OCEJWCD

Java EE 6 Web Profile

Java EE 6 Web Profile SDK

- Servlet 3.0
- JavaServer Pages (JSP) 2.2
- Expression Language (EL) 2.2
- Debugging Support for Other Languages (JSR-45) 1.0
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JavaServer Faces (JSF) 2.0
- Common Annotations for Java Platform (JSR-250) 1.1
- Enterprise JavaBeans (EJB) 3.1 Lite
- Java Transaction API (JTA) 1.1
- Java Persistence API (JPA) 2.0
- Bean Validation 1.0
- Managed Beans 1.0
- Interceptors 1.1
- Contexts and Dependency Injection for the Java EE Platform (CDI) 1.0
- Dependency Injection for Java 1.0

JSR 315 (Servlet 3.0)

- Extensibility and web framework pluggability (with web.xml fragments)
- Ease of development (EoD) through annotations:
    - @WebServlet
    - @WebFilter
    - @WebListener
- * Asynchronous processing

Deployment Descriptors

- The web.xml file is optional beginning with Java EE 6. Developers can use in-code annotations to configure components.
- web.xml in WEB-INF/ in a WAR
- web-fragment.xml in META-INF/ in a JAR or in WEB- INF/lib in a WAR
- Annotations in source files

web-fragment.xml example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-fragment 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/web-fragment_3_0.xsd"
version="3.0">
    <name>WebFragment1</name>
    <servlet>
        <servlet-name>WebFragmentServlet</servlet-name>
        <servlet-class>
            net.hello.util.WebFragmentServlet
        </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>WebFragmentServlet</servlet-name>
        <url-pattern>/WebFragmentServlet</url-pattern>
    </servlet-mapping>
</web-fragment>
```

Configuration order

```
<web-app ...>
    <absolute-ordering>
        <name>WebFragment1</name>
        <name>WebFragment2</name>
    </absolute-ordering>
    ...
</web-app>
```

```
<web-fragment ...>
    <name>WebFragment1</name>
    <ordering>
        <after>
            <name>WebFragment2</name>
        </after>
    </ordering>
    ...
</web-fragment>
```

```
<web-fragment ...>
    <name>WebFragment2</name>
    ..
</web-fragment>
```

```
<web-fragment ...>
    <name>WebFragment3</name>
    <ordering>
        <before><others/></before>
    </ordering>
    ..
</web-fragment>
```

metadata-complete

```
<web-app 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/web-app_3_0.xsd"
version="3.0"
         metadata-complete="true">
    ...
</web-app>
```

Servlet Mapping

- Multiple and wildcard URL patterns
- Using annotations
- ServletContext.getNamedDispatcher()

```
RequestDispatcher rd =
    getServletContext().getNamedDispatcher("MyServletname");
rd.forward(request, response);
```

HTTP Methods

- **OPTIONS**: Request the communication options available on the request/response chain
- **GET**: Request to retrieve information identified by the Request-URL
- **HEAD**: Identical to the GET except that it does not return a message body, only the headers
- **POST**: Request for the server to accept the entity enclosed in the body of the HTTP message
- **PUT**: Request for the server to store the entity enclosed in the body of the HTTP message
- **DELETE**: Request for the server to delete the resource identified by the request URI
- **TRACE**: Request for the server to invoke an application layer loop-back of the request message
- **CONNECT**: Reserved for use with a proxy that can switch to being a tunnel

@WebServlet url pattern

```
@WebServlet("/HelloServlet")
@WebServlet(value="/HelloServlet")
@WebServlet(urlPatterns="/HelloServlet")
```

```
@WebServlet({"/HelloServlet1", "/HelloServlet2"})
@WebServlet(value={"/HelloServlet1", "/HelloServlet2"})
@WebServlet(urlPatterns={"/HelloServlet1", "/HelloServlet2"})
```

@WebServlet
Init Parameters

