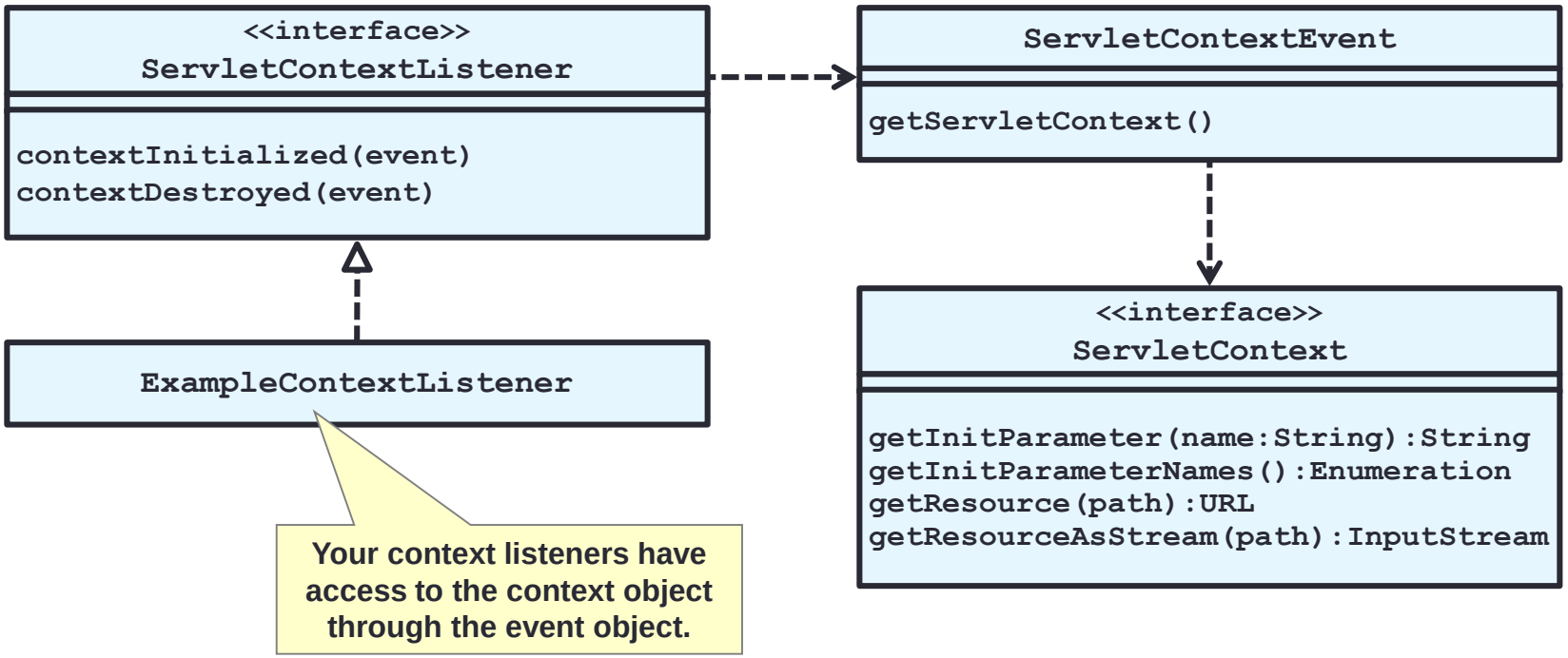
WEB ADVANCED

Servlet Event Listeners

Listener Interface	Event
javax.servlet.ServletContextListener	Initialization and destruction of the web context
javax.servlet.http.HttpSessionListener	Creation and invalidation of the session
javax.servlet.ServletRequestListener	A servlet request has started being processed by web components
javax.servlet.ServletContextAttributeListener	Attribute added, removed, or replaced on the web context (servlet context)
javax.servlet.http.HttpSessionAttributeListener	Attribute added, removed, or replaced in a session
javax.servlet.ServletRequestAttributeListener	Attribute added, removed, or replaced on the request (ServletRequest)
javax.servlet.http.HttpSessionActivationListener	Activation and passivation of the session

