Netzprogrammierung
HTTP Kommunikation

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

1. Vereinheitlichte Dienstnutzung in Java
2. Eigene Dienste
3. Push und Pull in HTTP
4. Authentifizierung in HTTP
5. Anfragen in HTTP
6. Zustand in Web Anwendungen

Vereinheitlichte Dienstnutzung in Java

URI, URL, URN

Freie Universität Berlin

- 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,...)

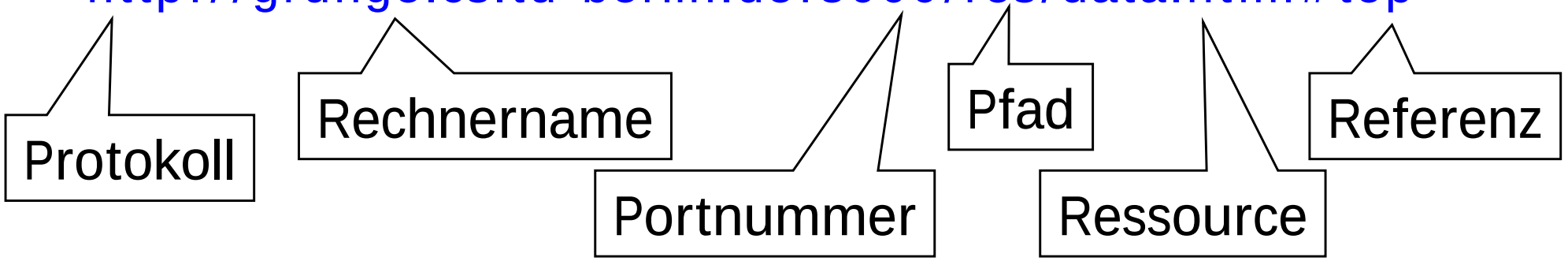
URI, URL, URN

- 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

```
>java URLComponents http://grunge.cs.tu-berlin.de:8000/res/data.html#top
Authority: grunge.cs.tu-berlin.de:8000
File: /res/data.html
Host: grunge.cs.tu-berlin.de
Path: /res/data.html
Port: 8000
Query: null
Ref: top
UserInfo: null

>java URLComponents http://user:pass@grunge.cs.
                                    tu-berlin.de:8000/res/data.html?q=fub
Authority: user:pass@grunge.cs.tu-berlin.de:8000
File: /res/data.html?q=fub
Host: grunge.cs.tu-berlin.de
Path: /res/data.html
Port: 8000
Query: q=fub
Ref: null
UserInfo: user:pass
```

URLConnection

- Man kann sich über eine „Verbindung“ zu einer durch eine URL bezeichnete Ressource „verbinden“
  - URLConnection openConnection()
- Jeweiliges Protokoll implementiert sein
- In Sun Java 1.5.0_06 in Windows:

```
>jar tf rt.jar|grep sun.net.www.protocol|
                                    grep URLConnection.class
sun/net/www/protocol/ftp/FtpURLConnection.class
sun/net/www/protocol/gopher/GopherURLConnection.class
sun/net/www/protocol/mailto/MailToURLConnection.class
sun/net/www/protocol/http/HttpURLConnection.class
sun/net/www/protocol/jar/JarURLConnection.class
sun/net/www/protocol/file/FileURLConnection.class
```

Beispiel: Dateien lesen (lassen)

Freie Universität Berlin

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

public class GetFile {
    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 GetFile http://www.inf.fu-berlin.de
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>FU-Berlin: Institut f&uuml;r Informatik</title>
    <base href="http://www.inf.fu-berlin.de">
    <link rel="stylesheet" type="text/css" href="http://www.inf.fu-
    berlin.de/styles/homepage.css">
    <!--script language="JavaScript" src="fuinf.js"-->
    <!--/script-->
    <style>
...
>java GetFile 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

Freie Universität Berlin

```
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 GetFile 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
```

URLConnection

- Zustände von URLConnection
  - Erzeugt
  - Verbunden
  - Geschlossen

- connect() wechselt Erzeugt -> Verbunden, falls noch nicht Verbunden
- Einige Operationen, die eine Verbindung brauchen, wechseln implizit zu Verbunden

java.net.URLConnection

- Abfragen der Ressource und Eigenschaften
  - Object getContent()
  - String getHeaderField(String name)
  - InputStream getInputStream()
  - OutputStream getOutputStream()
- Übliche Header
  - getContentEncoding()
  - getContentLength()
  - getContentType()
  - getDate()
  - getExpiration()
  - getLastModifed()
- Sind unter Umständen errechnet oder leer

