Netzprogrammierung
3. Internet-Dienste und Nutzung in Java

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de

Überblick

- Internet Dienste
- HTTP
- FTP
- Vereinheitlichte Dienstnutzung in Java
- Eigene URL Schemas

Internet Dienste

Was ist das Internet

- Eine weltweiter *Verbund von Rechnern*, die über Netze Daten austauschen können.
  - Hardware-bezogene Sicht
  - Zusammenschalten von lokalen Netzen zum Internet
  - Dabei notwendige Verarbeitung von Datenpaketen
- Eine *Protokollfamilie*
  - Netzbezogene Sicht
  - Protokollspezikationen
- Ein *offenes System*, in dem Dienste genutzt und angeboten werden können.
  - Nutzungs- und anwendungsbezogen
  - Beschreibt die Anwendungsmöglichkeiten des Internet

Internet als Protokollfamilie

- *Request For Comments*-Dokumente (RFC) definieren alle technischen Aspekte des Internet
- RFC 1738 :
  T. Berners-Lee, L. Masinter, und M. McCahill. Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force, December 1994.
- Internet Engineering Taskforce IETF erstellt RFCs
  http://www.ietf.org/rfc.html
- Standardisierungsprozeß ist als RFC standardisiert:
  The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force, RFC 3160, August 2001

IETF Arbeitsfelder (7/02)

- Applications Area
- General Area
- Internet Area
- Operations and Management Area
- Routing Area
- Security Area
- Sub-IP Area
- Transport Area

IETF Workinggroups Internet Area (7/02)

atommib	AToM MIB
dhc	Dynamic Host Configuration
dnsext	DNS Extensions
idn	Internationalized Domain Name
ifmib	Interfaces MIB
ipcdn	IP over Cable Data Network
ipoib	IP over InfiniBand
iporpr	IP over Resilient Packet Rings
ipv6	IP Version 6 Working Group
itrace	ICMP Traceback
l2tpext	Layer Two Tunneling Protocol Extensions
magma	Multicast & Anycast Group Membership
mobileip	IP Routing for Wireless/Mobile Hosts
pana	Protocol for carrying Authentication for Network Access
pppext	Point-to-Point Protocol Extensions
zeroconf	Zero Configuration Networking

Ausgewählte Standards und RFCs

Protokoll	Beschreibung	RFCs	STD
	Internet Ofcial Protocol Standards	1880	1
	Assigned Numbers	1700	2
	Host Requirements - Communications	1122	3
	Host Requirements - Applications	1123	3
IP	Internet Protocol	791	5
IP	Subnet Extension	950	5
IP	Broadcast Datagrams	919	5
IP	Broadcast Datagrams with Subnets	922	5
ICMP	Internet Control Message Protocol	792	5
IGMP	Internet Group Multicast Protocol	1112	5
UDP	User Datagram Protocol	768	6
TCP	Transmission Control Protocol	793	7
TELNET	Telnet Protocol	854, 855	8
FTP	File Transfer Protocol	959	9

Ausgewählte Standards und RFCs

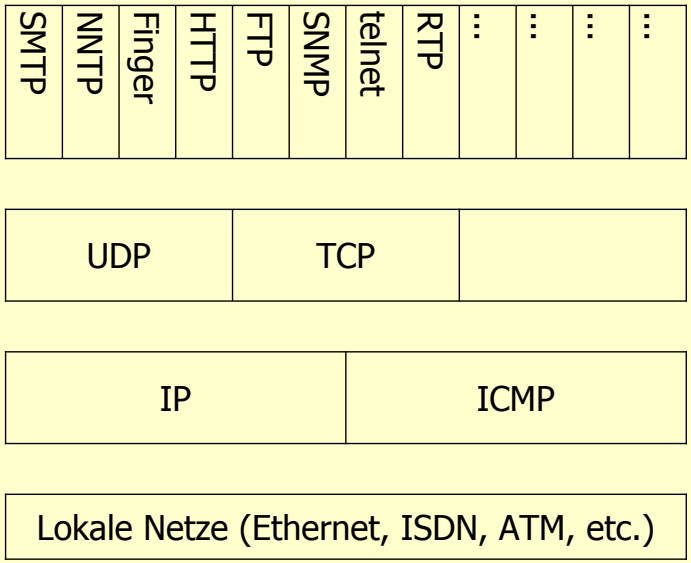
Protokoll	Beschreibung	RFCs	STD
SMTP	Simple Mail Transfer Protocol	821	10
SMTP-SIZE	SMTP Service Ext for Message Size	1870	10
SMTP-EXT	SMTP Service Extensions	1869	10
MAIL	Format of Electronic Mail Messages	822	11
CONTENT	Content Type Header Field	1049	11
NTPV2	Network Time Protocol, Version 2	1119	12
DOMAIN	Domain Name System	1034, 1035	13
DNS-MX	Mail Routing and the Domain System	974	14
SNMP	Simple Network Management Protocol	1157	15
SMI	Structure of Management Information	1155	16
Concise-MIB	Concise MIB Definitions	1212	16
MIB-II	Management Information Base-II	1213	17
NETBIOS	NetBIOS Service Protocols	1001, 1002	19
ECHO	Echo Protocol	862	20

Ausgewählte Standards und RFCs

Protokoll	Beschreibung	RFCs	STD
DISCARD	Discard Protocol	863	21
CHARGEN	Character Generator Protocol	864	22
QUOTE	Quote of the Day Protocol	865	23
USERS	Active Users Protocol	866	24
DAYTIME	Daytime Protocol	867	25
TIME	Time Server Protocol	868	26
TFTP	Trivial File Transfer Protocol	1350	33
RIP	Routing Information Protocol	1058	34
TP-TCP	ISO Transport Service on top of the TCP	1006	35
ETHER-MIB	Ethernet MIB	1643	50
PPP	Point-to-Point Protocol (PPP)	1661	51
PPP-HDLC	PPP in HDLC Framing	1662	51
IP-SMDSIP	Datagrams over the SMDS Service	1209	52

Internet-Protokolle und -Dienste

- Einordnung von Internet-Protokollen:



Internet als dienstorientiertes offenes System

- Internet Dienste sind (zumeist) definiert durch
  - Aufgabe
  - Portnummer auf dem der Dienst angeboten wird
  - Transportprotokoll (TCP oder/und UDP)
  - Protokoll

