            <member>
             <name>faultString</name>
            <value>
              <string>No such method.</string>
            </value>
            </member>
            </struct>
        </value>
    </fault>
</methodResponse>
```

XML-RPC Response Format

- HTTP Header

```
HTTP/1.1 200 OK
Date: Wed 18 Dec 2001 11:21:23 GMT
Server: Apache/1.3.12 (Unix) Debian/GNU PHP/4.0.2
Connection: close
Content-Type: text/xml
Content-Length: 818
```

XML-RPC Response Format

- HTTP Header
  - Response Code: nur 200 OK
  - Server: SW Version, …
  - Content-Type : text/xml
  - Contet-Length

-
-
-
-
-
-
-
-
-

XML-RPC

Die Java Einbindung

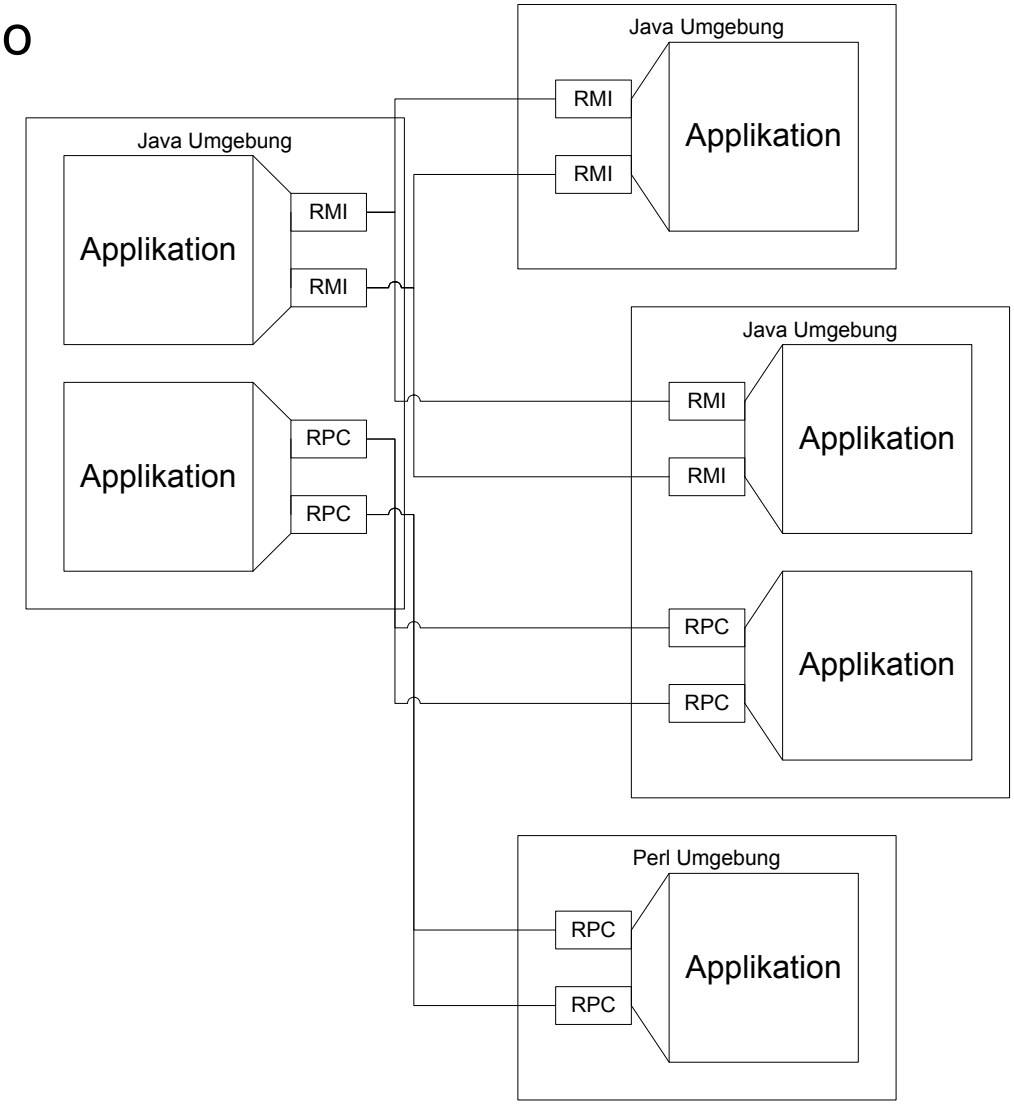
(Apache) XML-RPC – Client-Server Kommunikation

Einleitung

- Wie passt XML-RPC in die Kommunikationslandschaft von Java?
- Wie sieht ein typischer Java XML-RPC Server aus?
- Wie sieht ein typischer Java XML-RPC Client aus?
-
- Was versteht man unter einem Handler?
- Wie arbeitet ein Handler mit Client / Server zusammen?
-
- Wie werden die XML-RPC Datentypen in Java dargestellt?

(Apache) XML-RPC – Client-Server Kommunikation

Szenario



XML-RPC in Java

- Data Mapping

XML-RPC	Java bzw. vom Parser generiert
`int int4`	`int bzw Integer`
`Boolean`	`boolean bzw. Boolean`
`String`	`String`
`double`	`double bzw. Double`
`dateTime.iso8602`	`java.util.Date`
`struct`	`java.util.Hashtable`
`array`	`java.util.Vector`
`base64`	`byte[]`
`nil`	`null`

XML-RPC in Java

- Die Library
  - **`http://ws.apache.org/xmlrpc/`**
  - **`http://www.apache.org/dyn/closer.cgi/ws/xmlrpc/`**

- Die Installation
  - Unzip oder selber neu übersetzen:
  - Verzeichnisse bei neuem Aufbau der Archive (aus dem Quellcode)
    - **`.`** Ant **`build.xml`** und **`build.properties`**
    - **`bin/`** temporäres Verzeichnis (nur für build)
    - lib/ Verzeichnis mit den jar Files
    - examples/ Beispiele und Anleitungen
    - src/ Java Quelldateien.
    - xdocs/ XmlRpc Dokumentation in DocBook Format.
    - docs/ Dokumentation in HTML Format.

XML-RPC in Java : diesen Schritt können Sie überspringen!

- **Ant Tasks**
  - `C:\xmlrpc-1.1\source\xmlrpc-1.1>ant -projecthelp`
    `Buildfile: build.xml`
    `Main targets:`

    ```
    compile-tests  Compiles testing source code
    docs           Generates the HTML documentation
                   (including JavaDoc)
    install-jar    Installs .jar file in ${lib.repo}
    jar            Builds the two JAR files from source
    test           Runs unit and run-time tests
    ```
- **Beispiel**
  - **aus build.xml:**
  - ```
    <target name="install-jar" depends="jar"
     description="Installs .jar file in ${lib.repo}">
        <copy todir="${lib.repo}" filtering="no">
          <fileset dir="${build.dir}">
            <include name="${final.name}.jar"/>
            <include name="${final.name}-applet.jar"/>
          </fileset>
        </copy>
      </target>
    ```
    ```

