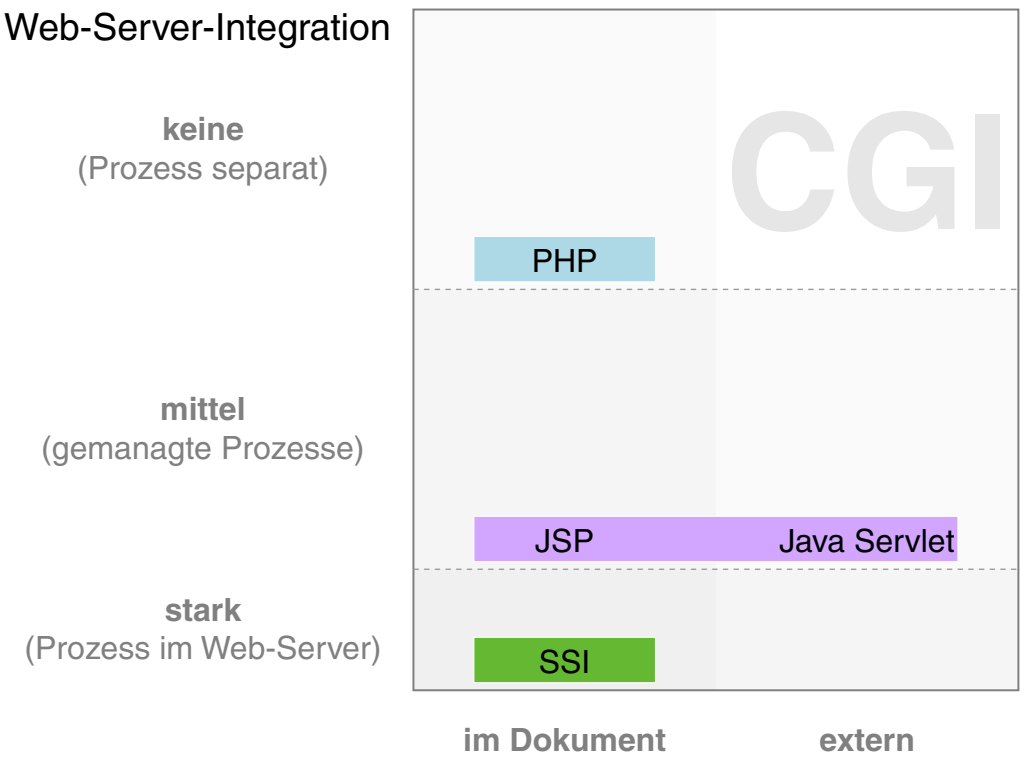
Kapitel WT:IV

IV. Server-Technologien

- Web-Server
- Common Gateway Interface CGI
- Java Servlet
- Java Server Pages JSP
- Active Server Pages ASP
- Exkurs: reguläre Ausdrücke
- PHP Hypertext Preprocessor
- Perl, Python, Ruby

Web-Server

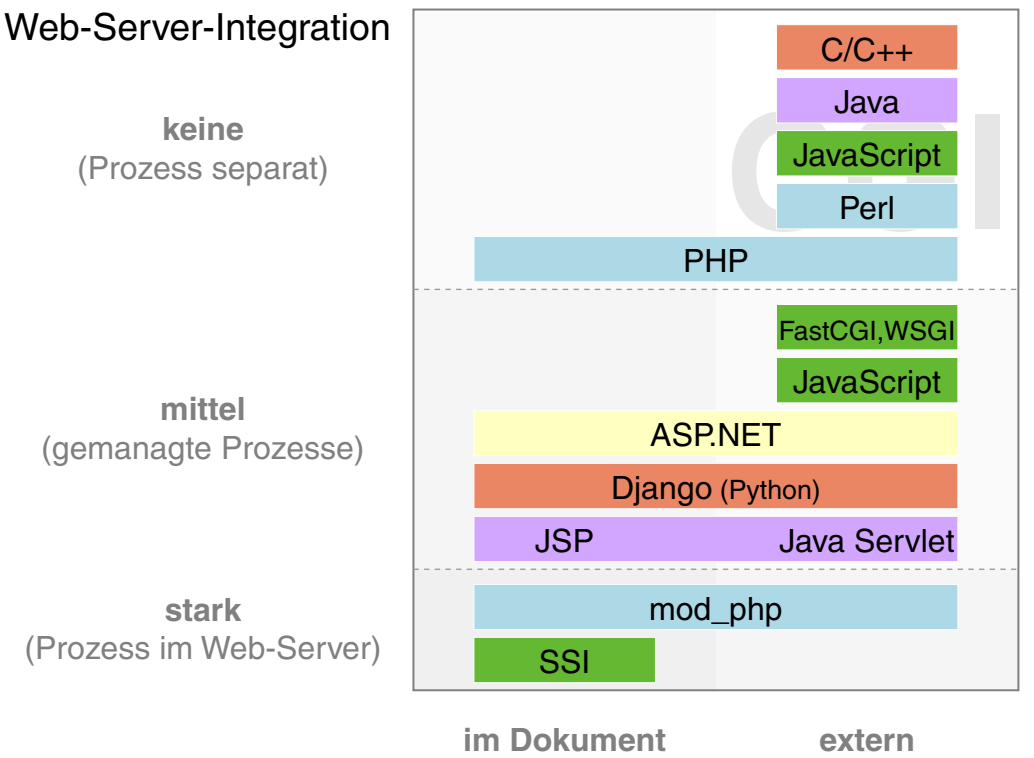
Einordnung von Server-Technologien [Stein 2012 - 2020]



- x-Achse: Wo befindet sich der Code zur Dokumenterzeugung?
- y-Achse: Wie stark ist die Technologie in den Web-Server integriert?

Web-Server

Einordnung von Server-Technologien [Stein 2012 - 2020]



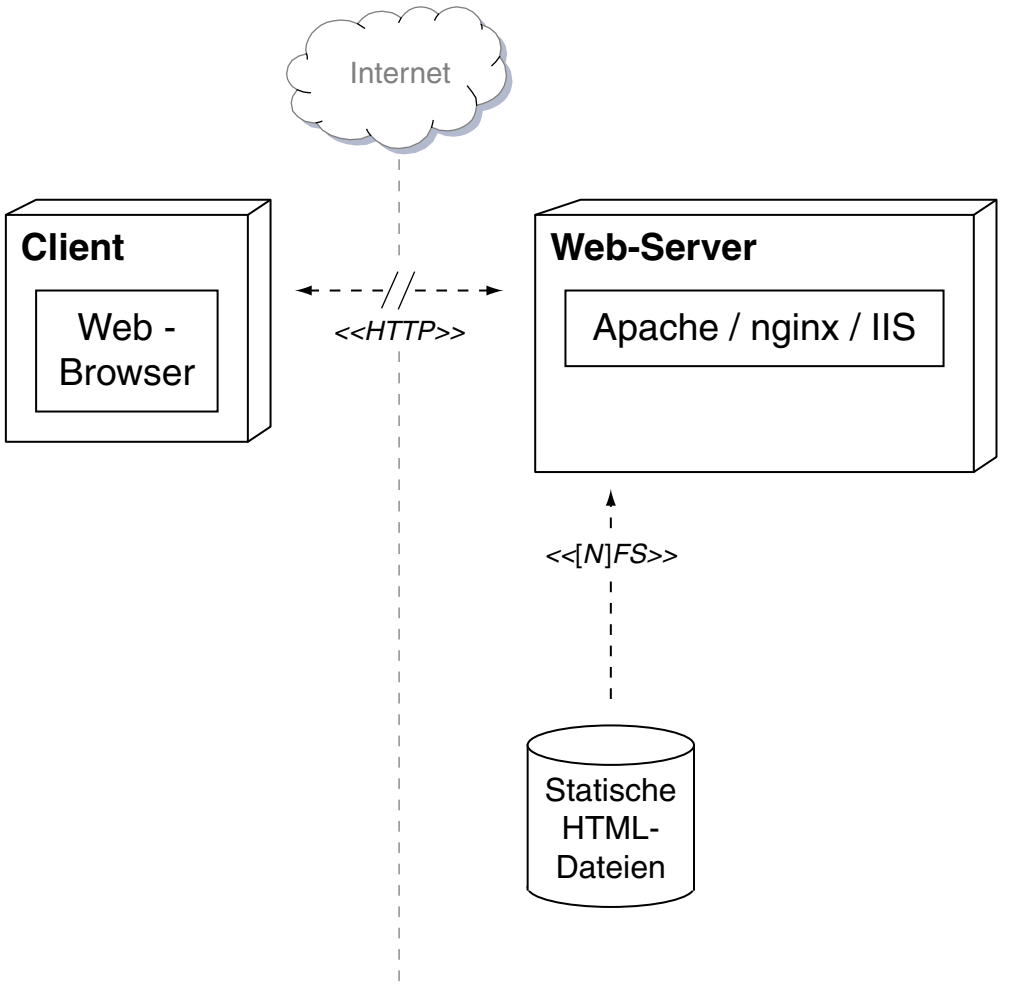
- x-Achse: Wo befindet sich der Code zur Dokumenterzeugung?
- y-Achse: Wie stark ist die Technologie in den Web-Server integriert?

Bemerkungen:

- Ein Web-Server ist ein Computersystem, das Anfragen verarbeitet, die gemäß des HTTP-Protokolls übermittelt werden. Der Begriff „Web-Server“ kann sich auf das gesamte Computersystem oder auf die Software beziehen, welche die HTTP-Anfragen beantwortet. Ein Web-Server wird auch als HTTP-Dämon bezeichnet; oft heißt das Programm, das den Web-Server implementiert, `httpd`.
- *Separate* Prozesse können mittels CGI angebunden und direkt durch das Betriebsystem ausgeführt werden. Vorteil: einfache Anbindung des Containers „Betriebssystem“ in Form einer Shell. Nachteil (u.a.): zeitaufwändiger Start eines Prozesses.
- Zur effizienten Verwaltung und Ausführung von Programmen mit Web-Funktionalität können spezialisierte Container / Application-Server persistente, *gemanagte* Prozesspools vorhalten. Beispiele:
  - FastCGI, mod_fcgid [apache.org]
  - JavaScript [nodejs.org]
  - JSP, Java Servlet [apache.org]
- Beispiele für Prozesse *im* Web-Server:
  - SSI (mod_include) [apache.org]
  - mod_php [apache.org]
  - mod_perl [apache.org]
- Überblick über Apache-Module [apache.org, Wikipedia]

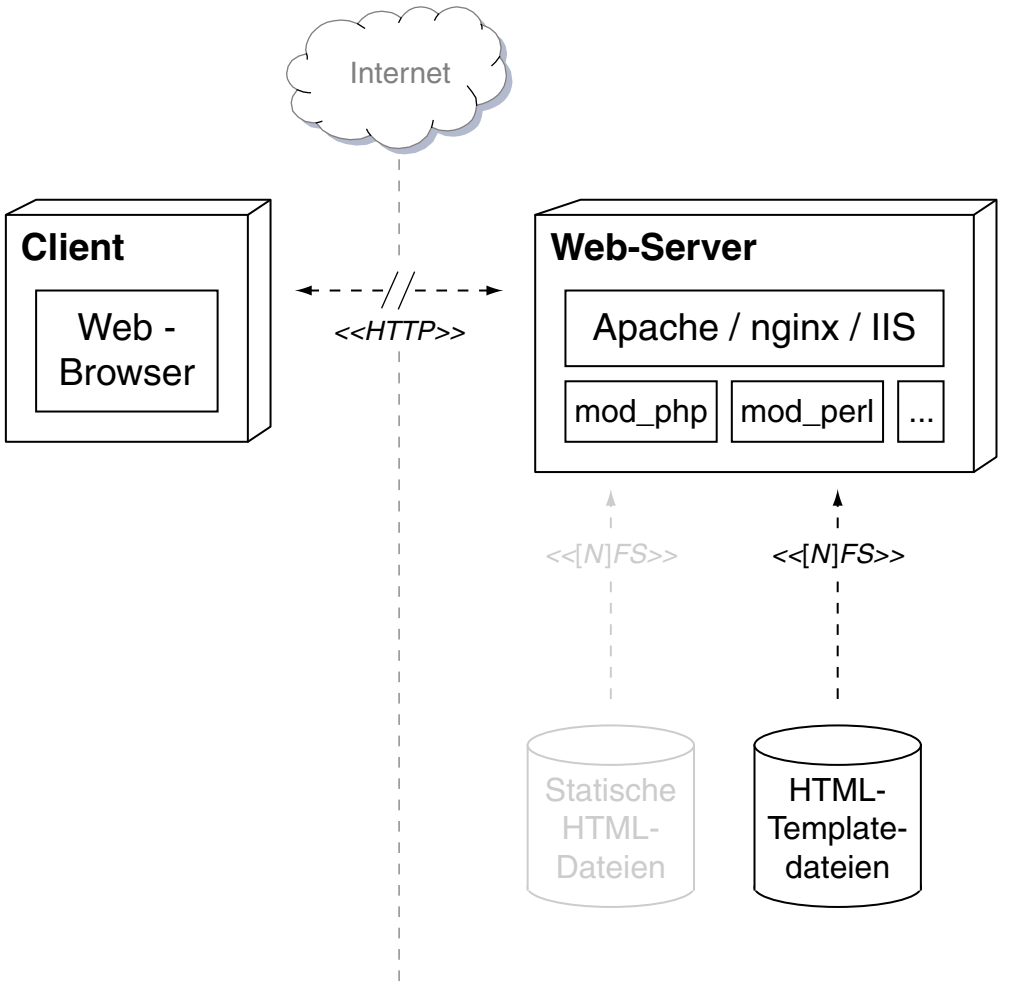
Web-Server

Deployment-Diagramm [Deployment DB-Server]