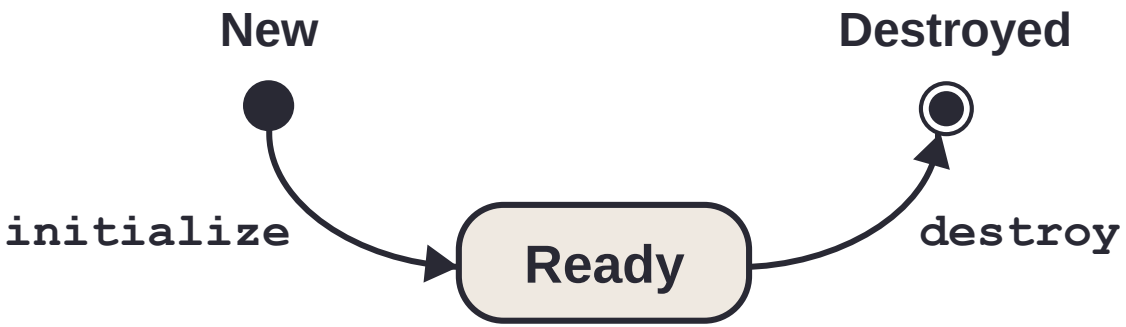
Developing a Servlet Context Listener

- `ServletContextListener` Interface



Servlet Context Listener

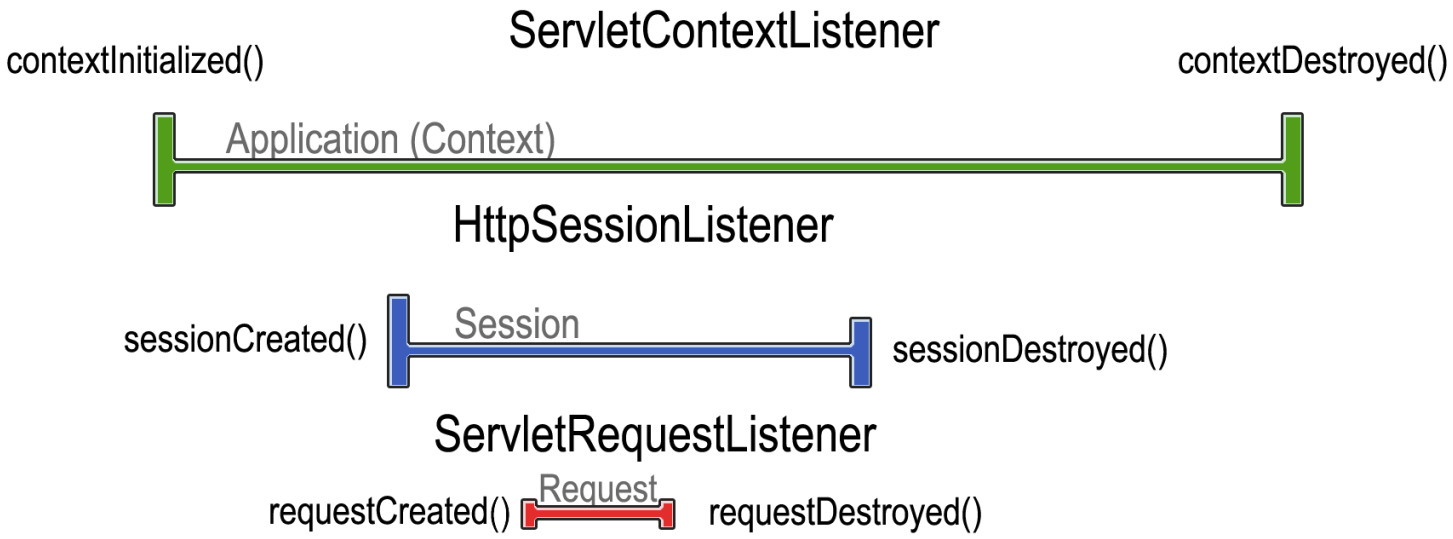
- Ciclo di vita Web Application



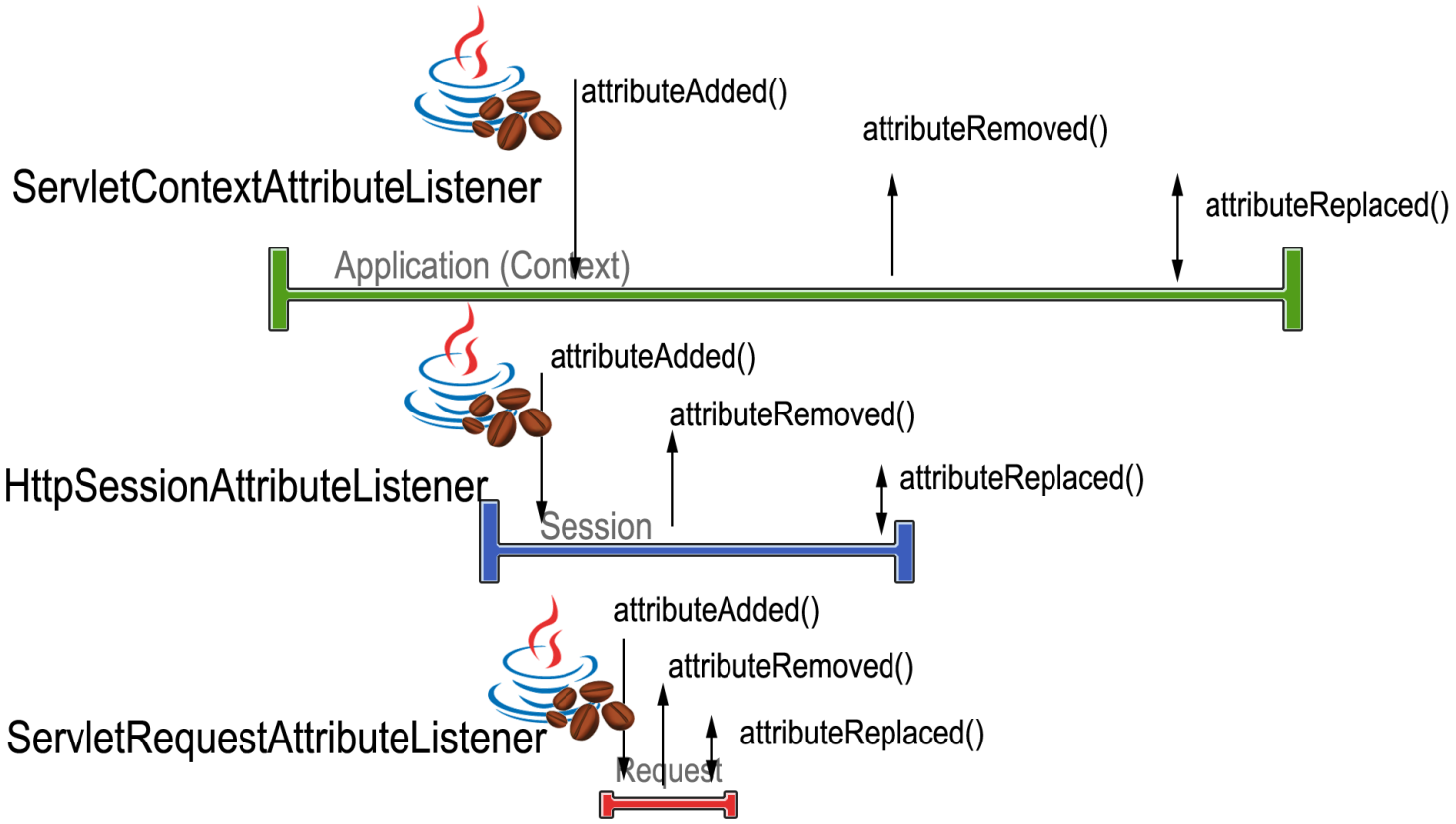
Developing a Servlet Context Listener

```
 1 package com.examples.lesson05.listeners;
 2
 3 import javax.servlet.ServletContextEvent;
 4 import javax.servlet.ServletContextListener;
 5 import javax.servlet.annotation.WebListener;
 6
 7 @WebListener()
 8 public class ExampleContextListener implements
 9              ServletContextListener {
10
11   @Override
12   public void contextInitialized(ServletContextEvent sce) {
13     sce.getServletContext().setAttribute("listenerValue",
14                             "Context Listener Value");
15   }
16
17   @Override
18   public void contextDestroyed(ServletContextEvent sce) {
19   }
20 }
```