XML-RPC in Java

- Anforderungen
  - falls Sie einen speziellen Parser einsetzen wollen, geben Sie diesen einfach im CLASSPATH an!
  - XML-RPC verwendet standardmässig einen ML SAX Parser; Sie können getrost den mit Java mitgelieferten SAX Parser verwenden indem Sie J2SDK1.4+ verwenden oder einen Parser direkt angeben.
  - Sie können auch im Programm einen eigenen Parser festlegen
    - **`XmlRpc.setDriver("ihrEigenerSAXDriver");`**
  - oder als System Property
    - **`java -Dsax.driver=ihrEigenerSAXDriver xmlrpcAnwendung`**

XML-RPC in Java

- Bau eines XML-RPC Servers
  - Warum zuerst ein Server?
    - dann haben wir wenigstens einen Service zur Verfügung!
    - der Bau eines Servers ist komplexer als jener eines Clients
      - der Service muss publiziert werden!
      - der Service muss mit den Anfragen zurecht kommen!
  - XML-RPC enthält einen fertigen Web Server: **`WebServer`**
    - dieser ist speziell für XML-RPC ausgerichtet

XML-RPC in Java

- Testen der Installation
  - ```
    @echo off
    @echo Testen der XML-RPC Installation
    IF NOT DEFINED JAVA_HOME GOTO NOJAVAHOME
    SET CLASSPATH=c:\xmlrpc-1.1\xmlrpc-1.1.jar;C:\xmlrpc-1.1\xmlrpc-1.1-
    applet.jar;
    Rem ------------------------------------------------------------------
    Rem Web Server als neues Fenster
    Rem ------------------------------------------------------------------
    start/separate TestenDesServers.bat
    Rem ------------------------------------------------------------------
    Rem Web Client
    Rem ------------------------------------------------------------------
    @echo Client und Server werden in separaten Fenstern gestartet!
    %JAVA_HOME%\bin\java org.apache.xmlrpc.XmlRpcClient
    http://localhost:8080 echo test 123
    pause
    GOTO ENDE
    :NOJAVAHOME
    @echo Sie muessen JAVA_HOME definieren
    @echo Beispiel: c:\J2SDK1.4.1_02
    :ENDE
    ```
    ```

XML-RPC in Java

- Testen der Installation
  - Ausgaben
    - Serverseitig
      - **Testen der XML-RPC Installation**
        Client und Server werden in separaten Fenstern gestartet!
        Usage: java org.apache.xmlrpc.WebServer <port>
        started web server on port 8080

    - Clientseitig
      - **Testen der XML-RPC Installation**
        Client und Server werden in separaten Fenstern gestartet!
        [test, 123]
        Press any key to continue . . .

(Apache) XML-RPC – Praxis

XML-RPC in Java

- Aufbau eigener Client und Server
  - Clientseitig haben Sie die Möglichkeit
    - synchron
      oder
    - asynchron zu kommunizieren
  - Serverseitig haben Sie die Möglichkeit
    - mit dem mitgelieferten WebServer
      oder
    - über Servlets XML-RPC einzusetzen.

XML-RPC in Java – mit der **WebServer** Klasse

