互联网应用开发技术

Web Application Development

第7课
WEB后端–SERVLET

Episode Seven

Servlet

陈昊鹏

chen-hp@sjtu.edu.cn

Web Application

Development

- Servlet
  - What is Servlet
  - Servlet API
  - Examples

RE*in*

REliable, INtelligent & Scalable Systems

- Web applications are of the following types:
  - Presentation-oriented: A presentation-oriented web application generates interactive web pages containing various types of markup language (HTML, XHTML, XML, and so on) and dynamic content in response to requests.
  - Service-oriented: A service-oriented web application implements the endpoint of a web service. Presentation-oriented applications are often clients of service-oriented web applications.
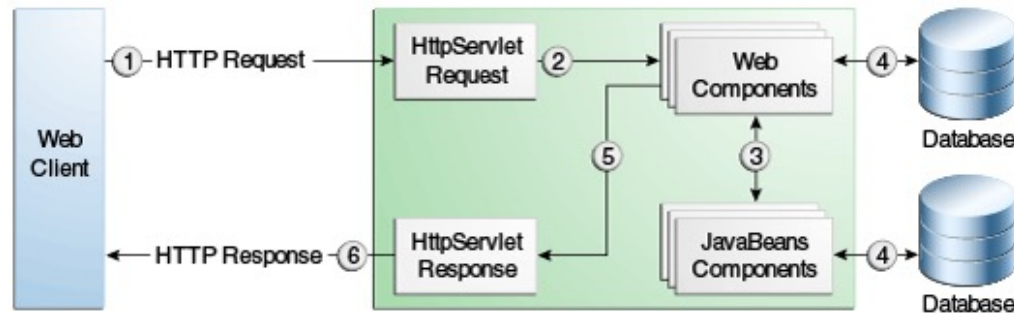
- A web application consists of
  - web components;
  - static resource files, such as images and cascading style sheets (CSS);
  - and helper classes and libraries.

- The web container provides many supporting services that enhance the capabilities of web components and make them easier to develop.

- However,
  - because a web application must take these services into account, the process for creating and running a web application is different from that of traditional stand-alone Java classes.

- The process for creating, deploying, and executing a web application can be summarized as follows:
  1. Develop the web component code.
  2. Develop the web application deployment descriptor, if necessary.
  3. Compile the web application components and helper classes referenced by the components.
  4. Optionally, package the application into a deployable unit.
  5. Deploy the application into a web container.
  6. Access a URL that references the web application.

```
servlet

command
        servlet
```

- A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
  - Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.
  - For such applications, Java Servlet technology defines HTTP-specific servlet classes.

- The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets.
  - All servlets must implement the `Servlet` interface, which defines lifecycle methods.
  - When implementing a generic service, you can use or extend the `GenericServlet` class provided with the Java Servlet API.
  - The `HttpServletclass` provides methods, such as `doGet` and `doPost`, for handling HTTP-specific services.

- Container types
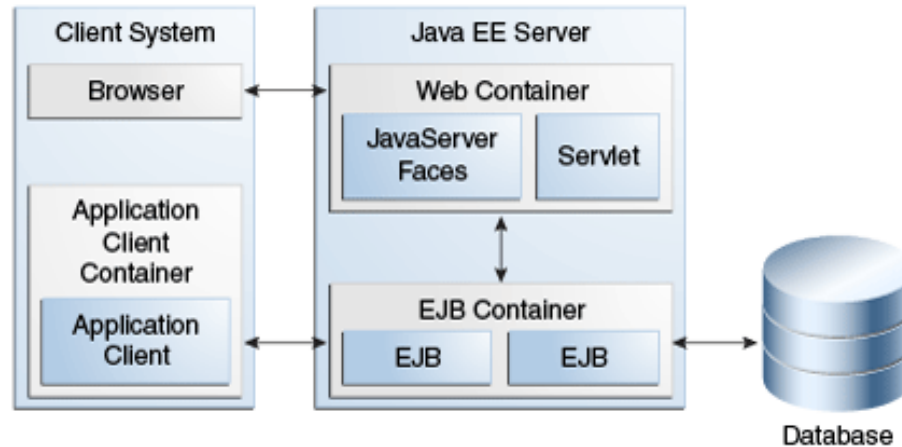
- The following is a typical sequence of events:
  1. A client (e.g., a Web browser) accesses a Web server and makes an HTTP request.
  2. The request is received by the Web server and handed off to the servlet container.
     - The servlet container can be running in the same process as the host Web server, in a different process on the same host, or on a different host from the Web server for which it processes requests.
  3. The servlet container determines which servlet to invoke based on the configuration of its servlets, and calls it with objects representing the request and response.
  4. The servlet uses the request object to find out who the remote user is, what HTTP POST parameters may have been sent as part of this request, and other relevant data.
     - The servlet performs whatever logic it was programmed with, and generates data to send back to the client. It sends this data back to the client via the response object.
  5. Once the servlet has finished processing the request, the servlet container ensures that the response is properly flushed, and returns control back to the host Web server.