Servlet Event Listeners



Servlet Event Listeners



HTTPSessionListener: esempio annotation

- ShoppingListSessionSetupListener

```
@WebListener
public class ShoppingListSessionSetupListener implements HttpSessionListener {

  @Override
  public void sessionCreated(HttpSessionEvent se) {
    ShopList shopList = new ShopList();
    shopList.add(createItem("Apple", "food"));
    shopList.add(createItem("Tomatoes", "food"));
    shopList.add(createItem("Radio", "electronics"));
    shopList.add(createItem("Flashlight", "electronics"));
    shopList.add(createItem("Mop", "cleaning"));
    shopList.add(createItem("Broom", "cleaning"));
    shopList.add(createItem("Screwdriver", "misc"));
    shopList.add(createItem("Light bulb", "misc"));
    shopList.add(createItem("Duct tape", "misc"));
    se.getSession().setAttribute("shopList", shopList);
  }

  @Override
  public void sessionDestroyed(HttpSessionEvent se) {
  }

  private ShopItem createItem(String name, String cat) {
    final ShopItem shopItem = new ShopItem();
    shopItem.setCategory(cat);
    shopItem.setName(name);
    shopItem.add();
    return shopItem;
  }
}
```

HTTPSessionListener: esempio annotation

- ShopListView

```
</thead>
<tbody>
  <c:forEach items="${sessionScope.shopList.items}" var="item" varStatus="status">
    <c:choose>
      <c:when test="${item.category eq 'food'}">
        <c:set var="rowColor" value="${((status.count mod 2) eq 0)?'#f0ebb8':'#ded890'}"/>
      </c:when>
      <c:when test="${item.category eq 'electronics'}">
        <c:set var="rowColor" value="${((status.count mod 2) eq 0)?'#b7c0e8':'#9093de'}"/>
      </c:when>
      <c:when test="${item.category eq 'cleaning'}">
        <c:set var="rowColor" value="${((status.count mod 2) eq 0)?'#b7e8b9':'#90de93'}"/>
      </c:when>
      <c:otherwise>
        <c:set var="rowColor" value="${((status.count mod 2) eq 0)?'#ffffff':'#f0f0f0'}"/>
      </c:otherwise>
    </c:choose>
    <tr style="background-color: ${rowColor}">
      <td style="background-color: ${rowColor}">${item.name}</td>
      <td style="background-color: ${rowColor}">${item.count}</td>
```

HTTPSessionListener

- ShopListView

```
        <td>
          <form action="ShopListServlet" method="POST">
            <input type="hidden" name="id" value="${item.id}"/>
            <input type="hidden" name="action" value="add"/>
            <input type="submit" value="+"/>
          </form>
        </td>
        <td>
          <form action="ShopListServlet" method="POST">
            <input type="hidden" name="id" value="${item.id}"/>
            <input type="hidden" name="action" value="remove"/>
            <input type="submit" value="-"/>
          </form>
        </td>
        <td>
          <form action="ShopListServlet" method="POST">
            <input type="hidden" name="id" value="${item.id}"/>
            <input type="hidden" name="action" value="delete"/>
            <input type="submit" value="Delete"/>
          </form>
        </td>
      </tr>
    </c:forEach>
  </tbody>
</table>
```

HTTPSessionListener

- ShopListServlet

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    request.getRequestDispatcher("/shopListView.jsp").forward(request, response);
  }

protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    String action = request.getParameter("action");
    ShopList shopList = (ShopList) session.getAttribute("shopList");
    String message = null;
    ShopItem modifiedItem = null;
    if ("new".equals(action)) {
      modifiedItem = new ShopItem();
      modifiedItem.setName(request.getParameter("itemName"));
      modifiedItem.setCategory(request.getParameter("itemCategory"));
      modifiedItem.add();
      shopList.add(modifiedItem);
      message = "New item:";
     }
    if ("add".equals(action)) {…}
    if ("remove".equals(action)) {…}
    if ("delete".equals(action)) {…}
    request.setAttribute("message", message);
    request.setAttribute("modified-item", modifiedItem);
    doGet(request, response);
  }
}
```

HTTPSessionListener

- **Bean ShopList**

```
public class ShopList {

  private int id = 1;
  private String name;
  private final List<ShopItem> items;

  public ShopList() {
    items = new ArrayList<ShopItem>();
  }
  public void add(ShopItem item) {
    item.setId(id);
    id++;
    items.add(item);
  }
  public ShopItem get(int id) {…}
  public ShopItem remove(int id) {…}
  public List<ShopItem> getItems() { return items;  }
}
```

- Bean ShopItem

