Verteilte Systeme - Zusammenfassung

Jan Fässler & Chregi Glatthard

4. Semester (FS 2013)

If you use this documentation for a exam, you should offer a beer to the authors!

Inhaltsverzeichnis

1 Networking

1.1 InetAddress

Static factory methods

- getByName(String name)
- getByAddress (4/16 bytes)
- getAllByName(String host)
- getLocalHost()

Instance methods

- byte[] getAddress()
- String getHostAddress()
- String getHostName()
- String getCanonicalHostName()
- boolean isReachable(int timeout)
- boolean isMulticastAddress()

1.2 Network Interfaces

Listing 1: Network Interfaces and its addresses

```java
public static void main(String[] args) throws SocketException {
  Enumeration<NetworkInterface> interfaces = NetworkInterface.getNetworkInterfaces();
  while(interfaces.hasMoreElements()){
    NetworkInterface intf = interfaces.nextElement();
    Syst  em.out.print(intf.getName());
    System.out.println(" ["+intf.getDisplayName()+"]");
    Enumeration<InetAddress> adr = intf.getInetAddresses();
    while(adr.hasMoreElements()){
      System.out.println("\t" + adr.nextElement());
    }
    byte[] hardwareAddress = intf.getHardwareAddress();
  }
}
```

1.3 Sockets

Abstraction through which an application may send and receive data through the network. A Socket is identified by Hostname/IP and port number.

Stream Sockets

- Use TCP as end-to-end protocol
- Provide a reliable byte-stream
- Connection oriented: Socket represents one end of a TCP connection

Datagaram Sockets

- Use UDP as protocol
- Not connection oriented, not reliable

1.3.1 Controlling Socket Behaviors

Blocking & Timeouts

ServerSocket.accept / InputStream.read
read or accept call will not block for more than a fixed number of msec otherwise, InterruptedIOException is thrown (get/setSoTimeout(int timeout))

Socket constructor
Uses a system-defined timeout, cannot be changed by Java API (Solution: use connect)

OutputStream.write
Cannot be interrupted / caused to time-out by Java API

Keep-Alive

- TCP provides a keep-alive mechanism
- Probe messages are sent after a certain time
- Application only sees keep-alive working if the probes fail!
- Per default keep-alive is disabled
- Default timeout: 2h (7200 secs)

Send / Receive Buffer Size

- When a Socket is created, the OS must allocate buffers to hold incoming & outgoing data
- Receive buffer size may also be specified on server socket (for accepted sockets which immediately receive data)

No Delay

- TCP tries to avoid sending small packets
- Buffers data until it has more to send, combines small packets with larger ones
- Necessary if application has to be efficient
- Default: false

1.3.2 Closing Connections

close()

- Once an endpoint (client or server) closes the socket, it can no longer send or receive data
- Close can only be used to signal the other end that the caller is completely finished communicating

shutdownOutput()

- Closes output-stream, no more data can be may be written (IOException)
- All data written before shutdownOutput can be read by receiver

shutdownInput()

- Closes the input stream
- Any undelivered data is (silently) discarded, read operations will return -1

s.close() / s.shutdownOutput()

- Data may still be waiting to be delivered to the other side
- By default, socket tries to deliver remaining data, but if socket crashes, data may be lost without notification to sender (as close returns immediately)

1.3.3 User Datagram Protocol

- UDP allows to address applications over ports
- UDP adds another layer of addressing (ports) to that of IP
- UDP detects some form of data corruption that may occur in transit and discards corrupted messages
- UDP retains message boundaries

2 Internet

2.1 Protocol

GET

- Access of content from the server
- Idempotent, i.e. the side effects of N¿0 identical requests is the same as for a single request (f(f(x)) = f(x))

POST
Comparable to GET but Method must not necessarily be idempotent and Request data is transferred in the body of the request

HEAD

- Identical to GET, except that the server must not return the body
- Can be used to request meta information (headers) about the resource