- The lifecycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps.

1. If an instance of the servlet does not exist, the web container
    - Loads the servlet class.
    - Creates an instance of the servlet class.
    - Initializes the servlet instance by calling the `init` method.
2. Invokes the `service` method, passing request and response objects.

- If it needs to remove the servlet, the container finalizes the servlet by calling the servlet's `destroy` method.

Creating a Servlet

- Use the @WebServlet annotation to define a servlet component in a web application.
  - The annotated servlet must specify at least one URL pattern.
  - This is done by using the urlPatterns or value attribute on the annotation.

```java
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
@WebServlet("/report")
public class MoodServlet extends HttpServlet {
  ...
```

- The web container initializes a servlet after loading and instantiating the servlet class and before delivering requests from clients.

- The `Servlet` interface is the central abstraction of the Java Servlet API.
  - All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface.
  - The two classes in the Java Servlet API that implement the Servlet interface are `GenericServlet` and `HttpServlet`.
  - For most purposes, Developers will extend `HttpServlet` to implement their servlets.

HTTP Specific Request Handling Methods

- `HttpServlet` class has methods to aid in processing HTTP-based requests:
  - `doGet` for handling HTTP GET requests
  - `doPost` for handling HTTP POST requests
  - `doPut` for handling HTTP PUT requests
  - `doDelete` for handling HTTP DELETE requests
  - `doHead` for handling HTTP HEAD requests
  - `doOptions` for handling HTTP OPTIONS requests
  - `doTrace` for handling HTTP TRACE requests