```
public class ShopItem {

  private int id;
  private String name;
  private String category;
  private int count;

  public int getId() {return id;  }
  public void setId(int id) {    this.id = id;  }

  public String getName() {    return name;  }
  public void setName(String name) {this.name = name;  }

  public int getCount() { return count;  }
  public void setCount(int count) {    this.count = count;  }

  public String getCategory() {    return category;  }
  public void setCategory(String category) {
      this.category = category; }

  public void add() {    count++;  }

  public void remove() {    count--;  }
}
```

Internazionalizzazione

- È possibile che un testo contenuto in una pagina, inclusi i messaggi di errore, debba essere localizzato per l'utente corrente. JSP consente di utilizzare bundle di messaggi localizzati con una pagina in modo semplice.
- Per utilizzare i18n in JSP:
  - Creare file di properties per ogni lingua
  - I nomi di file devono includere le impostazioni nazionali supportate, ad esempio `messages.properties` (file predefinito), `messages_en_US.properties`, `messages_de.properties`
  - I file dei messaggi devono essere inseriti nel `CLASSPATH` (nella directory di un package Java nell'IDE)
  - Utilizzare il linguaggio EL o le librerie fmt nelle pagine JSF per leggere un messaggio localizzato

Demo sull'internazionalizzazione

- Un file A `messages.properties` può contenere:
  `greeting=Howdy`
- All'interno di una pagina JSP, è possibile visualizzare il messaggio di saluto localizzato utilizzando
  le librerie fmt

```
<fmt:setLocale value="en"/>
<fmt:setBundle basename="it.esempio.i18n.messages" var="messages"/>
<fmt:message bundle="${messages}" key="item"/>
```

- I messaggi localizzati vengono utilizzati anche per
  i messaggi di errore di conversione e convalida.

Internazionalizzazione:esempio

- ShopListFormatted.jsp

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
  <head>
<fmt:setLocale value="${param.lang}"/>
<fmt:setBundle basename="it.esempio.i18n.shoplist" var="messages"/>
<h4><fmt:message bundle="${messages}" key="madeWith"/></h4>
<fmt:message var="title" bundle="${messages}" key="title"/>
   <title>${title}</title>
 </head>
 <h1>${title}</h1><table>
 <thead>
   <tr>
     <th><fmt:message bundle="${messages}" key="item"/></th>
     <th><fmt:message bundle="${messages}" key="quantity"/></th>
   </tr>
 </thead>
 <tbody>
   <c:forEach items="${sessionScope.shopList.items}" var="item" varStatus="status">
    […]         </c:forEach>
 </tbody>
</table>
<fmt:message bundle="${messages}" key="count">
 <fmt:param value="${itemCount}"/>
</fmt:message>
<br>

</body>
</html>
```

Internazionalizzazione:esempio

- it.esempio.i18n.ShopListFormatted_en.properties

```
madeWith=Internationalization and formatting using JSTL
title=Shopping List
item=Item
quantity=Qty
count=There are {0} items in your shopping list.
```

- it.esempio.i18n.ShopListFormatted_es.properties

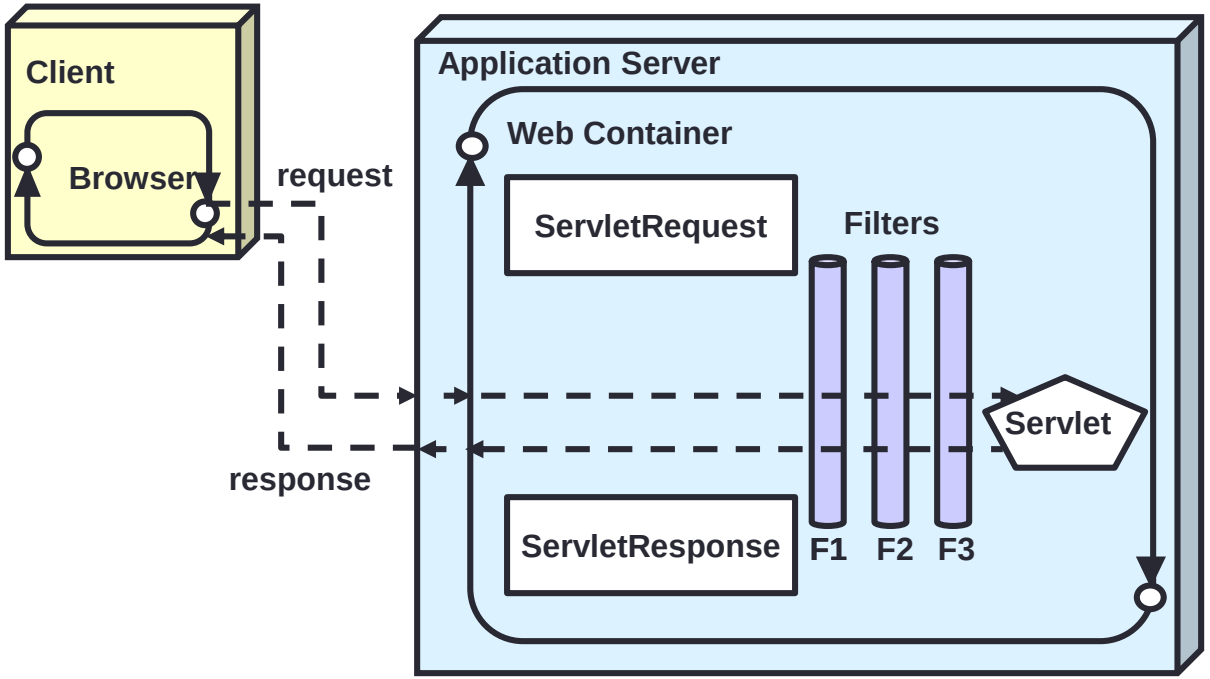
```
madeWith=Internacionalizaci\u00f3n y formato con JSTL
title=Lista de compras
item=Art\u00edculo
quantity=Cantidad
count=Hay {0} art\u00edculos en tu lista.
```

