Programmieren in Java

Vorlesung 06: Webprogrammierung

Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2013

Inhalt

Vorlesungsüberblick

Webprogrammierung

- Einführung
- Java API: Client
- Java API: Server

Vorlesungsüberblick

Bisher

- Einfache Klassen, Enum, Tests
- Zusammengesetzte Klassen (Interfaces), Collections
- Abstraktion mit Klassen, Refactoring mit Eclipse
- GUI mit Swing
- Testen abstrakter Klassen, Mock-Objekte, Pattern

Geplant

- Webprogrammierung mit Servlets
- Generics
- Vergleiche in Java: equals, compareTo, hashCode, Iterator.
- Exceptions, Input/Output-Hierarchie, XML, Serialization
- Rekursive Klassen, Reflection

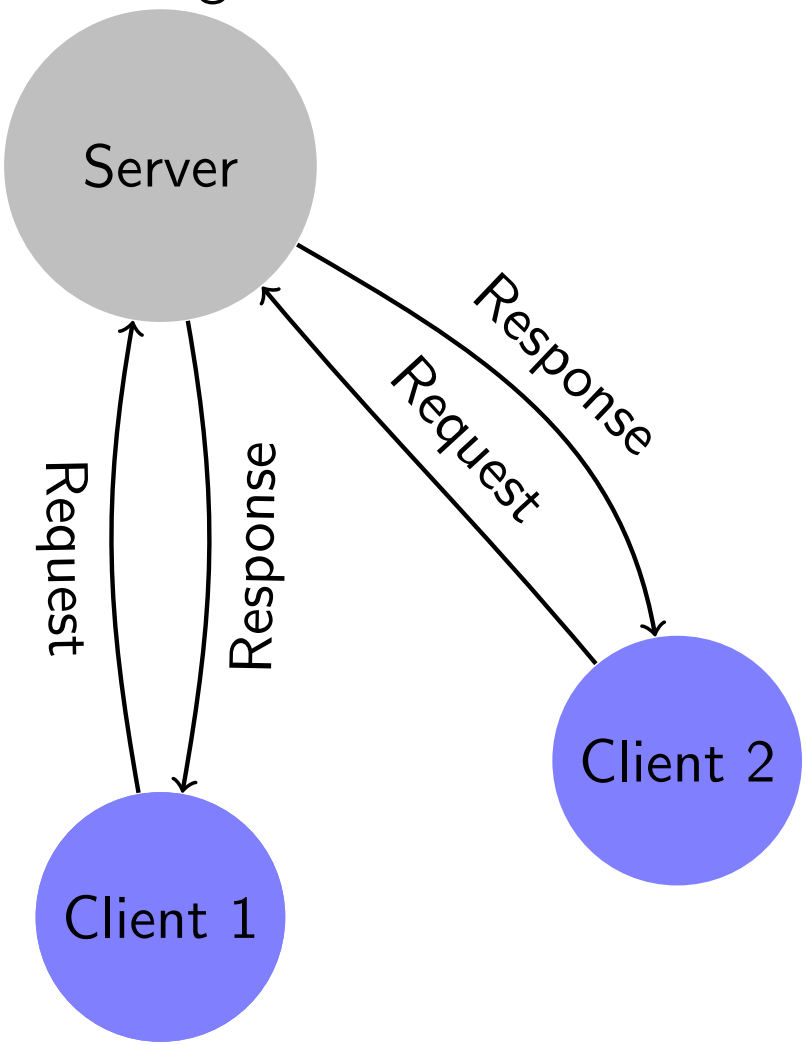
Was ist Webprogrammierung?

- Applikationen bestehen aus kommunizierenden Komponenten
- Typischerweise: Server und Clients
- Ein weites Feld, viele Technologien
- hier:
  - nur ,,Reinschnuppern"
  - Eindruck von der Struktur einer solchen Applikation
  - Weitere Informationen:
    Anders Møller, Michael Schwartzbach:
    ,, An Introduction to XML and Web Technologies"

HTTP

Prinzip

Geläufiges Protokoll für Client-Server Kommunikation



- Clients stellen **Requests**
  - Abfragen von Serverdaten (z.B. ,,Gib mir diese Datei!")
  - Nachricht an den Server (z.B. ,,Ich bin ab jetzt offline!")
- Server antworten mit **Response**:
  - Status (200 OK, 404 NOT FOUND)
  - Inhalt:

```
<!doctype html>
<html itemscope="itemscope">
<head>
<me+ ...
```

HTTP

Requests

- Anfrage geht an eine http-**URL**
  http://⟨server⟩**:**⟨port⟩**/**⟨path⟩**/**⟨to⟩**/**⟨resource⟩**?**⟨query⟩
  `http://www.google.de:80/search?q=hello`
- **Request Methods**:

  GET Abfrage von Daten.
  Sollte keine Zustandsänderung auf dem Server verursachen.