OPTIONS (1.1)
Returns information about the communication options available on the specified resource (or on the server in general if request URI=*)

PUT (1.1)
Stores a web page on the server (rarely implemented)

DELETE (1.1)
Removes a web resource from the servver (rarely implemented)

TRACE (1.1)
Returns the request as it was accepted by server ($\Rightarrow$ debugging)

CONNECT (1.1)
Implemented by Proxy Server capable to provide an SSL tunnel

2.1.1 Response Codes

200-299: Success

200	OK
201	Created
202	Accepted

300-399: Redirections

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other (e.g. after POST)
304	Not Modified
305	Use Proxy
307	Temporary Redirect

400-499: Client Error

400	Bad Request
401	Unauthorized

402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
407	Proxy Authentication Required
408	Request Time-out
411	Length Required
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type

500-599: Server Error

500	Internal Server Error
501	Not Implemented
503	Service Unavailable
505	HTTP Version not supported

2.2 Request Headers

Host server host

Referer host from which the request is initiated

Accept data types supported by the client

Accept-Language language supported by client

Accept-Encoding encodings supported by client, e.g. gzip or deflate

User-Agent browser details, supplies server with information about the type of browser making the request

Connection: Keep-Alive browser is requesting the use of persistent TCP connections

2.3 Response Headers

Content-Type MIME-Type of content

Content-Length size of body (in bytes)

Content-Encoding compression algorithms

Location used by redirections

Date timestamp when the response was created

Last-Modified modification date of resource (assumed by server)

Expires date after which the result is considered stale

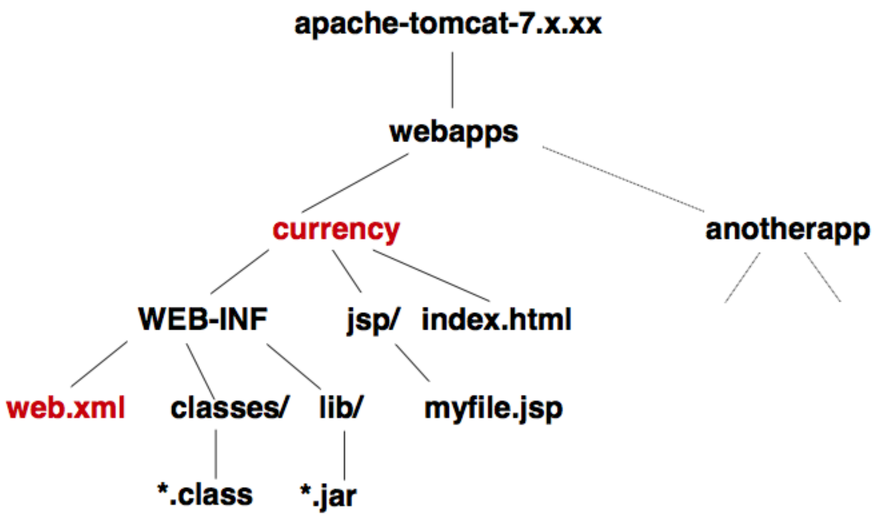
Server information about the server

Transfer-Encoding specifies type of transformation

Cache-Control information about cache handling (e.g. no-cache disables caching)

WWW-Authenticate information about authentication method

2.4 Servlet



Listing 2: Servlet Example

```java
public class Converter extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response) throws
      IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String amount = request.getParameter("amt");
    String from = request.getParameter("from");
    String to = request.getParameter("to");
    String res = computeResult(amount, from, to);
    out.println("<html>\n<body bgcolor=\"white\">");
    out.println("<h1>Currency Converter</h1>");
    out.println(amount + " " + from + " = " + res);
    out.println("</body>\n</html>");
  }
  String computeResult(String amount, String from, String to){...}
}
```