- Z.B.: Web Dienst
  - Übertragen von HTML Seiten
  - Port 80
  - TCP
  - HTTP

- Z.B.: Usenet Dienst
  - Übertragen von News
  - Port 119
  - TCP
  - NNTP

Beispiel: HTTP Protokoll

Client

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.04Gold (Win95; I)
Host: megababe.isdn:80
Accept: image/gif, image/jpeg, image/pjpeg, */*
```

```
HTTP/1.0 200 OK
Last-Modified: Sun, 15 Mar 1998 11:26:50 GMT
MIME-Version: 1.0
Date: Fri, 20 Mar 1998 16:43:11 GMT
Server: Roxen-Challenger/1.2beta1
Content-type: text/html
Content-length: 2990

<HTML><HEAD><TITLE>TU Berlin ---
```

Server

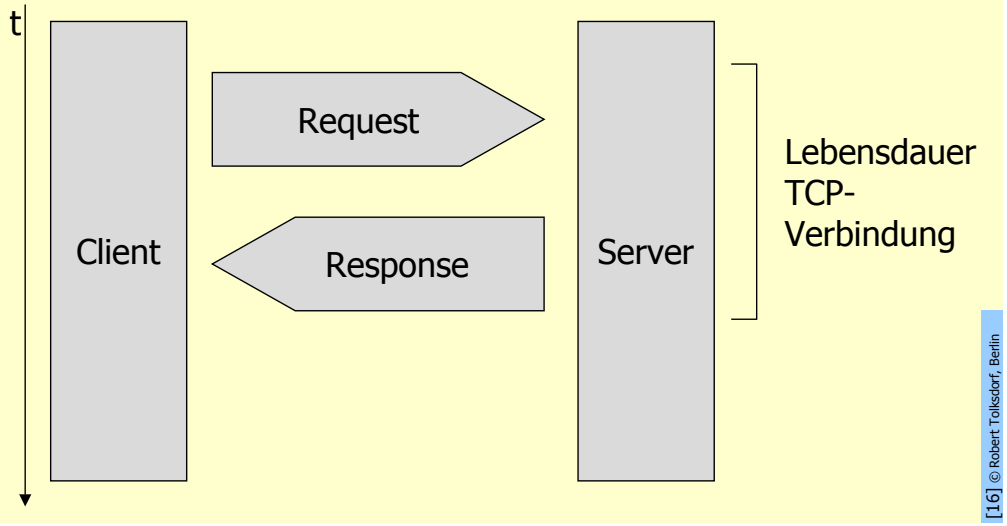
HTTP (Überblick)

Hypertext Transfer Protocol

- Aufgabe:
  Transfer von Informationen zwischen Web-Servern und Clients
- Port:
  80 ist für HTTP reserviert
- Transportprotokoll:
  TCP (leider)
- Protokoll:
  R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee. *Hypertext Transfer Protocol - HTTP/1.1*. RFC 2616, http://www.w3.org/Protocols/rfc2616/rfc2616.txt

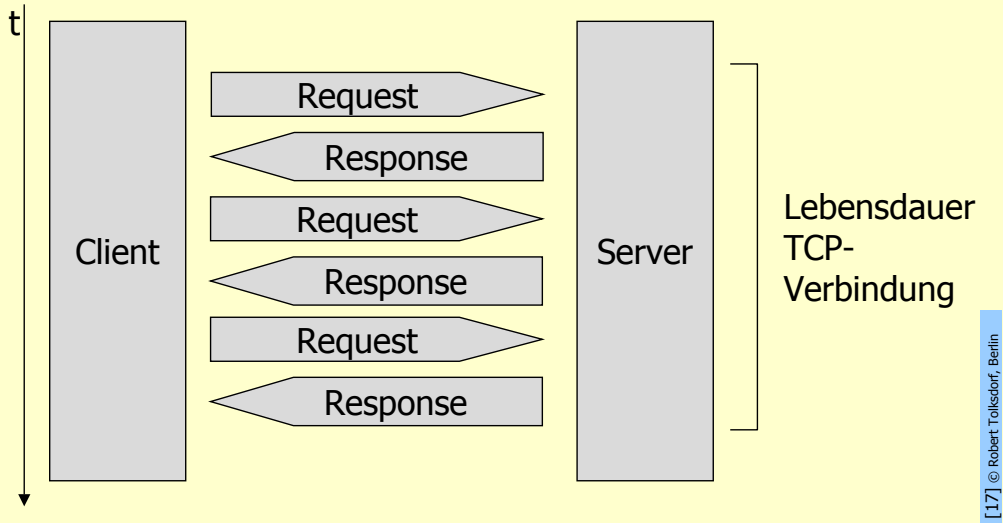
HTTP

- Zustandsloses Protokoll
- Request mit Response beantwortet



HTTP

- HTTP 1.1 erweitert Protokoll um interaktionslange Lebensdauer der TCP Verbindung



Anfragen

Aufbau Request

- Request besteht aus
  - Request
  - Request-Beschreibung durch Header
    - Allgemeine Beschreibungen
    - Request-spezifische Beschreibungen
    - Beschreibung eventuell beiliegenden Inhalts
- Beispiel:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.04Gold (Win95; I)
Host: megababe.isdn:80
Accept: image/gif, image/jpeg, image/pjpeg, */*
```


Requests in HTTP

- Format: *Methode URI* HTTP/*x.y*
- URI (Unique Resource Identifier) ist
  - Absoluter Pfad auf Server
  - Voll-qualifizierte URL bei Anfrage an Proxy
  - *, Authority
- GET
  - Anforderung einer Informationseinheit vom Server
    - GET /Style/CSS/ HTTP/1.1 an Server www.w3.org
    - GET http://www.w3.org/Style/CSS/ HTTP/1.1 an Proxy http-proxy.fu-berlin.de
  - Beantwortet mit Code, Headern, Inhalt
- HEAD
  - Anforderung der Beschreibung einer Informationseinheit vom Server
    - GET /Style/CSS/ HTTP/1.1 an Server www.w3.org
    - GET http://www.w3.org/Style/CSS/ HTTP/1.1 an Proxy http-proxy.fu-berlin.de
  - Beantwortet mit Code, Headern