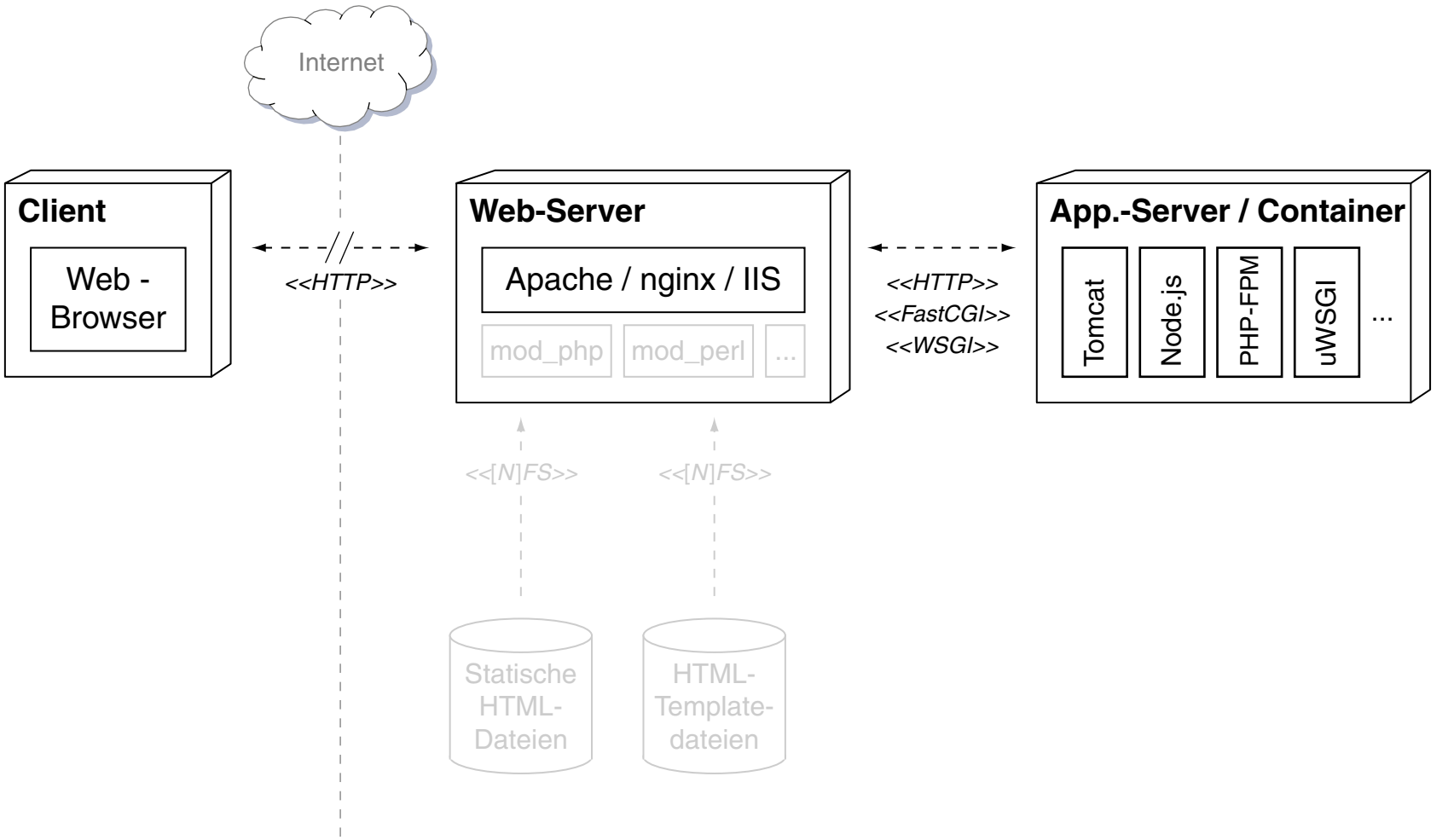
Web-Server

Deployment-Diagramm [Deployment DB-Server]



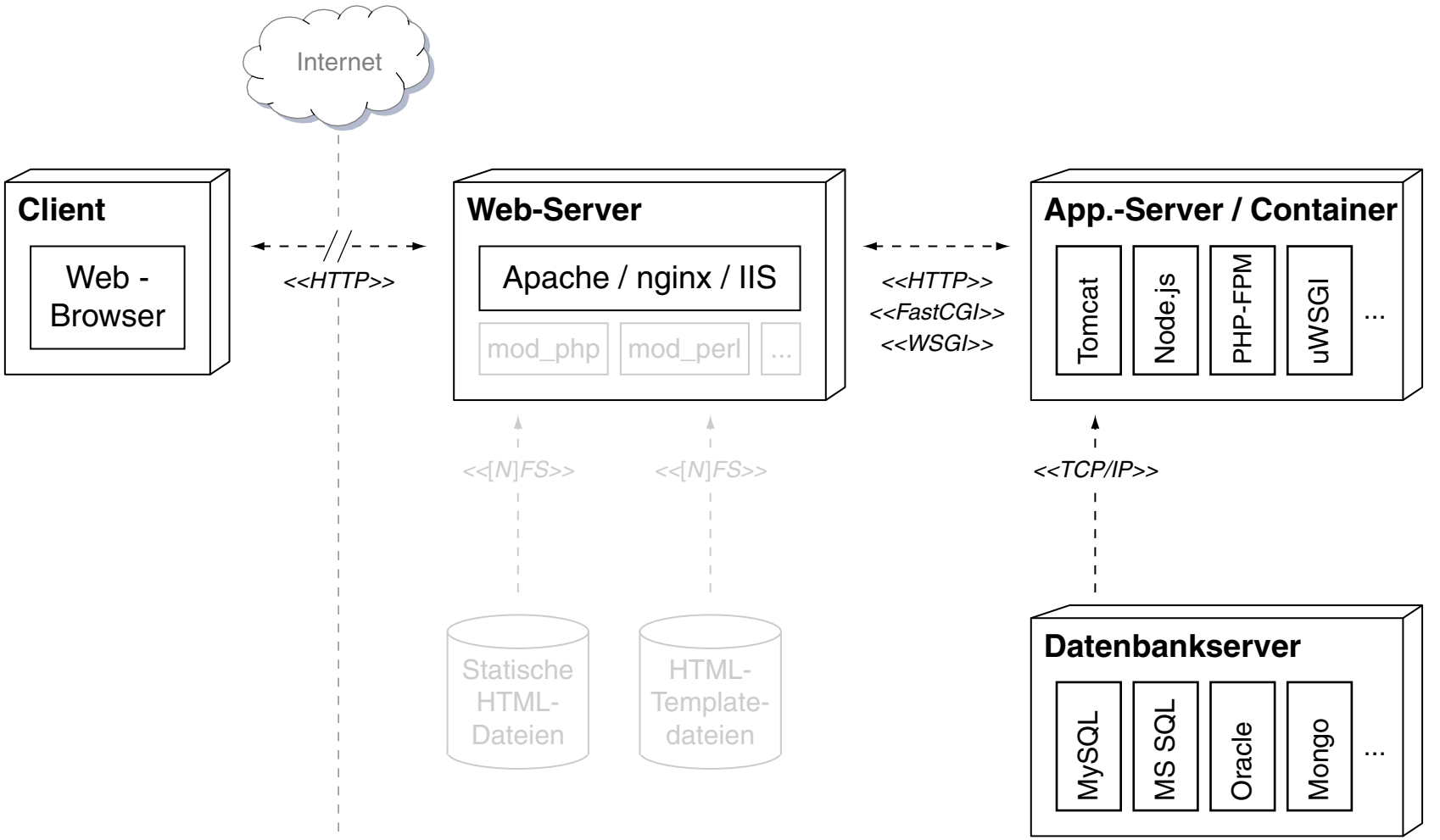
Web-Server

Deployment-Diagramm [Deployment DB-Server]



Web-Server

Deployment-Diagramm [Deployment DB-Server]



Web-Server

Wichtige Konfigurationseinstellungen

IP-Adresse, Hostname	Bei lokalem Betrieb die IP-Adresse 127.0.0.1 bzw. `localhost`.
Port	Üblicherweise lauscht der HTTP-Dämon auf Port 80, der HTTPS-Dämon auf Port 443 *(well-known Ports)*.
ServerRoot	Verzeichnis für Konfigurations-, Fehler-, und Log-Dateien.
DocumentRoot	Verzeichnis für statische HTML-Dateien.
Default-HTML-Dateiname	Spezifiziert die URL nur ein Verzeichns, wird nach einer Default-Datei gesucht. Üblich sind `index.html` oder `index.htm`.
CGI-Skripte	Physische und virtuelle Verzeichnisse für CGI-Skripte.
Log-Dateien	Protokollierung der Zugriffe (`access.log`) und Fehler (`error.log`)
Timeouts	Spezifiziert, wie lange der Web-Browser auf eine Antwort vom Server warten soll, und wie lange der Server versuchen soll, Daten an den Web-Browser zu schicken.
MIME-Typen	Dateiformate, die der Web-Server kennt.