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
```

java.net.URLConnection

- Setzen von Eigenschaften der Anfrage
  - setAllowUserInteraction(boolean b)
    Anfrage findet in Interaktion mit Nutzer statt
  - setDoInput(boolean b)
    Client will von Verbindung lesen (Normalwert: true)
  - setDoOutput(boolean b)
    Klient will auf Verbindung schreiben (Normalwert: false)
  - setIfModifiedSince(String s)
    IfModifiedSince-Header setzen
  - setUseCaches(boolean b)
    Zwischenspeichern von Daten erlauben (Normalwert: true)
  - setRequestProperty(String key, String value)
    Anfrageheader setzen

Beispiel: Sprache einstellen

Freie Universität Berlin

```
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
```

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://np.ag-nbi.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://np.ag-nbi.de

Push und Pull Interaktion

Client-Pull und Server-Push

- HTTP Interaktion mit Anfrage und Antwort wird durch Client-Pull und Server-Push erweitert
- Client-Pull
  - Client lädt Inhalte in regelmäßigen Abständen nach
  - Server löst das Verhalten durch zusätzlichen Header aus
- Server-Push
  - Server schickt mehrere Antworten nacheinander
  - Client ersetzt jeweils darstellung

Client-Pull

- Server gibt zusätzlichen Antwortheader an:
  - *Intervall-in-Sekunden*;HREF="*URL*"
- Beispiel
  - Refresh: 5;HREF="http://boersenkurse.de"

Dynamische Zeitanzeige

Freie Universität Berlin

```
import java.net.*;
import java.io.*;
public class TimePull {
   public static void main(String[] argv) {
      try {
       String myURL="http://"+(InetAddress.getLocalHost()).getCanonicalHostName();
       ServerSocket serverSocket= new ServerSocket(80);
       while (true) {
           Socket connection=serverSocket.accept();
           // Zeile lesen (und wegwerfen)
           (new BufferedReader(new
    InputStreamReader(connection.getInputStream()))).readLine();
           PrintWriter pw = new PrintWriter(connection.getOutputStream());
           pw.println("HTTP/1.1 200 Ok\n"+ "Content-type: text/html\n"+
                      "Refresh: 1;HREF="+myURL+"\n\n"+
                      "<html><head><title>Hello</title></head>\n"+
                      "<body><p>Es ist hier gerade "+new java.util.Date()+
                      "</body></html>\n");
           pw.flush();
           connection.close();
       }    // Interaktion fertig
    } catch (Exception e) { System.err.println(e.getMessage());  }
   }
}
```

Server-Push

- Server liefert eine Antwort vom Medientyp multipart/mixed an den Klienten
- Markierung trennt mehrere vollständigen Antwortteile
- Klient ersetzt Darstellung durch jeweil neuen Antwortteil
- Server verzögert Auslieferung von neuem Antwortteil

```
200 Ok
Content-type: multipart/mixed;boundary=Seite

--Seite
Content-type: text/html

<html><head><title>Ping</title></head><body>
<h1>PING!</h1></body></html>

--Seite
Content-type: text/html

<html><head><title>Pong</title></head><body>
<h1>PONG!</h1></body></html>
```

Dynamischer Inhalt

```
import java.net.*;
import java.io.*;
public class PingPongPush {
    public static void main(String[] argv) {
        try {
        ServerSocket serverSocket= new ServerSocket(80);
        while (true) {
           Socket connection=serverSocket.accept();
           // Zeile lesen (und wegwerfen)
           (new BufferedReader(new

    InputStreamReader(connection.getInputStream()))).readLine();
           PrintWriter pw = new
    PrintWriter(connection.getOutputStream());
           pw.println("HTTP/1.1 200 Ok\n"+
                      "Content-type: multipart/mixed;boundary=Seite\n");
```

Dynamischer Inhalt

