JSP & Servlets

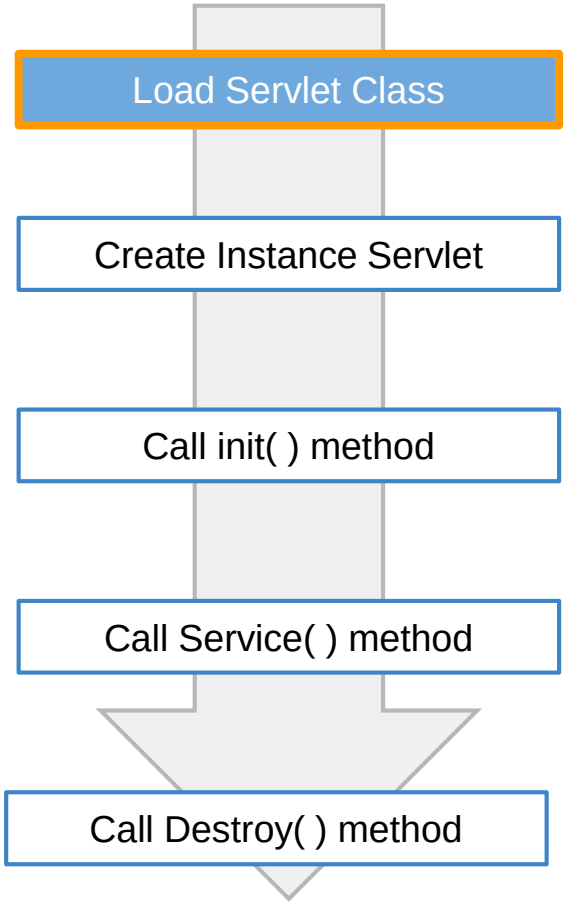
Servlets

Leerstof voor vandaag

Levensloop van een servlet
Servlet hiërarchie
Scope objecten

Lifecycle overview

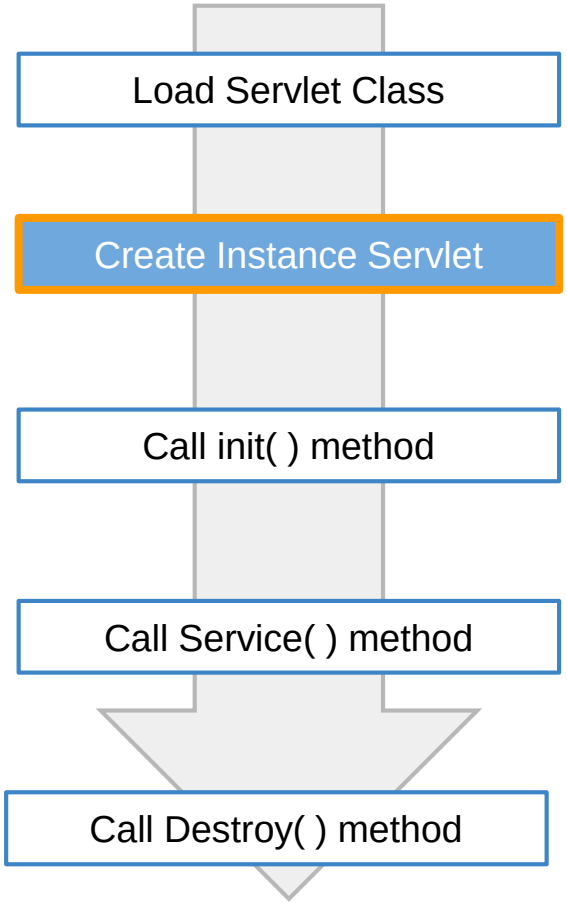
Lifecycle overview



Eerste stap:

De class wordt ingeladen, maar er wordt nog geen geheugen gereserveerd.