  POST Nachricht an den Server, die seinen internen Zustand verändert.

  Es sind noch weitere Methoden vorhanden (`DELETE`, . . .);
  diese werden aber wenig genutzt.
- Rest des Requests: **Header** mit weiteren Parametern, eventuell gefolgt von einem **Datenstrom**, der zum Server hochgeladen werden soll.

HTTP

Response

Die Antwort des Servers besteht, unter anderem, aus:

- **Response Code**:
  - 200 OK,
  - 400 BAD REQUEST,
  - 404 NOT FOUND,
  - `https://en.wikipedia.org/wiki/List_of_HTTP_status_codes`
- **Content-Type**, eine genormte Bezeichnung für die Art bzw. das Format der Übermittelten Daten,
- und einem Datenstrom, der die vom Client gewünschten Daten enthält.

Java API: Client

Download von einem HTTP-Server

```
// Create an URL object.
URL url = new URL("http://localhost:8080/Java2013git/MonopolySnapshot");
// Open the connection to the server located at the url
// Note the need to cast into a connection for HTTP!
HttpURLConnection con = (HttpURLConnection)url.openConnection();
// set the request method (GET is also the default)
con.setRequestMethod("GET");
// Access the response code
System.out.println("" + con.getResponseCode() + con.getResponseMessage());
// Get an input stream for the data the server is sending...
InputStream download = con.getInputStream();
// Read 50 bytes from the server
byte[] data = new byte[50];
download.read(data);
// Close the stream
download.close();
```

Java API: Client

Query an einen Server

Das Übermitteln von sehr einfachen und kurzen Daten an den Server kann direkt über den **Query String** geschehen:

```
String urlPrefix = "http://localhost:8080/helloworld/Hello";
// Encode the query, to allow characters not allowed in URLs
// (always use "UTF-8" as the second argument)
String query = URLEncoder.encode("What is the time?", "UTF-8");
// assertEquals("What+is+the+time%3F", query);
URL url = new URL(urlPrefix + "?" + query);
...
```

Java API: Client

Upload zu einem HTTP-Server

```
// Create an URL object and open the connection.
URL url = new URL("http://localhost:8080/Java2013git/MonopolySnapshot");
HttpURLConnection con = (HttpURLConnection)url.openConnection();
// set the request method to POST
con.setRequestMethod("POST");
// enable upload
con.setDoOutput(true);
// get the output stream to the server
OutputStream upload = con.getOutputStream();
// write data to upload to the stream and close it
upload.write(...);
upload.close();
// Access the response code; no further upload possible after this point
System.out.println("" + con.getResponseCode() + con.getResponseMessage());
```

Java API: Server

Komponenten

- Servlet Container (Web Server)
  - Übernimmt low-level Kommunikation
  - Implementierungen: Apache Tomcat, Jetty, ...
  - Typischerweise verwaltet durch System-Administrator
  - Für den Webprogrammierer: Testumgebung, z.B. mit Eclipse plugins [1]
- Servlet
  - Spezielle Klasse, die einzelne Anfragen behandelt.
  - Instanziierung und Aufruf der Anfrage-Methoden durch Servlet Container
  - Implementierung durch Webprogrammierer
    $\Longrightarrow$ Unser Fokus

[1] `http://proglang.informatik.uni-freiburg.de/teaching/java/2013/eclipse-jee.html`

