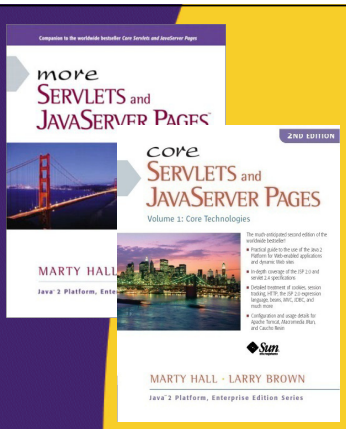
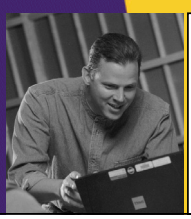
Servlet and JSP Filters

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/msajsp.html

© 2009 Marty Hall





Agenda

- Filter basics
- Accessing the servlet context
- Using initialization parameters
- Blocking responses
- Modifying responses

Filters: Overview

- Associated with any number of servlets or JSP pages
- Examine request coming into servlets or JSP pages, then:
  - Invoke the resource (i.e., the servlet or JSP page) in the normal manner.
  - Invoke the resource with modified request information.
  - Invoke the resource but modify the response before sending it to the client.
  - Prevent the resource from being invoked and instead redirect to a different resource, return a particular status code, or generate replacement output.

Advantages of Filters

- **Encapsulate common behavior.**
  - Have 30 different servlets or JSP pages that need to compress their content to decrease download time? Make 1 compression filter and apply it to all 30 resources.
- **Separate high-level access decisions from presentation code.**
  - Want to block access from certain sites without modifying the individual pages to which these access restrictions apply? Create an access restriction filter and apply it to as many pages as you like.
- **Apply wholesale changes to many different resources.**
  - Have a bunch of existing resources that should remain unchanged except that the company name should be changed? Make a string replacement filter and apply it wherever appropriate.

Steps to Creating Filters

1. **Create class that implements Filter interface.**
   - Methods: doFilter, init, destroy
2. **Put filtering behavior in doFilter.**
   - Args: ServletRequest, ServletResponse, FilterChain
3. **Call doFilter method of the FilterChain.**
   - This invokes next filter (if any) or actual resource
4. **Register the filter with the appropriate servlets and JSP pages.**
   - Use `filter` and `filter-mapping` in *web.xml*.
5. **Disable invoker servlet.**
   - See earlier slide

The doFilter Method

- **Basic format**

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws ServletException, IOException {
  …
  chain.doFilter(request,response);
}
```

- **Note on first two arguments**
  - They are of type ServletRequest and ServletResponse, not HttpServletRequest and HttpServletResponse.
    - Do a typecast if you need HTTP-specific capabilities
- **Note on final argument**
  - It is a FilterChain, not a Filter. Its doFilter method is different – two arguments only.

A Simple Reporting Filter

```
public class ReportFilter implements Filter {
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
      throws ServletException, IOException {
    HttpServletRequest req =
      (HttpServletRequest)request;
    System.out.println(req.getRemoteHost() +
                       " tried to access " +
                       req.getRequestURL() +
                       " on "+new Date() + ".");
    chain.doFilter(request,response);
  }
```

A Simple Reporting Filter (Continued)

