Principles and Techniques of DBMS 5 Servlet

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Overview

- Servlet
  - What is Servlet
  - Servlet API
  - Examples
  - Web Application Structure

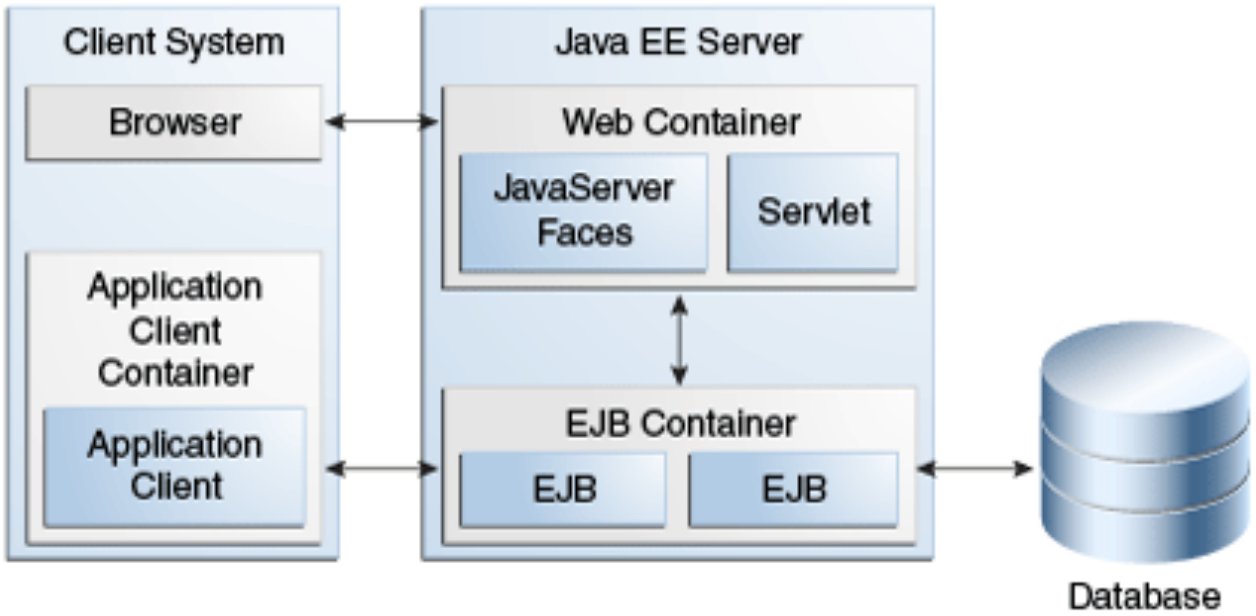
What is servlet?

- 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.

Servlet Container

- Container types



An Example

- 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.

Servlet Lifecycle

- 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

REliable, INtelligent & Scalable Systems

- 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.

```
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

- 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.

An Example: mood

REliable, INtelligent & Scalable Systems

```
@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();
        }
    }
```

An Example: mood

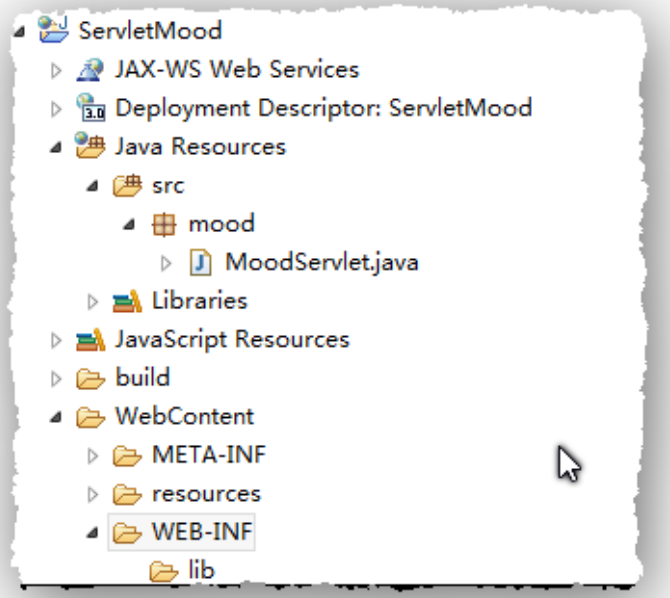

```
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        processRequest(request, response);
    }

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

}
```

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>
```

An Example: mood



Filtering Requests and Responses

REliable, INtelligent & Scalable Systems

- 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.