Lifecycle overview

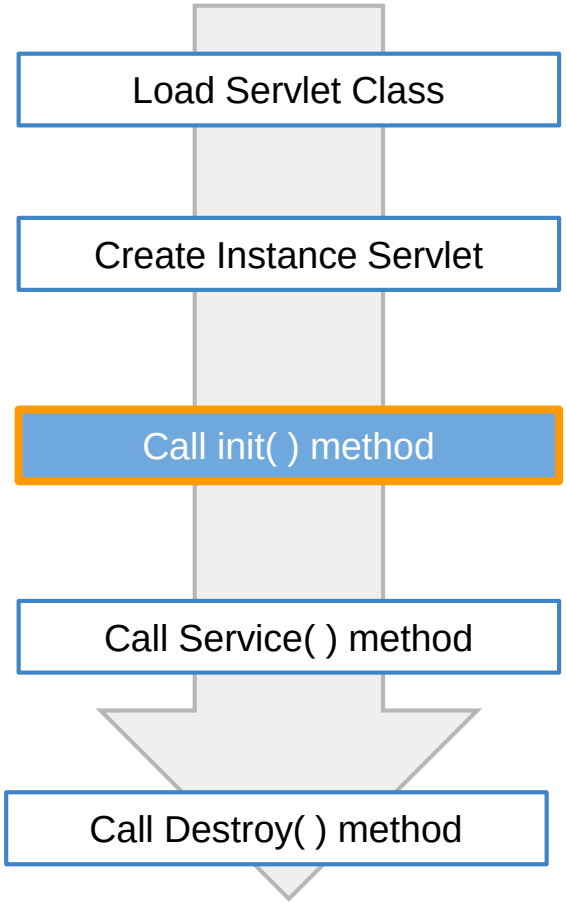


1 * aangeroepen!

Tomcat → krijgt request binnen:

- Instance aanmaken v Servlet
- Server stuurt request naar deze instance

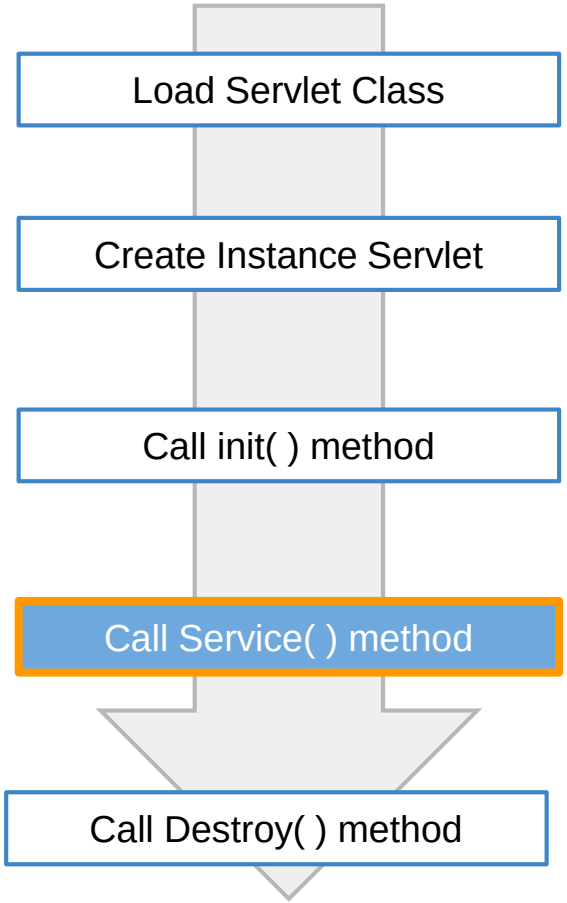
Lifecycle overview



Wordt slechts 1 maal aangeroepen, direct na de creatie van de Servlet.

→ Initialisaties uitvoeren

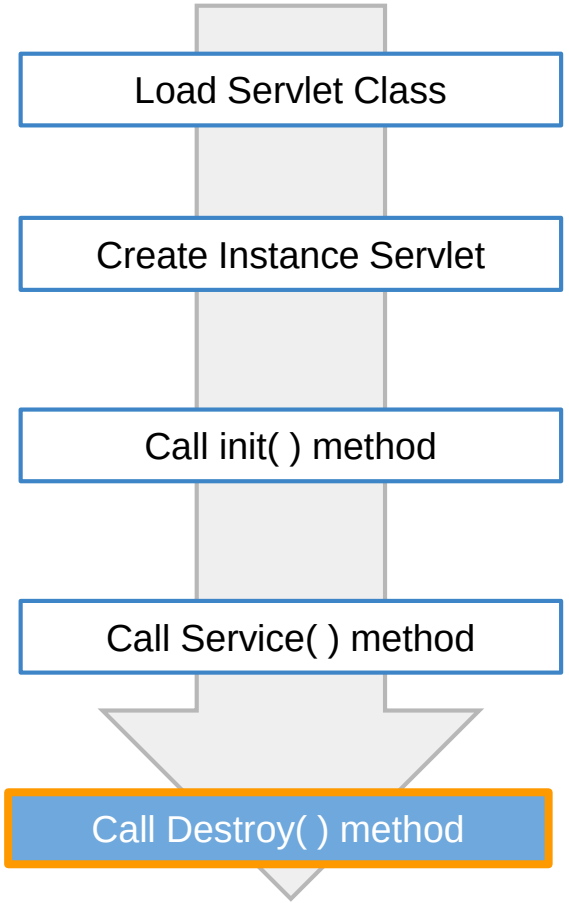
Lifecycle overview



De servlet instance is daarna klaar om requests te verwerken

- De webserver roept de method doGet op bij een GET request
- De webserver roept de method doPost op bij een POST request