```
            try {
               while(true) {
                     pw.println("--Seite\nContent-type: text/html\n\n"+
                                 "<html><head><title>Ping</title></head><body>"+
                          "<h1>PING!</h1></body></html>");
                     pw.flush();
                     try {
                        Thread.sleep(Integer.parseInt(argv[0])*1000);
                     } catch (Exception e) {}
                     pw.println("--Seite\nContent-type: text/html\n\n"+
                                 "<html><head><title>Pong</title></head><body>"+
                                 "<h1>PONG!</h1></body></html>");
                     pw.flush();
                     try {
                        Thread.sleep(Integer.parseInt(argv[0])*1000);
                     } catch (Exception e) {}
               }
            } catch (Exception e) { System.err.println(e.getMessage());}
         }
      } catch (Exception e) { System.err.println(e.getMessage());  }
   }
}
```

Authentifizierung in HTTP

Interaktion zur Authentifizierung

- Seiten im Web können Zugriffsschutz tragen
- Interaktion zum Abruf
  - Normales GET
  - Antwort 401 und WWW-Authenticate: Header, der Nachweis in unterschiedlichen Schemata anfordert
  - Weiteres GET mit Authorization: Header, der je nach Schema Parameter trägt
  - Antwort 200

Zugangsschutz auf Web-Servern

Freie Universität Berlin

- Beispiel am Apache Server zum Schutz von
  http://www.inf.fu-berlin.de/inst/ag-nbi/lehre/0304/V_NP/geheim/index.html
- .htaccess:

  AuthUserFile /import/htdocs/inst/ag-nbi/lehre/0607/V_NP/geheim/.htpasswd

  AuthGroupFile /dev/null

  AuthName "Zugang zur geheimen Seite"

  AuthType Basic

  require user Nutzer
- .htpasswd:

  Nutzer:ALwPfINObhEus
  - erzeugt mit /usr/apache/bin/htpasswd -c .htpasswd Nutzer
  - Passwort ist
    ganzgeheim

ReadProtectedURL/1

Freie Universität Berlin

```
import java.net.*;
import java.io.*;
public class ReadProtectedURL {
  public static void main(String[] argv) {
    int response=200;
    try {
      // Normal verbinden
      URL page=new URL(argv[0]);
      URLConnection connection=page.openConnection();
      connection.connect();
      System.out.println("Length: "+connection.getContentLength());
      System.out.println("Typ: "+connection.getContentType());
      // Ist es eine HTTP Verbindung?
      if (connection instanceof java.net.HttpURLConnection) {
      // Ja, Response-Code erfragen
     response=((java.net.HttpURLConnection)connection).getResponseCode();
      System.out.println("HTTP Result: "+ response);
```

ReadProtectedURL/2

```
if (response==401) {   // Ist es ein 401?
  // Herausforderung ermitteln
  String realm=connection.getHeaderField("WWW-Authenticate");
  System.out.println("Realm: "+ realm );
  // Realm: Basic realm="Zugang zur geheimen Seite"
  // Hier eigentlich: Komplexe Behandlung der Herausforderung
  realm=realm.substring(12);
  // Neu anforden aber mit Authorization Header
  URL pageAuth = new URL(argv[0]);
  connection=pageAuth.openConnection();
  byte[] userPass="Nutzer:ganzgeheim".getBytes();
   // in Base64 Kodierung
  String b64UserPass=new sun.misc.BASE64Encoder().encode(userPass);
  connection.setRequestProperty("Authorization", "Basic "+ b64UserPass);
  connection.connect();
 }
}
```

ReadProtectedURL/3

```
    BufferedReader br = new BufferedReader(
       new InputStreamReader(connection.getInputStream()));
      while (true) {
      String l =br.readLine();
      if (l==null) {
          break;
      } else {
          System.out.println(l);
      }
    }
    } catch (Exception e) {
      System.err.println(e.getMessage());
      return;
    }
  }
}
```

Anfragen in HTTP

Parameter für Web-Skripte

Freie Universität Berlin

- Zwei Arten der Übermittlung von Parametern an Skripte:
  - GET: Daten werden in URL kodiert
  - POST: Daten werden kodiert über Standardeingabe geliefert
- HTML:

```
<html><body>
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"
   method="get"> <input name="Eingabe" type="text">
   <input type="submit" value="Per GET">
</form>
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"
   method="post">
  <input name="Eingabe" type="text">
   <input type="submit" value="Per POST">
</form>
</body>
</html>
```

Echo Skript

- Serverseitig:

```
#!/usr/local/bin/perl -w
use strict;
use CGI;
my $cgi = new CGI; // Dekodiert Parameter je nach Methode
print $cgi->header(),
      $cgi->start_html('Echo'),
      $cgi->h1('Echo der Eingabe:'),
      $cgi->pre($cgi->param('Eingabe')),
      $cgi->end_pre(), end_html();
```

Kodierung von Eingabewerten

- Parameter haben bestimmten Übergabeformat
  - $Name_1=Wert_1\&Name_2=Wert_2$
- Dieser *Query String* wird
  - bei GET an die URL mit ? getrennt angehängt
    - http://flp.cs.tu-berlin.de/~tolk/echo.cgi?Eingabe=Hallo!
  - bei POST als Inhalt übermittelt und beim Web-Server über stdin einem Skript übergeben
- Query String selber muss kodiert werden
  - Um Transport zu sichern
  - Um bedeutungstragende Zeichen (=, & etc.) übertragen zu können
  - Medientyp der Nachricht ist application/x-www-form-urlencoded

Kodierung von Werten

- Namen und Werte anpassen
  - Leerzeichen -> +
  - Reservierte Zeichen -> %HH
    - CR LF -> %0D%0A
    - & -> %26
    - %&" "+#äß -> %25%26%22+%22%2B%23%E4%DF
- Name-Wert Paare gruppieren
  - $Name_1$=$Wert_1$
- Parameter zu Query-String gruppieren
  - $Name_1$=$Wert_1$&$Name_2$=$Wert_2$

- Dekodierung entsprechend

URLEncoding in Java

- java.net.URLEncoder:
  - public static String encode(String s, String enc) throws UnsupportedEncodingException
  - enc ist registrierte Zeichensatzbenennung wie “ISO-8859-1“
- java.net.URLDecoder:
  - public static String decode(String s, String enc) throws UnsupportedEncodingException

Beispiel: PostURL

```
import java.net.*;
import java.io.*;
public class PostURL {
  public static void main(String[] argv) throws IOException {
    URL url = new URL(argv[0]);
    HttpURLConnection http =(HttpURLConnection)url.openConnection();
    http.setDoOutput(true);
    PrintWriter out = new PrintWriter(http.getOutputStream());
    out.println(URLEncoder.encode("Eingabe","ISO-8859-1") + "=" +
                URLEncoder.encode("%&\" \"+#äß","ISO-8859-1"));
    out.close();
    BufferedReader in = new BufferedReader(new
        InputStreamReader(http.getInputStream()));
    String line;
    while ((line = in.readLine()) != null)
        System.out.println(line);
    in.close();
  }
}
```

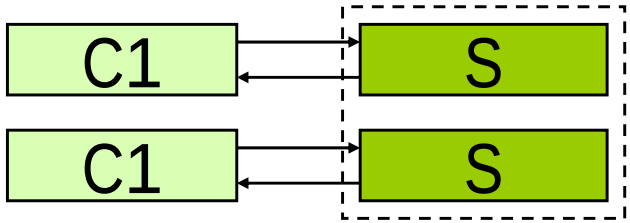
Ausführen