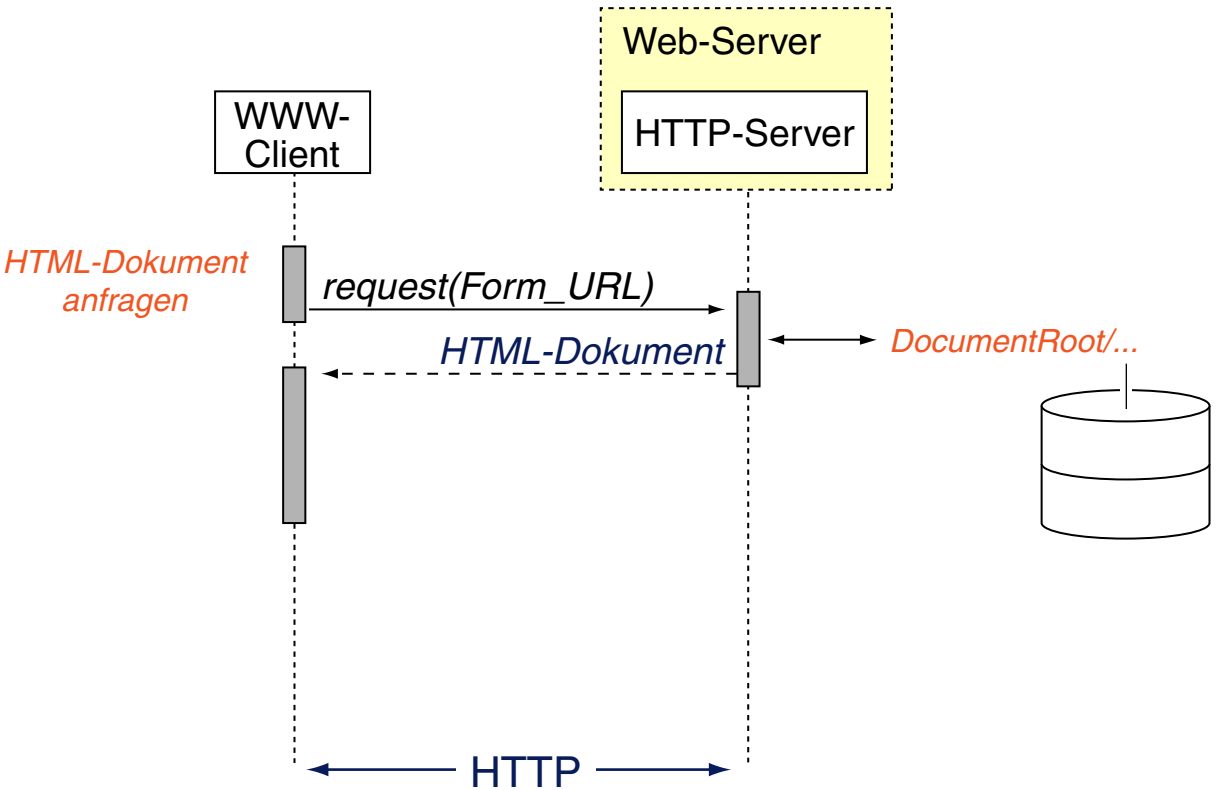
Web-Server

Wichtige Konfigurationseinstellungen

IP-Adresse, Hostname	Bei lokalem Betrieb die IP-Adresse 127.0.0.1 bzw. `localhost`.
Port	Üblicherweise lauscht der HTTP-Dämon auf Port 80, der HTTPS-Dämon auf Port 443 *(well-known Ports)*.
ServerRoot	Verzeichnis für Konfigurations-, Fehler-, und Log-Dateien.
DocumentRoot	Verzeichnis für statische HTML-Dateien.
Default-HTML-Dateiname	Spezifiziert die URL nur ein Verzeichns, wird nach einer Default-Datei gesucht. Üblich sind `index.html` oder `index.htm`.
CGI-Skripte	Physische und virtuelle Verzeichnisse für CGI-Skripte.
Log-Dateien	Protokollierung der Zugriffe (`access.log`) und Fehler (`error.log`)
Timeouts	Spezifiziert, wie lange der Web-Browser auf eine Antwort vom Server warten soll, und wie lange der Server versuchen soll, Daten an den Web-Browser zu schicken.
MIME-Typen	Dateiformate, die der Web-Server kennt.

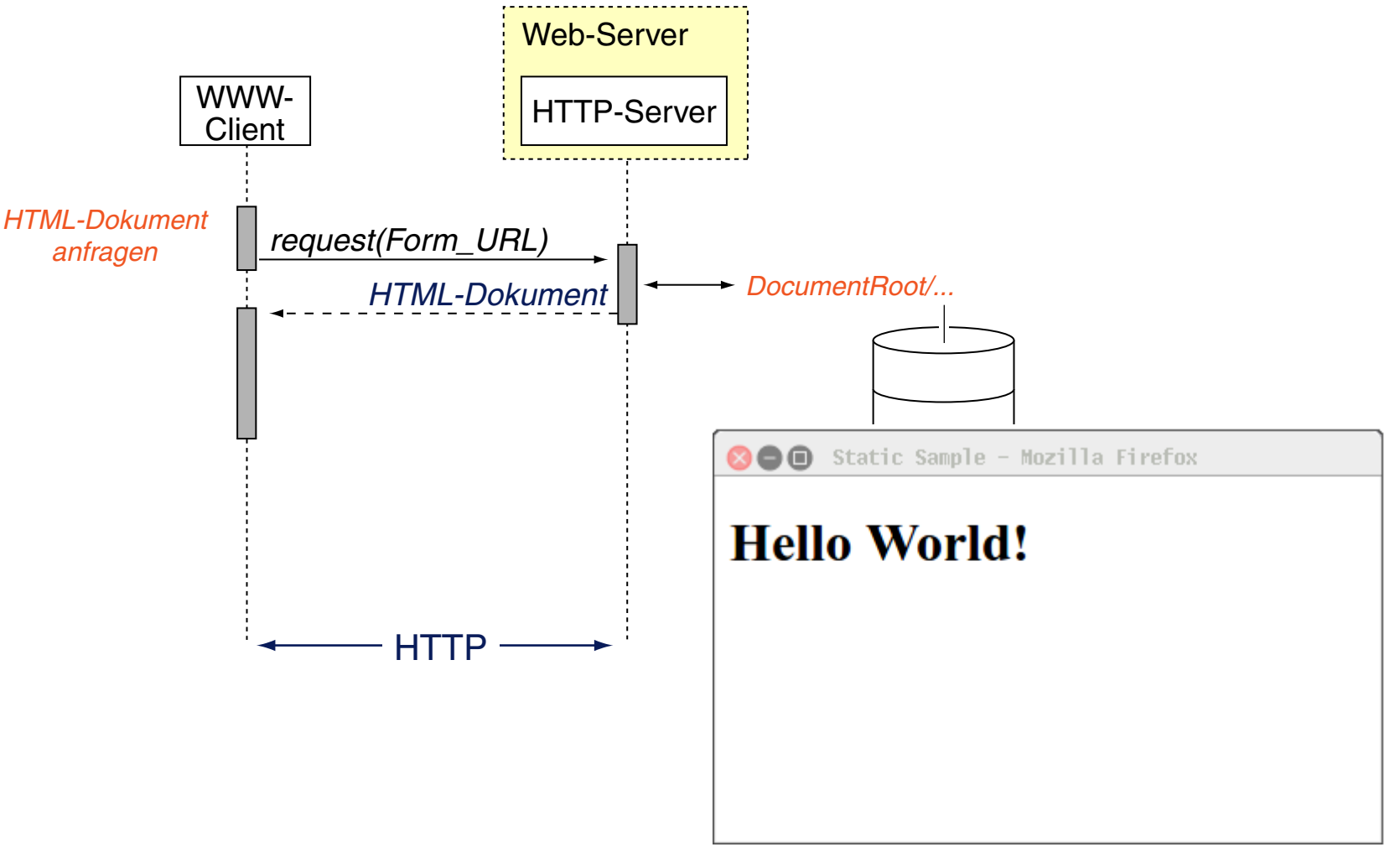
Web-Server

Ablauf einer statischen Seitenauslieferung [Vergleiche: CGI, Servlet, JSP]



Web-Server

Ablauf einer statischen Seitenauslieferung [Vergleiche: CGI, Servlet, JSP]



[Statisch: Source, Ausführung]

Web-Server

Apache HTTP-Server: Historie [Wikipedia]

1995 Version 0.6.2. Sammlung von Patches für den NCSA Web-Server (National Center for Supercomputing Applications) an der Universität von Illinois.

1998 Version 1.3. Grundstein für Apaches Erfolg. [apacheweek]

1999 Gründung der Apache Software Foundation. [apacheweek]

2002 Version 2.0. Modularisierung der Web-Server-Software. [apacheweek]

2005 Version 2.2. Unterstützung von Dateien > 2 GB, überarbeitete Authentifizierung, verbessertes Caching.

2012 Version 2.4. Deutlich performanter, geringerer Resourcenverbrauch.

2020 Aktuelle Version des Apache HTTP-Servers: [apache.org]

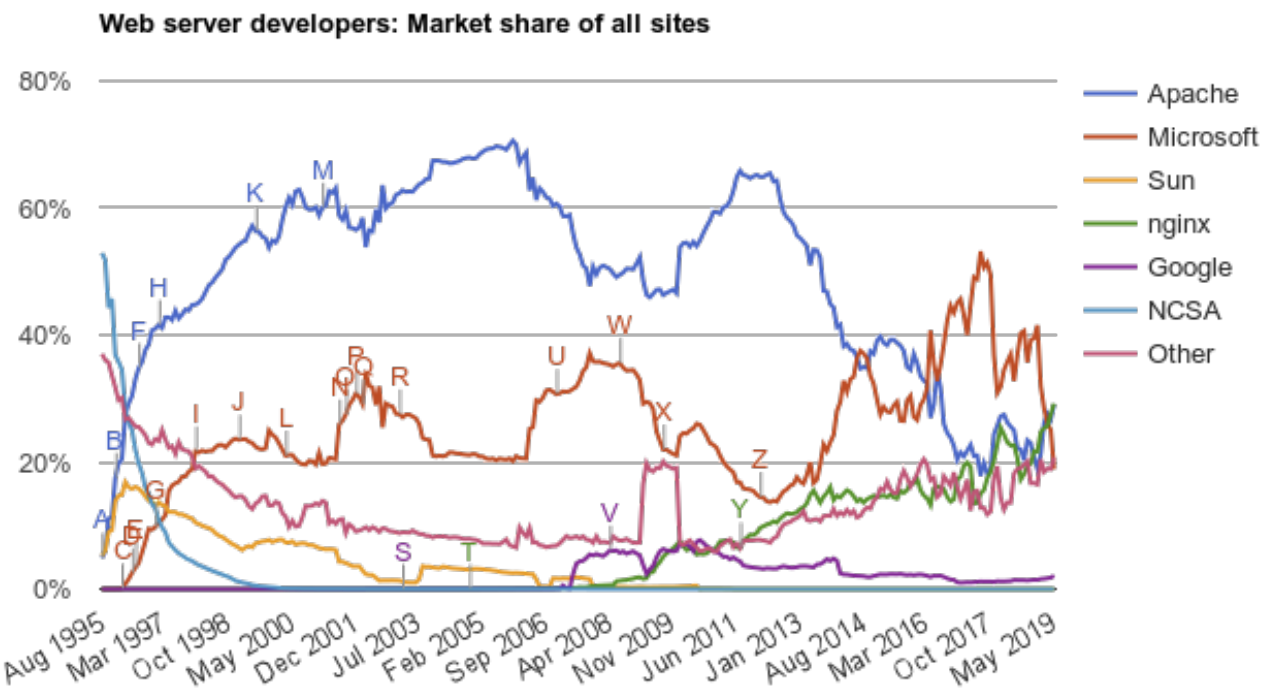
Web-Server

Apache HTTP-Server: Verbreitung

2019 1.326.664.693 Websites (= unique Hostnames) [internetlivestats: live statistics]

235.011.143 active Websites

8.726.985 Web-Servers (aka Web-Facing Computers)



[www.netcraft.com]

Bemerkungen:

- Der Marktanteil bezieht sich auf den Anteil an allen Websites. [netcraft.com]
- Zählen von Web-Servern: [netcraft.com]
- Dokumentation des Apache HTTP-Servers: [apache.org]
- Glossar mit Fachbegriffen im Zusammenhang mit dem Apache HTTP-Server und Web-Server im Allgemeinen: [apache.org]

Web-Server

Apache HTTP-Server: httpd.conf

Die zentrale Konfigurationsdatei `httpd.conf` bzw. `apache2.conf` liegt im Verzeichnis `/etc/httpd/` bzw. `/etc/apache2/`. Funktionsabschnitte:

1. Global Environment. Randbedingungen zur Arbeitsweise.
2. Main Server Configuration. Anweisungen zur Arbeitsweise.
3. Virtual Hosts. Einrichtung virtueller Hosts.

Web-Server

Apache HTTP-Server: httpd.conf

Die zentrale Konfigurationsdatei `httpd.conf` bzw. `apache2.conf` liegt im Verzeichnis `/etc/httpd/` bzw. `/etc/apache2/`. Funktionsabschnitte:

1. Global Environment. Randbedingungen zur Arbeitsweise.
2. Main Server Configuration. Anweisungen zur Arbeitsweise.
3. Virtual Hosts. Einrichtung virtueller Hosts.

Konfigurationsanweisungen *(Directives)* sind in sogenannten Containern gruppiert. Die Syntax ist XML-ähnlich, hat aber nichts damit zu tun.

Container	**Beschreibung**
`<IfDefine>`, `<IfModule>`	Bedingte Ausführung von Direktiven.