Requests in HTTP

- PUT
  - Abspeichern einer Informationseinheit auf einem Server
  - PUT /index.html HTTP/1.1
  - Beantwortet mit Code, Headern
- POST
  - Hinzufügen von Informationen zu einer Informationseinheit
  - POST /speichere.cgi HTTP/1.1
    Daten daten daten
  - Beantwortet mit Code, Headern, eventuell Inhalt
- DELETE
  - Löschen einer Informationseinheit auf einem Server
  - DELETE /index.html HTTP/1.1
  - Beantwortet mit Code, Headern

Requests in HTTP

- TRACE
  - Server schickt erhaltenen Inhalt zurück
- CONNECT
  - Sagt Proxy, dass er Tunnel aufbauen soll
  - Tunnel: Verpacken eines Protokolls A in ein anderes Protokoll B, so dass die Anwendung A spricht, aber B benutzt

Requests in HTTP

- OPTIONS
  - Informationen über Fähigkeiten des Servers
  - Überträgt alle Allow-Header
  - OPTIONS * HTTP/1.1
    Host: www.inf.fu-berlin.de
  - HTTP/1.1 200 OK
    Date: Tue, 25 Nov 2003 11:29:16 GMT
    Server: Apache/1.3.26 Ben-SSL/1.48 (Unix) Debian GNU/Linux mod_perl/1.26 PHP/4.1.2
    Content-Length: 0
    Allow: GET, HEAD, OPTIONS, TRACE

Web Client

- Aufgabe:
  Holen Sie die Eingangsseite eines Web-Servers
  - Bauen Sie dazu einen TCP Socket zu Port 80 auf
  - Schicken Sie die Zeile GET / HTTP/1.0 zu dem Server
  - Lesen Sie alle Antwortzeilen

WebClient/1

```
import java.io.*;
import java.net.*;

public class WebClient {

  public static void main(String[] argv) {
    InetAddress host = null;
    Socket socket = null;

    try {
      socket = new Socket(argv[0],80);
    } catch (IOException iOExc) {
      System.err.println("Problem bei der Verbindungsaufnahme\n"+
                         iOExc.getMessage());
      return;
    }
```


WebClient/2

```
    try {
      OutputStream os = socket.getOutputStream();
      PrintWriter pw = new PrintWriter(os);
      pw.println("GET / HTTP/1.0\n");
      //      pw.println("GET / HTTP/1.1\nHost: "+argv[0]+"\n");
      pw.flush();
      InputStream is = socket.getInputStream();
      BufferedReader br = new BufferedReader(new InputStreamReader(is));
     while (true) {
      String l =br.readLine();
      if (l==null) {
          break;
      } else {
          System.out.println(l);
      }
     }
    } catch (IOException iOExc) {
      System.err.println("Problem beim Lesen\n"+ iOExc.getMessage());
      return;
    }
  }
}
```


Allgemeine und Anfrage-Header

Allgemeine Header in Anfrage und Antwort

- Date: Tue, 15 Nov 1994 08:12:31 GMT
  Datum des Abschickens der Anforderung im RFC 1123 Format
- Via: *Protokollversion Host* …
  Weg der Nachricht, z.B. Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
- Upgrade: *Protokoll*
  Wunsch nach Verwendung eines neueren Protokolls z.B.: Upgrade: HTTP/2.0
- Connection: close
  Verbindung nach Ergebnisübermittlung abbauen
- Trailer: *Trailer-Header*
  Nach dem Inhalt folgen weitere Kopfzeilen geschickt
- Transfer-Encoding: *Encoding*
  Wie die Mitteilung für den Transfer kodiert wurde
  - chunked: Mitteilung in Stücken transportiert, Initiale Hexzahl gibt jeweils Anzahl von Zeichen an
  - identity: Mitteilung unkodiert geschickt
  - gzip, compress, deflate: Komprimierte Übertragung

Allgemeine Header in Anfrage und Antwort

- Cache-Control: *Direktive*
  Steuert das Caching von Anfragen und Antworten
  - no-cache: Antwort darf nicht zur Beantwortung anderer Anfragen genutzt werden
  - no-store: Antwort- oder Anfragemitteilungen dürfen nicht gespeichert werden
  - weitere: max-age, max-stale, min-fresh, no-transform, only-if-cached, public, private, must-revalidate, proxy-revalidate, s-maxage
- Pragma: no-cache
  Entspricht Cache-Control: no-cache
- Warning: *Freitext*
  Zusätzlicher Hinweis

Request Header

- Host: *Name*
  Aus der URL ermittelter Name des Rechners von dem angefordert wird. Pflichtheader in HTTP 1.1
- If-Modified-Since: *Datum*
  Änderung der Informationseinheit seit *Datum*
  - Ja: 200 und Inhalt schicken
  - Nein: 304 und Inhalt nicht schicken
- If-Unmodified-Since: *Datum*
  Änderung der Informationseinheit seit Datum
  - Ja: 412 und nicht verarbeiten
  - Nein: Normal verarbeiten (als sei If-Unmodified-Since: nicht vorhanden)

Request Header

- Max-Forwards: *Anzahl*
  Wie oft ein OPTIONS oder TRACE weitergeleitet werden darf
- Range: *Bytebereich*
  Nur Teile der Information anfordern, Antwort ist dann 216
  Range: bytes=500-999
- Expect: *Token*
  Client erwartet bestimmte Eigenschaften von Server/Proxy
  (Falls nicht: 417)

Request Header

- From: *Mailadresse*
  Nutzer
- User-Agent: *Produkt*[*/ Version*]
  Browser z.B.
  (Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
- Referer: *URL*
  Seite auf der ein Link auf die angeforderte Seite stand
- Authorization: *Nachweis*
  Autorisierungsnachweis falls mit 401 angefordert
- Authorization: username="Mufasa",
  realm="testrealm@host.com",
  response="6629fae49393a05397450978507c4ef1"
- Proxy-Authorization: *Nachweis*
  Autorisierungsnachweis für Proxy, falls mit 407 angefordert

Request Header

- Accept: *Medienart/Variante;* q=*Qualität;* mxb=*Maximale Größe*
  - Accept: text/postscript; mxb=200000
- Accept-Charset: Zeichensatz