```
public static void main(String[] args) {
   try {
       // Aufruf <http://localhost:9090/>.
       WebServer server = new WebServer(9090);
       // Dienst addHandler(Dienstname, Dienstobjekt)
       // Dienst = Methode
       server.addHandler("helloWorld", new
                                  HelloWorldServer());
   } catch (Exception exception) {
       System.err.println("[HelloWorldServer]" +

       exception.toString());
   }
}
```

XML-RPC in Java – Der Client

- Bau eines XML-RPC Clients
  - Grundschema eines Clients
    1. Definition der URL, bei der der Dienst angeboten wird
       - `XmlRpcClient serverAnbindung = new XmlRpcClient(server_url);`
    2. Definition der Parameterliste
       - `Vector params = new Vector();`
    3. Aufruf der Remote Methode
       - `String result =(String) serverAnbindung.execute("helloWorld.wieGehts", params);`
    4. Verarbeiten des Ergebnisses

XML-RPC in Java – der Client

```
ublic static void main (String [] args) {
  try {
  // Server Objekt
  XmlRpcClient serverAnbindung = new
                                    XmlRpcClient(server_url);
  // Parameterliste
  Vector params = new Vector();
  // RPC
  String result =
  (String)serverAnbindung.execute("helloWorld.wieGehts",params);
  // verarbeiten der Ergebnisse
  } catch (XmlRpcException exception) {
     System.err.println("[HelloWorldClient]XML-RPC Fault #" +
                Integer.toString(exception.code) + ": " +
                exception.toString());
      exception.printStackTrace();
  } catch (Exception exception) {
      exception.printStackTrace();
  }
```

XML-RPC in Java – der Client

- zum Debuggen steht Ihnen folgende Möglichkeit zur Verfügung
  - `XmlRpc.setDebug(true);`
- wie beispielsweise in
  - ```
    public static void main(String[] args) {
        HandlerClient handlerClient1 = new HandlerClient();
        try {
          XmlRpc.setDebug(true);
          // Server Objekt
          XmlRpcClient server = new
                          XmlRpcClient(server_url);
            ...
    ```
    ```

XML-RPC in Java – den Client debuggen

- Ausgabe
  -
```
POST /RPC2 HTTP/1.1
Content-Length: 219
Content-Type: text/xml
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.4.2_01
Host: localhost:8099
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

Beginning parsing XML input stream
startElement: methodCall
startElement: methodName
endElement: methodName
startElement: params
startElement: param
startElement: value
startElement: int
endElement: int
endElement: value  ...
endElement: params
endElement: methodCall
Spent 30 millis parsing
method name: HandlerTest.XMLRPCHandler
inparams: [5, 3]
```

XML-RPC in Java – *XML-RPC Handler*

- Serverseitig
  - bevor eine Methode remote genutzt werden kann, muss sie serverseitig registriert werden.
    - **`addHandler()`** registriert einen Handler
    - **`removeHandler()`** entfernt eine Methode
  - falls eine Methode nur erlaubte Datentypen verwendet, kann sie "automatisch" registriert werden:
    - zuerst muss ein HTTP Server definiert werden, mit dem die HTTP Transaktionen behandelt werden.
    - dann kann dieser Server die Methode registrieren
      - ```
        WebServer server = new WebServer(8182);
        server.addHandler("NameDerRPCMethode",
                          new KlasseMitDerMethode());
        ```
    - der Server analysiert die Klasse (**`KlasseMitDerMethode`**) und deren Methoden und leitet Anfragen mit passender Signatur an diese weiter.
        ```