```
@WebServlet(
    "/HelloServlet",
    initParams={
        @WebInitParam(name = "name", value = "Simon"),
        @WebInitParam(name = "email", value = "simon@hello.com")
    }
)
```

web.xml overrule @WebServlet

```
<servlet>
    <servlet-name>HelloServlet03</servlet-name>
    <servlet-class>net.hello.servlets.HelloServlet03</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet03</servlet-name>
    <url-pattern>/HelloServlet03</url-pattern>
</servlet-mapping>
```

```
@WebServlet(name="HelloServlet03", urlPatterns={"/HelloServlet031"})
public class HelloServlet03 extends HttpServlet {
    ...
}
```

@WebFilter annotation

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

@WebFilter Attributes

- **filterName**: Defines a filter name
- **urlPatterns**: Defines URL pattern
- **servletNames**: Defines servlet to which the filter will be applied
- **dispatcherTypes**: It associates a dispatcher to a filter
- **initParams**: Initialization parameter to the filter

@WebFilter Does Not Define the Order of Filters

```
@WebFilter(filterName="filterOne")
public class FilterOne
        implements Filter {
   ...
}
```

```
@WebFilter(filterName="filterTwo")
public class FilterTwo
        implements Filter {
   ...
}
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app ...>
    <filter-mapping>
        <filter-name>filterOne</filter-name>
        <url-pattern>/msg/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>filterTwo</filter-name>
        <url-pattern>/msg/*</url-pattern>
    </filter-mapping>
</web-app>
```

Web application listeners

- **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)

`net.hello.listeners`

Listeners and Events

- ServletContextListener -> ServletContextEvent
- HttpSessionListener -> HttpSessionEvent
- ServletRequestListener -> ServletRequestEvent
- ServletContextAttributeListener -> ServletContextAttributeEvent
- HttpSessionAttributeListener -> HttpSessionBindingEvent
- ServletRequestAttributeListener -> ServletRequestAttributeEvent

@WebListener annotation

```
@WebListener()
public class MyContextListener implements ServletContextListener {
    ...
}
```

```
<listener>
    <listener-class>
        com.hello.listeners.MyContextListener
    </listener-class>
</listener>
```

@ServletSecurity annotation

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Manager</web-resource-name>
        <url-pattern>/security</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>admin</role-name>
        <role-name>manager</role-name>
    </auth-constraint>
</security-constraint>
```

@ServletSecurity annotation

```
...
@WebServlet(name="SecurityServlet", urlPatterns={"/security"})
@ServletSecurity(
    httpMethodConstraints = {
        @HttpMethodConstraint(
            value = "GET", rolesAllowed = {"admin", "manager"}
        ),
        @HttpMethodConstraint(
            value = "POST", rolesAllowed = {"admin", "manager"}
        ),
        @HttpMethodConstraint(
            value = "TRACE", emptyRoleSemantic = EmptyRoleSemantic.DENY
        )
    }
)
public class SecurityServlet extends HttpServlet {
    ...
}
```

Mapping Role References

```
<servlet>
    <servlet-name>SecuredMethodServlet</servlet-name>
    <servlet-class>
        com.examples.lesson11.servlets.SecuredMethodServlet
    </servlet-class>
    <security-role-ref>
        <role-name>originalRoleName</role-name>
        <role-link>linkRoleName</role-link>
    </security-role-ref>
</servlet>
```

```
// Authenticate user if not already authenticated
if (request.authenticate(response)) {
    // Use role link name
    if (request.isUserInRole("linkRoleName") {
        // User authenticated is originalRoleName
    }
}
else {
    request.getRequestDispatcher(
        "/errors/access_denied.jsp").forward(request, response);
}
```