ISO-8859-1	ISO-8859-2	ISO-8859-3
ISO-8859-4	ISO-8859-5	ISO-8859-6
ISO-8859-7	ISO-8859-8	ISO-8859-9
ISO-2022-JP	ISO-2022-JP-2	ISO-2022-KR
UNICODE-1-1	UNICODE-1-1-UTF-7	UNICODE-1-1-UTF-8
US-ASCII	…	

Request Header

- Accept-Encoding: *Kodierung*

Binäre Daten	binary	Inhaltscodierung
8-Bit-Daten	8bit	
7-Bit-Daten	7bit	
uuencode-kodiert	quoted-printable	
base64-kodiert	base64	
…		
komprimiert	gzip, compress, deflate	Transfer-codierung
in Teilen	chunked	
unkodiert	identity	

- Accept-Language: *Sprachkürzel*
  - Accept-Language: de, en

Inhaltsheader

Inhalts-Header

- Content-Encoding: *Kodierung*
  Kodierung des Inhalts
- Content-Transfer-Encoding: *Kodierung*
  Transferkodierung
- Content-Type: *Medienart*
  Medientyp des Inhalts
  Content-Type: text/html
- Content-Language: *Sprachkürzel*
  Sprache des Inhalts
- Content-Length: *Länge*
  Länge des Inhalts in Byte
- Content-Range: *Range*
  Beschreibung des Ausschnitts bei Teilanforderung

Inhalts-Header

- Content-Location: *URI*
  Verweis auf eigentlichen Inhalt
- Content-MD5: *MD5Checksum*
  Message Digest für Inhalt zur Integritätsprüfung
- Expires: *Datum*
  Kann nach Datum aus Caches gelöscht werden
- Last-Modified: *Datum*
  Letzte Änderung

Antworten

Aufbau Response

- Response besteht aus
  - Antwort-Code
  - Response-Beschreibung durch Header
    - Allgemeine Beschreibungen
    - Response-spezifische Beschreibungen
    - Beschreibung eventuell beiliegenden Inhalts
- Beispiel:

```
HTTP/1.0 200 OK
Last-Modified: Sun, 15 Mar 1998 11:26:50 GMT
MIME-Version: 1.0
Date: Fri, 20 Mar 1998 16:43:11 GMT
Server: Roxen-Challenger/1.2beta1
Content-type: text/html
Content-length: 2990

<HTML><HEAD><TITLE>TU Berlin ---
```


Antwort Codes

- 200-er Codes: Erfolgreiche Ausführung
  - 200 – OK
    GET, HEAD, POST, TRACE erfolgreich, Antwort anbei
  - 201 – Created
    Erfolgreiches PUT oder POST
  - 202 – Accepted
    Für spätere Ausführung vermerkt
  - 203 – Non-Authoritative Information
    Metainformationen in Header stammen von Dritten
  - 204 – No Content
    Anfrage verarbeitet, kein Antwortinhalt notwendig
  - 205 – Reset Content
    Anfrage verarbeitet, Ansicht erneuern
  - 206 – Partial Content
    GET mit Teilanforderung erfolgreich, Teilantwort anbei

Antwort Codes

- 300-er Codes: Weitere Aktion des Client zur erfolgreichen Ausführung notwendig
  - 300 - Multiple Choices
    Verschiedene Versionen erhältlich, Accept-Header nicht eindeutig
  - 301 - Moved Permanently
    Verschoben (Location und URI Header geben Auskunft)
  - 302 - Found Moved Temporarily
    Verschoben (Location und URI Header geben Auskunft)
  - 303 See Other
    Andere Resource laden (Location und URI Header geben Auskunft)
  - 304 - Not Modied
    Bei GET mit If-Modified-Since Header
  - 305 Use Proxy
    Muss durch Proxy angesprochen werden (Location hat dessen Adresse)
  - 307 Temporary Redirect
    Umleitung bei GET, HEAD

Antwort Codes

- 400-er Codes: Nicht erfolgreich, Fehler bei Client
  - 400 - Bad Request
    Falsche Request Syntax
  - 401 - Unauthorized
    Passwort notwendig
  - 403 – Forbidden
    Ohne Angabe von Gründen verweigert
  - 404 - Not Found
    Nicht auffindbar
  - 405 - Method Not Allowed
    Methode für die Resource nicht zugelassen
  - 406 - Not Acceptable
    Information vorhanden aber nicht passend zu Accept-Headern
  - 407 Proxy Authentication Required
    Zuerst Autentifizierung bei Proxy nötig, der Proxy-Authenticate Header mit schicken muss
  - 408 - Request Timeout
    Timeout bei Übermittlung der Requests

Antwort Codes

- 409 Conflict
  Methode steht in Konflikt mit Zustand des Servers, Client kann Konflikt aufheben
- 410 Gone
  Permanent und absichtlich nicht auffindbar
- 411 Length Required
  Content- Length Header ist notwendig
- 412 Precondition Failed
  Bedingungen der Anfrage (in Headern) unerfüllbar
- 413 Request Entity Too Large
  Anfrage zu groß
- 414 Request-URI Too Long
  URI zu lang
- 415 Unsupported Media Type
  Unbekanntes Inhaltsformat
- 416 Requested Range Not Satisfiable
  Teilanforderung falsch beschrieben
- 417 Expectation Failed
  Expect Header unerfüllbar

Antwort Codes

- 500-er Codes: Nicht erfolgreich, Fehler bei Server
  - 500 - Internal Server Error
  - 501 - Not Implemented
    Angeforderte Methode nicht unterstützt
  - 502 - Bad Gateway
    Weiterer benutzer Server nicht erreichbar
  - 503 - Service Unavailable
    Server kann Dienst gerade nicht erbringen (Retry-After Header)
  - 504 - Gateway Timeout
    Weiterer benutzer Server antwortet nicht rechtzeitig
  - 505 HTTP Version Not Supported
    Unbekannte HTTP Version

Response Header

- Server: *Produkt*
  Server-Produkt
  Server: CERNb-HTTPD/3.0 libwww/2.17
- Accept-Ranges: *Token*
  Inwiefern der Server Teilübertragungen unterstützt
  Accept-Ranges: bytes
  Accept-Ranges: none
- Retry-After: *Datum*
  Bei 503: Zeitpunkt zur Wiederholung der Anfrage
  Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
  Retry-After: 120
- Age: *Sekunden*
  Geschätztes Alter der Resource

