

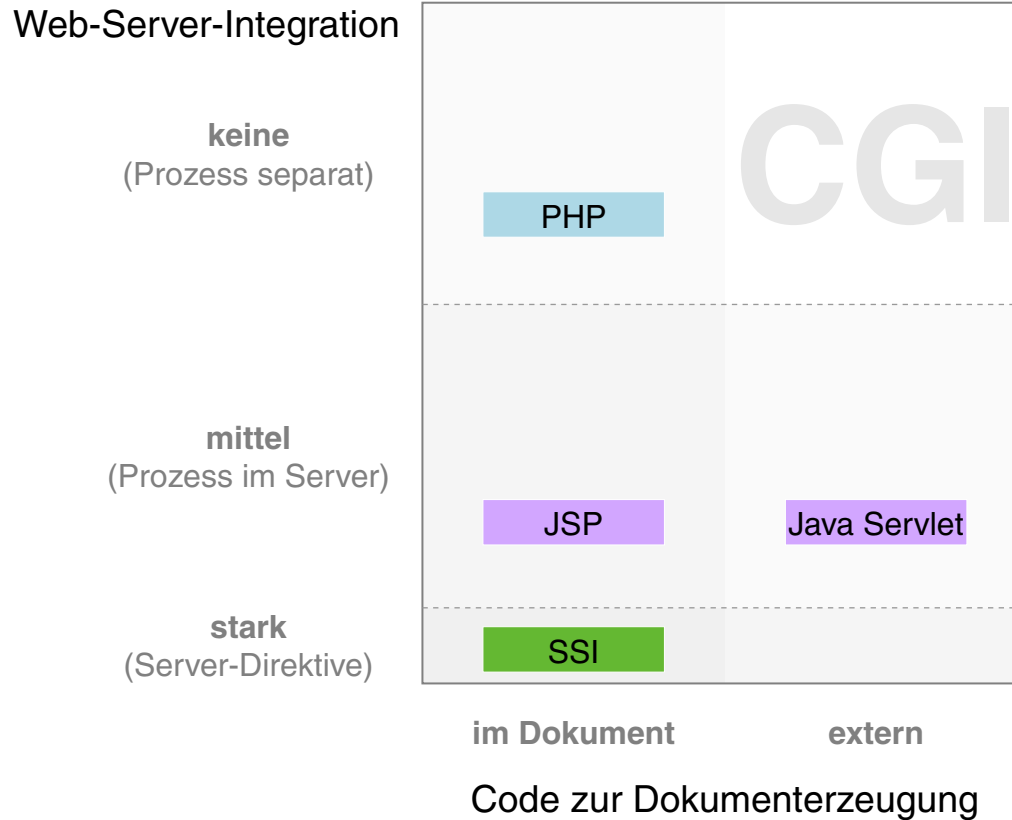
# Kapitel WT:IV

## 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 - 2019]



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

# Web-Server

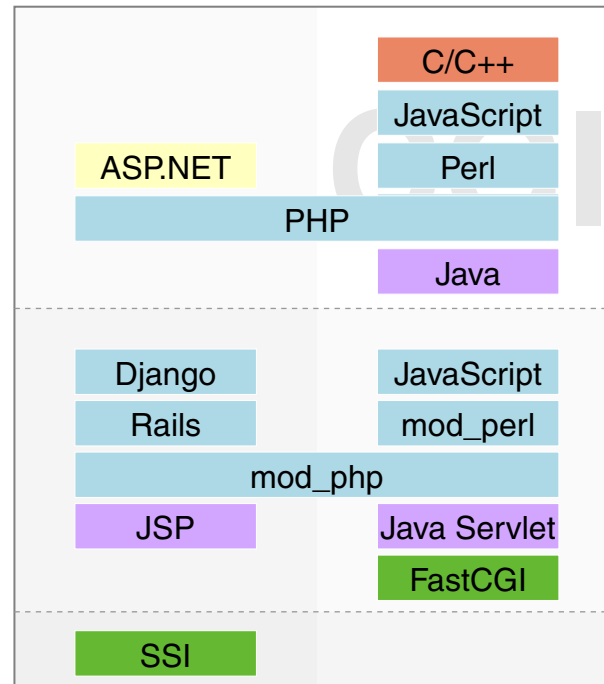
## Einordnung von Server-Technologien [Stein 2012 - 2019]

### Web-Server-Integration

**keine**  
(Prozess separat)

**mittel**  
(Prozess im Server)

**stark**  
(Server-Direktive)



im Dokument

extern

Code zur Dokumenterzeugung

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

## Bemerkungen:

- ❑ A web server is a computer system that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. The term can refer to the entire system, or specifically to the software that accepts and supervises the HTTP requests. [\[Wikipedia\]](#)
- ❑ Ein Web-Server wird auch als HTTP-Dämon bezeichnet; oft heißt das Programm, das den Web-Server implementiert, `httpd`.
- ❑ Separate Prozesse werden mittels CGI angebunden und direkt durch das Betriebssystem ausgeführt. Vorteil: einfache Anbindung des „Containers Betriebssystem“ in Form einer Shell. Nachteil (u.a.): zeitaufwändiger Start eines Prozesses.
- ❑ Prozesse *im* Web-Server sind durch Erweiterungsmodule (`mod_xxx`), zusätzliche Bibliotheken oder einen integrierten Container realisiert [\[Einordnung\]](#). Beispiele:
  - FastCGI, `mod_fcgid` [\[apache.org\]](#)
  - `mod_php` [\[apache.org\]](#)
  - `mod_perl` [\[apache.org\]](#)
  - JavaScript [\[nodejs.org\]](#)
  - JSP, Java Servlet [\[apache.org\]](#)

Überblick über Apache-Module [\[apache.org\]](#), [Wikipedia\]](#)

# Web-Server

## Wichtige Konfigurationseinstellungen

---

IP-Adresse, Hostname	Bei lokalem Betrieb die IP-Adresse 127.0.0.1 bzw. <code>localhost</code> .
Port	Üblicherweise lauscht der HTTP-Dämon auf Port 80 ( <i>well-known Port</i> ); andere Einstellungen sind möglich.
ServerRoot	Verzeichnis im Dateisystem des Server-Rechners für Konfigurations-, Fehler-, und Log-Dateien der Server-Software.
DocumentRoot (“Web-Space”)	Verzeichnis im Dateisystem des Server-Rechners, in dem sich die auszuliefernden HTML-Dateien befinden.
Default-HTML-Dateiname	Spezifiziert die URL nur ein Verzeichns, wird nach einer Default-Datei gesucht. Üblich sind <code>index.html</code> oder <code>index.htm</code> .
CGI-Skripte	Angabe des physischen Verzeichnisses und eines virtuellen Verzeichnisses (meist <code>/cgi-bin</code> ) für CGI-Skripte. Aufruf mit <code>https://Servername/cgi-bin/Skriptname</code> .
Log-Dateien	Protokollierung der Zugriffe ( <code>access.log</code> ), Fehler ( <code>error.log</code> )
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