Lifecycle overview

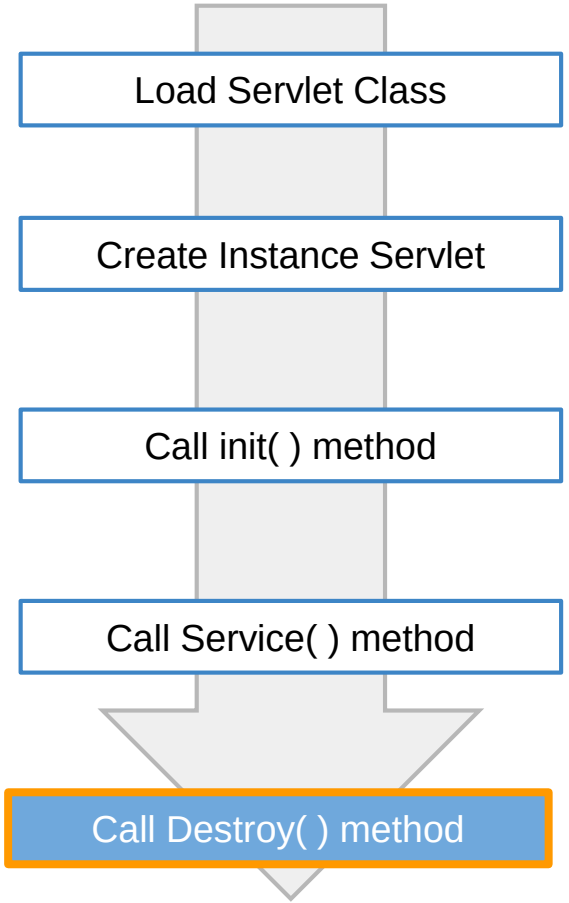


Wordt eenmalig op het einde van de lifecycle opgeroepen → Als alle Servlet request afgehandeld zijn.

Wordt gebruikt om database connecties te sluiten, achtergrond taken te stoppen, e.a.

Of als Tomcat afgesloten wordt!

Einde van een servlet instance



Wanneer sterft een servlet instance?

- bij het stoppen van de website.
- bij het verwijderen van de website
- bij het stoppen van de webserver

Voorwaarde:

de servlet instance is **NIET** bezig met een request te verwerken

Servlet initialisatie parameters

Algemeen

Servlet initialisatie parameters

= waarden die een servlet leest uit **web.xml of uit de annotatie.**

Deze waarden zijn enkel en **alleen beschikbaar voor die ene servlet**

Voordeel:

gemakkelijk aanpasbaar, geen tussenkomst van een programmeur nodig

Voorwaarde:

@webservlet niet enkel associëren met een URL maar ook benoemen.

(logisch: in web.xml moeten we kunnen verwijzen naar die servlet)

Hoe?

Servlet associëren, waarden definiëren en uitlezen → zie volgende dia's

Aanpassingen aan @WebServlet

Parameters van @WebServlet:

- urlPatterns: definieert welke URL bij de servlet hoort
- name: definieert de naam van de servlet

Opdracht

Wijzig in de **WelkomServlet** volgende regel:

```
@WebServlet("/welkom")
```

naar

```
@WebServlet(urlPatterns="/welkom",
name="welkomservlet")
```

Initialisatieparameters via annotaties.

```
@WebServlet(urlPatterns="/welkom",
name="WelkomServlet",
initParams=@WebInitParam(name="naam",
value="Kenneth"))
public class WelkomServlet extends HttpServlet{
…
}
```

Initialisatieparameters via web.xml

Voeg toe tussen <web-app …> en </web-app> in web.xml:

```
<servlet>
        <servlet-name>welkomservlet</servlet-name>
        <servlet-class>be.vdab.componenten.WelkomServlet</servlet-class>
        <init-param>
                <param-name>voornaam</param-name>
        <param-value>Luigi</param-value>
        </init-param>
        …
```

Vul hier zelf aan met de parameters familienaam, aantalkinderen en gehuwd

```
</servlet>
```

De initialisatieparameters lezen

getInitParameter

- De waarde van een initialisatieparameter lees je met de overgeërfde method **getInitParameter()**.