```
>java PostURL http://flp.cs.tu-berlin.de/~tolk/echo.cgi
Eingabe=%25%26%22+%22%2B%23%E4%DF
<!DOCTYPE HTML
   PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<HTML LANG="en-US"><HEAD><TITLE>Echo</TITLE>
</HEAD><BODY><H1>Echo der Eingabe:</H1><PRE>%&"
   "+#äß
</PRE></PRE></BODY></HTML>
```

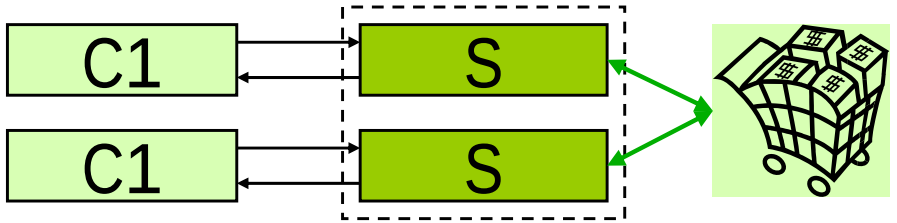
Zustand in Web Anwendungen

Zustand in HTTP

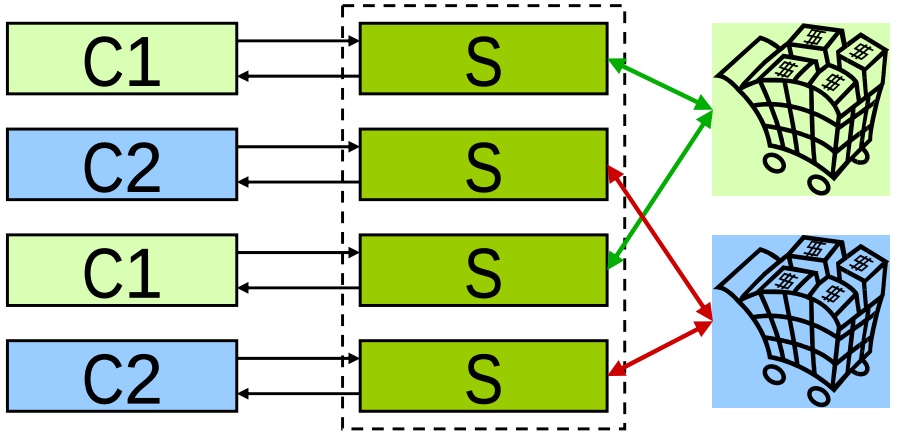
- HTTP ist zustandslos
  - Zwei Interaktionen sind unabhängig voneinander



- Zustand aber oft benötigt
  - Transaktionen auf Datensatz beim Server (z.B. Warenkorb)

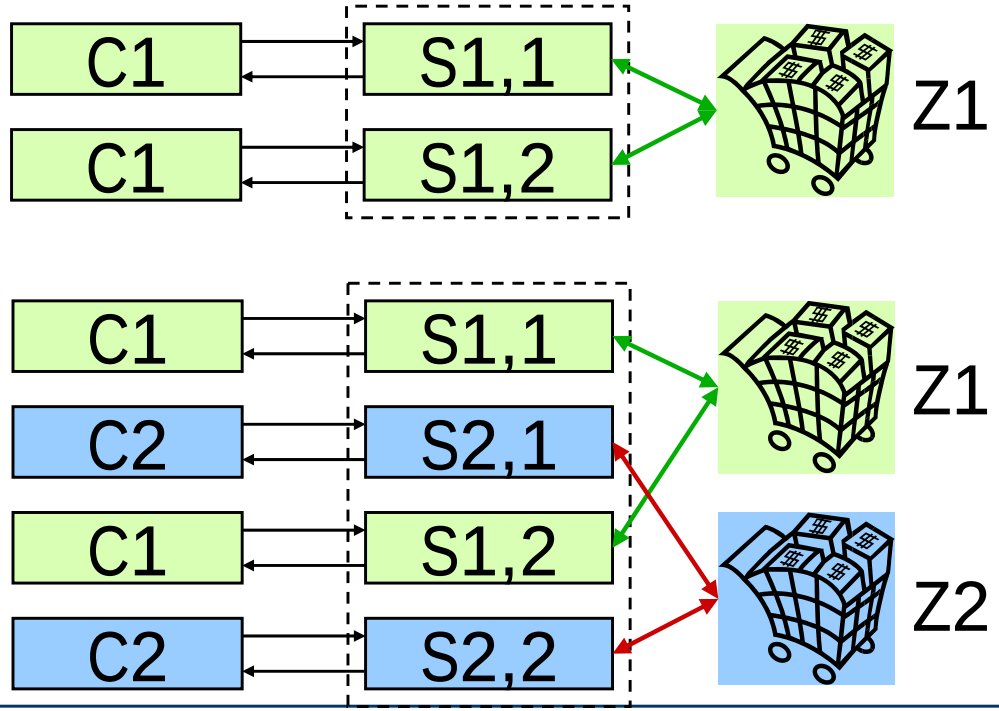


  - Unterscheidung von Klienten zur
    - Personalisierung
    - Authentifizierung
    - ...



Sessions

- Einführung von Sitzungen (Sessions)
- Sitzung: Folge von Interaktionen, die einen gemeinsamen Zustand haben
- Identifikation in der Interaktion durch eindeutige Sitzungsnummer (Session-ID)
- Ermittlung des Zustand auf Basis der Session-ID



Zustand in HTTP

- Client aus HTTP- und Socket-Informationen eindeutig identifizieren?
- Session-ID=
  (Browsername x User x Betriebssystem x IP-Adresse)
- *Nicht* eindeutig, weil:
  - Informationen bis auf IP-Adresse nicht immer vorhanden
  - IP-Adresse nicht eindeutig
    - Mehrere Nutzer auf einem Rechner
    - Proxy/Firewall/NAT Problematik: Keine individuellen IP-Adressen nach aussen
    - Mehrere Browser-Sessions des gleichen Nutzers
- => Session-ID muss in der Interaktion immer zwischen Klient und Server ausgetauscht werden

Zustand in HTTP

1. Versteckte Formularfelder enthalten Session-ID

- Formularfelder in HTML:
  - <input type=typ name="name" ...> z.B.:
    - <input type="text" name="PLZ"> Texteingabe
    - <input type="password" name="pw">Passwordeingabe
    - Weitere Auswahlen und Textfelder
    - <input type="hidden" name="SessionID"
      value="977e5d8ae8500c456ab1fca6cbaa12af">
- Bei Submit wird ein Query-String

  PLZ=14195&pw=geheim&SessionID=977e5d8ae8500c456ab1fca6cbaa12af

  erzeugt und an den Server übermittelt
- Server wertet Session-ID aus und baut sie in Formulare auf Ergebnisseite ein