- Typically when developing HTTP-based servlets, a Servlet Developer will only concern himself with the `doGet` and `doPost` methods.

```java
@WebServlet("/report")
public class MoodServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html lang=\"en\">");
            out.println("<head>");
            out.println("<title>Servlet MoodServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet MoodServlet at " + request.getContextPath() + "</h1>");
            request.setAttribute("mood", "sleep");
            String mood = (String) request.getAttribute("mood");
            out.println("<p>Duke's mood is: " + mood + "</p>");
            if (mood.equals("sleepy")) {
                out.println("<img src=\"resources/images/duke.snooze.gif\" alt=\"Duke sleeping\"/><br/>");
            } else{
                
                ……
            }
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    processRequest(request, response);
}

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

}
```
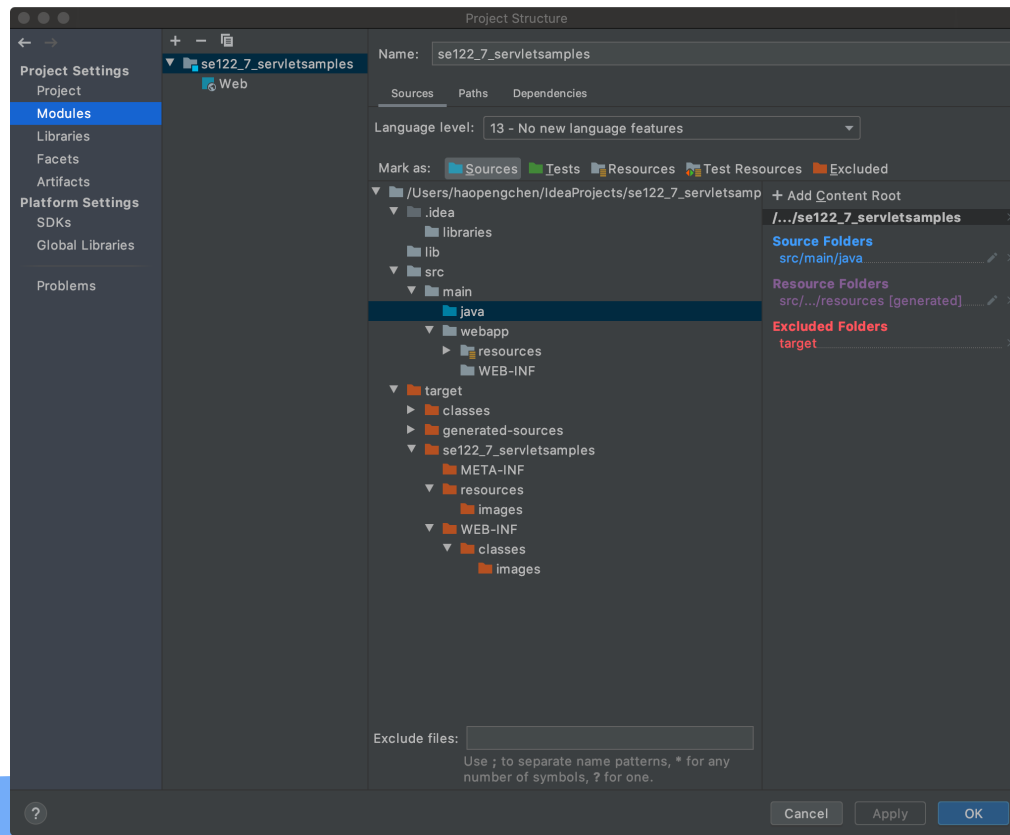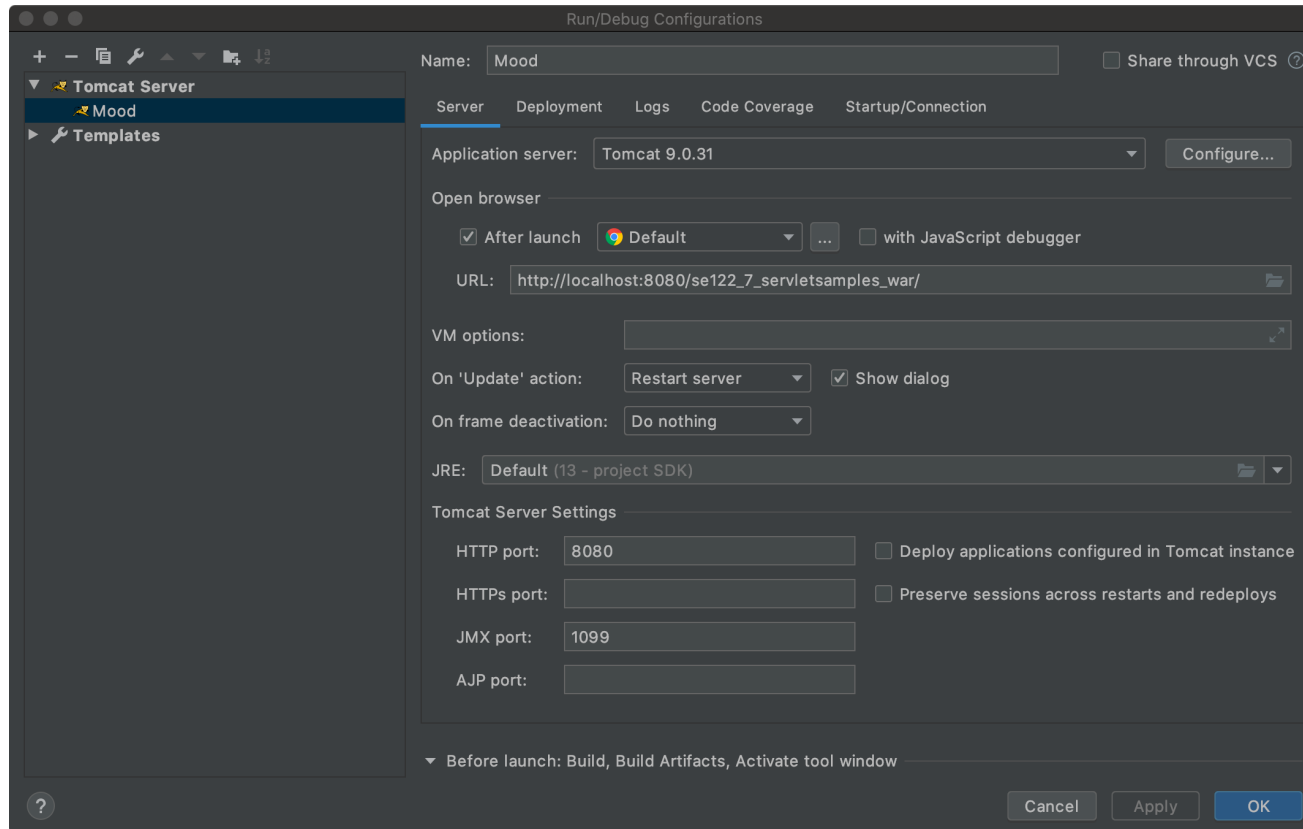
Servlet Sample
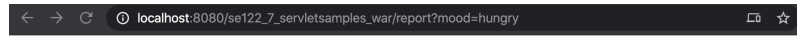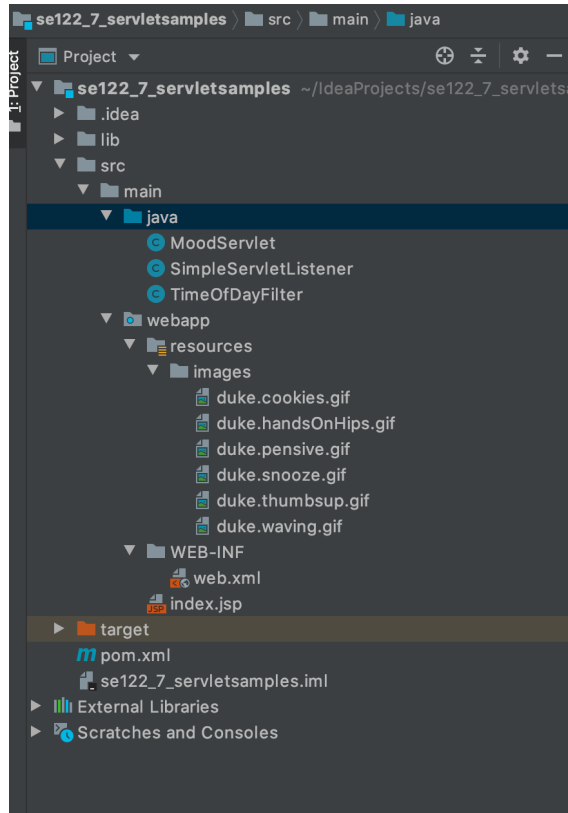
- New Maven Project – Create from archtetype
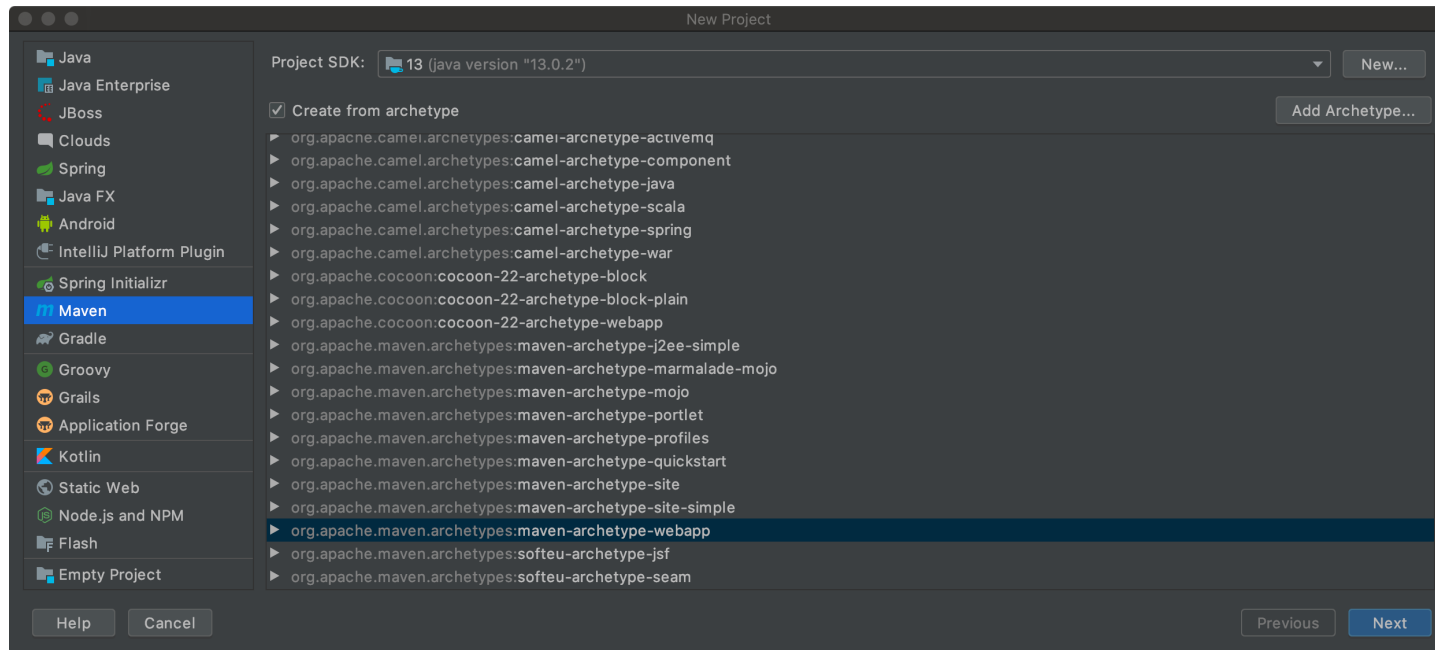
Servlet Sample