XML-RPC in Java – *XML-RPC Handler automatische Registrierung*

```
• public class HandlerServer {
    public HandlerServer() {
      System.out.println("[HandlerServer]Konstruktor");
    }
    public static void main(String[] args) {
      try {
        XmlRpc.setDebug(true);
        // Aufruf <http://localhost:8080/RPC2>.
        System.out.println("[Server]Instanziern des Web Servers:
                                              localhost:8099");
        WebServer server = new WebServer(8099);
        // Instanz dieses Servers bilden und auf Anfragen warten
        System.out.println("[Server]Instanziern des XML-RPC
                                                    Servers");
        server.addHandler("HandlerTest", new XMLRPCHandler());
      } catch (Exception exception) {
        System.err.println("[Server]Fehler " +
                                    exception.toString());
      }
    }
  }
  // Client : server.execute("HandlerTest.XMLRPCHandler", params);
```

XML-RPC in Java – XML-RPC Handler

- Serverseitige, explizite Registrierung eines Handlers
  1. zuerst muss die Handlerklasse definiert werden
     - diese muss die das Interface **`XmlRpcHandler`** implementieren
     - diese implementiert die Methode
       `public Object execute(String methodenName, Vector parameter)`
     - die Methode execute() ruft ihrerseits die Servicemethoden auf

```
public Object execute(String methodenName, Vector parameter) {
  ...
  return new Integer(summe(p1,p2));
}
```

  2. dann muss in der Serverklasse dieser Handler registriert werden
     - `server.addHandler("eigenerHandler", new XMLRPCHandler());`
  3. schliesslich erfolgt der Methodenaufruf im Client
     - `Object result=Server.execute("eigenerHandler.summe",params);`
- **Beispiel:**
  - ClientFürHandlerklasse, XMLRPCHandler, ServerMitHandlerklasse

XML-RPC in Java – XML-RPC Handler

- Einfache Definition und Einsatz eines Handlers (ohne Implementierung des **`XmlRpcHandler`** Interfaces)

1. zuerst muss die Handlerklasse definiert werden
   - diese implementiert die Servicemethoden (gibt ein **`Object`**)
     - ```
       public Integer summe(double p1, double p2) {
        ...
        return new Integer(p1+p2);
       }
       ```
2. dann muss in der Serverklasse dieser Handler registriert werden
   - `server.addHandler("eigenerHandler", new XMLRPCHandler());`
3. schliesslich erfolgt der Methodenaufruf im Client
   - `Object result=Server.execute("eigenerHandler.summe",params);`

- **Beispiel:**
  - SummenHandler, SummenServer, SummenClient
       ```

XML-RPC in Java – XML-RPC Handler

- Einfache Definition und Einsatz eines Handlers (ohne Implementierung des **`XmlRpcHandler`** Interfaces)
  - Erweiterung
    - anstelle von Summen sollen auch Differenzen und ... als Methoden möglich sein
    - Vorgehen:
      - wir übergeben eine Hashtabelle mit den Methoden
      - in der Handler Klasse analysieren wir den Inhalt der Hashtabelle
      - je nach Parameter rufen wir die eine oder andere Methode auf
  - Beispiel
    - Flaeche, FlaechenClient, FlaechenServer

XML-RPC in Java – XML-RPC Handler

- Einfache Definition und Einsatz eines Handlers (ohne Implementierung des **`XmlRpcHandler`** Interfaces)
  - FlaechenClient
  - ```
    Double radius = new Double(args[0]);
    Hashtable procedureHash = new Hashtable();
    procedureHash.put("typus", "kreis");
    procedureHash.put("radius", radius);
    Vector params = new Vector();
    params.addElement(procedureHash);
    // Methodenaufruf (Prozeduraufruf)
    result = client.execute("flaeche.anyFlaeche",params);
    ```
  - FlaechenServer
  - ```
    server.addHandler("flaeche",new Flaeche());
    ```

XML-RPC in Java – XML-RPC Handler

- Einfache Definition und Einsatz eines Handlers (ohne Implementierung des **`XmlRpcHandler`** Interfaces)
  - FlaechenHandler
  -
```
public class Flaeche {
 public Double rechteckFlaeche(double laenge, double breite) {
    return new Double(laenge*breite);}
 public Double kreisFlaeche(double radius) {
    return new Double(radius*radius*Math.PI);}
  // Erweiterung: beliebige Auswahl der geom. Figur
  public Double anyFlaeche(Hashtable argumente){
    Double wert; wert = new Double(0);
    String procedureTypus = (String)argumente.get("typus");
    if (procedureTypus.equals("kreis")) {
      Double radius = (Double)(argumente.get("radius"));
      wert = kreisFlaeche(radius.doubleValue());
    }if (procedureTypus.equals("rechteck")) {
      Double laenge = (Double)(argumente.get("laenge"));
      Double breite = (Double)(argumente.get("breite"));
      wert =
rechteckFlaeche(laenge.doubleValue(),breite.doubleValue());
    }
    return wert;
}}
```

XML-RPC in Java – XML-RPC Handler

- Zwischenspeichern auf dem Server / im Handler
  - falls Sie Objekte, Daten, Informationen für weitere Aufrufe benötigen, können Sie auch
    - im Handler eine statische oder Objekt- Variable definieren
    - diese im Handler mit einer set-Methode setzen
    - diese in einem späteren Aufruf mit einer get-Methode wieder abfragen und weiterverwenden.