Woher Session IDs nehmen?

- Beispiel: http://ws.apache.org/axis
- String sessionID=
  org.apache.axis.utils.SessionUtils.generateSessionId();

- java -cp axis.jar;commons-logging.jar;
  commons-discovery.jar;. SessionID

- 43B04BB7719E14AC1105FCDB20F0CBD2

- Andere Bibliotheken natürlich auch nutzbar

Freie Universität Berlin

2. Session-ID in URLs in Verweisen (URL Rewriting)

- www.amazon.de: Einstieg per HTTP auf URL mit schon kodierter Session-ID geleitet:
  http://www.amazon.de/exec/obidos/tg/browse/-/301128/ref=cs_nav_tab_1/028-1096689-7395702

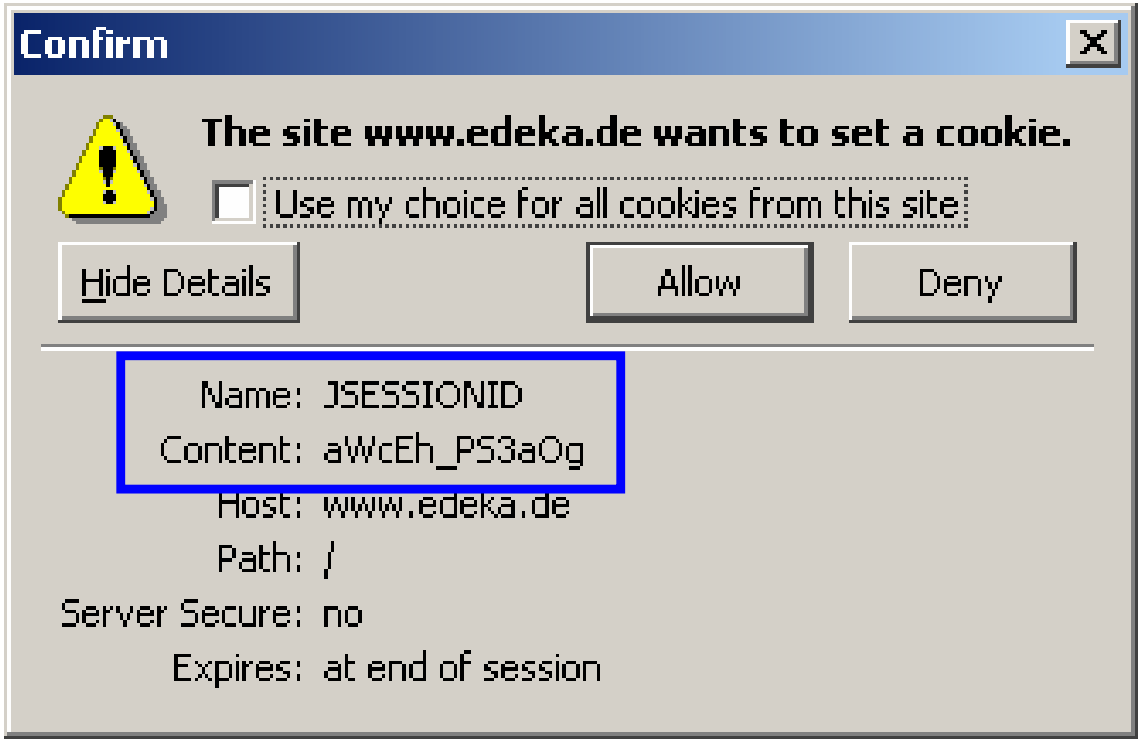
http://www.amazon.de/exec/obidos/tg/browse/-/299956/ref=cs_nav_tab_3/028-1096689-7395702

http://www.amazon.de/exec/obidos/tg/stores/static/-/general/international-gateway/ref=cs_nav_sn_1_new/028-1096689-7395702

URLs

- Session-ID kann unterschiedlich kodiert werden
- Abbildung von Pfad auf Informationen ist Server-Sache
  - Im Pfad:
    http://www.amazon.de/exec/obidos/tg/browse/-/301128/ref=cs_nav_tab_1/028-1096689-7395702
  - Im Query-String:
    http://www.cyberport.de/webshop/cyberportShop.omeco?ORDER=&PHPSESSID=8823f90c85597aedc87100cd91a4c7fd&FINANZING=
- Vorteil:
  - Zustand kann außerhalb von Formulareingaben gehalten werden
  - Portabel
- Nachteile:
  - Alle Verweise müssen entsprechend markiert werden
  - Alte Session-ID kann in Bookmark sein
  - Gültige Session-ID kann einfach an andere Nutzer gelangen

3. Cookie Mechanismus zum Speicher der Session-ID



- Cookie ist kleiner Datensatz, der bei Klienten gespeichert ist
- Server kann ihn setzen
- Klient schickt ihn bei jeder weiteren Interaktion mit
- Implementiert mit zusätzlichen HTTP-Headern

Cookies

- Server schickt Set-Cookie Header in HTTP
  Set-Cookie: *Name*=*Wert*; expires=*Datum*; path=*Pfad*; domain=*Internet-Domäne*; secure
- Wenn der Klient will, speichert er den Cookie
- Beispiele:
  - www.edeka.de
    Set-Cookie: JSESSIONID=a3nwR5on0lhe; path=/
  - www.bmw.de
    Set-Cookie: WEBTRENDS_ID=160.45.114.204-1073465686.192079; path=/
  - www.amazon.de