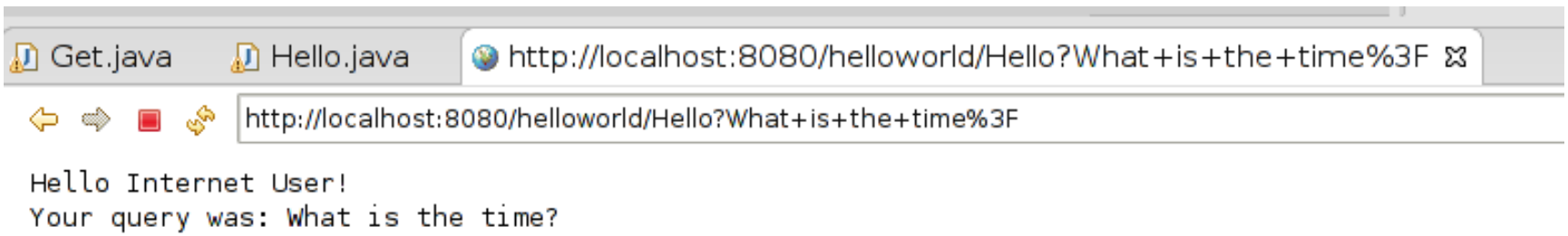
Java API: Server

Hello World Servlet

```
// Implement a servlet by extending HttpServlet.
// Specify the location on the server as an @WebServlet annotation
@WebServlet("/Hello")
public class Hello extends HttpServlet {
  // Handler for GET requests. Request and response are available from the parameters
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Get the query string from the request
    String q = request.getQueryString();
    if (q == null) {q = "<none>";} // default when no query given
    else { q = URLDecoder.decode(q, "UTF-8"); } // decode the request
    // set the repsonse code
    response.setStatus(HttpServletResponse.SC_OK);
    // indicate that the response is plain text
    response.setContentType("text/plain");
    // Transmit the content of the response with a java.io.PrintWriter
    PrintWriter w = response.getWriter();
    w.println("Hello Internet User!\n Your query was: " + q);
  }}
```