- Beispiel
  - GetSetHandler (mit Objektvariable : die Klasse wird beim Registrieren im Server instanziert, genau einmal)
  - GetSetServer
  - GetSetClient

XML-RPC in Java – XML-RPC und Servlets

- Sie können XML-RPC auch zusammen mit Servlets einsetzen.
- Grundsätzlicher Programmaufbau
  1. bilden einer Instanz von **`XmlRpcServer`**
  2. registrieren des Handlers beim **`xmlserver`**-Objekt
  3. Verbindung mit der **`doPost()`** Methode
- Beispiel

```
public class XmlRpcServlet extends HttpServlet {
     private XmlRpcServer xmlrpc;
     public void init(ServletConfig config) throws ServletException {
     if("true".equalsIgnoreCase(config.getInitParameter("debug")))
             XmlRpc.setDebug(true);
     xmlrpc = new XmlRpcServer();
     try {
             xmlrpc.addHandler("beispiel", new SummenHandler());
     } catch (Exception x) {
             x.printStackTrace(System.err);
     }
}
```

XML-RPC in Java – XML-RPC und Servlets

- Beispiel für die Anbindung an **`doPost()`**

```
public void doPost(HttpServletRequest req, HttpServletResponse res) throws
                                ServletException, IOException {
    byte[] result = xmlrpc.execute(req.getInputStream());
    res.setContentType("text/xml");
    res.setContentLength(result.length);
    OutputStream output = res.getOutputStream();
    output.write(result);
    output.flush();
 }
}
```

XML-RPC in Java – XML-RPC asynchron nutzen

- Die Änderung der remote Methode bei einem asynchronen Aufruf ist denkbar einfach
  - anstelle des Aufrufes **`server.execute("name", parameter);`** wird
  - **`server.executeAsync("name", parameter, callbackThread);`**

- Falls Sie anstelle des **`callbackThreads`** einfach **`null`** angeben, werden die Ergebnisse nicht mehr lesbar...

- Die Callback-Klasse muss das Interface **`org.apache.xmlrpc.AsyncCallback`** implementiert werden.
  - dieses besitzt zwei Methoden
    - **`public void handleResult(Object result, URL url, String method);`**
    - **`public void handleError(Exception exception, URL url, String method);`**

XML-RPC in Java – Peer-to-Peer

- Das letzte Beispiel zeigte, wie ein (oder mehrere) Client einen laufenden Server kontrollieren können.
- von diesem Beispiel zu einem P2P (Peer-to-Peer) System ist der Unterschied sehr klein!
- Die Definition und Implementierung von Beispielen zu P2P überlasse ich Ihnen, und hoffe Sie nutzen Ihre Kreativität voll aus!

XML-RPC

- Was ich nicht gezeigt habe
  - XML-RPC gibt's für fast alle Sprachen
    - Perl
    - PHP
    - JavaScript ?
    - ...
    - was auch immer

-
-
-
-
-
-
-
-
-

XML-RPC

und die Web Service Landschaft

Die Web Service Vision

- Idee
  - basierend auf vorhandener Technologie einen Service Web aufbauen
- Probleme
  - Security
  - Verrechnung (Micro-Payments)
- Ansätze
  - WSDL + SOAP + UDDI

  auf der Basis von
  - XML, HTTP / SMTP (und TCP/IP)
- Positionierung von XML-RPC
  - leichtgewichtige Lösung für einfachste Anwendungsfälle

Software Entwicklung mit XML-RPC

- Debugging
  - Client, Server und Handler müssen zusammen getestet werden
    - deswegen fange ich in der Regel mit einem autonomen Handler an, den ich separat testen kann
    - dann folgt der Server, obschon dies schwieriger ist als die Entwicklung des Client (dieses Vorgehen entspricht teilweise RUP: risikoreiche Programme werden zuerst erstellt)
    - und schliesslich der Client
      darauf kann man sich freuen!
- Netzwerkprobleme
  - Latenzzeit
    - das Netzwerk muss funktionsfähig sein (TCP/IP)
      - das lässt sich einfach überprüfen (Web)
- Authentifizierung und Autorisierung
  - haben wir weggelassen, weil dieser Teil analog zu anderen Java Programmen geschehen kann (JAAS)