Programming Filters

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

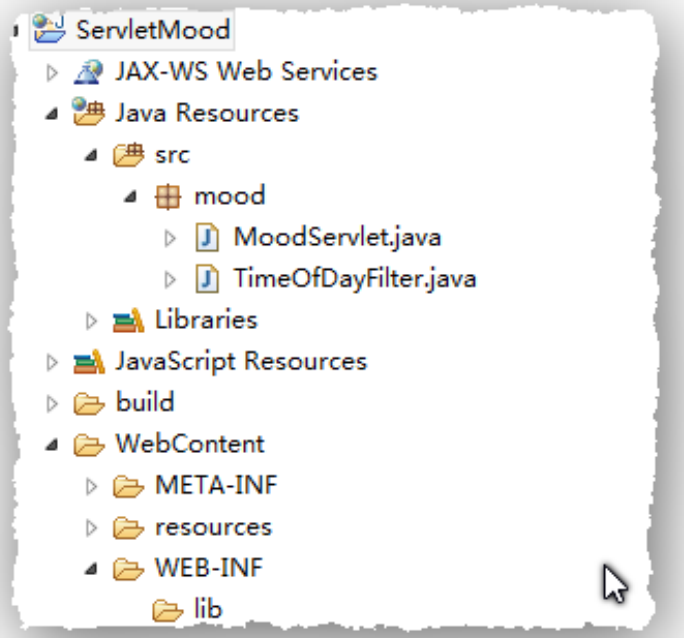
```
import javax.servlet.Filter;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

@WebFilter(filterName = "TimeOfDayFilter",
 urlPatterns = {"/*"},
 initParams = {
   @WebInitParam(name = "mood", value = "awake")}
)
public class TimeOfDayFilter implements Filter { ....
```

Programming Filters

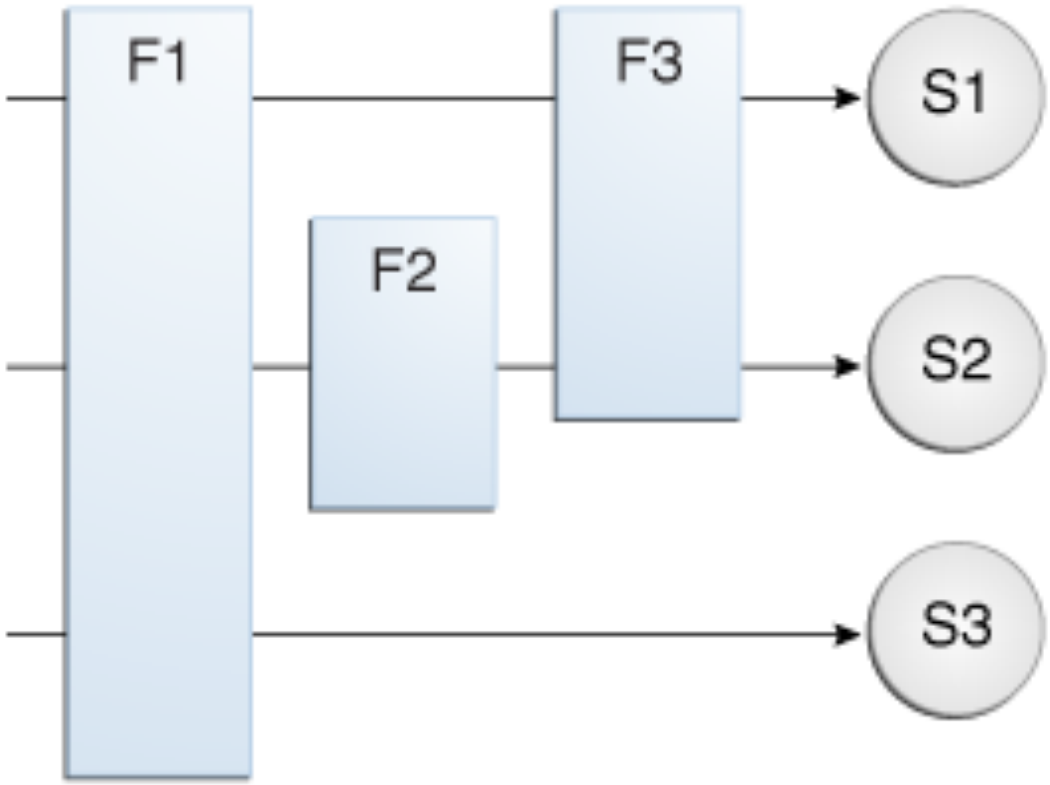
REliable, INtelligent & Scalable Systems

```
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() {
    }
}
```