- De waarde wordt als een String teruggegeven.

De oproep **moet** gebeuren in de method **init( )**
(ook overgeërfd van de base class HttpServlet).

Tomcat roept deze methode op onmiddellijk nadat de webserver de instance maakte met de constructor.

voorbeeld:

```
zaakvoerder.setVoornaam(this.getInitParameter("voornaam"));
```

De method init van een servlet

Opdracht

1. Voeg een initialisatieparameter toe met jouw naam.
2. De startpagina moet nu in een heading jouw naam tonen.
3. Test je programma uit.
4. Doe hetzelfde nu maar via **web.xml.** Wijzig in deze file jouw naam.
5. Test je programma uit.
6. Maak opdracht 2 in de cursus (p. 33)!

doGet() en doPost()

doGet() methode

Webbrowser → Nieuwe pagina → Webserver
Parameters kunnen toegevoegd aan URL (=query string)

BV:
http://localhost:8080/Webcomponents/Welkom?message=”Hello”

Opvragen in doGet() methode:
String message= request.getParameter(“message”);

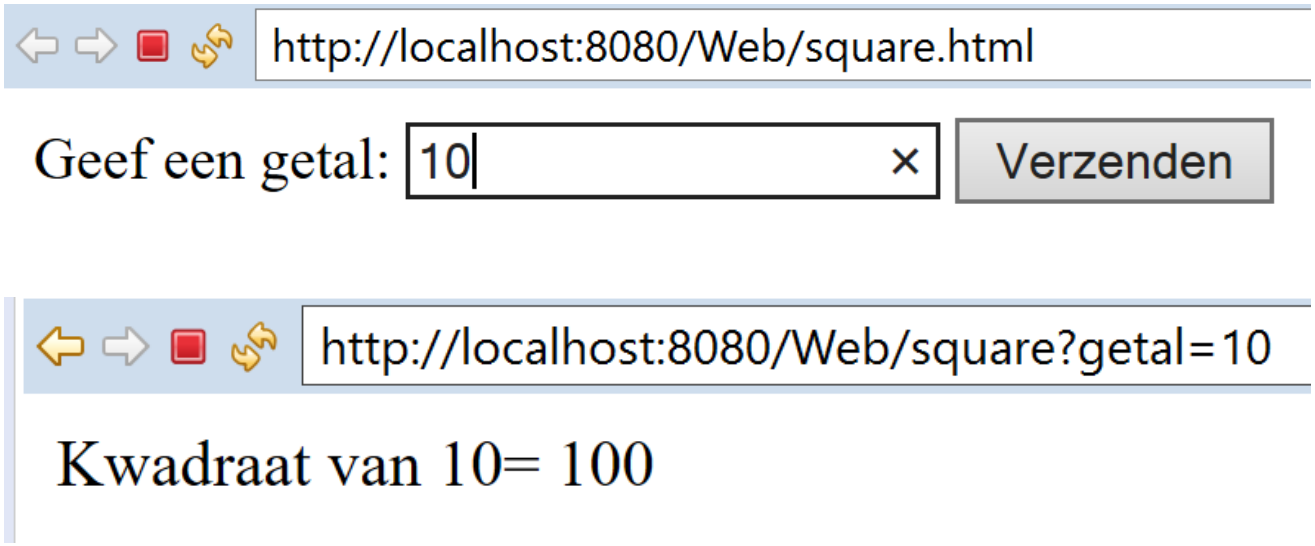
doGet() methode

Zelf formulier maken:

```
<form action="welkom" method="get">
        <input type="text" name="message" />
        <input type="submit" value="Verzenden" />
</form>
```

Opvragen in doGet() methode van WelkomServlet:
String message= request.getParameter(“message”);

Opdracht 3: p. 36



doPost() methode

Webbrowser → Nieuwe pagina → Webserver
Parameters kunnen toegevoegd aan de body.