XML-RPC – wie geht's weiter

- SOAP Simple Object Access Protocol
  - beschreiben wir an an derer Stelle
  - ist eine Weiterentwicklung von XML-RPC
    - zuverlässiger
    - professioneller (Datentypen, Handshaking, ..)
    - Standard : W3C
  - ist ein XML Protokoll
    - Arbeitsgruppe XML Protokoll Architekturen bei W3C

XML-RPC – wie geht's weiter

- UDDI Universal Description, Discovery and Integration of Business for the Web
  - Verzeichnissystem für Web Services
    - Yellow Pages
    - White Pages
    - Green Pages
  - stammt von IBM
  - ist kein Standard
  - enthält chaotische Einträge
  - ist wohl noch nicht das wahre...
  - ist inhouse aber bereits voll nutzbar!

XML-RPC – wie geht's weiter

- WSDL Web Services Description Language
  - XML basiert
  - W3C unterstützt
  - beschreibt Dienste oder Endpunkte eines Service Webs
  - beschreibt die Bindung an XML Protokolle am Beispiel SOAP

  - Standardreferenz zum Spielen
    - XMETHODS.COM

    - dort kann man WSDL's aber auch ganze Clients für einfache Beispieldienste ansehen und herunterladen.

-
-
-
-
-
-
-
-
-

XML-RPC

XML Know How

Was sollten Sie über XML wissen?

- XML definiert nur eine Basissyntax
  - Anfangstag Inhalt Endtag
  - nur saubere Verschachtelungen <a>.. geht nicht
  - jedes Dokument besitzt genau eine Wurzel
  - Namen der Tags gehorchen bestimmten Regeln

- Unterschied zu HTML
  - keine vordefinierten Tags
  - keine verschobenen Verschachtelungen (siehe oben)
  - keine Formatierungen (in der Basissyntax; XSL bietet Styles)
  - Gross- und Kleinschreibung ist bei XML wichtig

  - es wird zwischen "wohlgeformt" und "gültig" unterschieden
    - wohlgeformt : wie oben
    - gültig : gemäss einer DTD (Document Type Definition) oder einem XML Schema