Response Header

- Location: *URI*
  Adresse unter der Resource aufzufinden ist
  Bei 201: Adresse der neu geschaffenen Resource
  Bei 3xx: URI für Umlenkung
- WWW-Authenticate: *Aufgabe*
  Bei 401: Client muss sich gegenüber Server ausweisen
- Proxy-Authenticate: *Aufgabe*
  Bei 407: Client muss sich gegenüber Proxy ausweisen

WebServer

- Aufgabe: Schreiben Sie einen Webserver, der alle Anfrage mit derselben HTML-Seite beantwortet.
  - Ein Webserver wartet an Port 80 auf Verbindungen
  - Er erhält über die Verbindung eine Zeile der Art
    GET */Pfad*
  - Er antwortet mit HTML Code und schließt die Verbindung
  - Vor der HTML Seite muß
    HTTP/1.0 200 Ok
    Content-Type: text/html
    Leerzeile
    stehen damit der Browser sie richtig anzeigt

WebServer/1

```
import java.net.*;
import java.io.*;
public class WebServer {
  public static void main(String[] argv) {
    // Nummer des Ports von Kommandozeile (Default 80)
    int port=80;
    if (argv.length==1) {
      try {
          port=java.lang.Integer.parseInt(argv[0]);
     } catch (Exception e) {}
    }
    // Hauptprogramm
    try {
      // Server initialisieren
      ServerSocket serverSocket= new ServerSocket(port);
```

WebServer/2

```
while (true) {
   Socket connection=serverSocket.accept();
   BufferedReader br = new BufferedReader(new InputStreamReader
                  (connection.getInputStream())); // Eine Zeile lesen
   String httpLine=br.readLine();
   // Antwort senden
   PrintWriter pw = new PrintWriter(connection.getOutputStream());
   pw.println("HTTP/1.0 200 Ok\n"+ "Content-type: text/html\n\n"+
         "<HTML><HEAD><TITLE>Hello</TITLE></HEAD>\n"+
         "<BODY><H1>Willkommen</H1>\n"+
         "<P>Das HTTP Kommando war:\n"+
         "<PRE>\n"+httpLine+"\n</PRE>\n</BODY></HTML>\n");
   pw.flush();
   connection.close();
     }    // Interaktion fertig
   } catch (Exception e) { System.err.println(e.getMessage());  }
  }
}
```

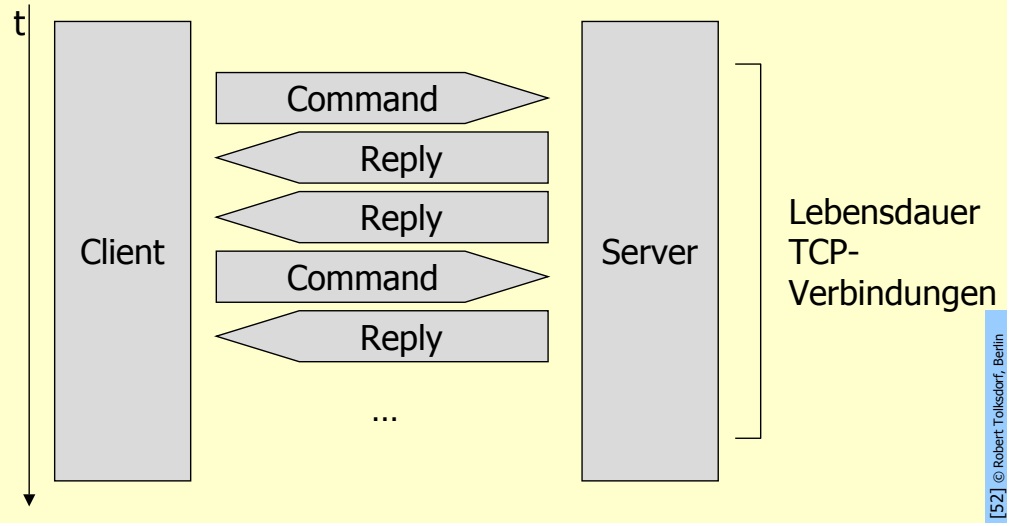

FTP

File Transfer Protocol

- Aufgabe:
  Transfer von Dateien zwischen FTP-Servern und Clients
- Ports:
  20 ist für FTP Kontrollverbindung reserviert
  21 ist für FTP Datenverbindung reserviert
- Transportprotokoll:
  TCP
- Protokoll:
  J. Postel und J. Reynolds. *FILE TRANSFER PROTOCOL (FTP)*, Oktober 1985. RFC 959, http://www.w3.org/Protocols/rfc2616/rfc2616.txt

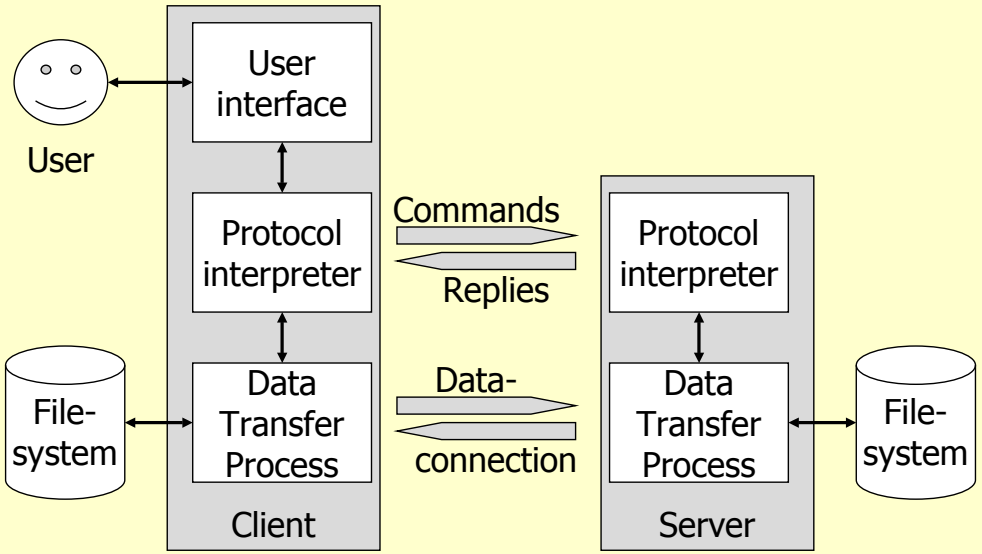
FTP

- Zustandshaltiges Protokoll
- Request mit Response beantwortet



FTP

- Modell:



Beispielsitzung

Login/Passwort-Eingabe

```
Connected to caramba.
220 ftp.cs.tu-berlin.de FTP server ready.
Name (ftp:tolk): ---> USER tolk
331 Password required for tolk.
---> PASS **********************
230 User tolk logged in.
```

Eingabe "dir"

```
ftp> ---> PORT 130,149,17,167,185,53
200 PORT command successful.
---> LIST
150 Opening ASCII mode data connection for /bin/ls.
total 33264
drwxr-xr-x  52 tolk    flp        8704          Jul 22 17:20 .
drwxr-sr-x  44 root    root       2560          Jun 25 14:23 ..
-rw-r--r--      1 tolk    flp        164352        Jul 16 09:22 NBI.ppt
[...]
226 Transfer complete.
12491 bytes received in 0.13 seconds (97.37 Kbytes/s)
```


Beispielsitzung

Eingabe "get test"

```
ftp> ---> PORT 130,149,17,167,185,54
200 PORT command successful.
---> RETR test
150 Opening ASCII mode data connection for test (6 bytes).
226 Transfer complete.
local: test remote: test
6 bytes received in 0.052 seconds (0.11 Kbytes/s)
```

Eingabe "get nofile"

```
ftp> ---> PORT 130,149,17,167,185,55
200 PORT command successful.
---> RETR nofile
550 nofile: No such file or directory.
```

Eingabe "quit"

```
ftp> ---> QUIT
221-You have transferred 6 bytes in 1 files.
221-Total traffic for this session was 13058 bytes in 2 transfers.
221-Thank you for using the FTP service on ftp.cs.tu-berlin.de.
221 Goodbye.
```


Kommandos zur Zugriffskontrolle

- Einloggen:
  - USER NAME (USER)
  - PASSWORD (PASS)
  - ACCOUNT (ACCT)
- Navigation
  - CHANGE WORKING DIRECTORY (CWD)
  - CHANGE TO PARENT DIRECTORY (CDUP)
  - STRUCTURE MOUNT (SMNT)
- Sitzungsmanagent
  - REINITIALIZE (REIN)
  - LOGOUT (QUIT)

Kommandos zu Übertragungsparametern

- Datenverbindung
  - DATA PORT (PORT)
    Datenport, falls nicht Standard
  - PASSIVE (PASV)
    Server wartet auf Datenverbindung
- Formate
  - REPRESENTATION TYPE (TYPE)
    Übertragungsrepräsentation (ASCII, Wortlänge etc.)
  - FILE STRUCTURE (STRU)
    Dateistruktur (File, Records, Seiten)
  - TRANSFER MODE (MODE)
    Übertragungsmodus (Strom, Blöcke, Komprimierung)

Kommandos zur Übertragung

- Dateitransfers
  - RETRIEVE (RETR)
  - STORE (STOR)
  - STORE UNIQUE (STOU)
  - APPEND (with create) (APPE)
  - ALLOCATE (ALLO) Platz reservieren
  - RENAME FROM (RNFR) + RENAME TO (RNTO)
  - ABORT (ABOR)
  - DELETE (DELE)
- Verzeichnisse
  - REMOVE DIRECTORY (RMD)
  - MAKE DIRECTORY (MKD)
  - PRINT WORKING DIRECTORY (PWD)
  - LIST (LIST)
  - NAME LIST (NLST)

Kommandos zur Übertragung

- Informationen
  - SYSTEM (SYST)
  - SITE PARAMETERS (SITE)
  - STATUS (STAT)
  - HELP (HELP)
- NOOP (NOOP)

Antwort Codes der Form xyz

- Antwortart-Codes für x:

1yz	Vorläufig positive Antwort, Server meldet sich wieder
2yz	Erfolgreiche Ausführung
3yz	Vorläufig positive Antwort, Client muß sich melden
4yz	Vorübergehen negative Antwort – erneut versuchen
5yz	Erfolglose Ausführung

- Betreff-Codes für y:

x0z	Probleme mit Anfragesyntax
x1z	Antworten auf Informationsanfragen
x2z	Antworten bzgl. Verbindungen
x3z	Antworten bzgl. Authentifizierungen
x5z	Antworten bzgl. Status des Dateisystems

Syntax

- 200 Command okay.
- 500 Syntax error, command unrecognized.
- 501 Syntax error in parameters or arguments.
- 202 Command not implemented, superfluous at this site.
- 502 Command not implemented.
- 503 Bad sequence of commands.
- 504 Command not implemented for that parameter.

Information

- 110 Restart marker reply.
- 211 System status, or system help reply.
- 212 Directory status.
- 213 File status.
- 214 Help message.
- 215 NAME system type
  (Where NAME is an official system name from the list in the Assigned Numbers document)

Verbindungen

- 120 Service ready in nnn minutes.
- 220 Service ready for new user.
- 221 Service closing control connection.
- 421 Service not available, closing control connection.
- 125 Data connection already open; transfer starting.
- 225 Data connection open; no transfer in progress.
- 425 Can't open data connection.
- 226 Closing data connection.
- 426 Connection closed; transfer aborted.
- 227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

Authentifizierung

- 230 User logged in, proceed.
- 530 Not logged in.
- 331 User name okay, need password.
- 332 Need account for login.
- 532 Need account for storing files.

Dateisystem

- 150 File status okay; about to open data connection.
- 250 Requested file action okay, completed.
- 257 "PATHNAME" created.
- 350 Requested file action pending further information.
- 450 Requested file action not taken. File unavailable
- 550 Requested action not taken. File unavailable
- 451 Requested action aborted. Local error in processing.
- 551 Requested action aborted. Page type unknown.
- 452 Requested action not taken. Insufficient storage space
- 552 Requested file action aborted. Exceeded storage allocation
- 553 Requested action not taken. File name not allowed.