```
  public void init(FilterConfig config)
      throws ServletException {
  }

  public void destroy() {
  }
}
```


Declaring the Reporting Filter

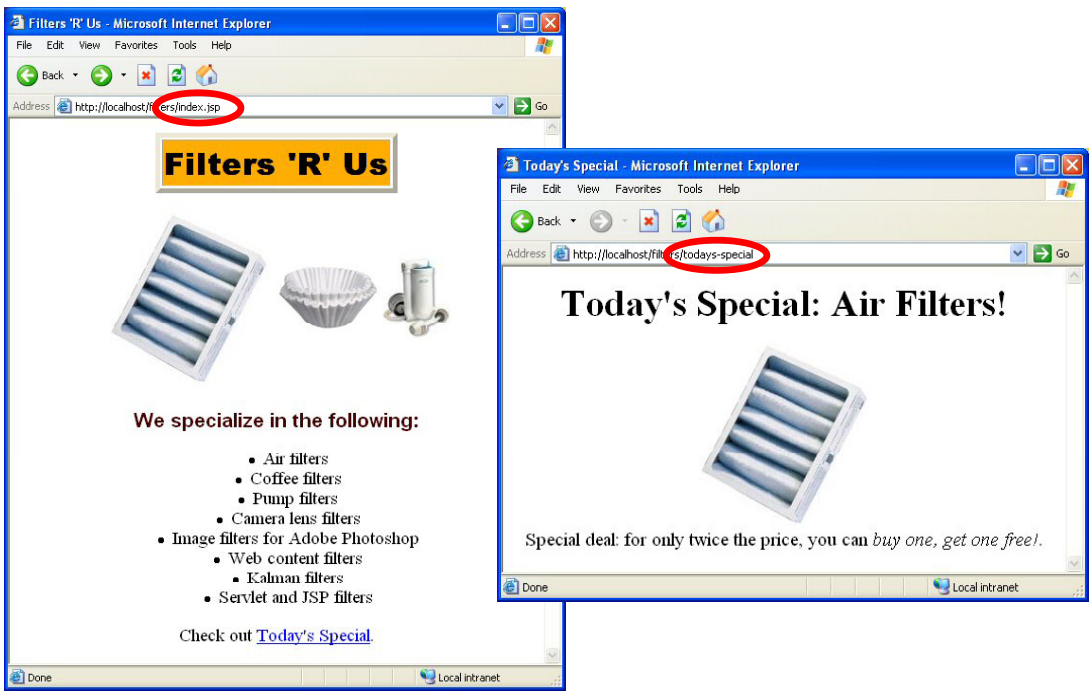
```
…
<web-app…>
  <!-- Register the name "Reporter"
       for ReportFilter. -->
  <filter>
    <filter-name>Reporter</filter-name>
    <filter-class>
      coreservlets.filters.ReportFilter
    </filter-class>
  </filter>
```

- **Important note**
  - Servers load filters into memory when the Web app first comes up. So, if that filter is not found, your *entire* Web app is disabled.

Associating Reporting Filter with Given URLs

```
<!-- Apply Reporter filter to home page. -->
<filter-mapping>
  <filter-name>Reporter</filter-name>
  <url-pattern>/index.jsp</url-pattern>
</filter-mapping>

<!-- Also apply Reporter filter to
     servlet named "TodaysSpecial". -->
<filter-mapping>
  <filter-name>Reporter</filter-name>
  <servlet-name>TodaysSpecial</servlet-name>
</filter-mapping>
…
</web-app>
```


Reporting Filter: Results



Reporting Filter (Results Continued)

- **Printouts to standard output akin to the following will result from the two accesses shown on previous page:**
  - purchasing.sun.com tried to access http://www.filtersrus.com/filters/index.jsp on Fri Apr 11 13:19:14 EDT 2008.
  - admin.microsoft.com tried to access http://www.filtersrus.com/filters/TodaysSpecial on Fri Apr 11 13:21:56 EDT 2008.
- **Point: A single filter can apply to lots of different resources in transparent manner**
  - The individual resources do not need any special code

Accessing the Servlet Context

- **What if filter wants to read or write Web application-wide parameters? Or it simply wants to log data?**
  - You use methods in ServletContext for this
- **Surprisingly, the doFilter method provides no access to the ServletContext**
  - Neither ServletRequest nor ServletResponse provides access to it either
- **Solution: store the ServletContext in init**
  - Call getServletContext on the FilterConfig argument that is passed to init
  - Store the result in an instance variable (field) of the filter
  - Access the field from the doFilter method

A Logging Filter

```
public class LogFilter implements Filter {
  protected FilterConfig config;
  private ServletContext context;
  private String filterName;

  public void init(FilterConfig config)
      throws ServletException {
    // In case it is needed by subclass.
    this.config = config;
    context = config.getServletContext();
    filterName = config.getFilterName();
  }
```


A Logging Filter (Continued)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws ServletException, IOException {
  HttpServletRequest req =
    (HttpServletRequest)request;
  context.log(req.getRemoteHost() +
              " tried to access " +
              req.getRequestURL() +
              " on " + new Date() + ". " +
              "(Reported by " +
              filterName + ".)");
  chain.doFilter(request,response);
}
```