Ein Beispiel

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Autor>
  <Person Pnr="1112">
    <Name>Huber</Name>
    <Vorname>Hubert</Vorname>
  </Person>
  <Person Pnr="1314">
    <Name>von Goisern</Name>
    <Vorname>Hubert</Vorname>
  </Person>
  <Person Pnr="000" />
- <!-- Platzhalter  -->
</Autor>
```

Elemente und Attribute

- Elemente und Attribute sind die Basiseinheiten von XML
  - Element
    - <StartTag> Elementinhalt </EndTag>
    - der Name des Tags ist frei wählbar, sofern die Zeichenkette irgendwie Sinn macht
  - Attribut
    - <StartTag Attribut='Wert' Attr="Werte">Elementinhalt</...>
    - Attributwerte müssen in einfachen oder doppelten Anführungszeichen stehen.
- Syntax
  - Namen sind casesensitiv (Gross- und Kleinschreibung)
  - Namen, die mit "xml" anfangen, sind für reserviert
  - Namen müssen mit einem Buchstaben oder _ anfangen, kann beliebig viele Zeichen und Ziffern und "." enthalten.

Wohlgeformt

- Ein XML-Dokument, welches den W3-Regeln für XML 1.0 folgt, wird als wohlgeformt bezeichnet.
  - Die Mindestanforderung ist ein Root-Element,
  - symmetrische Tags,
  - identische Tagnamen in Start- und Ende-Tag
  - und Attribute in Anführungszeichen, falls vorhanden.
  - Eine XML-Deklaration oder Processing Instruction muss nicht vorhanden sein. In der Regel ist das aber der Fall.

Kommentare und Entitäten

- <!– ist ein Kommentar //→

- Entity Referenzen
  - wenn Sie <, >, ... darstellen müssen, müssen Sie Entity Referenzen verwenden
  - anstelle von < verwenden Sie <
  - & &
  - > >
  - ' &aps;
  - " "

- Sie können auch Zeichen als UTF-8 darstellen
  - 1 0
  - A A

Unicode Codierschemata

- Übersicht

Anweisung	Bedeutung
encoding="UTF-8"	internationaler Zeichensatz auf Basis der ISO/IEC-10646-Norm mit 8 Bit Zeichenbreite
encoding="UTF-16"	internationaler Zeichensatz auf Basis der ISO/IEC-10646-Norm mit 16 Bit Zeichenbreite
encoding="ISO-8859-1"	ISO-Zeichensatz für westeuropäische Sprachen ISO-8859-1
encoding="ISO-8859-2"	ISO-Zeichensatz für osteuropäische Sprachen ISO-8859-2
encoding="ISO-8859-3"	ISO-Zeichensatz für südeuropäische Sprachen ISO-8859-3
encoding="ISO-8859-4"	ISO-Zeichensatz für nordeuropäische Sprachen ISO-8859-4
encoding="ISO-8859-5"	ISO-Zeichensatz für kyrillische Sprachen ISO-8859-5
encoding="ISO-8859-6"	ISO-Zeichensatz für arabische Sprachen ISO-8859-6
encoding="ISO-8859-7"	ISO-Zeichensatz für griechische Sprache ISO-8859-7
encoding="ISO-8859-8"	ISO-Zeichensatz für hebräische Sprache ISO-8859-8
encoding="ISO-8859-9"	ISO-Zeichensatz für türkische Sprache ISO-8859-9
encoding="ISO-8859-10"	ISO-Zeichensatz für nordische Sprache ISO-8859-10

Gültige Dokumente

Definition

Ein XML-Dokument das sowohl wohlgeformt (valid) ist als auch in seiner Struktur der ihm zugeordneten DTD oder dem Schema entspricht.

Die Gültigkeit wird mit einem Parser geprüft. Nicht alle Parser können auf Gültigkeit prüfen.

DTD – Document Type definition

- Typische DTD's
  - <!ELEMENT e (#PCDATA)> Element e darf beliebige Zeichen enthalten
  - <!ELEMENT e EMPTY> Element e enthält keinen Inhalt (aber Attr.)
  - <!ELEMENT e ANY> Element e kann Zeichen oder andere Daten enthalten
  - <!ELEMENT e (a,b*,c)> e besteht aus a und mehreren b's und c
  - <!ELEMENT e (b|c)> e besteht aus b oder c

- Attribute
  - attname CDATA #REQUIRED Attribut muss immer vorhanden sein
  - attname CDATA #IMPLIED Attribut kann, muss nicht vorhanden sein
  - attname (a|b|c) Attribut kann nur einen der Werte a,b,c annehmen
  - attname CDATA #FIXED "CHF" Attribut hat festen Wert "CHF"

DTD – Document Type definition

- DTD zur Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSPY v5 U (http://www.xmlspy.com)-->
<!ELEMENT Autor (Person+)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Person (Name?, Vorname?)>
<!ATTLIST Person
   Pnr (000 | 1112 | 1314) #REQUIRED
>
<!ELEMENT Vorname (#PCDATA)>
```

DTD – Document Type Definition, eingebunden in XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Autor SYSTEM
   "C:\eclipse\workspace\ApacheXML_RPC\src\xmlrpcstandaloneserver\
   Adresse.dtd">
<Autor>
   <Person Pnr="1112">
        <Name>Huber</Name>
        <Vorname>Hubert</Vorname>
   </Person>
   <Person Pnr="1314">
        <Name>von Goisern</Name>
        <Vorname>Hubert</Vorname>
   </Person>
   <Person Pnr="000"/>
   <!-- Platzhalter -->
</Autor>
```

XML Schema zur Adresse

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v5 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="Autor">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Person" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Name" type="xs:string"/>
	<xs:element name="Person">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Name" minOccurs="0"/>
				<xs:element ref="Vorname" minOccurs="0"/>
			</xs:sequence>
			<xs:attribute name="Pnr" use="required">
				<xs:simpleType>
					<xs:restriction base="xs:NMTOKEN">
						<xs:enumeration value="000"/>
						<xs:enumeration value="1112"/>
						<xs:enumeration value="1314"/>
					</xs:restriction>
				</xs:simpleType>
			</xs:attribute>
		</xs:complexType>
	</xs:element>
	<xs:element name="Vorname" type="xs:string"/>
</xs:schema>
```

-
-
-
-
-
-
-
-
-

XML-RPC

HTTP Basiswissen

Was Sie über HTTP wissen sollten

- Damit Sie HTTP nutzen können, müssen Sie zuerst TCP/IP in Gang setzen
- TCP/IP
  - spezielle IP Adresse
    - 127.0.0.1 : localhost
  - Sie können, wenn Sie wirklich Anhnung haben, pro PC mehrere IP Adressen haben
  - In der Regel werden Sie dynamisch (mit DHCP) eine IP Adresse von Ihrem Provider erhalten.

Was Sie über HTTP wissen sollten

- HTTP Befehle sind recht einfach
  - GET, PUT, POST, ...der Befehlssatz ist sehr klein!