Lifecycle Method Annotations

- @PostConstruct and @PreDestroy
- The methods target may be: Zero-argument, Return void, throw no checked exceptions, Non-final
- The @PostConstruct method is guaranteed to be called after the injection and before the init() method.
- If any exception is thrown from @PostConstruct, the object is abandoned.
- @PostConstruct and @PreDestroy are more general than init() and destroy(), because they allow nonpublic/non-interface methods.
- init() and destroy() methods are invoked by the servlet container—these are specific methods declared in the Servlet interface.
- @PostConstruct and @PreDestroy are supported by any container that supports dependency injection and can be applied to any method.
- Sequence: Servlet constructor -> @PostConstruct -> init -> destroy -> @PreDestroy

`LifeCycleServlet`

Asynchronous servlet

- Were added with Java EE 6
- Avoid blocking servlet threads that are waiting for external conditions to trigger completion
- Allow the HTTP response to be generated by an arbitrary thread
- Are distinct from AJAX techniques

Asynchronous Servlet Example

```
@WebServlet(name = "HelloAsyncServlet",
    urlPatterns = {"/HelloAsyncServlet"}, asyncSupported = true)
public class HelloAsyncServlet extends HttpServlet {
    private ExecutorService executorService =
            Executors.newCachedThreadPool();
    @Override
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        AsyncContext asyncContext =
            request.startAsync(request, response);
        asyncContext.addListener(new HelloAsyncListener());
        executorService.execute(new HelloAsyncWorker(asyncContext));
    }

    @Override
    public void destroy() {
        executorService.shutdownNow();
    }
}
```

AsyncListener Example

```
public class HelloAsyncListener implements AsyncListener {

    @Override
    public void onComplete(AsyncEvent event) throws IOException { }

    @Override
    public void onTimeout(AsyncEvent event) throws IOException { }

    @Override
    public void onError(AsyncEvent event) throws IOException {
        ...
        event.getAsyncContext().complete();
    }

    @Override
    public void onStartAsync(AsyncEvent event) throws IOException {
        ServletResponse response =
                event.getAsyncContext().getResponse();
        ...
    }
}
```

Runnable Example

```java
public class HelloAsyncWorker implements Runnable {

    private final AsyncContext asyncContext;

    public HelloAsyncWorker(AsyncContext asyncContext) {
        this.asyncContext = asyncContext;
    }

    @Override
    public void run() {
        ...
        HttpServletResponse response = (HttpServletResponse)
                asyncContext.getResponse();
        try {
            ...
        } catch (IOException ex) {
            ...
        } finally {
            asyncContext.complete();
        }
    }
}
```

Forwarding and Filtering

- Asynchronous handlers can dispatch or forward
- The target does not need to be asynchronous
- This allows fast-responding pages to be used in any case
- AsyncContext.dispatch invokes filters with a dispatcher type of ASYNC

Handling Multipart Forms

- Part
    - getContentType()
    - getInputStream()
    - delete()
    - write(String filename)
- request.getPart(String)
- Collection <Part> request.getParts()
- @MultipartConfig
    - fileSizeThreshold: The file size in bytes after which the file will be temporarily stored on disk. The default size is 0 bytes.
    - maxRequestSize: The maximum size allowed for a multipart/form-data request, in bytes.
    - maxFileSize: The maximum size allowed for uploaded files, in bytes.
    - location: An absolute path to a directory on the file system.

@MultipartConfig annotation

- **location**: An absolute path to a directory on the file system. The location attribute does not support a path relative to the application context. This location is used to store files temporarily while the parts are processed or when the size of the file exceeds the specified fileSizeThreshold setting. The default location is ""

- **fileSizeThreshold**: The file size in bytes after which the file will be temporarily stored on disk. The default size is 0 bytes

- **MaxFileSize**: The maximum size allowed for uploaded files, in bytes. If the size of any uploaded file is greater than this size, the web container will throw an exception (IllegalStateException). The default size is unlimited

- **maxRequestSize**: The maximum size allowed for a multipart/form-data request, in bytes. The web container will throw an exception if the overall size of all uploaded files exceeds this threshold. The default size is unlimited.