Applying Logging Filter to Entire Web Application

```
<web-app>
  …
  <filter>
    <filter-name>Logger</filter-name>
    <filter-class>
      coreservlets.filters.LogFilter
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>Logger</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  …
</web-app>
```


Logging Filter: Results

- **Log file:**
  - audits.irs.gov tried to access http://www.filtersrus.com/filters/business-plan.jsp on Tue Apr 15 15:16:15 EDT 2008. (Reported by Logger.)
  - ceo.enron.com tried to access http://www.filtersrus.com/filters/tax-shelter/ on Wed Apr 16 10:24:11 EDT 2008. (Reported by Logger.)

Using Filter Initialization Parameters

- **Reminder: who needs to customize servlet and JSP behavior?**
  - Developers.
    - They customize the behavior by changing the code of the servlet or JSP page itself.
  - End users.
    - They customize the behavior by entering values in HTML forms.
  - Deployers.
    - This third group is the one served by initialization parameters. Members of this group are people who take existing Web applications (or individual servlets or JSP pages) and deploy them in a customized environment.
- **Resources with initialization parameters**
  - Servlets, JSP pages, servlet context, filters, listeners.

Declaring Filter Initialization Parameters

```
<filter>
  <filter-name>LateAccessFilter</filter-name>
  <filter-class>
    coreservlets.filters.LateAccessFilter
  </filter-class>
  <init-param>
    <param-name>startTime</param-name>
    <param-value>2</param-value>
  </init-param>
  <init-param>
    <param-name>endTime</param-name>
    <param-value>10</param-value>
  </init-param>
</filter>
```

Reading Init Params: An Access Time Filter

```
public void init(FilterConfig config)
    throws ServletException {
  context = config.getServletContext();
  formatter =
    DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
                                   DateFormat.MEDIUM);
  try {
    startTime = Integer.parseInt
                (config.getInitParameter("startTime"));
    endTime = Integer.parseInt
                (config.getInitParameter("endTime"));
  } catch(NumberFormatException nfe) { // Malformed/null
    // Default: access at or after 10 p.m. but before 6
    // a.m. is considered unusual.
    startTime = 22; // 10:00 p.m.
    endTime = 6;    //  6:00 a.m.
  }
}
```


An Access Time Filter (Continued)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws ServletException, IOException {
  HttpServletRequest req =
    (HttpServletRequest)request;
  GregorianCalendar calendar =
    new GregorianCalendar();
  int currentTime =
    calendar.get(Calendar.HOUR_OF_DAY);
  if (isUnusualTime(currentTime,startTime,endTime)){
    context.log("WARNING: " +
                req.getRemoteHost() +
                " accessed " +
                req.getRequestURL() +
                " on " +
                formatter.format(calendar.getTime()));
  }
  chain.doFilter(request,response);
}
```

Blocking the Response

- **Idea**
  - Normal situation: call doFilter on FilterChain object
  - Unusual situation: redirect response or generate custom output
- **Generic Example**

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws ServletException, IOException {
  HttpServletRequest req =
    (HttpServletRequest)request;
  HttpServletResponse res =
    (HttpServletResponse)response;
  if (isUnusualCondition(req)) {
    res.sendRedirect("http://www.somesite.com");
  } else {
    chain.doFilter(req,res);
  }
}
```


A Banned Site Filter

```
public class BannedAccessFilter implements Filter {
  private HashSet<String> bannedSiteTable;

  public void init(FilterConfig config)
      throws ServletException {
    bannedSiteTable = new HashSet<String>();
    String bannedSites =
      config.getInitParameter("bannedSites");
    StringTokenizer tok =
      new StringTokenizer(bannedSites);
    while(tok.hasMoreTokens()) {
      String bannedSite = tok.nextToken();
      bannedSiteTable.add(bannedSite);
      System.out.println("Banned " + bannedSite);
    }
  }
  public void destroy() {}
```