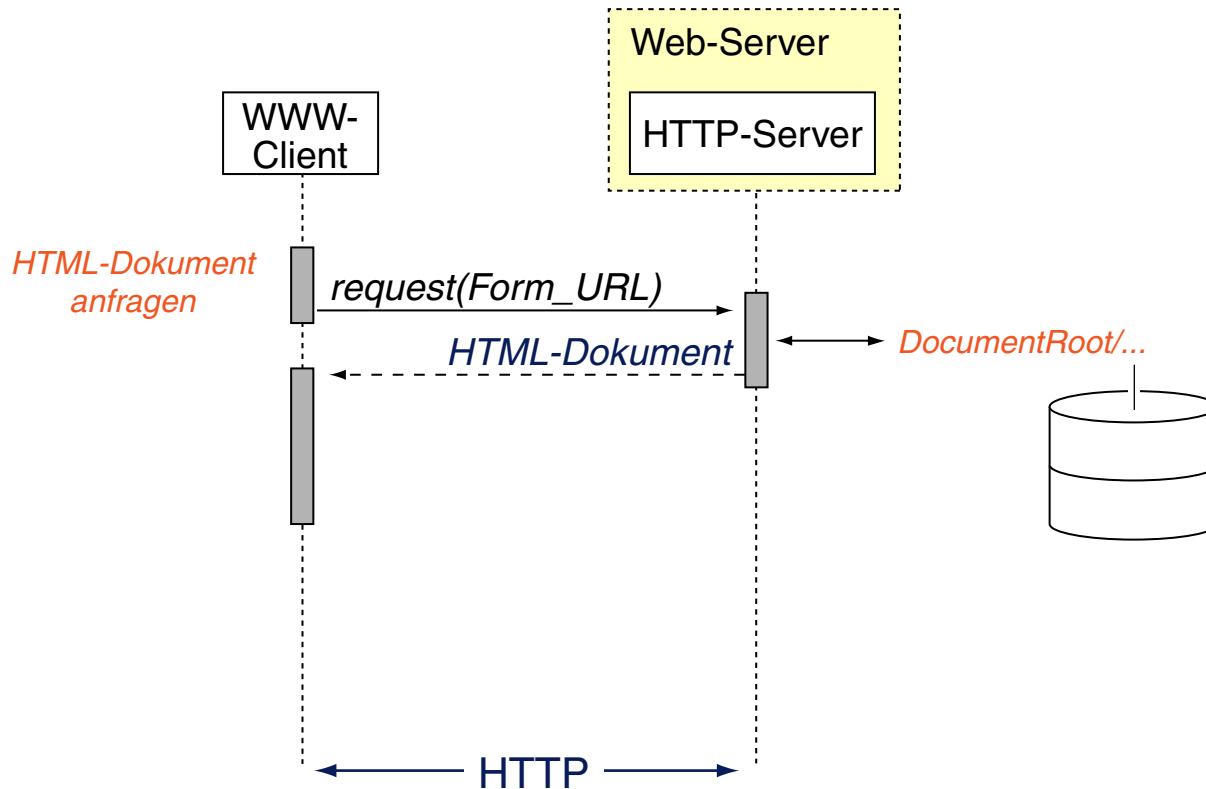
---

IP-Adresse, Hostname	Bei lokalem Betrieb die IP-Adresse 127.0.0.1 bzw. <code>localhost</code> .
Port	Üblicherweise lauscht der HTTP-Dämon auf Port 80 ( <i>well-known Port</i> ); andere Einstellungen sind möglich.
ServerRoot	Verzeichnis im Dateisystem des Server-Rechners für Konfigurations-, Fehler-, und Log-Dateien der Server-Software.
DocumentRoot ("Web-Space")	Verzeichnis im Dateisystem des Server-Rechners, in dem sich die auszuliefernden HTML-Dateien befinden.
Default-HTML-Dateiname	Spezifiziert die URL nur ein Verzeichns, wird nach einer Default-Datei gesucht. Üblich sind <code>index.html</code> oder <code>index.htm</code> .
CGI-Skripte	Angabe des physischen Verzeichnisses und eines virtuellen Verzeichnisses (meist <code>/cgi-bin</code> ) für CGI-Skripte. Aufruf mit <code>https://Servername/cgi-bin/Skriptname</code> .
Log-Dateien	Protokollierung der Zugriffe ( <code>access.log</code> ), Fehler ( <code>error.log</code> )
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.