Opvragen in doPost() methode:
String message= request.getParameter(“message”);

→ getParameter ziet zowel in body als in URL.

doPost() methode

Zelf formulier maken:

```
<form action="welkom" method="post">
        <input type="text" name="message" />
        <input type="submit" value="Verzenden" />
</form>
```

Opvragen in doPost() methode van WelkomServlet:
String message= request.getParameter(“message”);

doPost() methode

OF:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                        response.setContentType("text/html");
                        response.setCharacterEncoding("UTF-8");
                        PrintWriter out = response.getWriter();
                        out.println(
                                "<!DOCTYPE html><html><head><title>Using doPost!</title>"
                                + "</head><body><form action='welkom' method='post'>"
                                + "<input type='text' name='message' /> "
                                + "<input type='submit' value='Verzenden' />"
                                + "</form></body></html>");
                        out.close();
                }
```

Opvragen in doPost() methode van WelkomServlet:
String message= request.getParameter(“message”);

Opdracht 4: p. 39

Scope objecten

Algemeen

Request Scope

Beperkt tot afhandeling van request.

Bv: Afhandelen formulier

Session Scope

Gedurende de sessie van een cliënt

Bv: Winkelmandje van een gebruiker

Application Scope

Gedurende de sessie van alle cliënts

Bv: Aantal gebruikers op de website

Request Scope

- Get or Post parameters: Via formulier

- Attributen: Servlet1 verzend info naar Servlet2
  (komt later in de cursus)

Session Scope

Algemeen

- Een website gebruikt het HTTP protocol
- Het HTTP protocol is een **stateless** protocol

Wat is stateless?

- Elke request is een zelfstandige handeling
- Variabelen verdwijnen uit het geheugen nadat de request verwerkt is.

Voordeel:

- Veel gebruikers terzelfde tijd veroorzaken geen geheugen tekort
- Webserver mag gestopt en herstart worden

Het begrip Session

Wat is een session?

- Een stuk intern geheugen van de webserver die data voor één gebruiker/browser onthoudt.

- Deze data heten **session attributen**

- Elke session heeft een **uniek ID.** Zo'n ID bestaat uit een lange tekst met random tekens.

- De webserver houdt bij welke session er bij welke browser hoort. Dat kan op twee manieren:
    - Met een **tijdelijke cookie**
    - Door middel van **URL rewriting.**

Bijhouden met een tijdelijke cookie

Wanneer?

Als de cookies ingeschakeld zijn

Verloop:

1. Na aanmaak van een session stuurt de webserver een tijdelijke cookie via een response naar de browser. Deze cookie heeft als naam **JSESSIONID** en als waarde de ID van de session.

2. Bij een volgende request worden alle cookies, inclusief JSESSIONID naar de server gestuurt.

3. De server zoekt de session op.

Bijhouden met URL rewriting

Wanneer?

Als de cookies uitgeschakeld zijn

Verloop:

In de servlet method **sendRedirect** geven we de URL niet rechtstreeks mee maar wel via de method **encodeURL** van het **response object**:

```
out.print("<form method='get' action='" +
          response.encodeURL(request.getRequestURI()) + "'>");
```

Een session ophalen/aanmaken

Waar?

In de servlet methods **doGet** en **doPost**

Hoe?

Door de method **getSession** van de request parameter op te roepen **VOOR** een **forward** of een **sendRedirect.**

Voorbeeld

```
HttpSession session = request.getSession();
```

Opmerking

Indien geen session wordt gevonden, wordt hiermee een nieuw aangemaakt, tenzij de parameter **false** wordt meegegeven.

Een session attribuut toevoegen / wijzigen

Waar?

In de servlet methods **doPost**

Hoe?

Door de method **setAttribute** van de session op te roepen. Je geeft als parameter de **naam** en de **waarde** mee van het attribuut.

Voorbeeld

```
session.setAttribute("allePizzas", pizzaDAO.findAll());
```

Opmerking

De parameter waarde kan een **primitief type** zijn of **een class** dat de interface **Serializable** implementeert

De inhoud van een session attribuut lezen

Waar?

In de servlet methods **doGet**

Hoe?

Door de method **getAttribute** van de session op te roepen. Je geeft als parameter de **naam** mee van het attribuut.

Voorbeeld

```
Set<Long> mandje = (Set<Long>) session.getAttribute("mandje");
```

Opmerking

De method getAttribute geeft een object of Null terug indien het attribuut niet werd gevonden

Een session attribuut verwijderen

Waar?

In de servlet methods **doPost**

Hoe?

Door de method **removeAttribute** van de session op te roepen. Je geeft als parameter de **naam** mee van het attribuut.

De volledige session verwijderen

Waar? In de servlet methods **doPost**

Hoe? Door de method **invalidate** van de session op te roepen. De session en al zijn attributen worden hierdoor verwijdert.

Opmerking

Een session vervalt na x aantal minuten, anders dreigt het geheugen vol te komen zitten. Het aantal is bij Tomcat standaard op 30 ingesteld. Je kan dit wijzigen in **web.xml:**

```
<web-app ...>
<session-config>
        <session-timeout>20</session-timeout> <!-- 20 minuten -->
</session-config>
...
```

Demo

1) Tekst vragen aan gebruiker
2) Tekst opslaan in de sessie
3) Bij opnieuw openen, de tekst tonen
4) Test uit in verschillende browsers
5) Cookies uitschakelen in de browser
6) Url codering instellen
7) Test opnieuw uit
8) Maak opdracht 5, p. 51

Session Event handling

Afzonderlijk object → Op de hoogte brengen dat sessie beëindigd is.

Bv: Winkelkarretje verwijderen als sessie ten einde loopt.