An Example: mood



Filter-to-Servlet Mapping



Handling Servlet Lifecycle Events

REliable, INtelligent & Scalable Systems

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

Handling Servlet Lifecycle Events

REliable, INtelligent & Scalable Systems

```
@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 vs. Parameter

REliable, INtelligent & Scalable Systems

- 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");
```

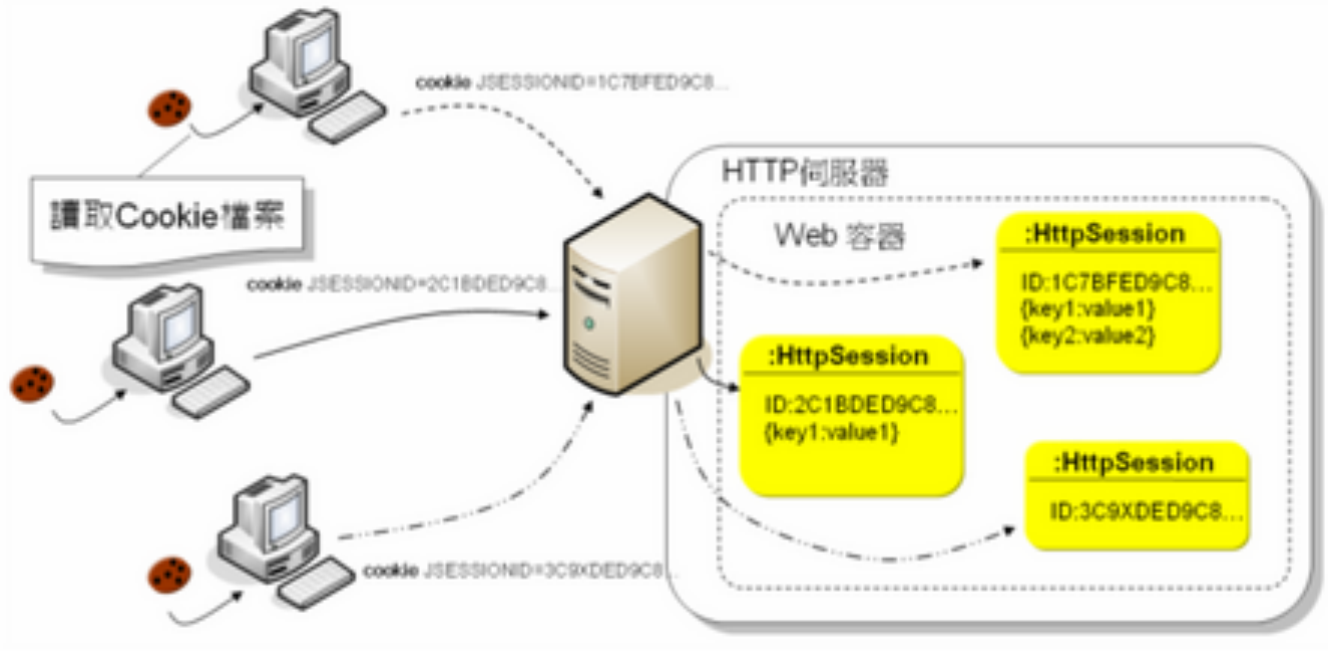
Maintaining Client State

- 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.

Accessing a Session

REliable, INtelligent & Scalable Systems

- 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.



Associating Objects with a Session

- You can associate object-valued attributes with a session by name.
  - 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.

Uploading Files

REliable, INtelligent & Scalable Systems

- Supporting file uploads is a very basic and common requirement for many web applications.

- `javax.servlet.annotation.MultipartConfig`, is used to indicate that the servlet on which it is declared expects requests to made using the `multipart/form-data` MIME type.
  - Servlets that are annotated with `@MultipartConfig` can retrieve the Part components of a given multipart/form-data request by calling the `request.getPart(String name)` or `request.getParts()` method.