---

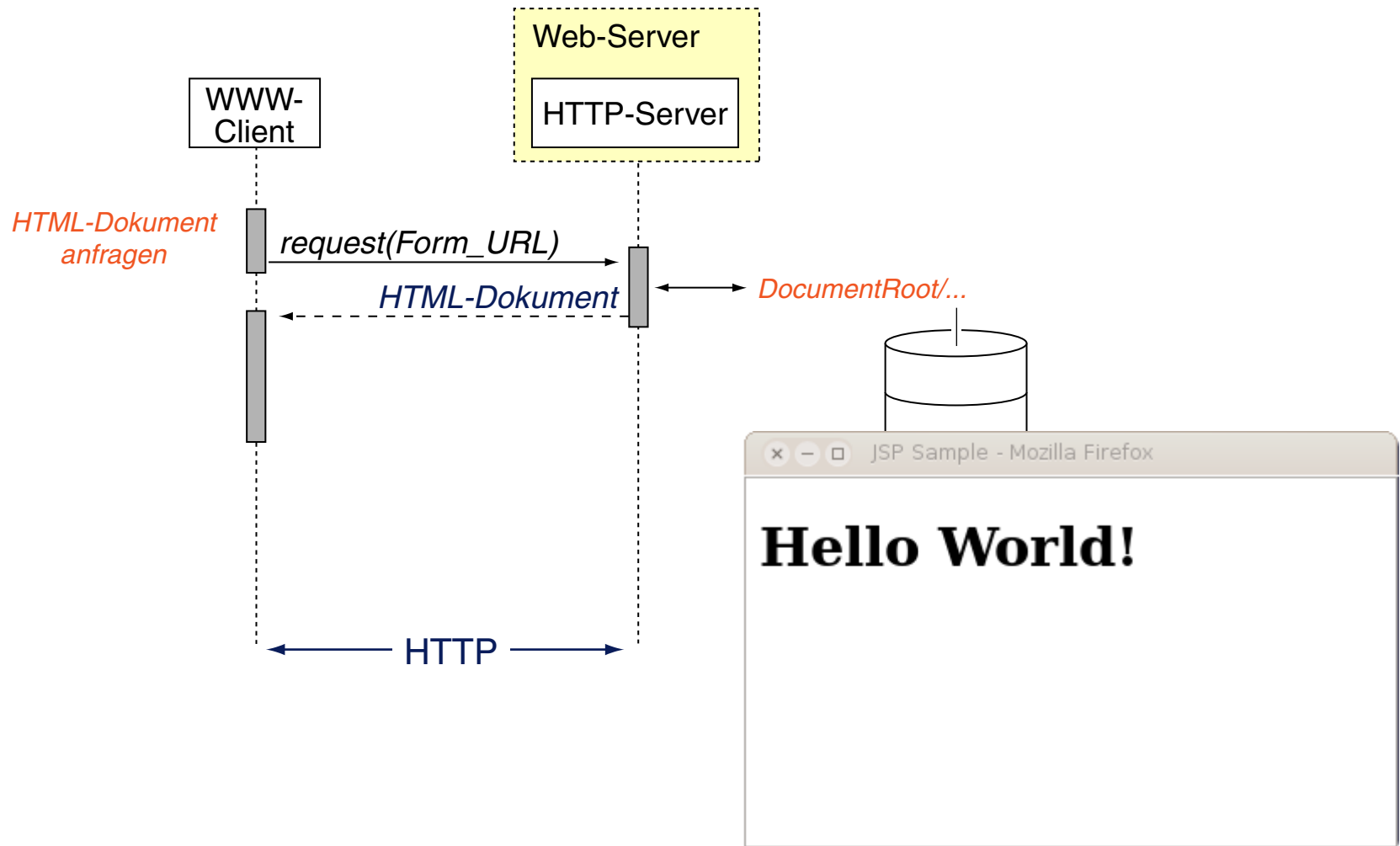
# Web-Server

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



# Web-Server

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



[Statisch: [Source](#), [Ausführung](#)]



# Web-Server

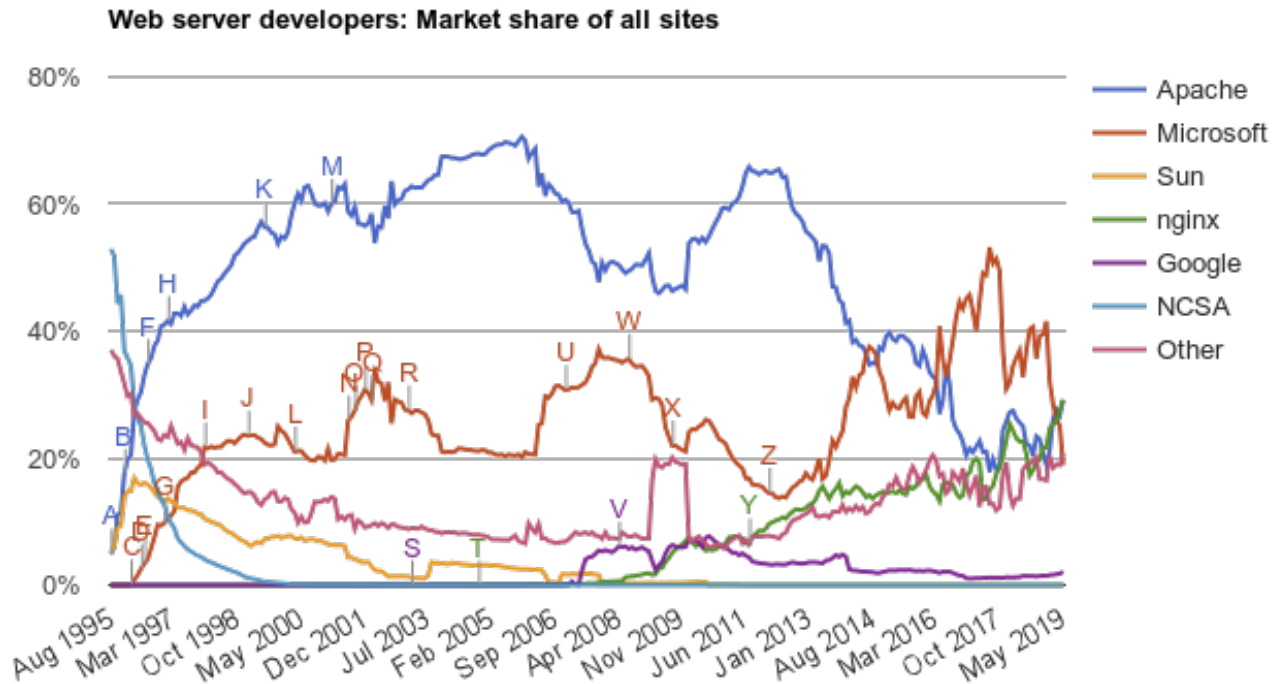
## Apache HTTP-Server: Historie [\[Wikipedia\]](#)

- 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 Ressourcenverbrauch.
- 2019 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](http://www.netcraft.com)]

## Bemerkungen:

- ❑ Der Marktanteil bezieht sich auf den Anteil an allen Websites. [[netcraft.com](http://netcraft.com)]
- ❑ Zählen von Web-Servern: [[netcraft.com](http://netcraft.com)]
- ❑ Dokumentation des Apache HTTP-Servers: [[apache.org](http://apache.org)]
- ❑ Glossar mit Fachbegriffen im Zusammenhang mit dem Apache HTTP-Server und Web-Server im Allgemeinen: [[apache.org](http://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
<code>&lt;IfDefine&gt;</code> , <code>&lt;IfModule&gt;</code>	Bedingte Ausführung von Direktiven.
<code>&lt;Directory&gt;</code> , <code>&lt;DirectoryMatch&gt;</code> <code>&lt;Files&gt;</code> , <code>&lt;FilesMatch&gt;</code> <code>&lt;Location&gt;</code> , <code>&lt;LocationMatch&gt;</code>	Spezifikation von Direktiven für Verzeichnisse, Dateien und URLs. Die <code>&lt;Match&gt;</code> -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](https://httpd.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](https://httpd.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. [\[apache.org\]](https://httpd.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 Passwörtern 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](http://apache.org)]
- ❑ Welche der globalen Vorgaben ein Anwender in der `.htaccess`-Datei überschreiben darf, wird mit der `allowoverride`-Direktive festgelegt. [[apache.org](http://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 Dateiergung 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 gekennzeichnet

### Beispiel:

```
<h3>Dynamisches HTML mit SSI</h3>  
Datum/Uhrzeit auf dem Server-Rechner:
```

```
Name dieser HTML-Datei:  
Installierte Server-Software:  
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.