    Set-Cookie: obidos_path_continue-shopping=continue-shopping-url=/tg/browse/-/301128%3Fsite-redirect%3Dde&continue-shopping-post-data=&continue-shopping-description=browse.gateway.301128; path=/; domain=.amazon.de

Cookie-Felder

- *Name*=*Wert*;
  Zeichenkette ohne Leerzeichen, Semikolon, Komma ->
  Ähnlich Query String kodieren
- domain=*IP-Domäne*
  Der Klient soll den Cookie bei einem GET an Rechner mitschicken, zu denen die IP-Domäne passt
  - mindestens drei Punkte erforderlich
  - bei Toplevel-Domänen (com, edu, net, org, gov, mil,...) nur zwei
- path=*Pfad*;
  Der Klient soll den Cookie bei einem GET an Rechner mitschicken, zu denen die IP-Domäne und der Pfad darin passt
  - / für alle Seiten
  - /foo für /foobar aber auch für /foo/bar.html

Cookie-Felder

- expires=*Datum*
  Altersgrenze ab der der Cookie nicht mehr gespeichert oder herausgegeben werden soll
  - *Datum:* „based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is GMT and the separators between the elements of the date must be dashes.”
  - Fehlt expires verfällt der Cookie am Ende der Nutzung des Klienten
  - Ist das Datum in der Vergangenheit soll der Klient den Cookie löschen
- secure
  Cookie nur an Server schicken, wenn dieser mit SSL (also https-Protokoll) kontaktiert wird

Cookies

- Wenn bei einer Anfrage nach einer URL
  - der Klient will,
  - die Domäne zur URL passt,
  - der Pfad zur URL passt,
  - bei mit secure markierten Cookies https benutzt wird
  - der Cookie nicht veraltet ist

  schickt der Klient das Name-Wert-Paar als Header mit

- Cookie: $Name_1=Wert_1$; $Name_2=Wert_2$; ...

- Server kann mehrere Cookies senden (mehrere Set-Cookie Header)
- Klient kann mehrere Cookies antworten (mehrere pro Header oder mehrere Header)

Cookies

- Vorteile:
  - Einfacher Mechanismus ohne Inhaltsänderung
  - Seiten müssen nicht generiert werden
  - Cookies sind (wahrscheinlich) persistent über Klientennutzungen
- Nachteile:
  - Schlechtes Image von Cookies
  - Cookies für „fremde“ Domains
  - Datenschutzaspekt / Nutzerprofile
  - Klientenverhalten heute frei konfigurierbar

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