`<Directory>`, `<DirectoryMatch>`
`<Files>`, `<FilesMatch>`
`<Location>`, `<LocationMatch>`
	Spezifikation von Direktiven für Verzeichnisse, Dateien und URLs. Die `<Match>`-Variante ermöglicht die Angabe regulärer Ausdrücke.

Web-Server

Apache HTTP-Server: .htaccess

`.htaccess`-Dateien dienen zur Spezifikation von Zugriffen im Web-Space (Verzeichnisbaum unterhalb DocumentRoot) von Web-Servern. [apache.org]

Web-Server

Apache HTTP-Server: .htaccess

`.htaccess`-Dateien dienen zur Spezifikation von Zugriffen im Web-Space (Verzeichnisbaum unterhalb DocumentRoot) von Web-Servern. [apache.org]

Beispiel:

`.htaccess`-Datei:

```
# Kommentar
AuthType Basic
AuthName "Service"
AuthUserFile /usr/maintenance/web/.htusers
AuthGroupFile /usr/maintenance/web/.htgroups
Require user Alice Bob Eve
Require group Support
```

`.htusers`-Datei:

```
Alice:INY8m5KMwIc
Bob:69gY8YPjQXeN6
Eve:INw2mPEH.owe2
Frank:INh6DHvyejvf2
Greg:INboWuvjjwQ7E
```

Web-Server

Apache HTTP-Server: .htaccess

`.htaccess`-Dateien dienen zur Spezifikation von Zugriffen im Web-Space (Verzeichnisbaum unterhalb DocumentRoot) von Web-Servern. [apache.org]

Beispiel:

`.htaccess`-Datei:

```
# Kommentar
AuthType Basic
AuthName "Service"
AuthUserFile /usr/maintenance/web/.htusers
AuthGroupFile /usr/maintenance/web/.htgroups
Require user Alice Bob Eve
Require group Support
```

`.htusers`-Datei:

```
Alice:INY8m5KMwIc
Bob:69gY8YPjQXeN6
Eve:INw2mPEH.owe2
Frank:INh6DHvyejvf2
Greg:INboWuvjjwQ7E
```

Direktive	Beschreibung
`AuthType`	Art der Authentifizierung, üblich ist `Basic`: Benutzer mit Passworten sind in einer anzugebenden Datei.
`AuthUserFile`	absoluter Pfad zur Datei von autorisierten Benutzern mit Passwort
`Require {user | group}`	Liste der autorisierten Benutzer bzw. Gruppen

Bemerkungen:

- Die `.htaccess`-Dateien werden von Web-Servern ausgewertet, die zum NCSA-Server kompatibel sind.
- Das `.htaccess`-Konzept kann von dem Anwender, der Inhalte in dem Web-Space eines Web-Servers pflegt, eingesetzt werden. Aus Performanzgründen sollte grundsätzlich auf den Einsatz von `.htaccess`-Dateien verzichtet werden, falls die Möglichkeit besteht, Vorgaben in der `httpd.conf`-Datei machen zu können. [apache.org]
- Welche der globalen Vorgaben ein Anwender in der `.htaccess`-Datei überschreiben darf, wird mit der `AllowOverride`-Direktive festgelegt. [apache.org]
- Standardmäßig gelten die Angaben einer `.htaccess`-Datei für das Verzeichnis, in dem die Datei gespeichert ist, einschließlich aller Unterverzeichnisse.
- Es ist sinnvoll, die sensiblen Dateien mit der Passwortinformation außerhalb des Web-Space des Web-Servers zu speichern.
- `.htaccess` ermöglicht viele Direktiven zur Zugriffsspezifikation:
  1. Optionen zum Verzeichnis-Browsing
  2. Optionen zum automatischen Weiterleiten
  3. Formulierung eigener Regeln zur Reaktion auf HTTP-Fehlermeldungen
  4. bedingte Auslieferung von Inhalten; z.B. können Web-Seiten abhängig von der Landessprache des benutzten Web-Browsers geliefert werden
  5. Optionen zur Komprimierung von Daten vor deren Übertragung zum Browser

Web-Server

Server Side Includes SSI [Einordnung] [apache.org] [SELFHTML]

Server Side Includes, SSI, sind die einfachste Möglichkeit, um HTML-Dokumente Server-seitig dynamisch zu verändern.

- SSI-Anweisungen sind Teil der HTML-Datei, maskiert als Kommentar:

  `<!--#`*Anweisung Parameter = "Wert"* `-->`

- HTML-Dateien, die SSI-Anweisungen enthalten, sind mit einer speziellen Dateiendung gekennzeichnet: `shtml, shtm, sht`

Web-Server

Server Side Includes SSI [Einordnung] [apache.org] [SELFHTML]

Server Side Includes, SSI, sind die einfachste Möglichkeit, um HTML-Dokumente Server-seitig dynamisch zu verändern.

- SSI-Anweisungen sind Teil der HTML-Datei, maskiert als Kommentar:

  `<!--#`*Anweisung Parameter = "Wert"* `-->`

- HTML-Dateien, die SSI-Anweisungen enthalten, sind mit einer speziellen Dateiendung gekennzeichnet: `shtml, shtm, sht`

Beispiel:

```
<h3>Dynamisches HTML mit Server Side Includes</h3>
Datum/Uhrzeit auf dem Server:   <!--#config timefmt="%d.%m.%Y, %H.%M" -->
                                <!--#echo var="DATE_LOCAL" --> Uhr <br>
Name dieser HTML-Datei:         <!--#echo var="DOCUMENT_NAME" --> <br>
Installierte Server-Software:   <!--#echo var="SERVER_SOFTWARE" --> <br>
Aufrufender Web-Browser:        <!--#echo var="HTTP_USER_AGENT" -->
<h3>Weitere Informationen:</h3> <!--#exec cmd="free" -->
```

Web-Server

Server Side Includes SSI [Einordnung] [apache.org] [SELFHTML]

Server Side Includes, SSI, sind die einfachste Möglichkeit, um HTML-Dokumente Server-seitig dynamisch zu verändern.

- SSI-Anweisungen sind Teil der HTML-Datei, maskiert als Kommentar:

  `<!--#Anweisung Parameter = "Wert" -->`

- HTML-Dateien, die SSI-
  Dateiendung gekennzei

Beispiel:

```
<h3>Dynamisches HTML mit S
Datum/Uhrzeit auf dem Serv


Name dieser HTML-Datei:
Installierte Server-Softwa
Aufrufender Web-Browser:
<h3>Weitere Informationen:</h3> <!--#exec cmd="free" -->
```

[SSI: Source, Ausführung]

Web-Server

Server Side Includes SSI (Fortsetzung)

Anweisung	Parameter
`#config`	`errmsg="`*String*`", sizefmt="`*Formatstring*`", timefmt="`*Formatstring*`"`
`#echo`	`var="`*Name*`"` Es sind eigene, CGI-Umgebungsvariablen sowie folgende Variablen erlaubt: `DOCUMENT_NAME`: Name der HTML-Datei `DOCUMENT_URI`: Pfad der HTML-Datei `LAST_MODIFIED`: Zeitstempel der HTML-Datei `QUERY_STRING_UNESCAPED`: unmaskierter GET-Übergabestring `DATE_LOCAL`: Datum und Uhrzeit nach Server `DATE_GMT`: Datum und Uhrzeit nach Greenwich-Zeit
`#exec`	`cmd="`*Pfad/Programmdatei*`"` `cgi="`*CGI-Pfad/CGI-Skript*`"`
`#fsize`	`file="`*Pfad/Datei*`"` `virtual="`*Pfad/Datei*`"`
`#flastmod`	`file="`*Pfad/Datei*`"` `virtual="`*Pfad/Datei*`"`
`#include`	`file="`*Pfad/Datei*`"` `virtual="`*Pfad/Datei*`"`

Common Gateway Interface CGI

Prinzip: Ein Programm außerhalb des Web-Servers stellt einen Zugang *(Gateway)* zu geschützten, für den Web-Server nicht erreichbaren Daten bereit.

Der hierfür standardisierte Kommunikationsmechanismus heißt CGI. [Einordnung]

Common Gateway Interface CGI

Prinzip: Ein Programm außerhalb des Web-Servers stellt einen Zugang *(Gateway)* zu geschützten, für den Web-Server nicht erreichbaren Daten bereit.

Der hierfür standardisierte Kommunikationsmechanismus heißt CGI. [Einordnung]

Aufruf eines CGI-Skripts aus einer HTML-Datei mit Übergabe von Anwenderdaten:

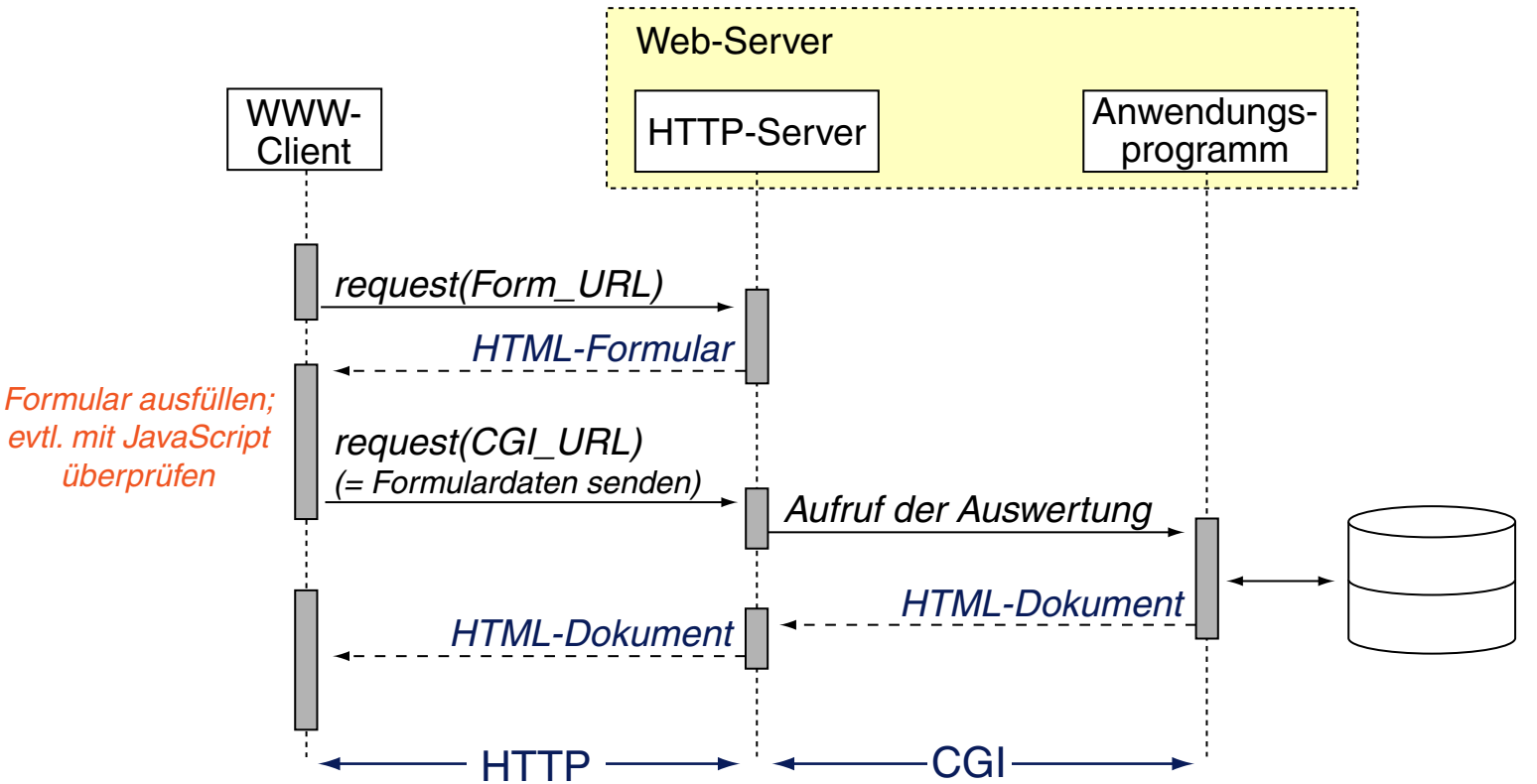
- Über ein Formular.
  ```
  <form action="/cgi-bin/sample.sh" method="get">
  ```