Java API: Server

Hello World Servlet in Action



Java API: Server

Uploads von Clients

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
         // Get an input stream
         InputStream s = request.getInputStream();
         // Prepare buffer (alternatively use java.io Classes) and read the data
         byte[] reqData = new byte[100];
         s.read(reqData);
         // Send a response
         response.getWriter().println("Got: " + Arrays.toString(reqData));
}
```

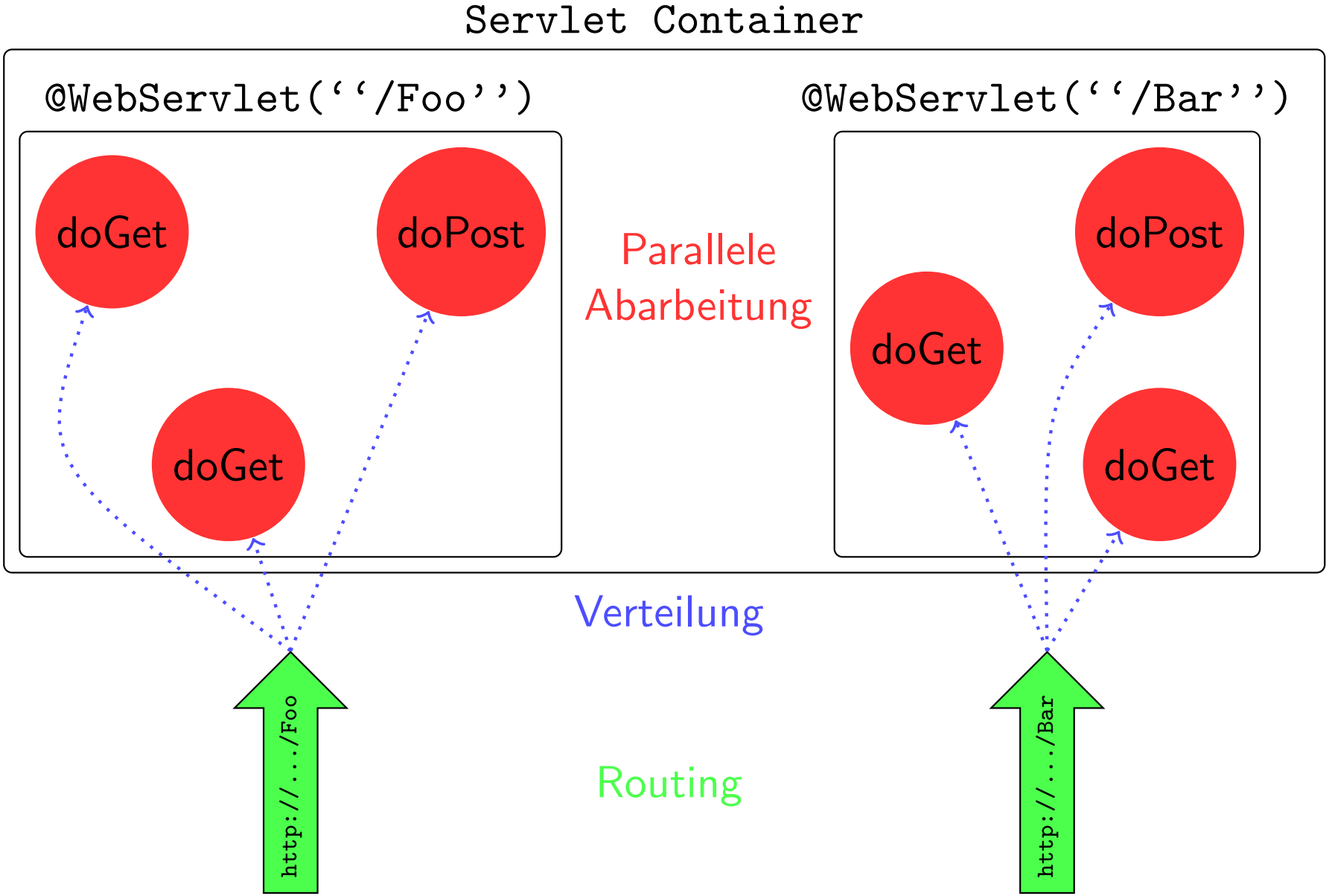
Achtung: für verlässlichen Betrieb müssen weitere Maßnahmen ergriffen werden, damit böswillige Clients den Server nicht blockieren können

- Verbindungszeit mit dem Client muss begrenzt werden
  (Anzahl der parallelen Verbindungen zum Server sind begrenzt.)
- Dies kann durch Konfiguration des Servers erreicht werden
  (nicht Teil dieser Vorlesung)

Request Verarbeitung

- **Routen** der Requests auf Servlets
- **Verteilung** der Requests auf ,,Worker-Threads”
- **Parallele Abarbeitung** der Requests

Request Verarbeitung



Request Verarbeitung

- **Routen** der Requests auf Servlets
- **Verteilung** der Requests auf ,,Worker-Threads"
- **Parallele Abarbeitung** der Requests:
  Daten, die über **mehrere Requests und/oder Servlets hinweg** gültig sein sollen, müssen speziell behandelt werden!
  $\Longrightarrow$ Der `ServletContext` kann solche Daten verwalten.

Servlet Context Beispiel

Gemeinsamer Zugriff auf einen String

```
@WebServlet("/Submit")
class Submit ... {
  public void doPost(...) {
    String req = request.getQueryString();
    ...
    // store the submitted data
    // in the shared context
    // – choose an arbitrary identifier:
    // "submit.data"
    // – remember that "submit.data"
    // holds a String
    this.getServletContext()
        .setAttribute("submit.data", req);
    ....
  }
}
```

```
@WebServlet("/Readout")
class Readout ... {
  public void doPost(...) {
    // retreive the currently submitted data
    // – we know it is a string,
    // and cast it accordingly
    String data =
     (String)this.getServletContext()
              .getAttribute("submit.data");
    if (data == null) { ... }
    ...
    response.getWriter().println(data);
  }
}
```

Nebenläufige Zustandsänderung

- `ServletContext` erlaubt nur ,,Zwischenlagern" einzelner Objekte.
- Request-Bearbeitung in Threads kann in beliebiger zeitlicher Vermischung passieren.
- Zustandsänderungen, die von gemeinsamen Daten abhängen, müssen zusätzlich geschützt werden.

Nebenläufige Zustandsänderung

Falsch:

```
Integer i = (Integer) this.getServletContext().getAttribute("requestCount");
this.getServletContext().setAttribute("requestCount", new Integer(i + 1));
```

- **Message Queue** (MQ) ordnet nebenläufige Anfragen sequentiell
- Arbeiterprozess *GameRunner* arbeitet Anfragen sequentiell ab
- Verteilt Antworten an entsprechende MQ der Spielers

Abarbeiten von Nebenläufigen Requests

Server Sicht