```

```

- ❑ Über eine Server Side Include Anweisung.

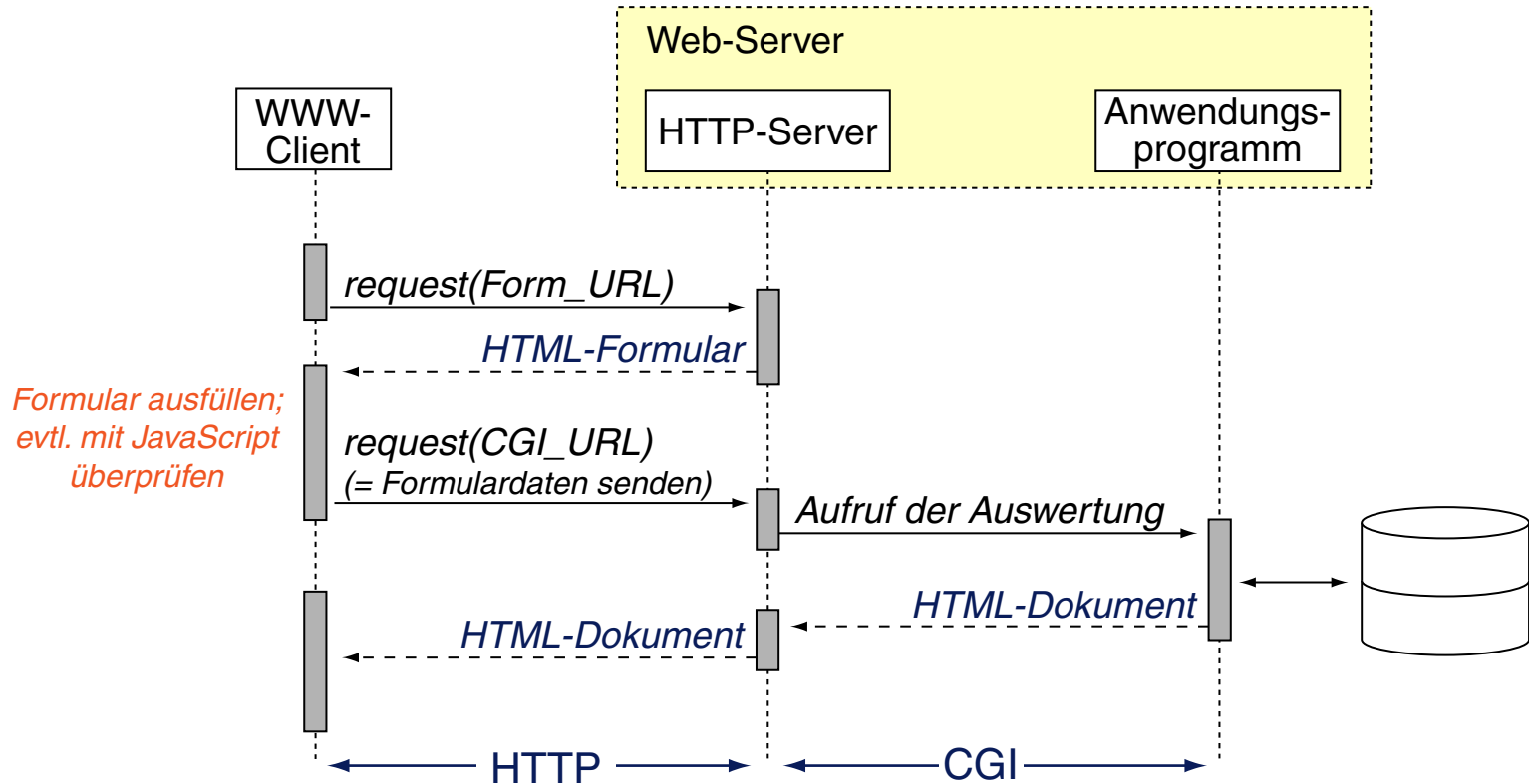
```
<!--#exec cgi="/cgi-bin/counter.pl" -->
```

- ❑ 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 einiger CGI-Variablen"
echo "Installierte Server-Software: Apache/2.2.22 (Ubuntu)"
echo "Aufrufender Web-Browser: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0)"
echo "Anfragemethode: Gecko/20100101 Firefox/37.0"
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.”*

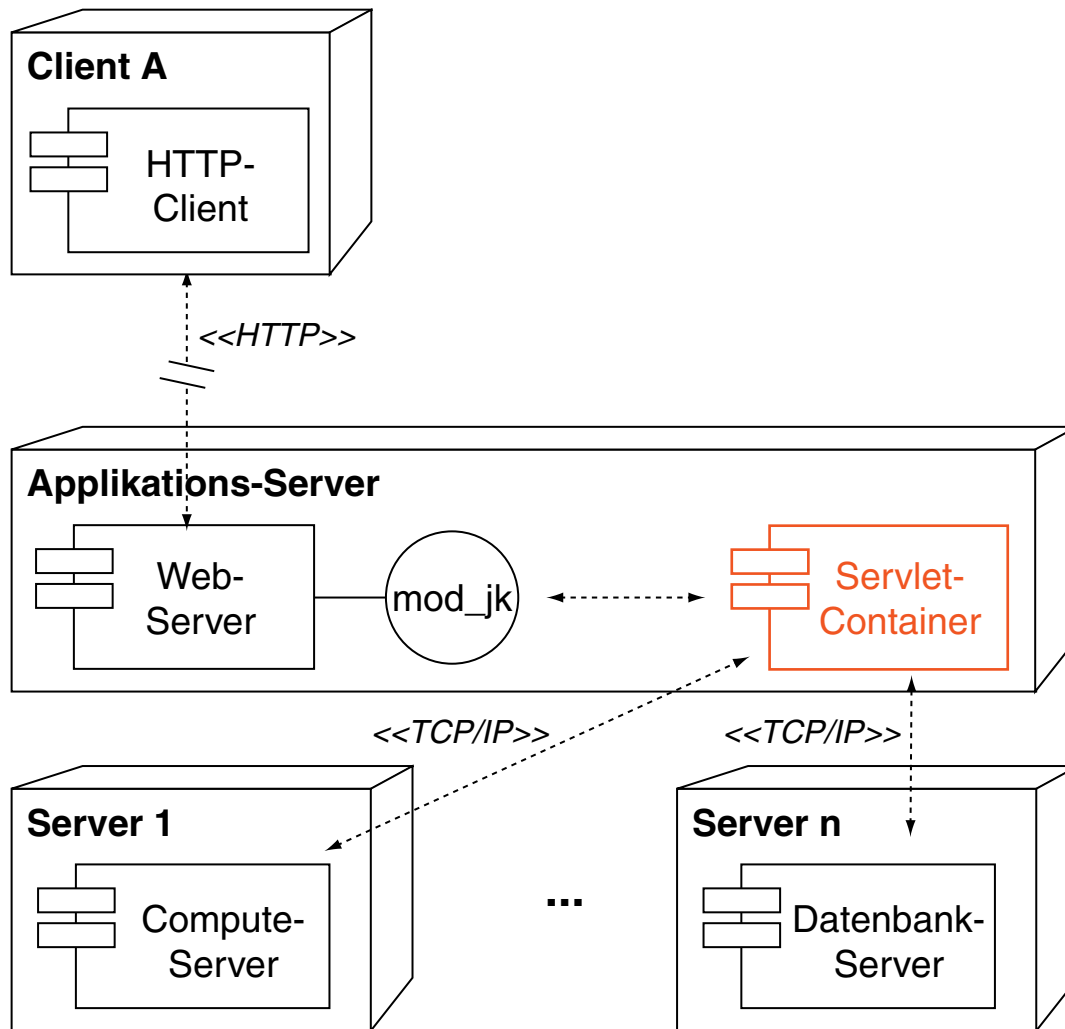
[Oracle [1](#), [2](#)]

- ❑ 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
<code>HttpServlet</code>	<code>service()</code> , <code>doGet()</code> , <code>doPost()</code>
<code>HttpServletRequest</code>	<code>get...()</code> -Methoden für CGI-Environment
<code>HttpServletResponse</code>	Streams als Ausgabekanal zum Client

# Java Servlet

## Einführung (Fortsetzung) [Deployment DB-Server]





## Bemerkungen:

- ❑ Das Deployment-Diagramm zeigt das Szenario eines Clients, der bei einem Web-Server eine URL anfragt, die evtl. auf ein Servlet verweist.