@MultipartConfig Example

```
@MultipartConfig(location="/tmp", fileSizeThreshold=1024*1024,
                 maxFileSize=1024*1024*5, maxRequestSize=1024*1024*5*5)
```

```
<servlet>
    <servlet-name>...</servlet-name>
    <servlet-class>...</servlet-class>
    <multipart-config>
        <location>/tmp</location>
        <max-file-size>20848820</max-file-size>
        <max-request-size>418018841</max-request-size>
        <file-size-threshold>1048576</file-size-threshold>
    </multipart-config>
</servlet>
```

```
<form action="RealImageStorageServlet"
      enctype="multipart/form-data" method="post">
    <input type="file" name="filename"/>
    <input type="submit" value="upload"/>
</form>
```

Multipart Servlet Example

```
@WebServlet(name = "RealImageStorageServlet",
            urlPatterns = {"/RealImageStorageServlet"})
@MultipartConfig(location="C:/SL314EE6/resources/upload/")
public class RealImageStorageServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {
        Part part = request.getPart("filename");
        String fileName = getFileName(part);

        if (fileName != null && fileName.length() > 0) {
            part.write(fileName);
            ...
        }
    }

    private String getFileName(Part part) {
        ...
    }
}
```

Context Dependency Injection beans.xml in WEB-INF

```
<?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>
```

Context Dependency Injection JavaBean

```
package net.hello.service;

public interface HelloService {
    public String greeting();
}
```

```
package net.hello.service;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class HelloServiceImpl implements HelloService {

    private final String[] MESSAGE = {"Hi", "Hello", "Good Morning"};

    @Override
    public String greeting() {
        return MESSAGE[(int)(Math.random() * 3)];
    }
}
```

Context Dependency Injection Servlet Example

```java
package net.hello.servlets;
...
import javax.inject.Inject;
import javax.servlet.annotation.WebServlet;
import net.hello.service.HelloService;

@WebServlet(name = "ServiceServlet", urlPatterns = {"/service.view"})
public class ServiceServlet extends HttpServlet {

    @Inject
    private HelloService helloService;

    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        ...
        out.println(helloService.greeting());
        ...
    }
    ...
}
```

Java Persistence API Structure

- An entity, defined by the programmer
- Mappings from entity to the database
- persistence.xml specifying the DataSource
- JNDI lookup service
- A DataSource connected to a connection pool
- A connection pool connected to the database
- The database with appropriate tables

JPA Entity Class

- Declare a public class
- Class must not be final; methods cannot be final.
- Class may be abstract.
- Class must not be inner.
- Annotate the class with javax.persistence.Entity.
- Declare the attributes; these must not be public.
- Declare public business, accessor, and mutator methods as necessary.
- Annotate the primary key field or accessor method using the Id annotation.
- Verify and override the default mapping.

JPA Entity Class Example

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Employee implements Serializable {
    @Id
    private int id;
    private String firstName;
    private String lastName;
    @Temporal(TemporalType.DATE)
    private Date birthDate;
    private float salary;

    public Employee() {}

    // ... other constructors and methods
}
```

Persistence Units

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/
persistence">
    <persistence-unit name="QuizPU" transaction- type="JTA">
        <jta-data-source>jdbc/QuizDB</jta-data-source>
        <properties/>
    </persistence-unit>
</persistence>
```

EntityManager Example

```
...
@WebServlet(...)
public class ListMemberJPAServlet extends HttpServlet {

    @PersistenceUnit(unitName = "HelloJPAPU")
    private EntityManagerFactory emf;

    protected void processRequest(...)
            throws ServletException, IOException {
        ...
        EntityManager em = emf.createEntityManager();
        TypedQuery<Member> query = em.createQuery(
                "SELECT m FROM Member m", Member.class);
        List<Member> ms = query.getResultList();
        ...
   }
    ...
}
```