- it.esempio.i18n.ShopListFormatted_it.properties

```
madeWith=Internazionalizzazione
title=Carrello
item=Articolo
quantity=Qta
count=Ci sono {0} articoli nel tuo carrello.
```

- http://localhost:8080/JSPServletAdv/shopListFormatted_one.jsp?lang=en
- http://localhost:8080/JSPServletAdv/shopListFormatted_one.jsp?lang=es
- http://localhost:8080/JSPServletAdv/shopListFormatted_one.jsp?lang=it

Filtri


Client
Browser
request
response
Application Server
Web Container
ServletRequest
ServletResponse
Filters
F1
F2
F3
Servlet

Updating the Request Data Filter: Example

```
1  package com.examples.lesson09.filters;
2
3  import java.io.IOException;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6  import javax.servlet.Filter;
7  import javax.servlet.FilterChain;
8  import javax.servlet.FilterConfig;
9  import javax.servlet.ServletException;
10 import javax.servlet.ServletRequest;
11 import javax.servlet.ServletResponse;
12 import javax.servlet.annotation.WebFilter;
13 import javax.servlet.annotation.WebInitParam;
14
15 @WebFilter(filterName = "RequestDataFilter", urlPatterns = {"*.jsp"},
16 initParams = {
17   @WebInitParam(name = "server-name", value = "Example Server")
18 })
19
20 public class RequestDataFilter implements Filter {
21
22   private String serverName;
23   private long lastExecTime = 0;
```

Updating the Request Data Filter: Example

```
24   @Override
25   public void doFilter(ServletRequest request, ServletResponse
          response, FilterChain chain)
26        throws IOException, ServletException {
27     Date now = new Date();
28
29     request.setAttribute("server-name", serverName);
30     request.setAttribute("server-date", new SimpleDateFormat("
                              dd MMM yyyy").format(now));
31     request.setAttribute("server-time", new SimpleDateFormat("
                              HH:mm:ss.SSS").format(now));
32     request.setAttribute("server-lastExecTime", lastExecTime);
33
34     long startTime = now.getTime();
35     chain.doFilter(request, response);
36     lastExecTime = System.currentTimeMillis() - startTime;
37
38   }
39
40
```

Updating the Request Data Filter: Example

```
41    @Override
42    public void destroy() {
43      serverName = null;
44    }
45
46    @Override
47    public void init(FilterConfig filterConfig) {
48      serverName = filterConfig.getInitParameter("server-name");
49    }
50  }
51
```

Configurazione del filtro

- Mediante annotation:

```
@WebFilter(filterName="RequestDataFilter",
urlPatterns={"*.jsp"},
dispatcherTypes={
   DispatcherType.FORWARD,
   DispatcherType.ERROR,
   DispatcherType.REQUEST,
   DispatcherType.INCLUDE},
initParams={
   @WebInitParam(
     name="server-name",
     value="Example Server"
    )
})
```

Configurazione del filtro

- Mediante deployment descriptor:

```
<filter>
    <filter-name>RequestDataFilter</filter-name>
    <filter-class>
       com.examples.lesson09.filters.RequestDataFilter
    </filter-class>
    <init-param>
       <param-name>server-name</param-name>
       <param-value>Example Server</param-value>
     </init-param>
</filter>
```