- `multipart/form-data` MIME type
  - A method for uploading files with forms in Browser
  - Scenario: the attachment of an email is attached with form, that is, the attachment is uploaded to server in multipart/form-data MIME type

Uploading Files

REliable, INtelligent & Scalable Systems

- The @MultipartConfig annotation supports the following optional attributes:
  - location:
    - An absolute path to a directory on the file system.
    - 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.

Uploading Files

- For, example, the @MultipartConfig annotation could be constructed as follows:

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

- Instead of using the @MultipartConfig annotation to hard-code these attributes in your file upload servlet, you could add the following as a child element of the servlet configuration element in the web.xml file.

```
<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>
```

Uploading Files

- The Servlet specification supports two additional `HttpServletRequest` methods:

```
Collection<Part> getParts()
Part getPart(String name)
```

- The `javax.servlet.http.Part` interface is a simple one, providing methods that allow introspection of each Part. The methods do the following:
  - Retrieve the name, size, and content-type of the Part
  - Query the headers submitted with a Part
  - Delete a Part
  - Write a Part out to disk

Uploading Files

REliable, INtelligent & Scalable Systems

- Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <title>File Upload</title>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 </head>
 <body>
   <form method="POST" action="upload" enctype="multipart/form-data" >
     File:
     <input type="file" name="file" id="file" /> <br/>
     Destination:
     <input type="text" value="/tmp" name="destination"/>
     </br>
     <input type="submit" value="Upload" name="upload" id="upload" />
   </form>
 </body>
</html>
```

Uploading Files

- FileUploadServlet.java

```
@WebServlet(name = "FileUploadServlet", urlPatterns = {"/upload"})
@MultipartConfig
public class FileUploadServlet extends HttpServlet {

    private final static Logger LOGGER =
            Logger.getLogger(FileUploadServlet.class.getCanonicalName());

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

        // Create path components to save the file
        final String path = request.getParameter("destination");
        final Part filePart = request.getPart("file");
        final String fileN = getFileName(filePart);
        File file = new File(fileN);
        final String fileName = file.getName();

        OutputStream out = null;
        InputStream filecontent = null;
        final PrintWriter writer = response.getWriter();
```

Uploading Files

REliable, INtelligent & Scalable Systems

- FileUploadServlet.java

```
        try {
            out = new FileOutputStream(new File(path + File.separator + fileName));;
            filecontent = filePart.getInputStream();

            int read;
            final byte[] bytes = new byte[1024];

            while ((read = filecontent.read(bytes)) != -1) { out.write(bytes, 0, read); }
            writer.println("New file " + fileName + " created at " + path);
            LOGGER.log(Level.INFO, "File {0} being uploaded to {1}",
                    new Object[]{fileName, path});