Typische Aufrufe aus einer HTML-Datei ohne Übergabe von Anwenderdaten:

- Über einen Verweis.
  ```
  <a href="/cgi-bin/statistik.py">Statistik</a>
  ```
- Über eine Grafikreferenz.
  ```
  <img src="/cgi-bin/counter.pl">
  ```
- Über eine Server Side Include Anweisung.
  ```
  <!-#exec cgi="/cgi-bin/counter.pl" ->
  ```
- Über ein automatisches Laden / Weiterleiten.
  ```
  <meta http-equiv="REFRESH" content="0; URL=/cgi-bin/welcome.sh">
  ```

Common Gateway Interface CGI

Ablauf einer Seitenauslieferung via CGI [Vergleiche: statisch, Servlet, JSP]



Bemerkungen:

- Anwendung von CGI: komplexe Berechnungen oder Datenverarbeitungsaufgaben, Anfragen und Updates bei Datenbanken.
- Jedes vom Betriebssystem (in einer Shell bzw. von einer Kommandozeile) ausführbare Programm kann als CGI-Programm dienen.
- Schritte bei der Kommunikation via CGI [RFC 3875] [Meinel/Sack 2004]:
  1. Der HTTP-Server erhält vom Client die Anforderung einer Informationsressource, die über ein (CGI-) Skript bzw. Programm bereitgestellt wird.
  2. Der HTTP-Server setzt auf Basis der Anforderung eine Reihe von Umgebungsvariablen.
  3. Der HTTP-Server startet das CGI-Skript bzw. das CGI-Programm.
     Geschieht der Aufruf via POST, so werden die Daten aus dem HTTP-Message-Body dem CGI-Skript über die Standardeingabe *(stdin)* zur Verfügung gestellt.
  4. Der HTTP-Server erhält die bei Ausführung des CGI-Skripts auf die Standardausgabe *(stdout)* geschriebenen Daten als Rückgabewert.
  5. Der HTTP-Server liefert die erhaltenen Daten an den Client aus.
- Die Ausgabe eines CGI-Skriptes enthält Entity- und Response-Header-Zeilen sowie den Body gemäß des HTTP-Protokolls. Die erste Zeile des Protokolls, die Status-Line, wird nicht vom CGI-Skript, sondern von dem Web-Server generiert; das CGI-Skript kann Angaben für den Status-Code generieren.

Common Gateway Interface CGI

Wichtige CGI-Umgebungsvariablen

Variable	Beschreibung
`CONTENT_LENGTH`	bei Aufruf via POST: Anzahl der übergebenen Zeichen
`CONTENT_TYPE`	bei Aufruf via POST: MIME-Typ der übergebenen Daten
`DOCUMENT_ROOT`	Pfad zu dem Web-Space des Web-Servers.
`HTTP_ACCEPT`	akzeptierte MIME-Typen des aufrufenden Browsers
`HTTP_ACCEPT_LANGUAGE`	Landessprache des aufrufenden Browsers
`HTTP_CONNECTION`	Informationen über den Status der HTTP-Verbindung
`HTTP_COOKIE`	Namen und Wert von Cookies, sofern vom Browser gesendet
`HTTP_HOST`	Domain-Namen bzw. IP-Adresse aus Adresszeile des Browsers
`HTTP_USER_AGENT`	Produkt- und Versionsinformationen zum Browser
`QUERY_STRING`	an die URL angehängte Daten, beginnend nach erstem „?“
`REQUEST_METHOD`	HTTP-Anfragemethode
`REQUEST_URI`	HTTP-Pfad des CGI-Skripts inklusive übergebenen Daten

Bemerkungen zur Codierung von Formulardaten:

- URL und Query sind durch ein „?“ voneinander getrennt.
- Formularvariablen einschließlich ihrer Daten sind durch „&“ voneinander getrennt.
- Name und Daten einer Formularvariablen sind durch „=“ voneinander getrennt.
- Leerzeichen in den eingegebenen Daten sind durch ein „+“ ersetzt.
- Zeichen mit ASCII-Werten zwischen 128 bis 255 sind durch hexadezimal codiert, eingeleitet durch ein Prozentzeichen. Beispiel: „%F6“ für „ö“

Common Gateway Interface CGI

Beispiel: Shell-Skript als CGI-Programm

In der HTML-Datei:

```
<a href="https://webtec.webis.de/cgi-bin/cgi-sample1.cgi">CGI-Aufruf</a>
```

Die Shell-Skript-Datei:

```
#!/bin/bash
echo "content-type: text/html"
echo ""   #Leerzeile gemäß HTTP-Protokoll.
echo "<!DOCTYPE html>"
echo "<html>"
echo "<head>"
echo "<meta http-equiv=\"content-type\" content=\"text/html; ...\">"
echo "<title>cgi-sample1</title>"
echo "</head>"
echo "<body>"
echo "<h3>Werte einiger CGI-Variablen</h3>"
echo "Installierte Server-Software: " $SERVER_SOFTWARE "<br>"
echo "Aufrufender Web-Browser: " $HTTP_USER_AGENT "<br>"
echo "Anfragemethode: " $REQUEST_METHOD "<br>"
echo "Query-String: " $QUERY_STRING "<br>"
echo "</body>"
echo "</html>"
```

Common Gateway Interface CGI

Beispiel: Shell-Skript als CGI-Programm

In der HTML-Datei:

```
<a href="https://webtec.webis.de/cgi-bin/cgi-sample1.cgi">CGI-Aufruf</a>
```

Die Shell-Skript-Datei:

```
#!/bin/bash
echo "content-type: text/html"
echo ""   #Leerzeile gemäß HTTP-Protokoll.
echo "<!DOCTYPE html>"
echo "<html>"
echo "<head>"
echo "<meta http-equiv=\"content-type\" content=\"text/html; ...\">"
echo "<title>cgi-sample1</title>"
echo "</head>"
echo "<body>"
echo "<h3>Werte einige
echo "Installierte Ser
echo "Aufrufender Web-
echo "Anfragemethode:
echo "Query-String: "
echo "</body>"
echo "</html>"
```

[CGI: Source, Ausführung]

Java Servlet

Einführung [Einordnung]

“A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model.”

[Oracle 1, 2]

Java Servlet

Einführung [Einordnung]

> *"A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model."*
>
> [Oracle 1, 2]

- Java Servlets können jede Art von Anfrage beantworten; ihr Einsatz geschieht hauptsächlich im Zusammenhang mit Web-Servern.
- Die Java Servlet API besteht aus den `javax.servlet`-Packages. [Javadoc] Diese gehören zu Java EE (nicht zu Java SE) bzw. seit 2019 zu Jakarta EE.
- Im Mittelpunkt der Servlet-Programmierung stehen:

Klasse bzw. Interface	**Konzepte**
`HttpServlet`	`service()`, `doGet()`, `doPost()`
`HttpServletRequest`	`get...()`-Methoden für CGI-Environment
`HttpServletResponse`	Streams als Ausgabekanal zum Client

Bemerkungen:

- Servlets werden in einem Servlet-Container verwaltet, der u.a. für die persistente Speicherung der Zustände und die Verfügbarkeit der Servlets zuständig ist. Der Servlet-Container gehört zur Vermittlungsschicht zwischen (Web-)Client und (Web-)Server und zählt somit zur Middleware.
- Der Servlet-Container kann entweder direkt oder über einen vorgeschalteten Web-Server angefragt werden. Im letzteren Fall wird eine Schnittstelle benögt, die zwischen denjenigen URLs unterscheidet, die der Web-Server bzw. der Servlet-Container bedienen soll. Im Apache-Web-Server stehen zur Realisierung einer solchen Schnittstelle die alternativen Module “mod_jk” und “mod_proxy” zur Verfügung. [apache.org]
- In der „klassischen“ Konfiguration ist der Servlet-Container ein Service, der zusätzlich oder anstelle eines Web-Servers installiert wird und mittels dem Servlets deployed werden. Alternativ kann eine Servlet-basierte Web-Anwendungen ihren eigenen Servlet-Container zur Installation mit bringen. Stichwort: *Embedded Servlet Container* [Tomcat, Jetty]

  Embedded Servlet-Container rücken die Container-Semantik in den Hintergrund (der Container dient nur *einer* Anwendung). Sie sind eine elegante Möglichkeit, eine Java-Anwendung verteilt – mit allen Vorteilen von standardisierten Web-Technologien – zur Verfügung zu stellen: weltweiter Zugriff via URL und HTTP, leistungsfähige Benutzeroberfläche via Browser, etc.

Java Servlet

Servlet-Lebenszyklus

Der Lebenszyklus eines Servlets wird von dem Container gesteuert, der das Servlet verwaltet.

Ein HTTP-Servlet wird durch einen HTTP-Request über eine URL angesprochen. Dann führt der Container folgende Schritte aus:

1. Überprüfung, ob eine Servlet-Instanz läuft. Falls nicht, wird
   (a) die Servlet-Klasse geladen,
   (b) eine Instanz der Servlet-Klasse erzeugt und
   (c) die Servlet-Instanz durch Aufruf der `init()`-Methode initialisiert.

2. Erzeugung eines `HttpServletRequest`-Objektes und eines `HttpServletResponse`-Objektes.

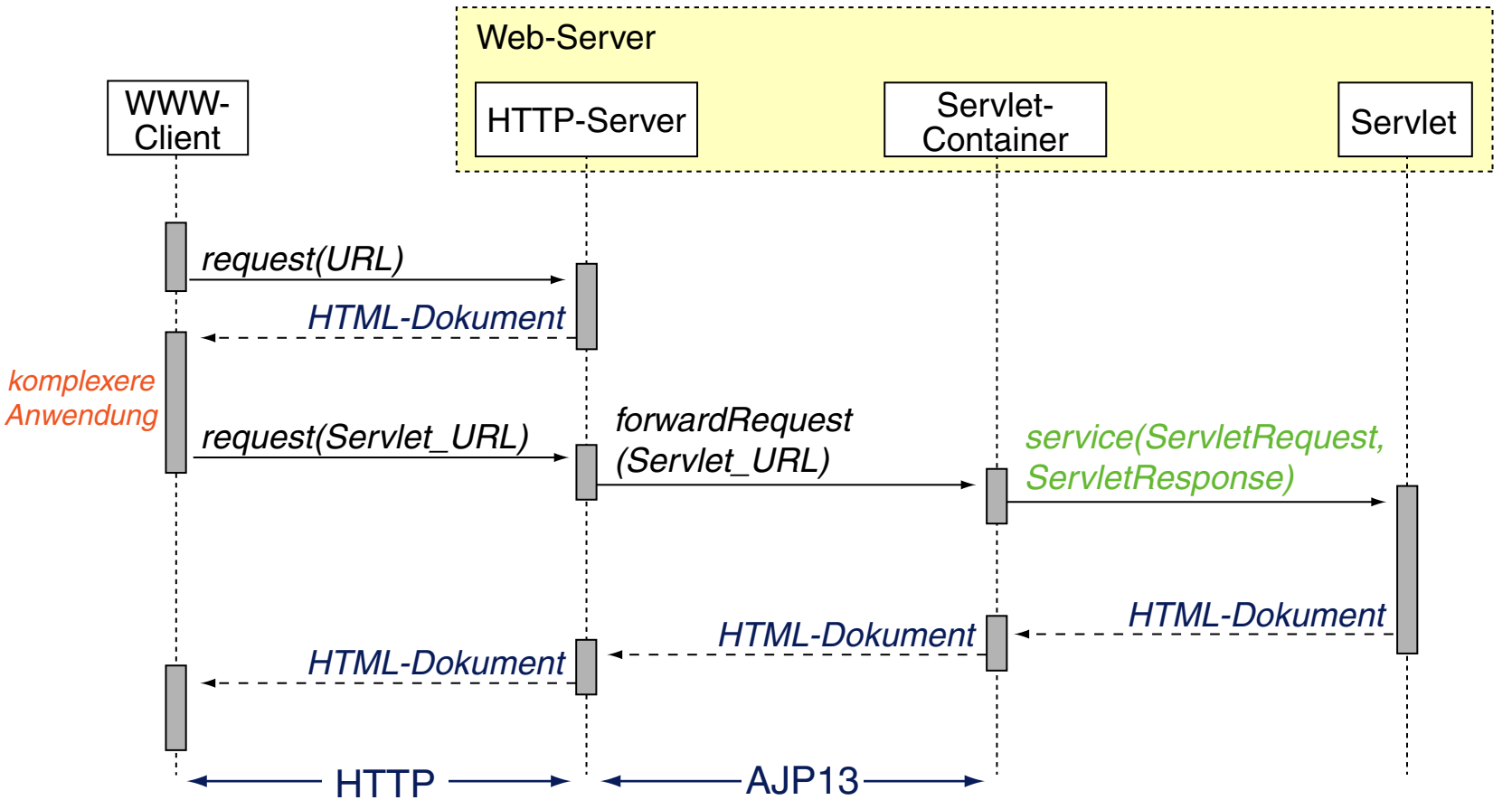
3. Aufruf der `service()`-Methode der Servlet-Instanz. Sie analysiert die Anfrage des Web-Servers und dispatched entsprechend; das Request-Objekt und das Response-Objekt werden mit übergeben.