```
@Weblistener
pulblic class myListener implements HttpSessionListener{
}
```

Wat is een listener

Wat is een listener ?

De servlet specificatie **definieert** enkele **gebeurtenissen** die kunnen optreden in je website.
Bij iedere van deze gebeurtenissen hoort een interface.

Een listener is **een class die zo'n interface implementeert**. Als zo'n gebeurtenis optreedt, roept de webserver in je listener class de method op die beschreven is in de interface die bij de gebeurtenis hoort.

Voorbeeld

(zie verderop)

Opmerking

Een listener moet steeds een default constructor hebben
Een listener moet threadsafe geschreven worden

Configureren van een listener

2 manieren

- Met de annotation **@WebListener**, die je schrijft voor de listener class

- In **web.xml**
  - ***voordeel:***
    herconfiguratie mogelijk zonder Java sources te wijzigen.
  - ***voorbeeld:***

```
<listener>
    <listener-class>
        be.vdab.be.mijnListener
    </listener-class>
</listener>
```

Volgorde van listeners

Bij gebruik van @WebListener

Volgorde onbepaald

Als meerdere listeners dezelfde interface implementeren en je registreert deze listeners met @WebListener, is de volgorde waarmee de webserver die listeners oproept onbepaald.

Bij registratie in web.xml

Volgorde volgens registratie

Als meerdere listeners dezelfde interface implementeren en je registreert deze listeners in web.xml, is de volgorde waarmee de webserver die listeners oproept, dezelfde als de volgorde waarin je de listeners registreert in web.xml

De belangrijkste gebeurtenissen (events)

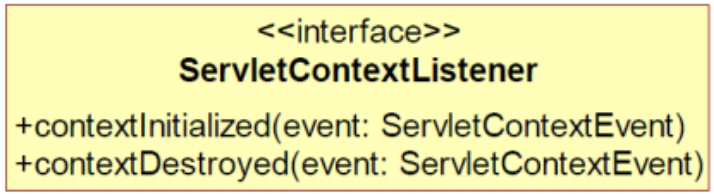
ServletContextListener

Gebeurtenis

De website is pas gestart of staat op het punt gestopt te worden

Interface

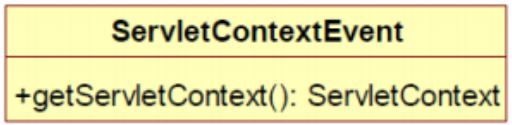
ServletContextListener



Methods

contextInitialized : De website is pas gestart
contextDestroyed: De website staat op punt gestopt te worden

Beide methods hebben een parameter van het type **ServletContextEvent**:



getServletContext: Geeft je toegang tot de servlet context.

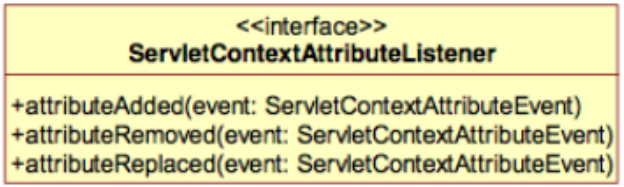
ServletContextAttributeListener

Gebeurtenis

Een servlet context attribuut werd aangemaakt, gewijzigd of verwijderd

Interface

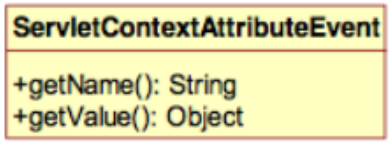
ServletContextAttributeListener



Methods

attributeAdded: een attribuut wordt aangemaakt
attributeRemoved: een attribuut wordt verwijderd
attributeReplaced: een attribuut wordt gewijzigd

Deze methods hebben een parameter van het type **ServletContextAttributeEvent**:



Methods van het type ServletContextAttributeEvent:

getName(): String Geeft de naam van het attribuut