- ❑ 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 – wie hier dargestellt – über einen vorgeschalteten Web-Server angefragt werden. Im letzteren Fall wird eine Schnittstelle benötigt, 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](http://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 können. Alternativ kann eine Servlet-basierte Web-Anwendungen ihren eigenen Servlet-Container zur Installation mit bringen. Stichwort: *Embedded Servlet Container* [[Tomcat](http://Tomcat), [Jetty](http://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:

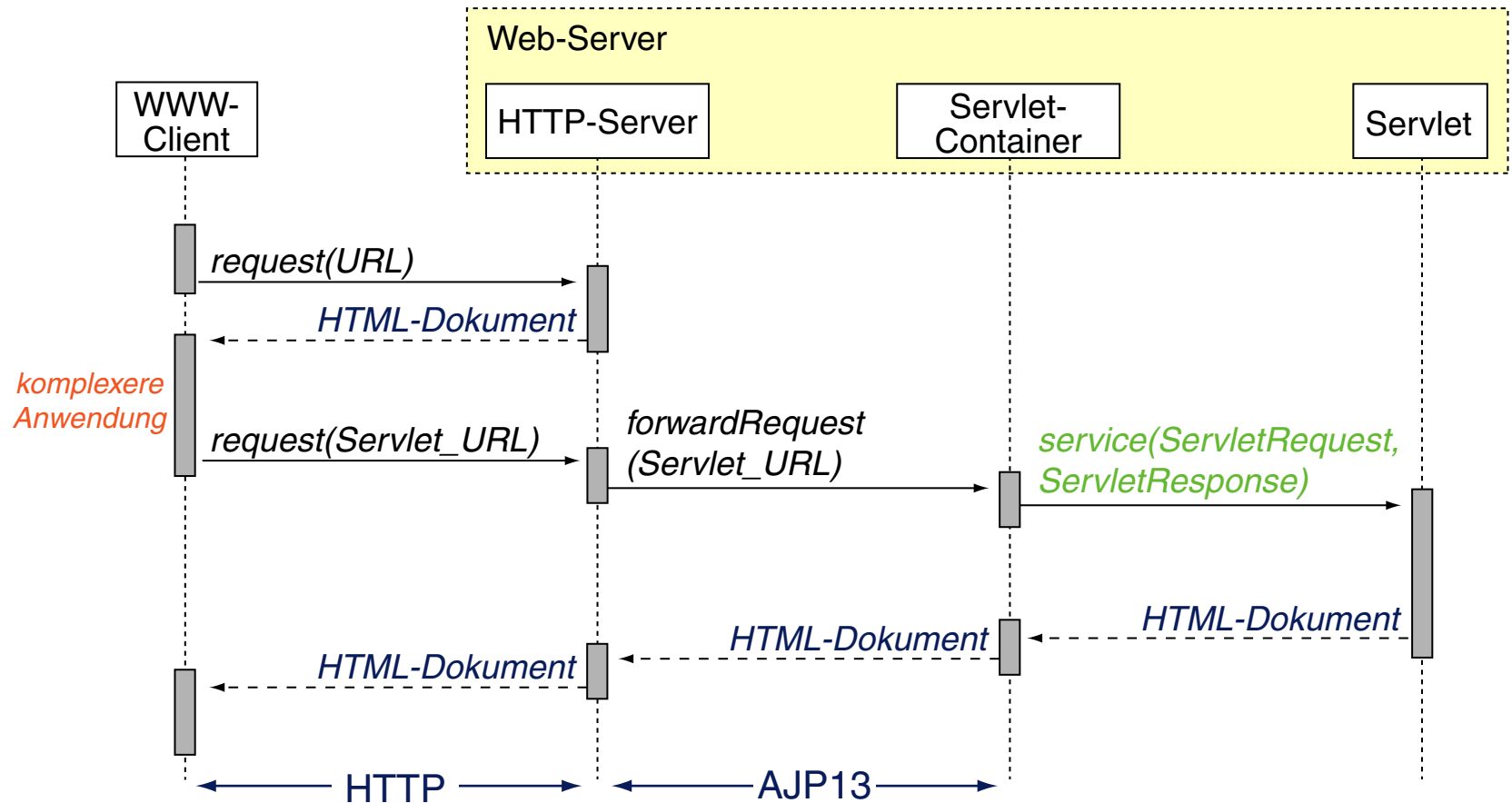
1. Überprüfung, ob eine Servlet-Instanz läuft. Falls nicht, wird
  - (a) die Servlet-Klasse geladen,
  - (b) eine Instanz der Servlet-Klasse erzeugt und
  - (c) die Servlet-Instanz durch Aufruf der `init()`-Methode initialisiert.
2. Erzeugung eines `HttpServletRequest`-Objektes und eines `HttpServletResponse`-Objektes.