Vereinheitlichte Dienstnutzung in Java

URI, URL, URN

- Uniform Resource Identifier URI: „A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource“ [RFC 2396]
  - Lediglich Syntax

```
absoluteURI   = scheme ":" ( hier_part | opaque_part )
relativeURI   = ( net_path | abs_path | rel_path ) [ "?" query ]
hier_part     = ( net_path | abs_path ) [ "?" query ]
opaque_part   = uric_no_slash *uric
uric_no_slash = unreserved | escaped |";"|"?"|":" |"@"|"&"|"="|"+"|"$"|",,
```

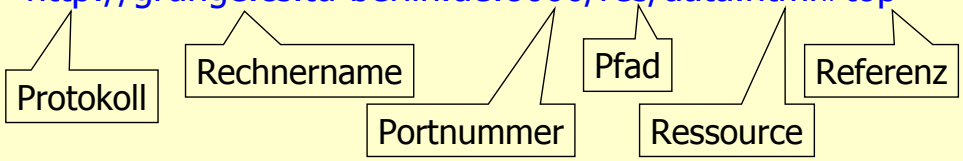
  - Beispiele:
    - ftp://ftp.is.co.za/rfc/rfc1808.txt
    - gopher://spinaltap.micro.umn.edu/00/Weather/Los%20Angeles
    - http://www.math.uio.no/faq/compression-faq/part1.html
    - mailto:mduerst@ifi.unizh.ch
    - news:comp.infosystems.www.servers.unix
    - telnet://melvyl.ucop.edu/
    - urn:isbn:n-nn-nnnnnn-n
  - URI-Schema typisiert URIs (ftp, gopher, fax, urn,…)

URI, URL, URN

- Uniform Resource Locator URL: „[…]a compact string representation for a resource available via the Internet.“ [RFC 1738]
  - Ist ein URI, dessen Schema auf die Zugreifbarkeit der Ressource im Netz hinweist
  - z.B. ftp://ftp.is.co.za/rfc/rfc1808.txt
- Uniform Resource Name URN: „[…] intended to serve as persistent, location-independent, resource identifiers and are designed to make it easy to map other namespaces“ [RFC 2141]
  - Ist eher URI, der Eigenschaft der Resource beschreibt
  - urn:isbn:n-nn-nnnnnn-n
  - URN-Namensraum strukturiert URNs (isbn,….)

URL

- URL Schemas sind für Internet-Dienste definiert und vereinheitlichen damit deren Nutzung syntaktisch:
  - http://grunge.cs.tu-berlin.de:8000/
  - ftp://ftp.cs.tu-berlin.de/pub/net/www
  - mailto:tolk@inf.fu-berlin.de
- Form:
  http://grunge.cs.tu-berlin.de:8000/res/data.html#top



- Bedeutung ist von Schema abhängig, URL ist nur als Syntax definiert

java.net.URL

- URLs als Objekte in Java: java.net.URL
- Konstruktoren:
  - Aus Zeichenkette:
    - URL(String spec)
  - Aus Komponenten:
    - URL(String protocol, String host, String file)
    - URL(String protocol, String host, int port, String file)
  - Relativ zu anderer URL
    - URL(URL context, String spec)
  - Mit eigenem Protokollobjekt
    - URL(String protocol, String host, int port, String file, URLStreamHandler handler)
    - URL(URL context, String spec, URLStreamHandler handler)

java.net.URL

- Bestandteile erfragen:
  - String getAuthority()
  - String getFile()
  - String getHost()
  - String getPath()
  - int getPort()
  - String getProtocol()
  - String getQuery()
  - String getRef()
  - String getUserInfo()
- Vergleichen:
  - boolean equals(Object obj)
  - boolean sameFile(URL other)
- Zeichenkettenrepräsentation
  - String toString()

java.net.URL

- Inhalt anfragen
  - Object getContent()
  - Object getContent(Class[] classes)

Beispiel

- Beispiel:

```
import java.net.*;
import java.io.*;
public class GetURLContent {
  public static void main(String[] argv) {
    try {
      URL page=new URL(argv[0]);
      Object content = page.getContent();
      System.out.println(content.getClass());
    } catch (Exception e) {
      System.err.println(e.getMessage());
      return;
    }
  }
```

```
>java GetURLContent http://www.inf.fu-berlin.de
class sun.net.www.protocol.http.HttpURLConnection$HttpInputStream
>java GetURLContent ftp://ftp.ietf.org
class sun.net.www.protocol.ftp.FtpURLConnection$FtpInputStream
```


URLConnection

- Vor dem Holen einer Seite sollten eventuell Header gesetzt werden
- Nach dem Holen einer Seite will man die Antwort-Header kennen
- java.net.URLConnection
- In java.net.URL:
  - URLConnection openConnection()
  - (getContent aus java.net.URL ist openConnection().getContent())

Beispiel: Sprache einstellen

```
import java.net.*;
import java.io.*;
public class GetURLLang {
  public static void main(String[] argv) {
    try {
      URL page=new URL(argv[0]);
      URLConnection connection=page.openConnection();
      connection.setRequestProperty("Accept-Language",argv[1]);
      connection.connect();
      System.out.println("Length: "+connection.getContentLength());
      System.out.println("Typ: "+connection.getContentType());
      System.out.println("Content:\n"+connection.getContent().getClass());
    } catch (Exception e) {
      System.err.println(e.getMessage());
      return;
    }
  }
}
```


Aufruf

```
>java GetURLLang http://www.cs.tut.fi/~jkorpela/multi/ de
Length: 2433
Typ: text/html
Content:
class sun.net.www.protocol.http.HttpURLConnection$HttpInputStream

>java GetURLLang http://www.cs.tut.fi/~jkorpela/multi/ en
Length: 1878
Typ: text/html
Content:
class sun.net.www.protocol.http.HttpURLConnection$HttpInputStream

>java GetURLLang http://www.cs.tut.fi/~jkorpela/multi/ fi
Length: 1848
Typ: text/html
Content:
class sun.net.www.protocol.http.HttpURLConnection$HttpInputStream
```