- Create **source** and **resource** folders

An Example: mood
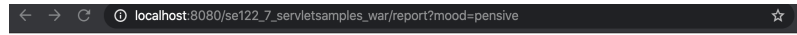
- The generated html page:

```
<html lang="en">
    <head>
        <title>Servlet MoodServlet</title>
    </head>
    <body>
        <h1>Servlet MoodServlet at /ServletMood</h1>
        <p>Duke's mood is: sleepy</p>
        <img src="resources/images/duke.snooze.gif" alt="Duke sleeping"/><br/>
    </body>
</html>
```

- A **filter** is an object that can transform the header and content (or both) of a request or response.
  - Filters differ from web components in that filters usually do not themselves create a response.
  - Instead, a filter provides functionality that can be "attached" to any kind of web resource.
  - Consequently, a filter should not have any dependencies on a web resource for which it is acting as a filter; this way, it can be composed with more than one type of web resource.

- The main tasks that a filter can perform are as follows:
  - Query the request and act accordingly.
  - Block the request-and-response pair from passing any further.
  - Modify the request headers and data. You do this by providing a customized version of the request.
  - Modify the response headers and data. You do this by providing a customized version of the response.
  - Interact with external resources.

- You define a filter by implementing the `Filter` interface.

```java
import javax.servlet.Filter;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

@WebFilter(filterName = "TimeOfDayFilter",
 urlPatterns = {"/*"},
 initParams = {
    @WebInitParam(name = "mood", value = "awake")}
)
public class TimeOfDayFilter implements Filter { ....
```
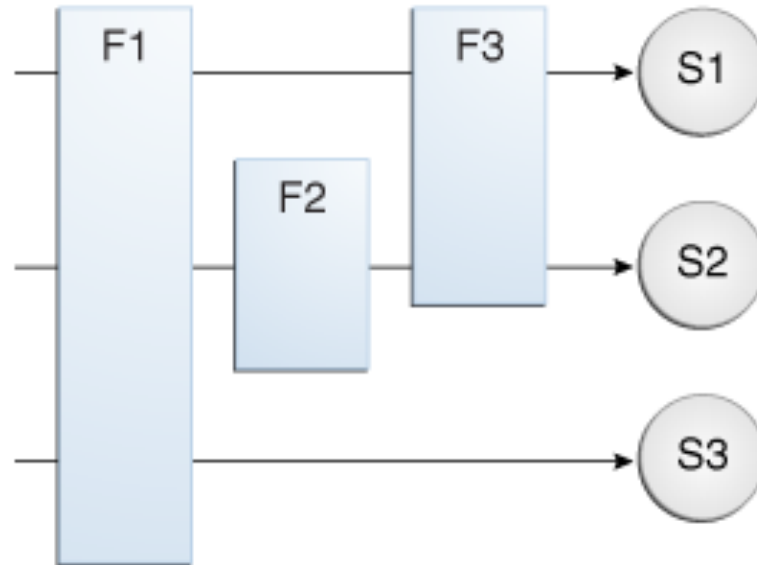
```java
public class TimeOfDayFilter implements Filter {
    String mood = null;
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        mood = filterConfig.getInitParameter("mood");
    }
    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
                    throws IOException, ServletException {
        Calendar cal = GregorianCalendar.getInstance();
        switch (cal.get(Calendar.HOUR_OF_DAY)) {
            case 23:case 24:case 1:case 2:case 3:case 4:case 5:case 6:
                mood = "sleepy";
                break;

            …………
        }
        req.setAttribute("mood", mood);
        chain.doFilter(req, res);
    }
    @Override
    public void destroy() {
    }
}
```

```
filter
```