getValue(): Object Geeft de waarde van het attribuut

HttpSessionListener

Gebeurtenis

Een session werd aangemaakt, verwijderd of vervalt

Interface

HttpSessionListener

Methods

sessionCreated
sessionDestroyed

Beide methods hebben een parameter van het type **HttpSessionEvent**

Method van het type HttpSessionEvent:

getSession(): HttpSession

→ Geeft de session die gemaakt of verwijderd wordt.

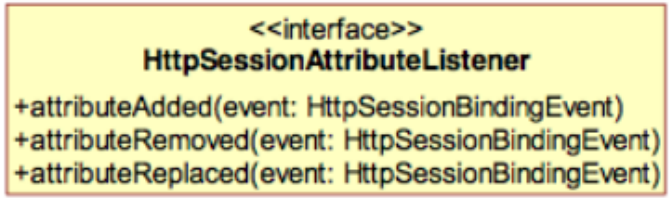
HttpSessionAttributeListener

Gebeurtenis

Een session attribuut werd aangemaakt, gewijzigd of verwijderd.

Interface

HttpSessionAttributeListener

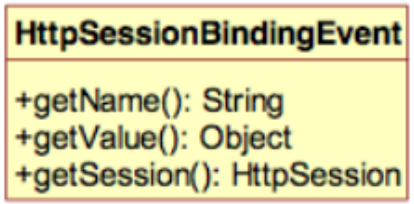


Methods

attributeAdded	een attribuut wordt aangemaakt
attributeRemoved	een attribuut wordt verwijderd
attributeReplaced	een attribuut wordt gewijzigd

Deze methods hebben een parameter van het type httpSessionBindingEvent

getName	Geeft de naam van het attribuut
getValue	Geeft de waarde van het attribuut
getSession	Geeft de bijhorende session



ServletRequestListener

Gebeurtenis

De website krijgt een request binnen of een request is helemaal verwerkt

Interface

ServletRequestListener

Methods

requestInitialized Een request komt binnen
requestDestroyed Een request is helemaal verwerkt

Deze methods hebben een parameter van het type ServletRequestEvent

getServletRequest
Geeft het bijhorend request
getServletContext
Geeft de servlet context

ServletRequestAttributeListener

Gebeurtenis

Een servlet attribuut werd aangemaakt, gewijzigd of verwijderd

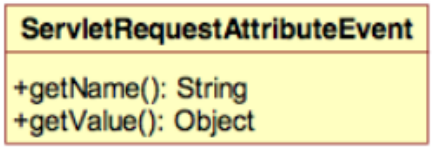
Interface

ServletRequestAttributeListener

Methods

attributeAdded	een attribuut wordt aangemaakt.
attributeRemoved	een attribuut wordt verwijderd.
attributeReplaced	een attribuut wordt gewijzigd.

Deze methods hebben een parameter van het type ServletRequestAttributeEvent



getName

Geeft je de naam van het attribuut

getValue

Geeft je de waarde van het attribuut

Application Scope

Een **servlet context** wordt gebruikt voor

- Servlet Context Initialisatieparameters
- Context attributen te bewaren
- Lokale server sources te lezen
- Informatie te bekomen over de servlet container

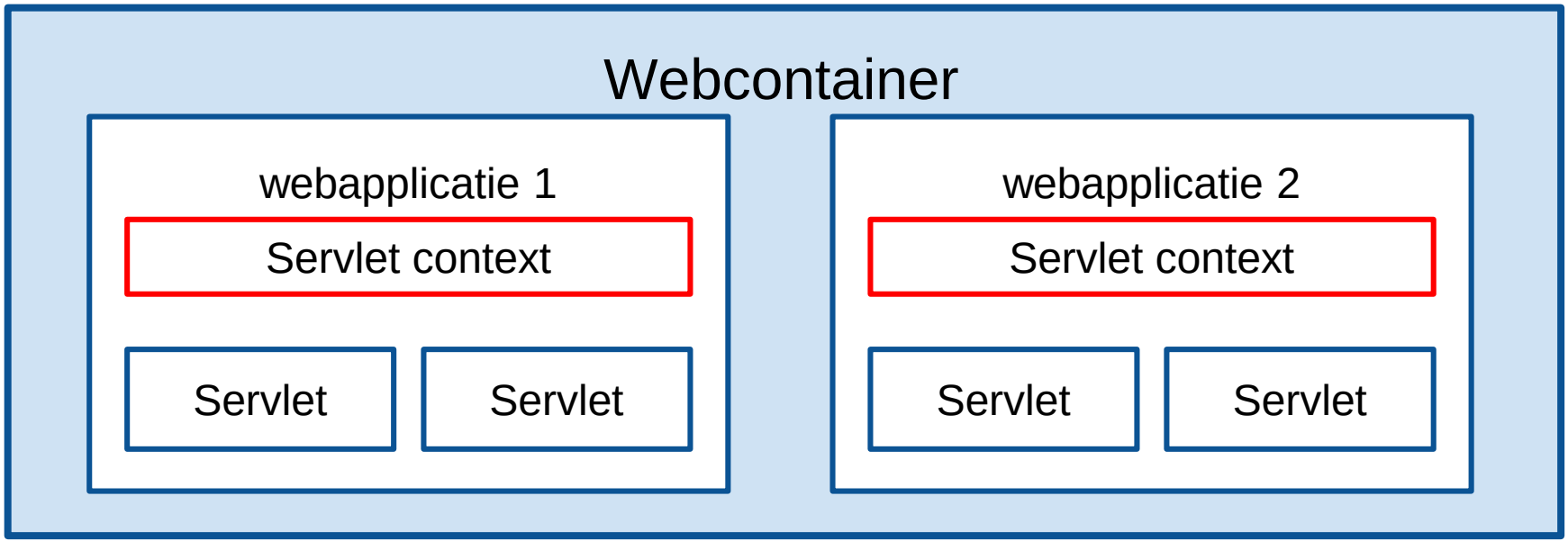
Iedere website heeft één servlet context.

Alle servlets van de website delen deze servlet context

Schematische voorstelling

Iedere website heeft één servlet context.

Alle servlets van de website delen deze servlet context



Servlet context initialisatie parameters

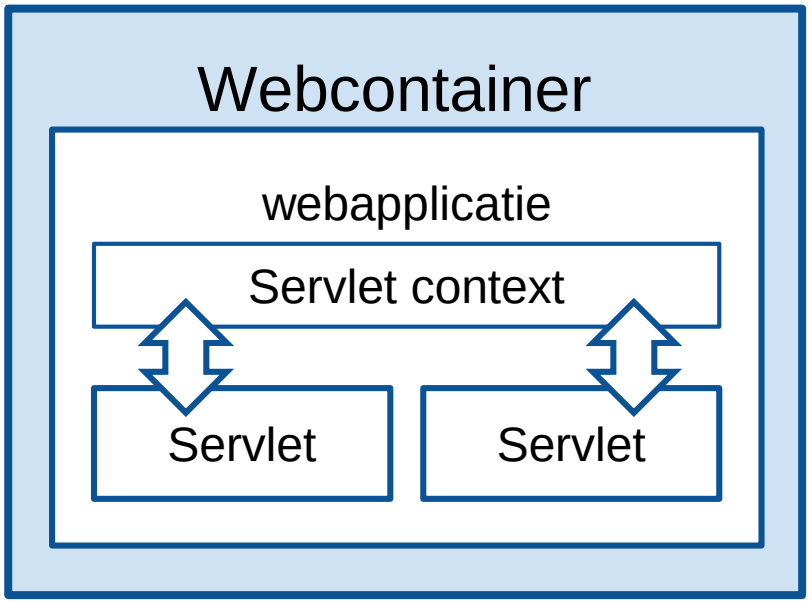
- Waarden die voor **ALLE** servlets beschikbaar zijn binnen eenzelfde webapplicatie

- Readonly

in web.xml:

```
<context-param>
        <param-name>...</param-name>
    <param-value>...</param-value>
</context-param>
```

getInitParameter()



De servlet context aanspreken vanuit een servlet

De servlet context aanspreken doe je:

Met de method **getServletContext()** die iedere servlet erft van HttpServlet.
Aanspreken in de method init().

of

Met de method **getServletContext()** van de parameter **request** van de methods doGet() en doPost()

```
ServletContext context =
 request.getSession().getServletContext();
```

Servlet context attributen

Komt één keer in het interne geheugen van de webserver voor
Wordt gedeeld door alle servlets van een website.

Synoniem: application scope variable

Iedere servlet van een website kan

- een servlet context attribuut toevoegen
- een servlet context attribuut wijzigen
- een servlet context attribuut verwijderen

Gezien meerdere gelijktijdige threads de code van servlets kunnen uitvoeren, moet je de toegang tot servlet context attributen thread-safe maken.

Servlet context attributen

De servlet context bevat methods om te werken met servlet context attributen:

setAttribute

setAttribute(naamAttribuut, waardeAttribuut)

Toevoegen nieuw attribuut of wijzigen bestaand attribuut

getAttribute

getAttribute(naamAttribuut)

De inhoud (type Object) van het attribuut

removeAttribute

removeAttribute(naamAttribuut)

Verwijderen van een attribuut

Demo

Samen opdracht 6 en 7 maken!