A Banned Site Filter (Continued)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
  throws ServletException, IOException {
HttpServletRequest req = (HttpServletRequest)request;
String requestingHost = req.getRemoteHost();
String referringHost =
  getReferringHost(req.getHeader("Referer"));
String bannedSite = null;
boolean isBanned = false;
if (bannedSiteTable.contains(requestingHost)) {
  bannedSite = requestingHost; isBanned = true;
} else if (bannedSiteTable.contains(referringHost)) {
  bannedSite = referringHost; isBanned = true;
}
if (isBanned) {
  showWarning(response, bannedSite);// Custom response
} else {
  chain.doFilter(request,response);
}} …
```


A Banned Site Filter (Continued)

```
private String getReferringHost
             (String refererringURLString) {
  try {
    URL referringURL =
      new URL(refererringURLString);
    return(referringURL.getHost());

  // Malformed or null
  } catch(MalformedURLException mue) {
    return(null);
  }
}
```

A Banned Site Filter (Continued)

```
private void showWarning(ServletResponse response,
                         String bannedSite)
    throws ServletException, IOException {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String docType =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n";
  out.println
    (docType +
     "<HTML>\n" +
     "<HEAD><TITLE>Access Prohibited</TITLE></HEAD>\n"+
     "<BODY BGCOLOR=\"WHITE\">\n" +
     "<H1>Access Prohibited</H1>\n" +
     "Sorry, access from or via " + bannedSite + "\n"+
     "is not allowed.\n" +
     "</BODY></HTML>");
}
```

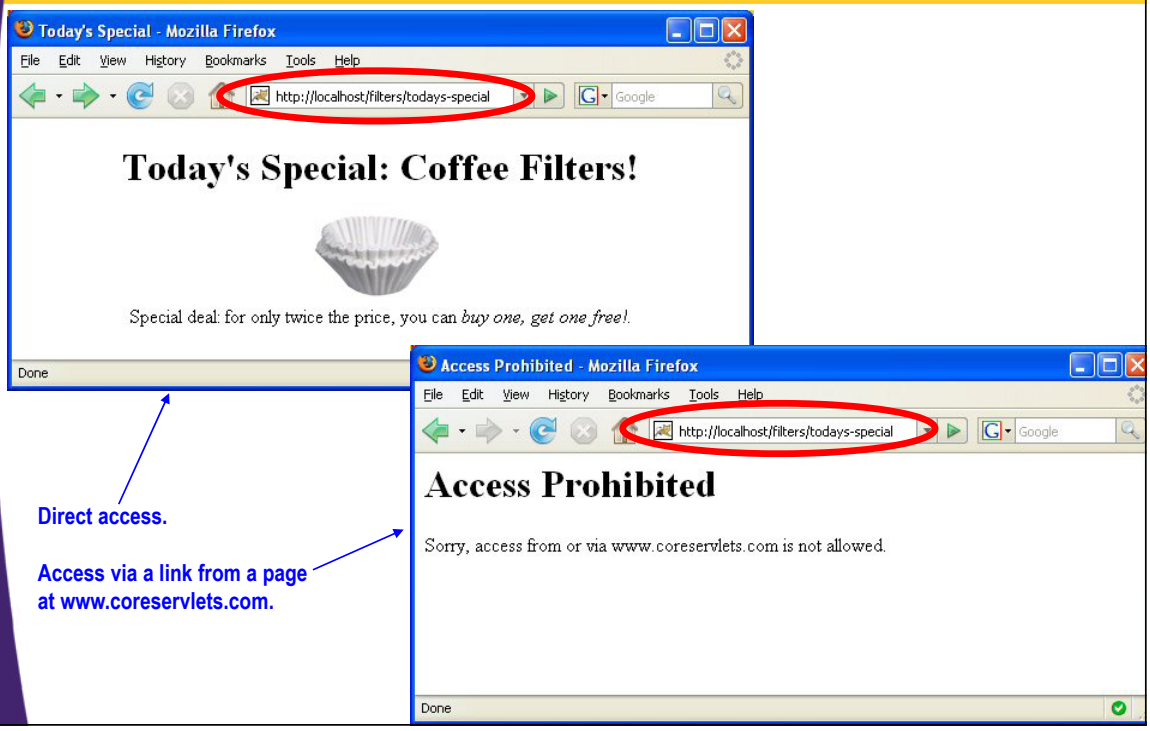

Registering the Banned Site Filter in web.xml