An Example: mood

Handling Servlet Lifecycle Events
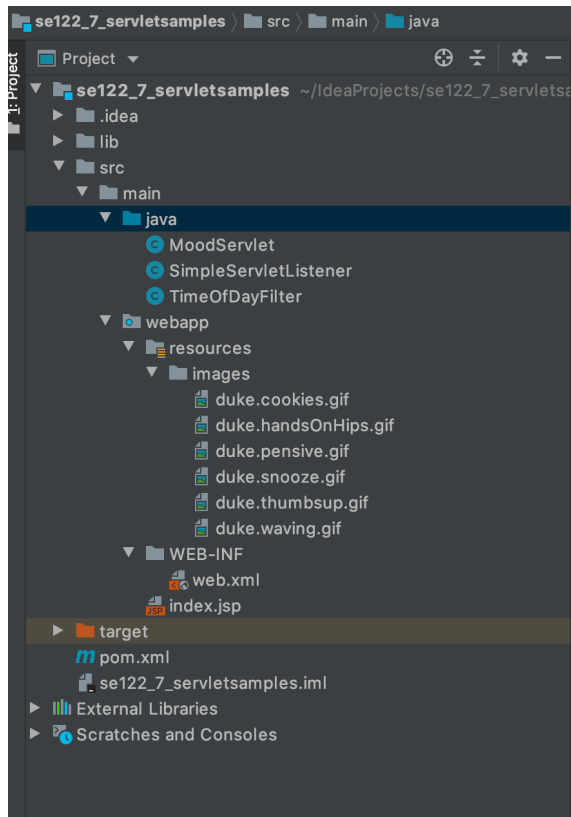
Object	Event	Listener Interface and Event
Web context	Initialization and destruction	javax.servlet.ServletContextListener and ServletContextEvent
Web context	Attribute added, removed, or replaced	javax.servlet.ServletContextAttributeListener and ServletContextAttributeEvent
Session	Creation, invalidation, activation, passivation, and timeout	javax.servlet.http.HttpSessionListener, javax.servlet.http.HttpSessionActivationListener, and HttpSessionEvent
Session	Attribute added, removed, or replaced	javax.servlet.http.HttpSessionAttributeListener and HttpSessionBindingEvent
Request	A servlet request has started being processed by web components	javax.servlet.ServletRequestListener and ServletRequestEvent
Request	Attribute added, removed, or replaced	javax.servlet.ServletRequestAttributeListener and ServletRequestAttributeEvent

```java
@WebListener()
public class SimpleServletListener implements ServletContextListener,
        ServletContextAttributeListener {

    static final Logger log =
            Logger.getLogger("mood.web.SimpleServletListener");

    @Override
    public void attributeAdded(ServletContextAttributeEvent event) {
        log.log(Level.INFO, "Attribute {0} has been added, with value: {1}",
                new Object[]{event.getName(), event.getValue()});
    }

    @Override
    public void attributeRemoved(ServletContextAttributeEvent event) {
        log.log(Level.INFO, "Attribute {0} has been removed",
                event.getName());
    }

    @Override
    public void attributeReplaced(ServletContextAttributeEvent event) {
        log.log(Level.INFO, "Attribute {0} has been replaced, with value: {1}",
                new Object[]{event.getName(), event.getValue()});
    }
}
```
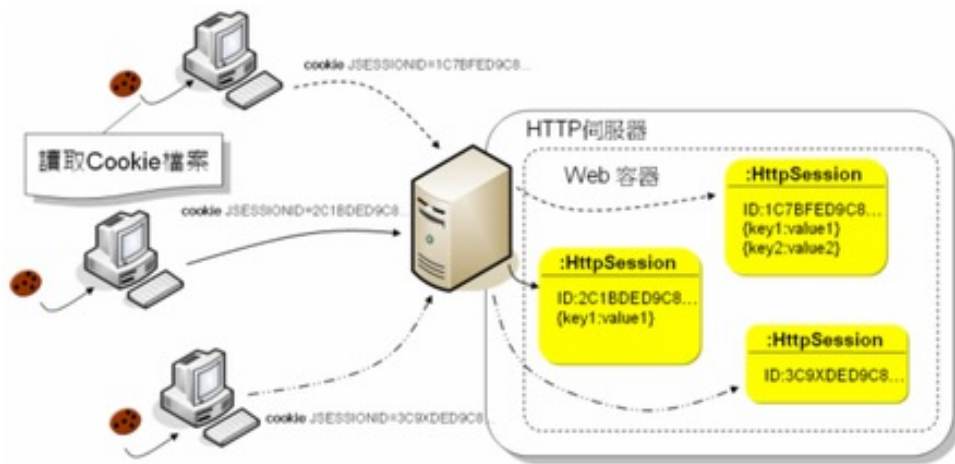
- **HttpRequest Attribute**
  ```
  request.setAttribute("mood", "sleepy");
  String mood = (String) request.getAttribute("mood");
  http://localhost:8080/ServletMood/report
  ```