Configurazione del filtro

```
<filter-mapping>
  <filter-name>RequestDataFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

- Il tag `<servlet-name>` può sostituire `<url-pattern>` per mapping rispetto al nome della servlet.
- Più filtri possono essere applicati alla stessa risorse, creando un Filter Chaining.

Filter Chaining

```
<servlet-mapping>
  <servlet-name>FrontController</servlet-name>
  </url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<filter-mapping>
  <filter-name>perfFilter</filter-name>
  <servlet-name>FrontController</servlet-name>
</filter-mapping>
```

```
<filter-mapping>
  <filter-name>auditFilter</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>
```

```
<filter-mapping>
  <filter-name>transformFilter</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>
```

Filtro

```
@WebFilter(filterName = "AccessFilter", urlPatterns = {
//Several filtered patterns
  "/page_header.jsp",  "/page_footer.jsp","/accessDenied.jsp"
})
public class AccessFilter implements Filter {

  @Override
  public void doFilter(ServletRequest request, ServletResponse response,
          FilterChain chain)
          throws IOException, ServletException {
        request.getRequestDispatcher("accessDenied.jsp")
        .forward(request, response);
  }

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
  }

  @Override
  public void destroy() {
  }
}
```

MVC with Dependency Injection

- Service `InfoService`:

```
1  public interface InfoService {
2
3    public String calculateServerName();
4
5    public String calculateServerMemory();
6
7    public String calculateServerFreeMemory();
8
9    public String calculateServerCores();
10
11   public String calculateServerTime();
12 }
13
```

MVC with Dependency Injection

- Service `InfoServiceImpl`:

```
1 @ApplicationScoped
2 public class InfoServiceImpl implements InfoService {
3
4   @Override
5   public String calculateServerName() {
6     return "TEST SERVER";
7   }
8
9   @Override
10  public String calculateServerMemory() {
11    return formatBytes(Runtime.getRuntime().maxMemory());
12  }
13
14  @Override
15  public String calculateServerFreeMemory() {
16   return formatBytes(Runtime.getRuntime().freeMemory());
17  }
```

MVC with Dependency Injection

- Service `InfoServiceImpl`:

```
18    @Override
19    public String calculateServerCores() {
20      return Integer.toString(Runtime.getRuntime().
                                availableProcessors());
21    }
22
23    @Override
24    public String calculateServerTime() {
25      return new SimpleDateFormat().format(new Date());
26    }
27
28    private String formatBytes(long bytes) {
29      …
30    }
31
32
```

MVC with Dependency Injection

- Controller `InfoServletCdiController`:

```
1 @WebServlet(name = "InfoServletCdiController", urlPatterns =
              {"/infoCdiController"})
2 public class InfoServletCdiController extends HttpServlet {
3
4   @Inject
5   private InfoService infoService;
6
7   @Override
8   protected void doGet(HttpServletRequest request, HttpServletResponse
                         response)
9       throws ServletException, IOException {
10
11     InfoBean infoBean = new InfoBean();
12     infoBean.setServerCores(infoService.calculateServerCores());
13     infoBean.setServerFreeMemory(
                          infoService.calculateServerFreeMemory());
14
```

MVC with Dependency Injection

```
15      infoBean.setServerMemory(infoService.calculateServerMemory());
16      infoBean.setServerName(infoService.calculateServerName());
17      infoBean.setServerTime(infoService.calculateServerTime());
18
19      request.setAttribute("infoBean", infoBean);
20
21      RequestDispatcher rd =
                    request.getRequestDispatcher("/infoServletView.jsp");
22      rd.forward(request, response);
23    }
24  }
25
26
```

- **WEB-INF/beans.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

MVC with Dependency Injection

- infoServletView.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>System information</title>
    <link rel="stylesheet" href="res/styles.css" type="text/css">
  </head>
  <body>
    <h1>System Information</h1>
    <p>JSP view</p>
    <ul>
      <li>Server Name: ${infoBean.serverName}</li>
      <li>Server Memory: ${infoBean.serverMemory}</li>
      <li>Server Free Memory ${infoBean.serverFreeMemory}</li>
      <li>Server Cores: ${infoBean.serverCores}</li>
      <li>Server Time: ${infoBean.serverTime}</li>
    </ul>
    <a href="index.html">Go Home</a>
  </body>
</html>
```

Grazie