```
…
<web-app>
  <filter>
    <filter-name>BannedAccessFilter</filter-name>
    <filter-class>
      coreservlets.filters.BannedAccessFilter
    </filter-class>
    <init-param>
      <param-name>bannedSites</param-name>
      <param-value>
        www.competingsite.com
        www.bettersite.com
        www.coreservlets.com
      </param-value>
    </init-param>
  </filter>
…
```

Registering the Banned Site Filter (Continued)

```
…
  <filter-mapping>
    <filter-name>BannedAccessFilter</filter-name>
    <url-pattern>/todays-special</url-pattern>
  </filter-mapping>
  <servlet>
    <servlet-name>TodaysSpecial</servlet-name>
    <servlet-class>
      coreservlets.TodaysSpecialServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TodaysSpecial</servlet-name>
    <url-pattern>/todays-special</url-pattern>
  </servlet-mapping>
  …
</web-app>
```


Filter in Action



Advanced Filters: Modifying the Response

1. **Create a response wrapper.**
   - Extend HttpServletResponseWrapper.
2. **Provide a PrintWriter that buffers output.**
   - Override getWriter method to return a PrintWriter that saves everything sent to it and stores that result in a field
3. **Pass that wrapper to doFilter.**
   - This call is legal because HttpServletResponseWrapper implements HttpServletResponse.
4. **Extract and modify the output.**
   - After call to doFilter method of the FilterChain, output of the original resource is available to you through whatever mechanism you provided in Step 2. Modify or replace it as appropriate.
5. **Send the modified output to the client.**
   - Original resource no longer sends output to client (output is stored in your response wrapper instead). *You* have to send the output. So, filter needs to obtain the PrintWriter or OutputStream from original response object and pass modified output to that stream.

A Reusable Response Wrapper

```
public class StringWrapper
             extends HttpServletResponseWrapper {
  private StringWriter stringWriter;

  public StringWrapper(HttpServletResponse response) {
    super(response);
    stringWriter = new StringWriter();
  }

  public PrintWriter getWriter() {
    return(new PrintWriter(stringWriter));
  }

  public ServletOutputStream getOutputStream() {
    return(new StringOutputStream(stringWriter));
  }

  public String toString() {
    return(stringWriter.toString());
  }

  public StringBuffer getBuffer() {
    return(stringWriter.getBuffer());
  }
}
```

StringWriter is builtin. But StringOutputStream is from my app. See source code online. (http://courses.coreservlets.com/Course-Materials/msajsp.html#Filters)

A Generic Modification Filter

```
public abstract class ModificationFilter implements Filter {
  private ServletContext context;
  private HttpServletRequest request;
  private HttpServletResponse response;

  public void doFilter(ServletRequest req,
                       ServletResponse resp,
                       FilterChain chain)
      throws ServletException, IOException {
    request = (HttpServletRequest)req;
    response = (HttpServletResponse)resp;
    StringWrapper responseWrapper =
      new StringWrapper(response);
    chain.doFilter(request, responseWrapper);
    String modifiedResponse =
      doModification(responseWrapper.toString());
    PrintWriter out = response.getWriter();
    out.write(modifiedResponse);
  }
```


A Generic Modification Filter (Continued)

```
  public abstract String doModification(String origResponse);

  public void init(FilterConfig config) {
    context = config.getServletContext();
  }

  public void destroy() {}

  public HttpServletRequest getRequest() {
    return(request);
  }

  public HttpServletResponse getResponse() {
    return(response);
  }

}
```

A Generic Replacement Filter

```
public abstract class ReplaceFilter
    extends ModificationFilter {
  private boolean isCaseInsensitive = false;

  public abstract String getTarget();

  public abstract String getReplacement();

  public void setCaseInsensitive(boolean flag) {
    isCaseInsensitive = flag;
  }

  public boolean isCaseInsensitive() {
    return(isCaseInsensitive);
  }
```


A Generic Replacement Filter (Continued)