- **HttpRequest Parameter**
  ```
  request.setParameter("mood", "sleepy");
  ```
  or
  ```
  http://localhost:8080/ServletMood/report?mood=sleepy

  String mood = (String) request.getParameter("mood");
  ```

REliable, INtelligent & Scalable Systems

- Many applications require that a series of requests from a client be associated with one another.
  - For example, a web application can save the state of a user's shopping cart across requests.
  - Web-based applications are responsible for maintaining such state, called a **session**, because HTTP is stateless.
  - To support applications that need to maintain state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions.

- Sessions are represented by an `HttpSession` object.
  - You access a session by calling the `getSession` method of a request object.
  - This method returns the current session associated with this request; or, if the request does not have a session, this method creates one.

REliable, INtelligent & Scalable Systems

- You can associate <mark>object-valued attributes</mark> with a <mark>session by name</mark>.
  - Such attributes are accessible by any web component that belongs to the same web context *and* is handling a request that is part of the same session.

```
HttpSession hs = request.getSession();
Shoppingcart cart = new Shoppingcart();
hs.setAttribute("cart", cart);
cart.add(item);

HttpSession hs = request.getSession();
Shoppingcart cart = hs.getAttribute("cart");
List<Object> ls = cart.getItems();
```

Finalizing a Servlet

- The web container may determine that a servlet should be removed from service
  - for example, when a container wants to reclaim memory resources or when it is being shut down.
  - In such a case, the container calls the `destroy` method of the `Servlet` interface.
  - In this method, you release any resources the servlet is using and save any persistent state. The `destroy` method releases the database object created in the `init` method.

- A servlet's service methods should all be complete when a servlet is removed.
  - The server tries to ensure this by calling the destroy method only after all service requests have returned or after a server-specific grace period, whichever comes first.

- The Java EE 8 Tutorial – Distributed Multitiered Applications
  - https://javaee.github.io/tutorial/overview004.html#BNAAY
- The Java EE 8 Tutorial – Java Servlet Technology
  - https://javaee.github.io/tutorial/servlets.html#BNAFD
- Java Servlet Specification
  - http://download.oracle.com/otn-pub/jcp/servlet-3_1-fr-eval-spec/servlet-3_1-final.pdf?AuthParam=1396178605_99ca893c1d2485541a2baf6a88414bf8

- *Web开发技术*
- *Web Application Development*

Thank You!