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 und TRACE. Die entsprechenden Java-Methoden heißen `doGet()`, `doPost()`, etc.
- ❑ 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.*;

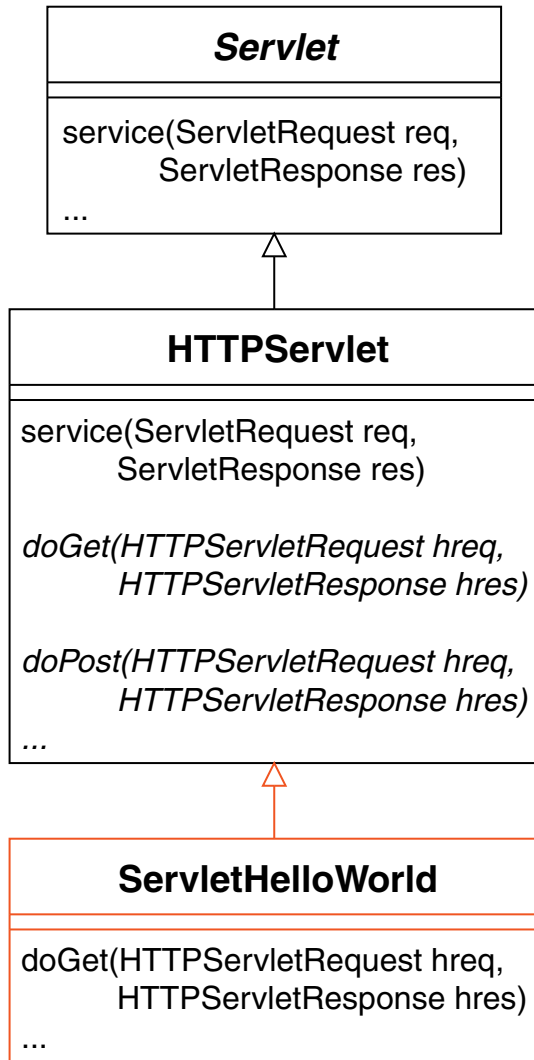
public class ServletHelloWorld extends HttpServlet {

    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

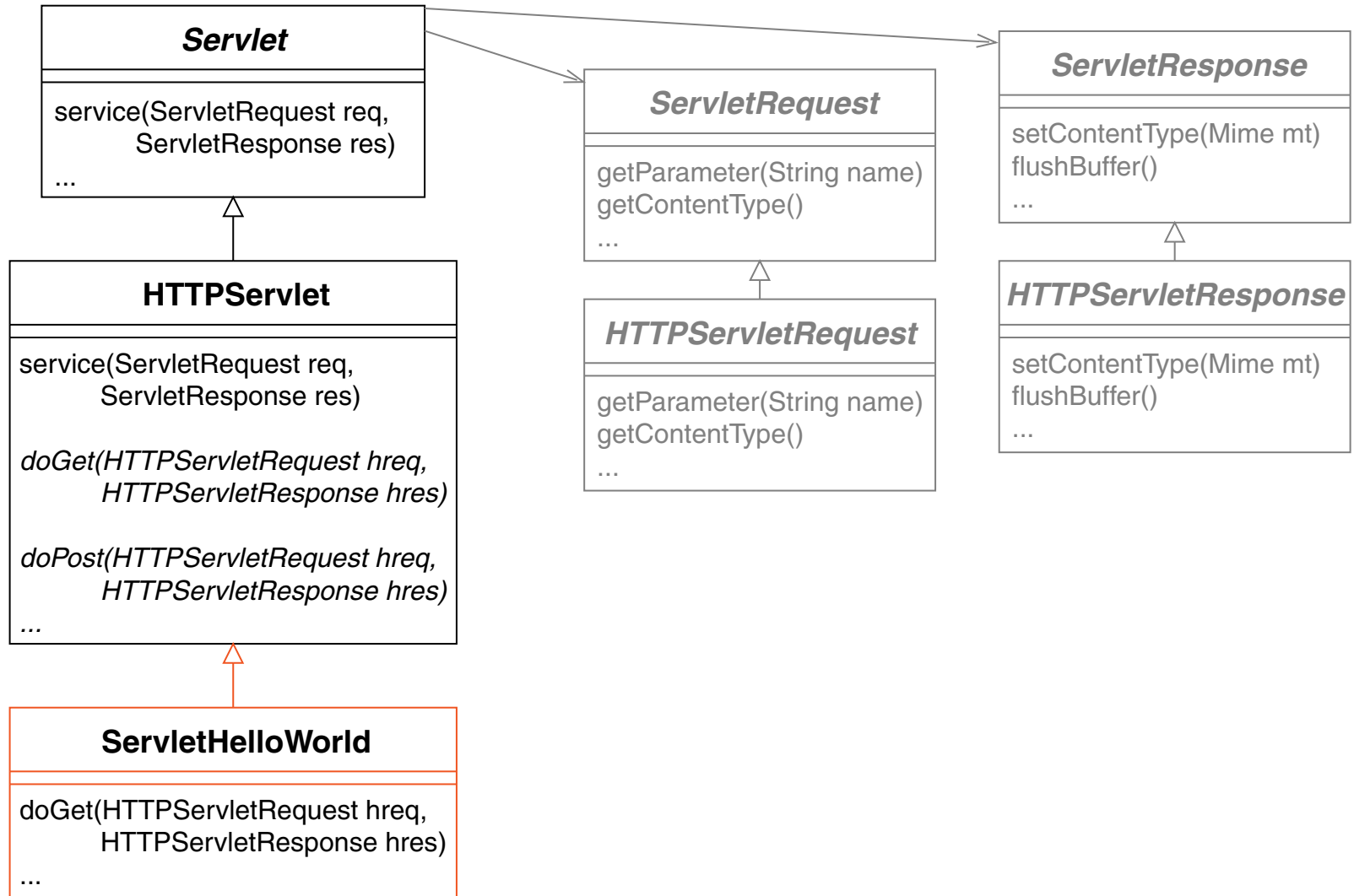
Implementierung folgt dem „Template Method“-Pattern





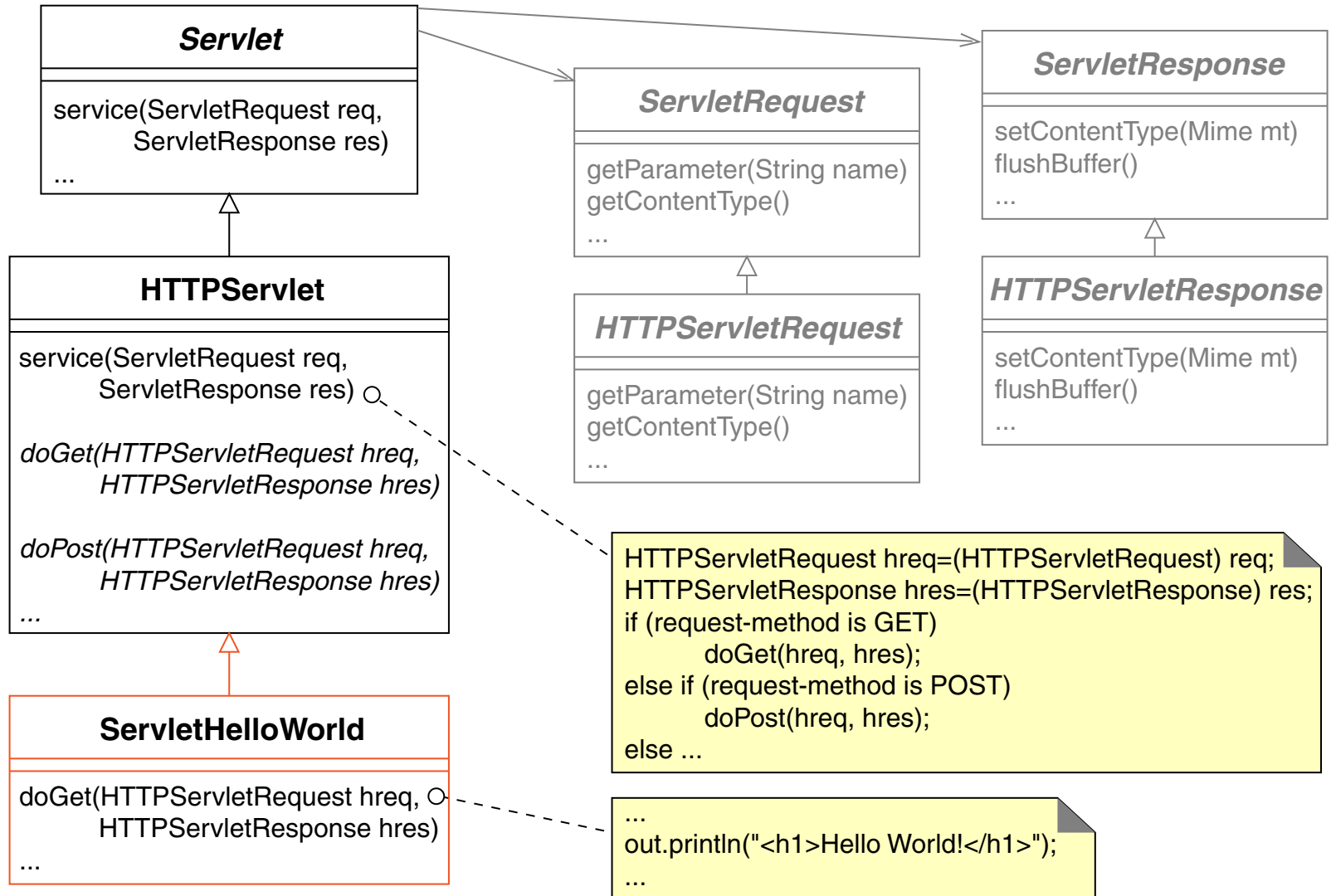
# Java Servlet

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)

Form with Servlet Call - Mozilla Firefox

Vorname:

Nachname:

Beruf:



URL Parameter - Mozilla Firefox

**Einlesen von URL Parametern**

- Vorname: Jean Claude
- Nachname: Lambert
- Beruf: Musiker

[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*



Mit obiger Verzeichnisstruktur geschieht der *Servlet*-Aufruf über einen HTTP-Server mit integrierten (1) bzw. externen (2) Servlet-Container wie folgt:

1. `http://Web-Server:8080/Context-Path/Servlet`