Listing 3: web.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
    sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <servlet>
        <servlet-name>CurrencyConverter</servlet-name>
        <servlet-class>ch.fhnw.ds.Converter</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CurrencyConverter</servlet-name>
    <url-pattern>/convert</url-pattern>
  </servlet-mapping>
</web-app>
```

3 Webservices

3.1 XML-RPC

simples RPC Protokoll über HTTP, benötigt keine lange Einarbeitungszeit

3.1.1 Primitive Datentypen

• int, i4	signed 32bit Integer
• string	ASCII string (no latin1)
• boolean	either 0 or 1
• double	double-precision floating point number
• dateTime.iso8601	z.B. 20050717T14:08:14
• base64	raw binary data, base64 encoded

Listing 4: Beispiele

```
<i4>13</i4>
<boolean>0</boolean>
```

3.1.2 Structs

- Struct enthält Members mit Name und Wert.
- können rekursiv sein (Structs die Structs enthalten)

Listing 5: Struct Beispiel

```
<i4>13</i4>
<struct>
  <member>
    <name>from</name>
    <value><i4>-5</i4></value>
  </member>
  <member>
    <name>to</name>
    <value><i4>5</i4></value>
  </member>
</struct>
```

3.1.3 Arrays

- Element-Typen können gemischt werden

Listing 6: Array

```
<array>
 <data>
  <value><i4>-5</i4></value>
  <value><string>44</string></value>
  <value><boolean>1</boolean></value>
 </data>
</array>
```

3.1.4 XML-RPC Request

Listing 7: Method Call

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>Echo.getEcho</methodName>
    <params>
```

```
      <param>
        <value>World</value>
      </param>
    </params>
</methodCall>
```

3.1.5 XML-RPC Response

Listing 8: Single Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>Hello World, welcome to XML-RPC</value>
    </param>
  </params>
</methodResponse>
```

Als Resultat kann nur ein Wert zurückkommen, dieser kann jedoch auch ein Struct oder ein Array sein.

Listing 9: Fault Result

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><i4>0</i4></value>
        </member>
        <member>
          <name>faultString</name>
          <value>No such handler: Echo.foo</value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

3.1.6 Apache XML-RPC Sample Server

Listing 10: Sample Server

```
import org.apache.xmlrpc.server.*;
import org.apache.xmlrpc.webserver.WebServer;

public class HelloServer {
  public static void main (String [] args) throws Exception {
    PropertyHandlerMapping phm = new PropertyHandlerMapping();
    phm.addHandler("Echo", ch.fhnw.ds.xmlrcp.echo.EchoImpl.class);
    WebServer server = new WebServer(80);
    XmlRpcServer xmlRpcServer = server.getXmlRpcServer();
    xmlRpcServer.setHandlerMapping(phm);
    server.start();
    System.out.println("Server started at port 80");
  }
}
```

Listing 11: Handler Class Server

```
public class EchoImpl {
public String getEcho(String name) {
return "[XML-RPC] Hello "+name+", welcome to XML-RPC";
}
}
```

Nur Instanzmethoden der Handlerklasse sind zugreifbar. Keine void Methoden. Public Default Constructor zwingend.

3.1.7 Apache XML-RPC Client

Listing 12: Handler Class Server

```java
import java.util.*;
import org.apache.xmlrpc.*;

public class HelloClient {
  public static void main (String [] args) throws Exception {
    XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
    config.setServerURL(new URL("http://localhost/xmlrpc"));
    XmlRpcClient client = new XmlRpcClient();
    client.setConfig(config);
    List params = new ArrayList();
    params.add(args[0]);
    Object result = client.execute("Echo.getEcho", params );
    System.out.println("The result is: "+result.toString());
  }
}
```

3.2 SOAP

3.2.1 WSDL (Web Services Description Language)

früher Web Services Definition Language

3.2.2 JAX-WS (Java API for XML Web Services

- Java-API zum Erstellen von Webservices
- Benutzt Annotationen um Entwicklung und Deployment von Clients und Service-Endpunkten zu vereinfachen
- Kommunikation über SOAP
- unterstützt erst WSDL 1.1

3.3 XML-RPC vs SOAP

4 REST