- GET-Header
  - ```
    GET /html/index.html
    User-Agent: Mozilla/5.2 [en] [WinNT; I]
    Host: www.webcentral.org
    Accept: */*
    ```
  - Host erlaubt die Eingabe eines Hosts beim Hosten von mehreren Domains.
- GET-Response
  - ```
    HTTP/1.0 200 OK
    Date: Mon, 29-Sept-03 12:27:21 GMT
    MIME-version: 1.0
    Content-type: text/html

    <html>
      <head>...
    </html>
    ```

Was Sie über HTTP wissen sollten

- POST-Befehl
  - um den POST Befehl testen zu können, kann man einen einfachen Server bauen und an diesen einen POST Request senden.
  -
```
public class HTTPTester {
  public HTTPTester() {
  }
  public static void main(String[] args) {
    HTTPTester HTTPTester1 = new HTTPTester();
    try {
      ServerSocket server = new ServerSocket(1234);
      Socket sock = server.accept();
      InputStream is = sock.getInputStream();
      while(true) {
        int i = is.read();
        if (i==-1) break;
        System.out.write(i);
      }
      System.out.println("\nVerbindung wird geschlossen");
    } catch (IOException ioe) {
    }
  }
}
```

Was Sie über HTTP wissen sollten

- POST-Header Beispielausgabe auf dem Server
  -
```
POST / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/msword, application/vnd.ms-
excel, application/vnd.ms-powerpoint, */*
Accept-Language: de-ch
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.0; .NET CLR 1.0.3705; .NET CLR 1.1.4322)
Host: localhost:1234
Content-Length: 63
Connection: Keep-Alive
Cache-Control: no-cache
```

Was Sie über HTTP wissen sollten

- POST Variante
  - Angabe des Encodings
    <form method="POST" action="http://localhost:1234"
    enctype="multipart/form-data">
- POST-Header Beispielausgabe auf dem Server
  - ```
    POST / HTTP/1.1
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
    application/msword, application/vnd.ms-excel,
    application/vnd.ms-powerpoint, */*
    Accept-Language: de-ch
    Content-Type: multipart/form-data; boundary=-----------------
    ---------7d3337251104a0
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
    .NET CLR 1.0.3705; .NET CLR 1.1.4322)
    Host: localhost:1234
    Content-Length: 463
    Connection: Keep-Alive
    Cache-Control: no-cache

    -----------------------------7d3337251104a0
    ```
    ```

Was Sie über HTTP wissen sollten

- POST Variante
  - Angabe des Encodings
    <form method="POST" action="http://localhost:1234"
    enctype="multipart/form-data">
- POST-Header Beispielausgabe auf dem Server
  –

```
-----------------------------7d3337251104a0
Content-Disposition: form-data; name="given"

Josef
-----------------------------7d3337251104a0
Content-Disposition: form-data; name="family"

Joller
-----------------------------7d3337251104a0
Content-Disposition: form-data; name="other"

Uster
-----------------------------7d3337251104a0
Content-Disposition: form-data; name="POST Methode"

Submit Query
-----------------------------7d3337251104a0--
Ende!
```

Was Sie über HTTP wissen sollten

- POST Variante
  - Angabe einer Datei

```
<form method="POST" action="http://localhost:1234"
          enctype="multipart/form-data">
          <p>
          Datei: <br><input name="file" type="file">
          <input name="POST Methode" type="submit">
  </form>
```

- POST-Header Beispielausgabe auf dem Server

```
POST / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/msword, application/vnd.ms-excel,
application/vnd.ms-powerpoint, */*
Accept-Language: de-ch
Content-Type: multipart/form-data; boundary=----------------
---------7d31a1c270322
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
.NET CLR 1.0.3705; .NET CLR 1.1.4322)
Host: localhost:1234
Content-Length: 12464
Connection: Keep-Alive
Cache-Control: no-cache
```

Was Sie über HTTP wissen sollten

- POST-Header Beispielausgabe auf dem Server

```
– -----------------------------7d31a1c270322
  Content-Disposition: form-data; name="file";
  filename="C:\Downloads\WHATSNEW.TXT"
  Content-Type: text/plain

  new features in this release of Xenu's Link Sleuth:
  http://home.snafu.de/tilman/xenulink.html
  ...
  and popup menu
  - launch report all the time

  -----------------------------7d31a1c270322
  Content-Disposition: form-data; name="POST Methode"

  Submit Query
  -----------------------------7d31a1c270322–
  Ende!
```

XML-RPC in Java Referenzen

- http://www.xmlrpc.com/spec

- http://ws.apache.org/xmlrpc/