Beispiel: Informationen über eine Seite holen

```
import java.net.*;
import java.io.*;
public class GetURL {
  public static void main(String[] argv) {
    try {
      URL page=new URL(argv[0]);
      URLConnection connection=page.openConnection();
      connection.connect();
      System.out.println("Length: "+connection.getContentLength());
      System.out.println("Typ: "+connection.getContentType());
      System.out.println("Content:\n"+connection.getContent().getClass());
    } catch (Exception e) {
      System.err.println(e.getMessage());
      return;
    }
  }
}
```


Aufruf

```
>java GetURL http://www.inf.fu-berlin.de
Length: 9489
Typ: text/html; charset=iso-8859-1
Content:
class sun.net.www.protocol.http.HttpURLConnection$HttpInputStream

>java GetURL http://www.inf.fu-berlin.de/styles/inst-title-600x400.jpg
Length: 11947
Typ: image/jpeg
Content:
class sun.awt.image.URLImageSource

>java GetURL ftp://ftp.ietf.org
Length: -1
Typ: content/unknown
Content:
class sun.net.www.protocol.ftp.FtpURLConnection$FtpInputStream
```


Beispiel: FTP Dateien lesen

```
import java.net.*;
import java.io.*;

public class GetFTPFile {
    public static void main(String[] argv) throws IOException {
      URL url = new URL(argv[0]);
      URLConnection connection = url.openConnection();
      BufferedReader in = new BufferedReader(new
          InputStreamReader(connection.getInputStream()));
      String line;
      while ((line = in.readLine()) != null)
        System.out.println(line);
      in.close();
    }
}
```


Aufruf

```
>java GetFTPFile ftp://ftp.inf.fu-berlin.de
total 28
d--x--x--x         2 0        0            4096 Dec 11  2002 bin
drwxr-xr-x         4 0        0            4096 Dec 17  2002 debian
d--x--x--x     2 0        0            4096 Dec 11  2002 etc
drwxr-xr-x     3 30000    1            4096 Dec  3 12:46 incoming
d--x--x--x     2 0        0            4096 Dec 11  2002 lib
drwxrwxr-t    34 30000    10005        4096 Oct 31 12:27 pub
-rw-r--r--     1 0        0             687 Dec 11  2002 welcome.msg

>java GetFTPFiles ftp://ftp.inf.fu-berlin.de/welcome.msg
Welcome, archive user %U@%R !

----------------------------------------------
You are connected to the anonymous ftp server
ftp.inf.fu-berlin.de (160.45.117.11) at
...
```

Beispiel: FTP Datei schreiben

```
public class PutFTPFile {
    public static void main(String[] argv) throws IOException {
    URL url = new URL("ftp://ftp:tolk%40inf.fu-berlin.de@"+
                      "ftp.inf.fu-berlin.de/incoming/npuebung");
    URLConnection connection = url.openConnection();
    connection.connect();
    PrintWriter out = new PrintWriter(connection.getOutputStream());
    out.println("Gib mir einen Keks");
    out.close();
    }
}
```

```
>java GetFTPFiles ftp://ftp.inf.fu-berlin.de/incoming
total 8
----------    1 0        0               0 Dec  3 12:45 .notar
drwx------    2 0        0            4096 Dec  3 12:46 lost+found
-r--r--r--    1 30000    1              20 Dec  3 16:41 npuebung
```


Eigene URL Schemata

Eigene URL Schemata

- URL Architektur in Java ist erweiterbar
- Dazu müssen definiert werden
  - eine Klasse Handler
  - eine Klasse *Schema*URLConnection
- Sie müssen in einem Paket stehen: package de.fuberlin.inf.nbi.*schema*
- Durch die Property java.protocol.handler.pkgs muss dem Laufzeitsystem mitgeteilt werden wo die eigenen Klassen stehen

```
java -Djava.protocol.handler.pkgs=de.fuberlin.inf.nbi
     GetURL daytime://nawab.inf.fu-berlin.de
```


Daytime URLs

- Ziel: Eigene Handler für das daytime „Protokoll“ schreiben

```
package de.fuberlin.inf.nbi.daytime;
import java.net.*;

public class Handler extends java.net.URLStreamHandler
{
    protected URLConnection openConnection(URL u) {
        return new DaytimeURLConnection(u);
    }
}
```

Daytime URLs

```
package de.fuberlin.inf.nbi.daytime;
import java.net.*;
import java.io.*;
public class DaytimeURLConnection extends java.net.URLConnection {
   Socket socket;

   public DaytimeURLConnection(URL url) {
      super(url);
   }
   public void connect() throws IOException {
   socket = new Socket(url.getHost(),13);
   }
   public Object getContent() throws IOException {
      connect();
      BufferedReader in=new BufferedReader(new InputStreamReader(
                              socket.getInputStream()));
      return in.readLine();
   }
}
```

- java -Djava.protocol.handler.pkgs=de.fuberlin.inf.nbi GetURL
  daytime://nawab.inf.fu-berlin.de

Zusammenfassung

Zusammenfassung

1. Internet als Protokollfamilie
   1. RFCs der IETF definieren Internetdienste
   2. Dienste durch Aufgabe, Port, TP, Prtokoll definiert
2. HTTP
   1. Request-Response
   2. Header
3. FTP
   1. Kommunikation über zwei Sockets
   2. Protokoll
4. Vereinheitlichte Dienstnutzung in Java
   1. URLs vereinheitlichen Dienste
   2. Komfortable Nutzung in Java
5. Eigene URL Schemata
   1. Eigene Schemata können durch Handler erzeugt werden

Literatur

- www.ietf.org
- T. Berners-Lee, R. Fielding, L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396. 1998
- Berners-Lee, T., Masinter, L., and M. McCahill, Editors. Uniform Resource Locators (URL), RFC 1738, December 1994.
- Moats, R. URN Syntax, RFC 2141, May 1997.
- Joint W3C/IETF URI Planning Interest Group. URIs, URLs, and URNs: Clarifications and Recommendations 1.0. W3C Note. 2001. http://www.w3.org/TR/uri-clarification. Auch RFC 3305.
- The Official IANA Registry of URI Schemes. http://www.iana.org/assignments/uri-schemes
- The Official IANA Registry of URN Namespaces. http://www.iana.org/assignments/urn-namespaces