Bemerkungen:

- Die `service()`-Methode erkennt die HTTP-Anfragen GET, POST, HEAD, PUT, DELETE, OPTIONS bzw. TRACE und ruft die entsprechenden Java-Methoden `doGet()`, `doPost()`, etc. auf.
- Falls der Container eine Servlet-Instanz entladen soll, ruft er dessen `destroy()`-Methode auf.

Java Servlet

Ablauf einer Seitenauslieferung via Servlet [Vergleiche: statisch, CGI, JSP]



Java Servlet

Charakteristika der Servlet-Technologie

- Portabilität
- Leistungsfähigkeit
- Effizienz
- Sicherheit
- Produktivität

Java Servlet

Charakteristika der Servlet-Technologie

- Portabilität

  Servlets sind über Betriebssysteme und Web-Server hinweg portabel.

- Leistungsfähigkeit

  Alle Konzepte von Java (Multithreading, Serialisierung, etc.) stehen zur Verfügung, einschließlich der gesamten Java-Bibliothek.

- Effizienz

  Eine Servlet-Instanz (ein Java-Objekt) wird nur einmal geladen und bleibt – mit seinem Zustand – im Speicher des Web-Servers.

- Sicherheit

  Java selbst ist sehr robust; zum Schutz des Web-Servers existieren darüberhinaus die Sicherheitsmechanismen des Java Security Managers. Stichwort: Servlet-Sandbox

- Produktivität

  Konzepte wie *Session Tracking, Cookie Handling* etc. erleichtern die Anwendungsentwicklung.

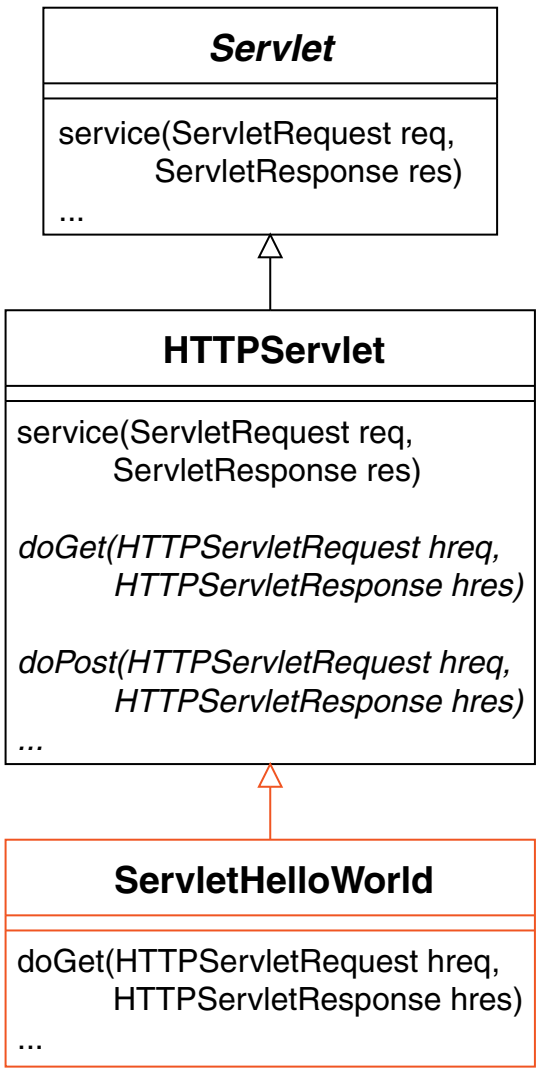
Java Servlet

Beispiel: HelloWorld [Servlet-Ausführung] [Source: ServletReadURLParam]

```
package servlet;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletHelloWorld extends HttpServlet {

    public void init() throws ServletException {
        // Nothing to do here.
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Use "response" to specify the HTTP response line and headers
        // (e.g. specifying the content type, setting cookies).
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser.
        out.print("<!DOCTYPE html>"
                + "<html><head><title>Hello World</title></head>"
                + "<body><h1>Hello World!</h1></body></html>");
    }
}
```