2. `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="http://xmlns.jcp.org/xml/ns/javaee/ .../web-app.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.



## Bemerkungen: (Fortsetzung)

- ❑ Aus Sicherheitsgründen sind fast alle Ports außer Port 80 bei Zugriffen von außerhalb gesperrt. Aufruf (1) ohne mod\_proxy ist dann nicht möglich.
- ❑ 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 Architektur, 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 Anmelde Daten:</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)



Registration - Mozilla Firefox

**Anmeldung**

Benutzername:

Anmelden



Registration - Mozilla Firefox

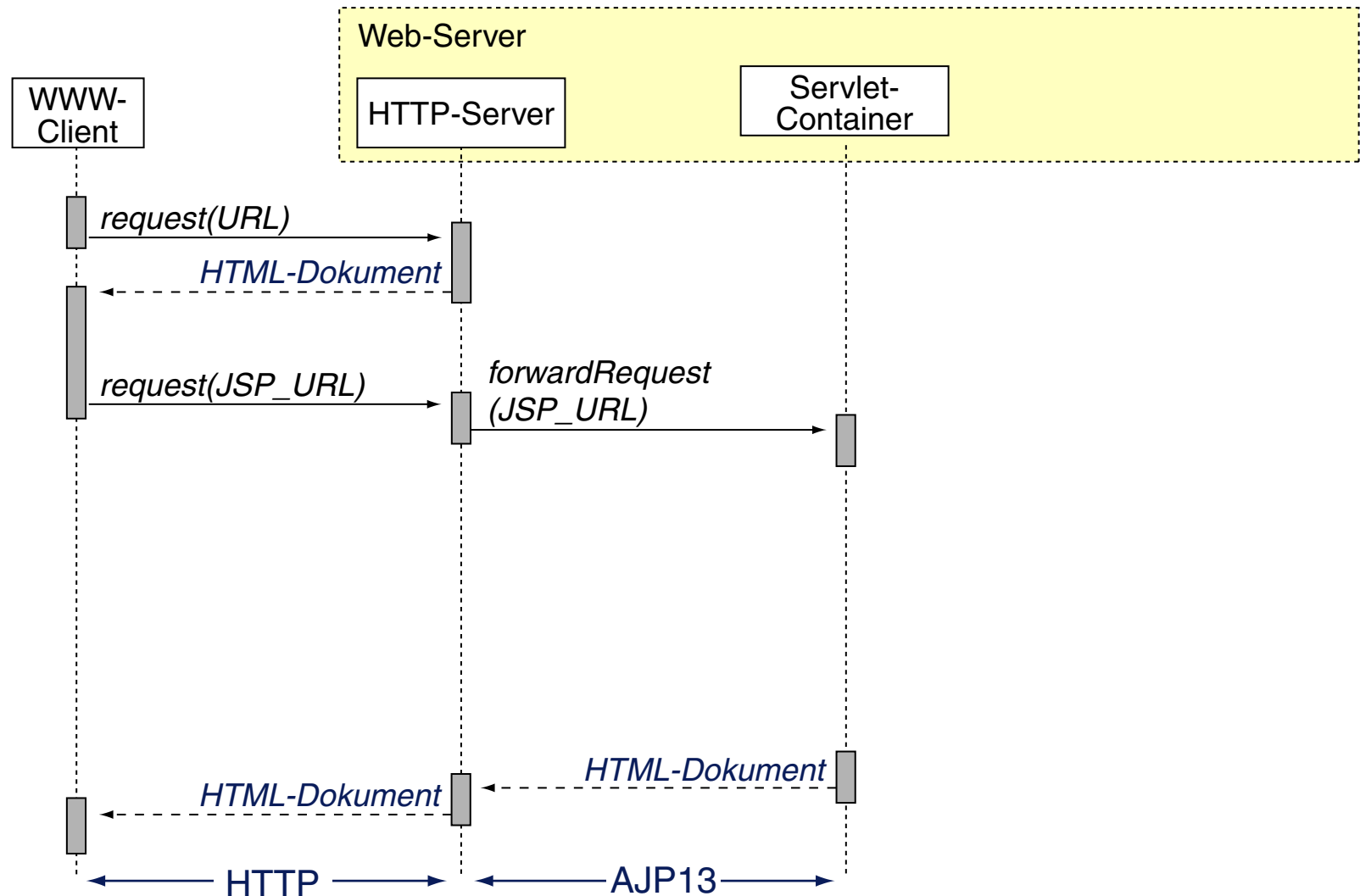
**Ihre Anmeldedaten:**

Benutzername: Jean Claude

[JSP: [Source](#), [Ausführung](#)]

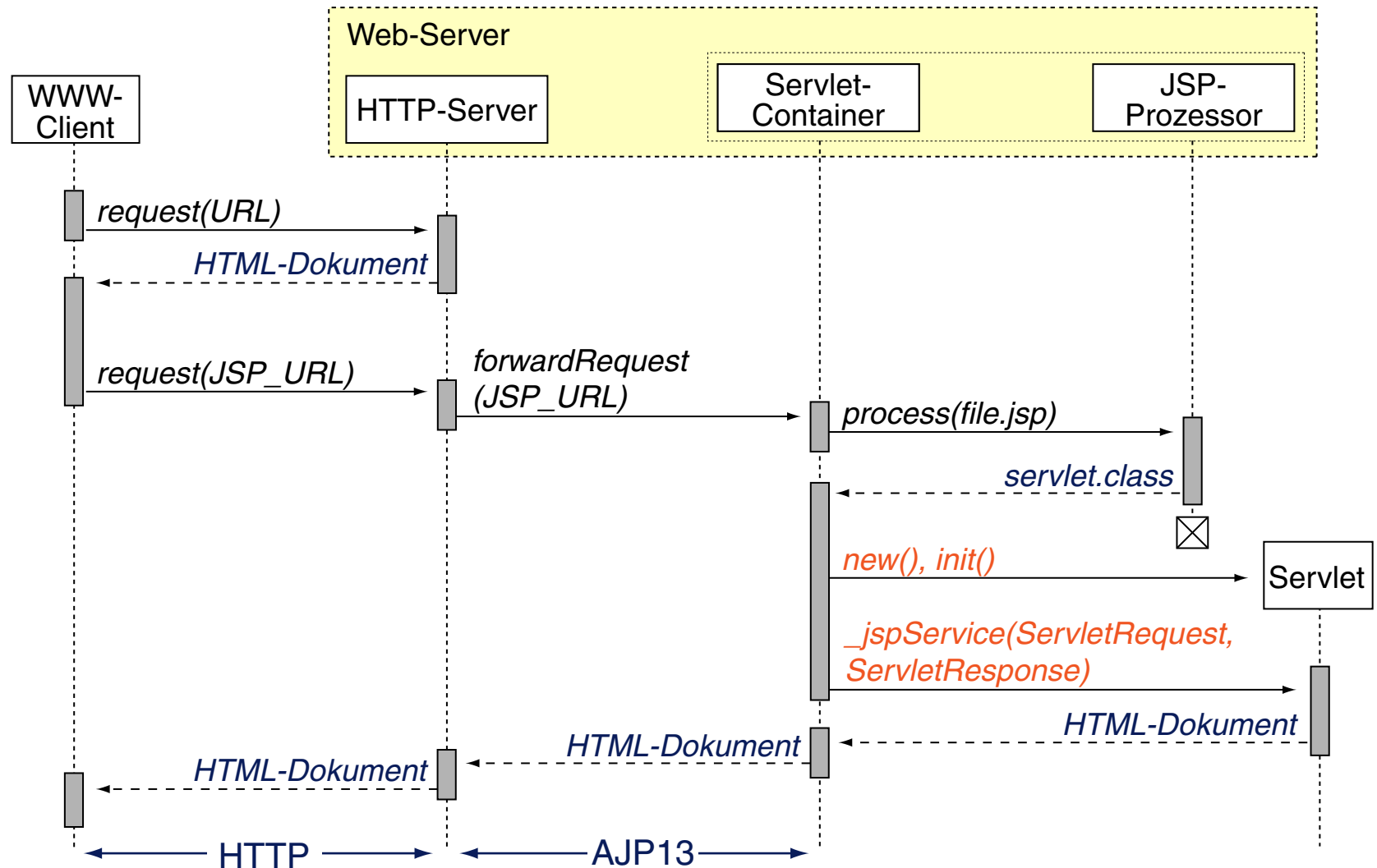
# Java Server Pages JSP

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. [\[Javadoc\]](#)

# 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. [\[Javadoc\]](#)

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

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

JSP-Dokument

JSP-  
Prozessor

```
public class SampleServlet extends HttpServlet {  
  ...  
  private int accessCount = 0;  
  public void _jspService(...) throws ServletException {  
    out.println("<h1>Hello World</h1>");  
    String user=request.getParameter("user");  
    out.println(clock.getDayOfMonth().toString());  
    ...  
  }  
}
```

# 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*.  
<http://www.moreservlets.com/>
- ❑ Oracle. *Java Servlet Tutorials*.  
<javaee.github.io/tutorial/servlets.html>
- ❑ Oracle. *Java Servlet Technology*.  
[www.oracle.com/technetwork/java/javaee/servlet](http://www.oracle.com/technetwork/java/javaee/servlet)  
[www.oracle.com/technetwork/java/javaee/overview](http://www.oracle.com/technetwork/java/javaee/overview)
- ❑ Vogel. *Apache Tomcat - Tutorial*.  
[www.vogella.com/tutorials/ApacheTomcat/article.html](http://www.vogella.com/tutorials/ApacheTomcat/article.html)