        } catch (FileNotFoundException fne) {
            writer.println("You either did not specify a file to upload or are "
                    + "trying to upload a file to a protected or nonexistent "
                    + "location.");
            writer.println("<br/> ERROR: " + fne.getMessage());
            LOGGER.log(Level.SEVERE, "Problems during file upload. Error: {0}",
                    new Object[]{fne.getMessage()});
        } finally {
            if (out != null) { out.close(); }
            if (filecontent != null) { filecontent.close(); }
            if (writer != null) { writer.close(); }
        }
    }
```

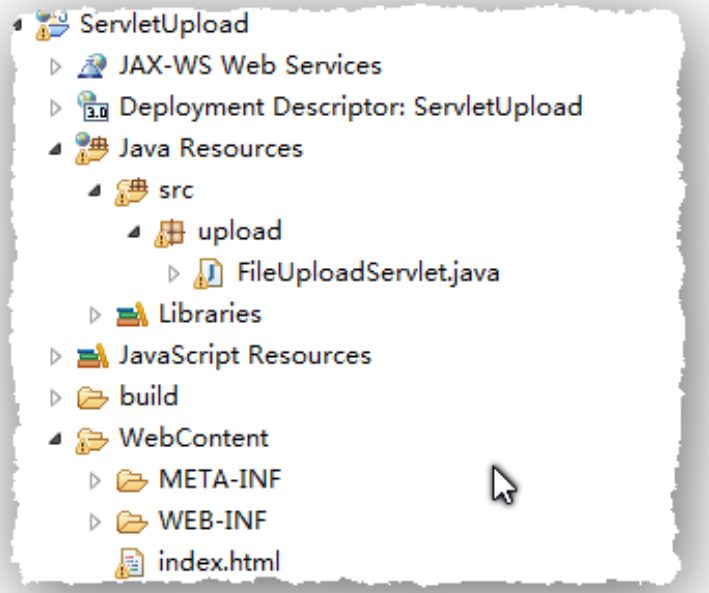
Uploading Files

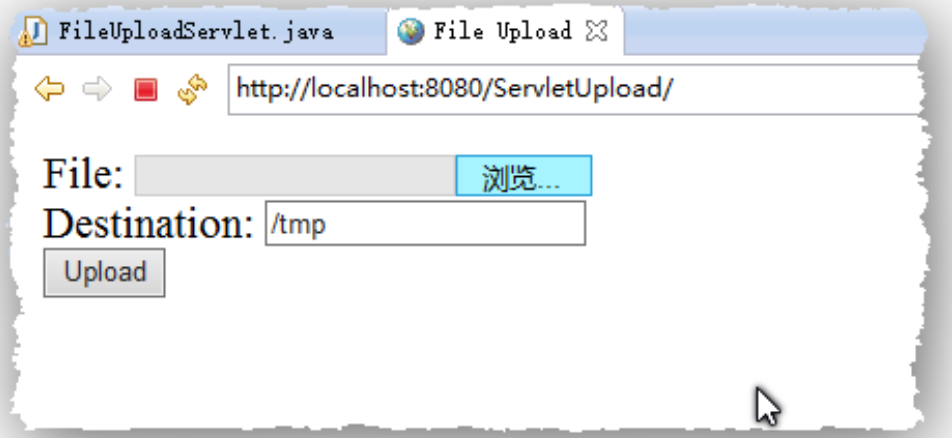
REliable, INtelligent & Scalable Systems

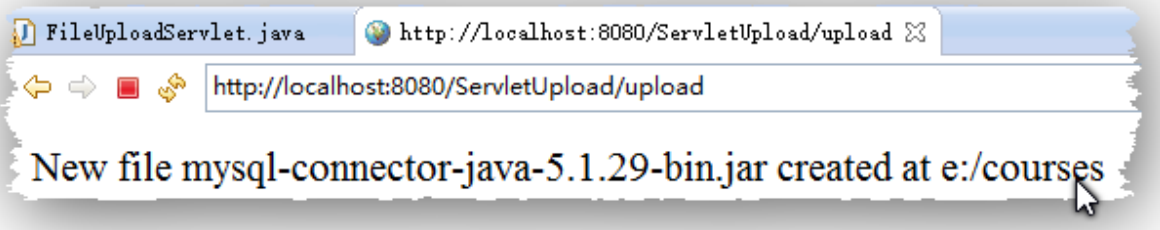
- FileUploadServlet.java

```
    private String getFileName(final Part part) {
        final String partHeader = part.getHeader("content-disposition");
        LOGGER.log(Level.INFO, "Part Header = {0}", partHeader);
        for (String content : part.getHeader("content-disposition").split(";")) {
            if (content.trim().startsWith("filename")) {
                return content.substring(
                        content.indexOf('=') + 1).trim().replace("\"", "");
            }
        }
        return null;
    }

    @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);
    }
}
```

Uploading Files







Web Applications Within Web Servers

REliable, INtelligent & Scalable Systems

- A Web application is rooted at a specific path within a Web server.
  - For example, a catalog application could be located at http://www.mycorp.com/catalog.

- A Web application may consist of the following items:
  - Servlets
  - JSP Pages
  - Utility Classes
  - Static documents (HTML, images, sounds, etc.)
  - Client side Java applets, beans, and classes
  - Descriptive meta information that ties all of the above elements together

Application Directory Structure

REliable, INtelligent & Scalable Systems

- An example:
  - /index.html
  - /howto.jsp
  - /feedback.jsp
  - /images/banner.gif
  - /images/jumping.gif
  - /WEB-INF/web.xml
  - /WEB-INF/lib/jspbean.jar
  - /WEB-INF/classes/com/mycorp/servlets/MyServlet.class
  - /WEB-INF/classes/com/mycorp/util/MyUtils.class

References

- Java EE 7 tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/servlets001.htm
- Java Servlet Specification
  - http://download.oracle.com/otn-pub/jcp/servlet-3_1-fr-eval-spec/servlet-3_1-final.pdf?AuthParam=1396178605_99ca893c1d2485541a2baf6a88414bf8

Thank You!