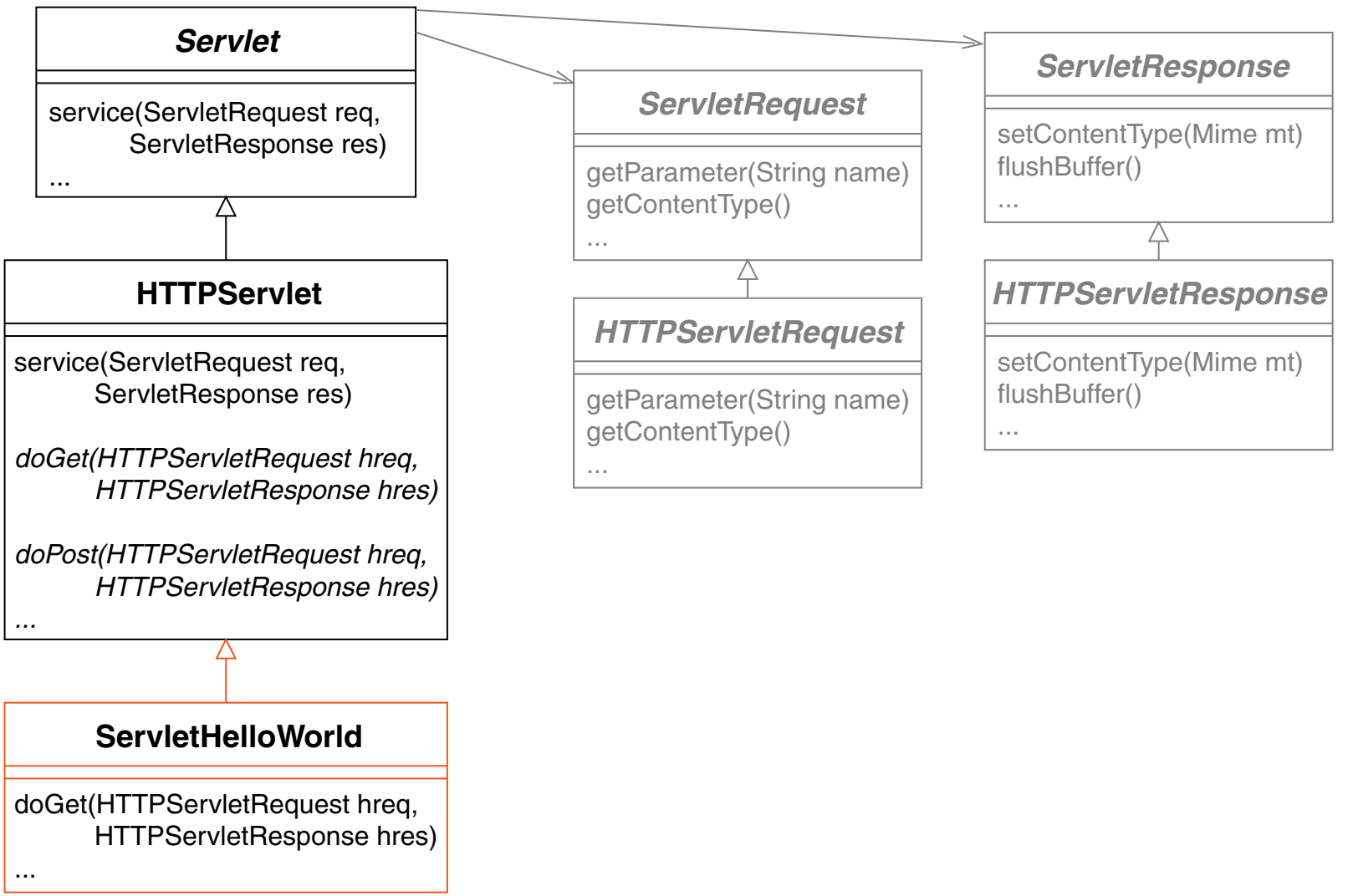
Java Servlet

Implementierung folgt dem „Template Method“-Pattern



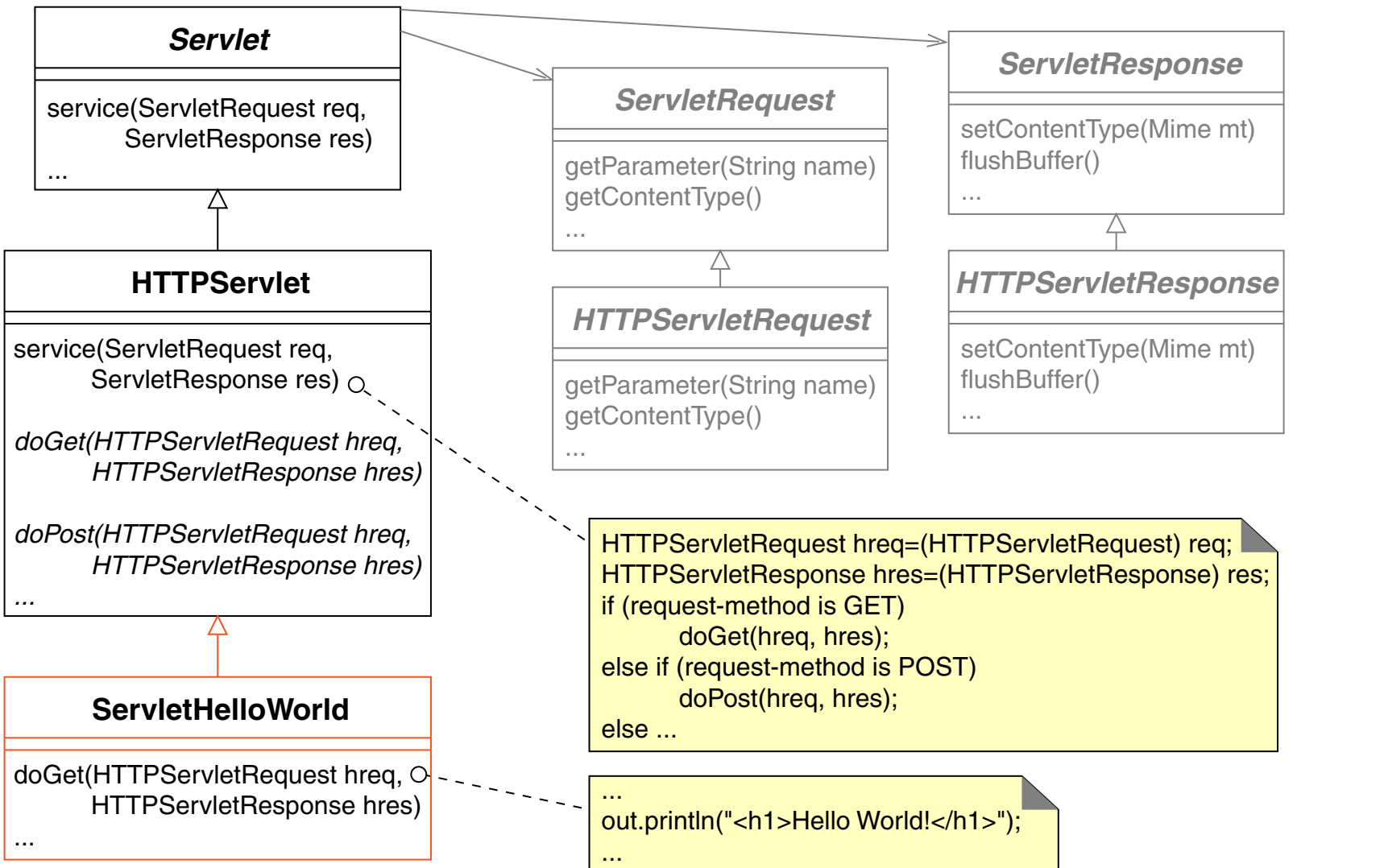
Java Servlet

Implementierung folgt dem „Template Method“-Pattern



Java Servlet

Implementierung folgt dem „Template Method“-Pattern



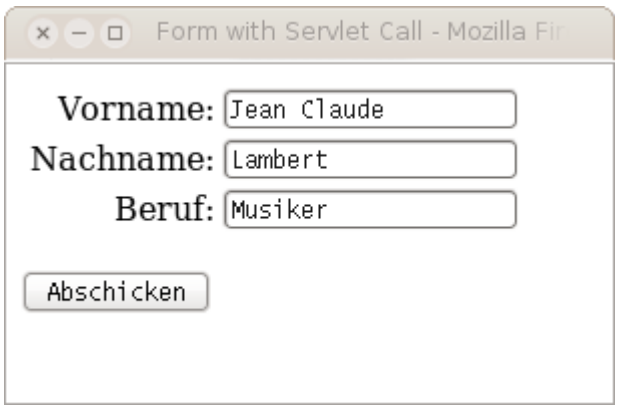
Java Servlet

Beispiel: URL-Parameter einlesen

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Form with Servlet Call</title>
  </head>
  <body>
    <form action="https://.../ReadURLParam" method="GET">
      <table border="0" cellpadding="0" cellspacing="4">
        <tr>
          <td align="right">Vorname:</td>
          <td><input name="vorname" type="text" size="20" ...></td>
        </tr>
        ...
      </table>
      <p>
        <input type="submit" name="z" value="Abschicken"> <br>
      </p>
    </form>
  </body>
</html>
```

Java Servlet

Beispiel: URL-Parameter einlesen (Fortsetzung)



[Servlet: Source, Ausführung]

Java Servlet

Beispiel: URL-Parameter einlesen (Fortsetzung) [Source: ServletHelloWorld]

```
package servlet;

import java.io.*; ...

public class ServletReadURLParam extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // Use "request" to read incoming HTTP headers (e.g. cookies)
        // and HTML form data (e.g. data the user entered and submitted)
        out.println("<!DOCTYPE html> <html>"
             + "<head><title>URL Parameter</title></head>"
             + "<body><h3>Einlesen von URL Parametern</h3>"
             + "<ul> <li>Vorname: " + request.getParameter("vorname")
             + "</li><li>Nachname: " + request.getParameter("nachname")
             + "</li><li>Beruf: " + request.getParameter("beruf")
             + "</li></ul> </body></html>");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Java Servlet

Deployment

Die Bereitstellung (*Deployment*) Servlet-basierter Web-Anwendungen ist standardisiert; der Java Community Process hat hierfür das `war`-Archivformat spezifiziert. Verzeichnisstruktur:

DocumentRoot/Context-Path.`war`

~

DocumentRoot/Context-Path/ `WEB-INF/classes/servlet`

Java Servlet

Deployment

Die Bereitstellung (*Deployment*) Servlet-basierter Web-Anwendungen ist standardisiert; der Java Community Process hat hierfür das `war`-Archivformat spezifiziert. Verzeichnisstruktur:

DocumentRoot/Context-Path.`war`

~

DocumentRoot/Context-Path/ `WEB-INF/classes/servlet`

- `web.xml`
- *Servlet1*`.class`
- *Servlet2*`.class`
- ...

Mit obiger Verzeichnisstruktur geschieht der *Servlet*-Aufruf über einen HTTP-Server mit Servlet-Container wie folgt:

`https://`*Web-Server/Context-Path/*`servlets/`*Servlet*

[Servlet-Ausführung] [Deployment] [mod_proxy: Servlet-Interaktion, apache.org]

Bemerkungen:

- Die Datei `web.xml` ist der Deployment-Descriptor und enthält die URL-Mappings zu den Servlets. Folgende web.xml-Datei beschreibt das `HelloWorld`-Servlet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.oracle.com/webfolder/.../web-app_3_1.xsd"
    version="3.1">
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>servlet.ServletHelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```

  Schema-Datei für `<web-app>`-Elemente.

- Die `HelloWorld`-Anwendung wurde in einem `war`-Archiv mit dem speziellen Namen `ROOT.war` bereitgestellt. Dadurch entfällt der *Context-Path* in der URL.

- Das Servlet-Deployment kann im laufenden Betrieb eines Web-Servers erfolgen.

Java Server Pages JSP

Java Server Pages JSP

Einführung [Einordnung]

Ziel: Pflege von statischen und dynamischen Inhalten in *einer* Datei.

Charakteristika, Technologie:

- dokumentenzentrierte Programmierung – direkt über den Web-Client
- Einbettung von Java-Code in HTML
- Java für dynamische, HTML für statische Dokumentbestandteile
- automatische Code-Generierung und Code-Verwaltung mit Servlets
- Zugriff auf und Verwendung des JavaBeans-Komponentenmodells

Anwendung:

- mittelgroße bis sehr große Projekte, einfache bis komplexe Probleme
- bei Forderung nach Portabilität, skalierbarer Architekur, hoher Performanz

Java Server Pages JSP

Einführung (Fortsetzung)

Java-Code wird in den HTML-Code einer `jsp`-Datei eingebettet:

```
<!DOCTYPE html>
<html>
  <head> <title>JSP Sample</title> </head>
  <body>
    <h1> <% out.print("Hello World!"); %> </h1>
  </body>
</html>
```

Java Server Pages JSP

Einführung (Fortsetzung)

Java-Code wird in den HTML-Code einer `jsp`-Datei eingebettet:

```
<!DOCTYPE html>
<html>
  <head> <title>JSP Sample</title> </head>
  <body>
    <h1> <% out.print("Hello World!"); %> </h1>
  </body>
</html>
```

[JSP: Source, Ausführung] [Servlet: generiert, manuell]

Java Server Pages JSP

Einführung (Fortsetzung) [generierte Web-Seite] [generiertes Servlet]

```
<!DOCTYPE html>
<html>
  <head> <title>Registration</title> </head>
  <body>




    <h3>Anmeldung</h3>
    <form action="registration.jsp" method="get">
      <table>
        <tr><td>Benutzername:</td>
            <td><input type="text" name="user"></td></tr>
        <tr><td><input type="submit" value="Anmelden"></td></tr>
      </table>
    </form>




  </body>
</html>
```

Java Server Pages JSP

Einführung (Fortsetzung) [generierte Web-Seite] [generiertes Servlet]

```
<!DOCTYPE html>
<html>
  <head> <title>Registration</title> </head>
  <body>

    <h3>Ihre Anmeldedaten:</h3>
    Benutzername:

  </body>
</html>
```

Java Server Pages JSP

Einführung (Fortsetzung) [generierte Web-Seite] [generiertes Servlet]

```
<!DOCTYPE html>
<html>
  <head> <title>Registration</title> </head>
  <body>
    <%
      String user = request.getParameter("user");
      if ( user == null || "".equals(user) ) {
    %>
    <h3>Anmeldung</h3>
    <form action="registration.jsp" method="get">
      <table>
        <tr><td>Benutzername:</td>
            <td><input type="text" name="user"></td></tr>
        <tr><td><input type="submit" value="Anmelden"></td></tr>
      </table>
    </form>
    <% } else { %>
    <h3>Ihre Anmeldedaten:</h3>
    Benutzername: <%= user %>
    <% } %>
  </body>
</html>
```

Vergleiche hierzu die PHP-Realisierung.

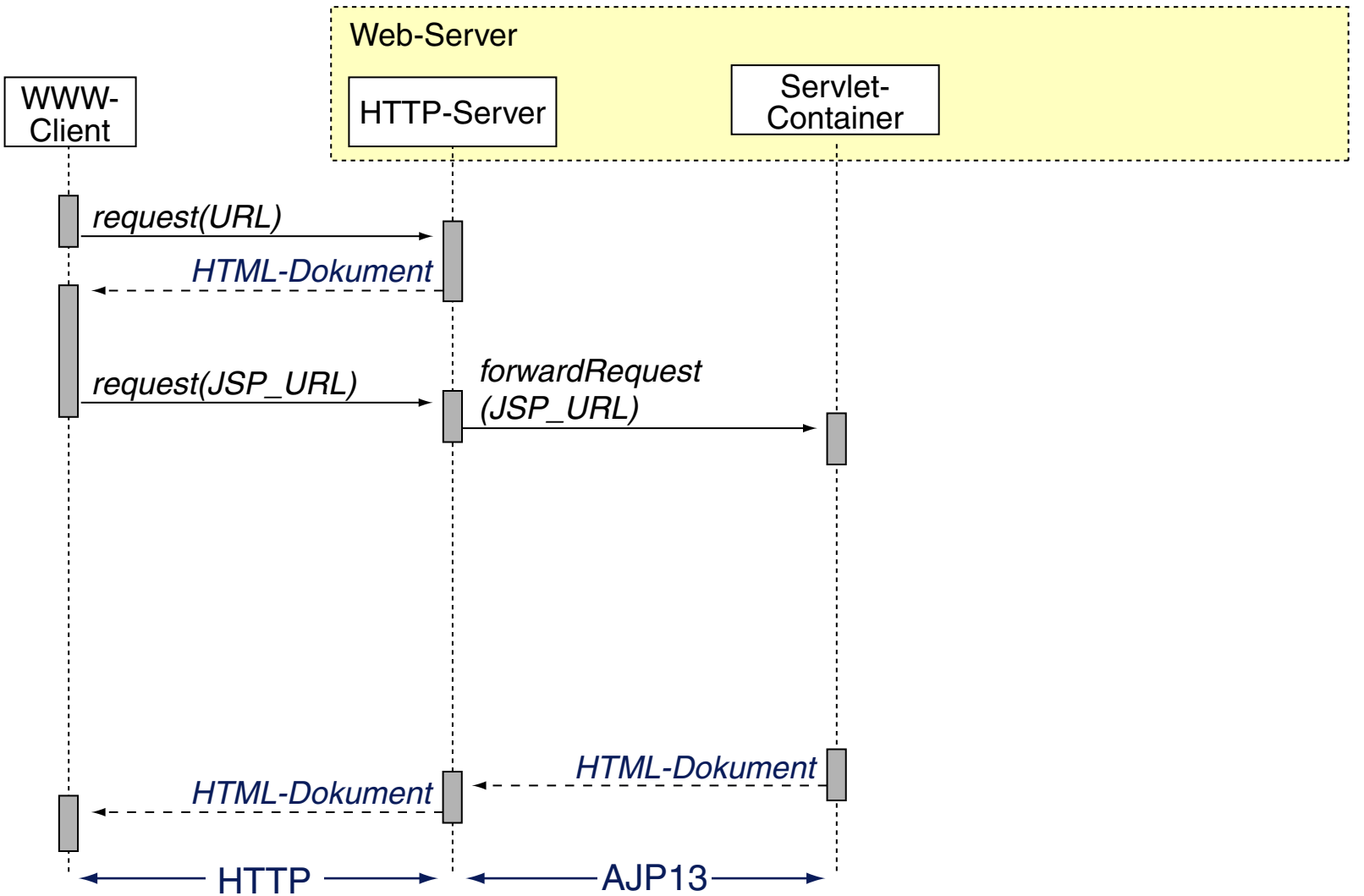
Java Server Pages JSP

Einführung (Fortsetzung)

[JSP: Source, Ausführung]

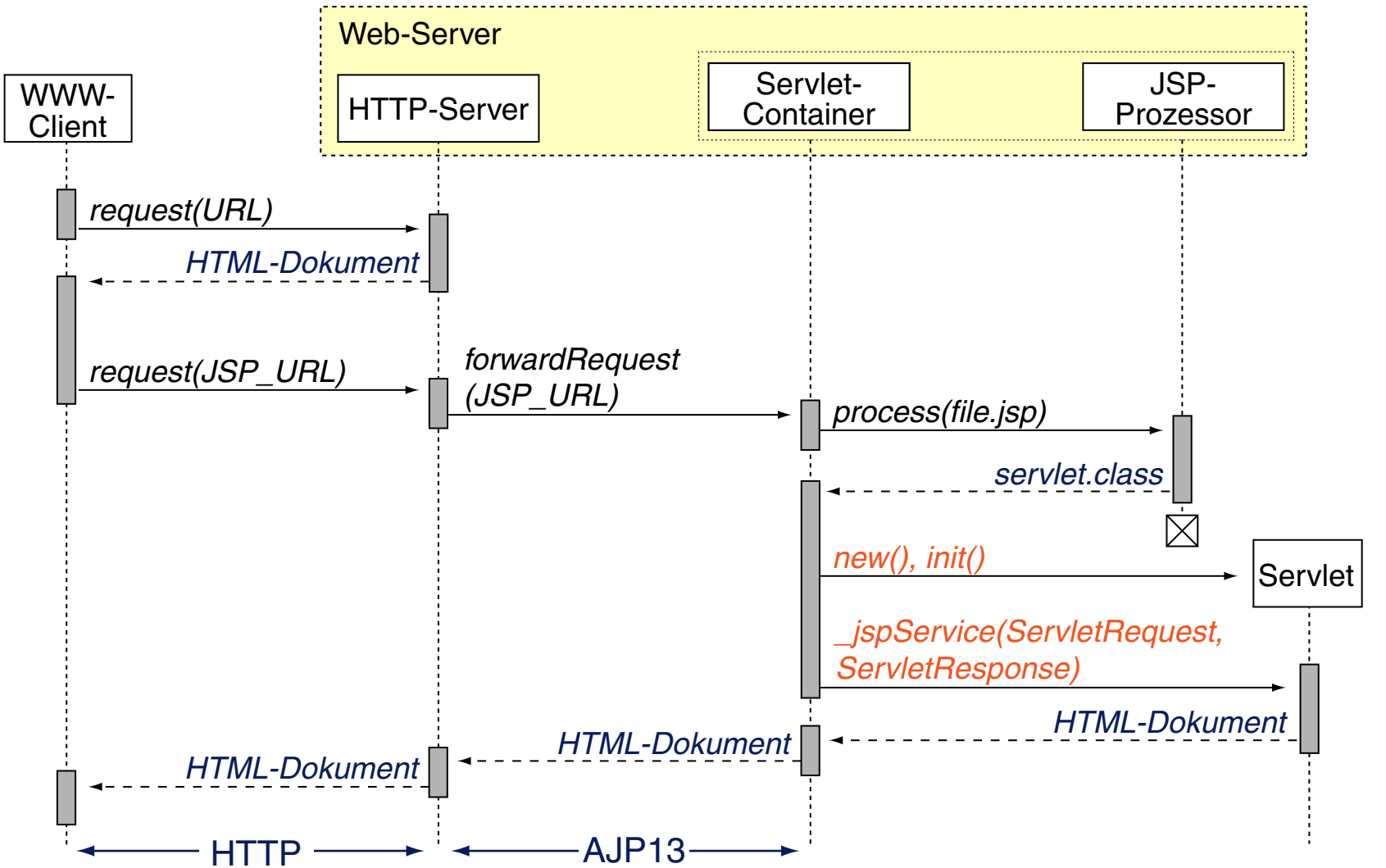
Java Server Pages JSP

Ablauf einer Seitenauslieferung via JSP [Vergleiche: statisch, CGI, Servlet]



Java Server Pages JSP

Ablauf einer Seitenauslieferung via JSP [Vergleiche: statisch, CGI, Servlet]



Bemerkungen:

- Die `jsp`-Datei kombiniert HTML-Code und Java-Code zur Erzeugung einer HTML-Seite.
- Die Generierung des Servlets aus der `jsp`-Datei geschieht „on the fly“. Dabei wird berücksichtigt, ob die `jsp`-Datei zwischenzeitlich verändert wurde.
- Das Action-Attribut `<... action="registration.jsp">` ist laut Referenz optional. Falls es fehlt, wird als Default-Wert die Datei selbst (hier: `registration.jsp`) genommen.
- Keine getrennten (CGI-)Programme zur Gestaltung eines Web-Dialogs: HTML-Form, HTML-Antwort sowie der Programmcode zur Verarbeitung sind in derselben Datei.

Java Server Pages JSP

Konzepte der JSP-Technologie

Der HTML-Code einer `jsp`-Datei wird um `out.println()`-Anweisungen ergänzt und in die `_jspService()`-Methode des Servlets integriert. [Javadoc] [Servlet: generiert]

Java Server Pages JSP

Konzepte der JSP-Technologie

Der HTML-Code einer `jsp`-Datei wird um `out.println()`-Anweisungen ergänzt und in die `_jspService()`-Methode des Servlets integriert. [Javadoc] [Servlet: generiert]

Zur Programmierung gibt es drei Konzepte:

1. Java.
   (a) Java-*Scriptlet*: beliebige Java-Anweisungen.
   HTML-Syntax: `<%` *Code* `%>` Code wird Teil der `_jspService()`-Methode.
   (b) Java-*Expression*: Wert, der zur Laufzeit ermittelt und als String ausgegeben wird.
   HTML-Syntax: `<%=` *Code* `%>` Code wird Teil der `_jspService()`-Methode.
   (c) Java-*Declaration*: beliebige Java-Anweisungen.
   HTML-Syntax: `<%!` *Code* `%>` Code ist außerhalb der `_jspService()`-Methode.

2. JSP-Aktionen.
   Anbindung von JavaBeans-Komponenten; verwendet XML-Syntax.

3. JSP-Direktiven.
   Anweisungen, die direkt vom JSP-Prozessor verarbeitet werden.

Java Server Pages JSP

JSP-Prozessor

```
...
<html>
 <body>
  <h1>Hello World</h1>
  <% String user=request.getParameter("user"); %>
  <%= clock.getDayofMonth() %>
  <%! private int accessCount = 0; %>
  ...
 </body>
</html>
```

JSP-Dokument

```
public class SampleServlet extends HttpServlet {
  ...

  public void _jspService(...) throws ServletException {



    ...
  }
}
```

Java Server Pages JSP

JSP-Prozessor

```
...
<html>
 <body>
  <h1>Hello World</h1>
   <% String user=request.getParameter("user"); %>
   <%= clock.getDayofMonth() %>
   <%! private int accessCount = 0; %>
   ...
 </body>
</html>
```

JSP-Dokument

```
public class SampleServlet extends HttpServlet {
  ...

  public void _jspService(...) throws ServletException {
    out.println("<h1>Hello World</h1>");

    ...
  }
}
```

Java Server Pages JSP

JSP-Prozessor

```
...
<html>
 <body>
  <h1>Hello World</h1>
  <% String user=request.getParameter("user"); %>
   <%= clock.getDayofMonth() %>
   <%! private int accessCount = 0; %>
   ...
 </body>
</html>
```

JSP-Dokument

```
public class SampleServlet extends HttpServlet {
  ...

  public void _jspService(...) throws ServletException {
    out.println("<h1>Hello World</h1>");
    String user=request.getParameter("user");

    ...
  }
}
```

Java Server Pages JSP

JSP-Prozessor

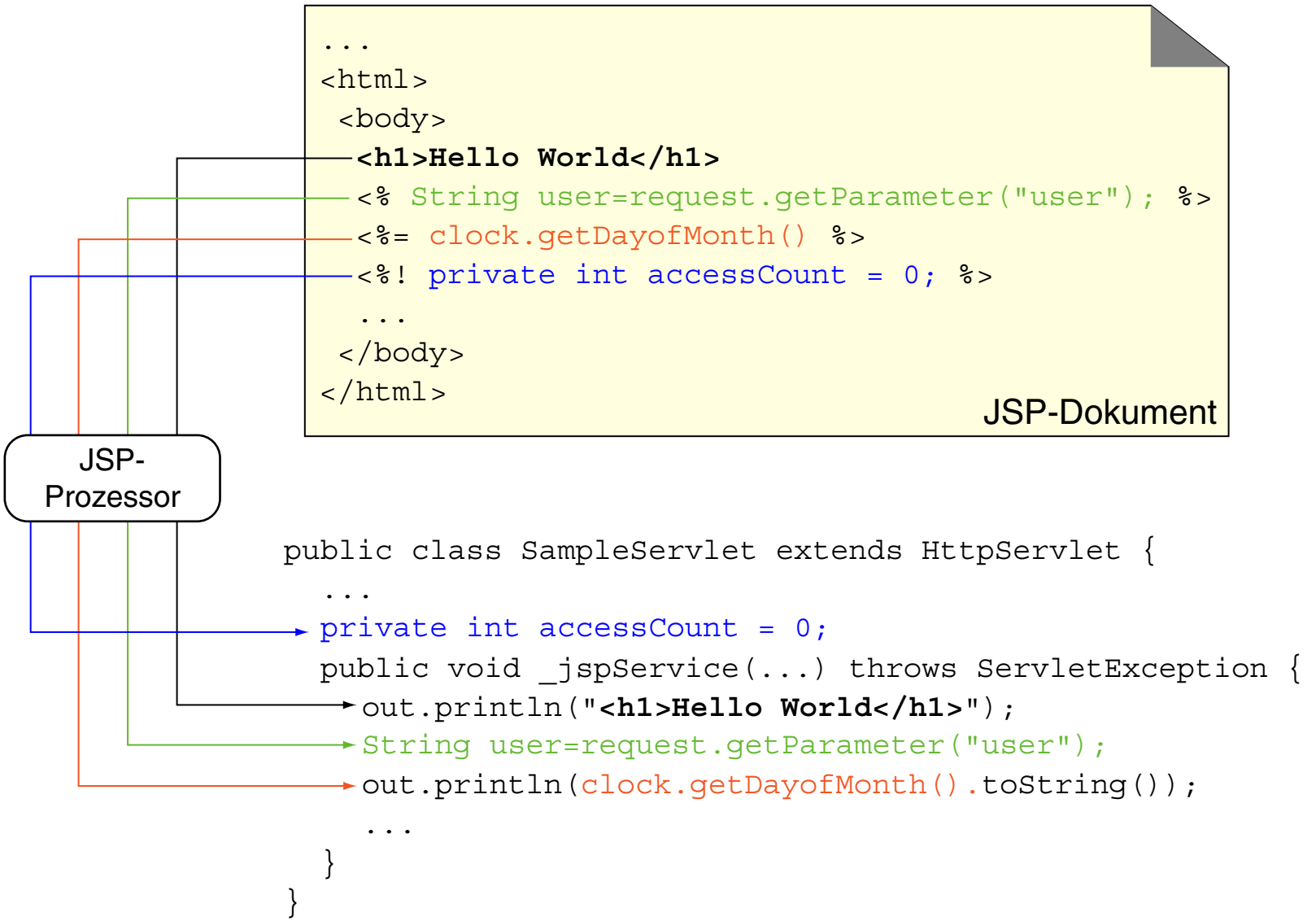
```
...
<html>
 <body>
  <h1>Hello World</h1>
  <% String user=request.getParameter("user"); %>
  <%= clock.getDayofMonth() %>
   <%! private int accessCount = 0; %>
   ...
 </body>
</html>
```

JSP-Dokument

```
public class SampleServlet extends HttpServlet {
  ...

  public void _jspService(...) throws ServletException {
    out.println("<h1>Hello World</h1>");
    String user=request.getParameter("user");
    out.println(clock.getDayofMonth().toString());
    ...
  }
}
```

Java Server Pages JSP

JSP-Prozessor



Server-Technologien

Quellen zum Nachlernen und Nachschlagen im Web

- Apache. *Apache HTTP Server Documentation.*
  httpd.apache.org/docs
- Hall. *Servlets and JavaServer Pages Tutorial Series.*
  courses.coreservlets.com
- Hall. *More Servlets and JavaServer Pages.*
  www.moreservlets.com/
- Oracle. *Java Servlet Tutorials.*
  javaee.github.io/tutorial/servlets.html
- Oracle. *Java Servlet Technology.*
  www.oracle.com/java/technologies/servlet-technology.html
  www.oracle.com/java/technologies/java-ee-glance.html
- Vogel. *Apache Tomcat - Tutorial.*
  www.vogella.com/tutorials/ApacheTomcat/article.html