```
  public String doModification(String orig) {
    if ((getTarget() == null) ||
        (getReplacement() == null)) {
      return(orig);
    } else {
      String target = getTarget();
      if (isCaseInsensitive()) {
        target = "(?i)" + target;
      }
      String replacement = getReplacement();
      return(orig.replaceAll(target, replacement));
    }
  }
}
```

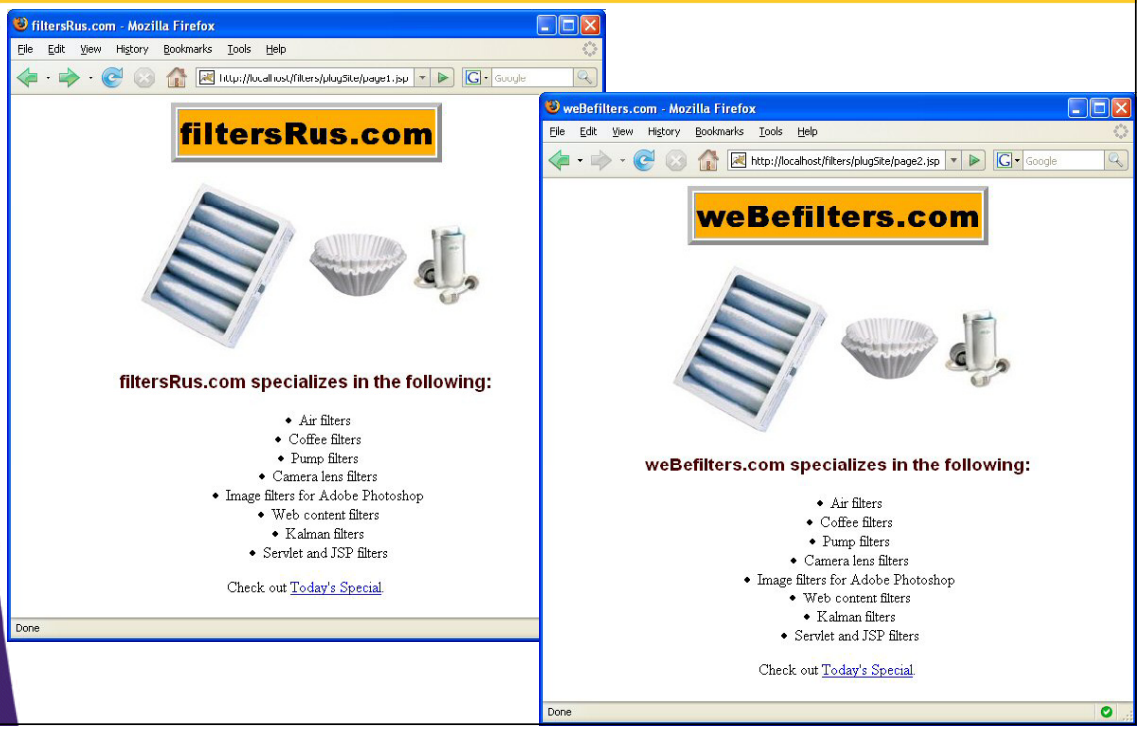
A Specific Replacement Filter

```
public class ReplaceSiteNameFilter
    extends ReplaceFilter {

  public String getTargetString() {
    return("filtersRus.com");
  }

  public String getReplacementString() {
    return("weBefilters.com");
  }
}
```


A Specific Replacement Filter (Continued)

```
<web-app…>
…
  <filter>
    <filter-name>ReplaceSiteNameFilter</filter-name>
    <filter-class>
      coreservlets.filters.ReplaceSiteNameFilter
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>ReplaceSiteNameFilter</filter-name>
    <url-pattern>/plugSite/page2.jsp</url-pattern>
  </filter-mapping>
…
</web-app>
```

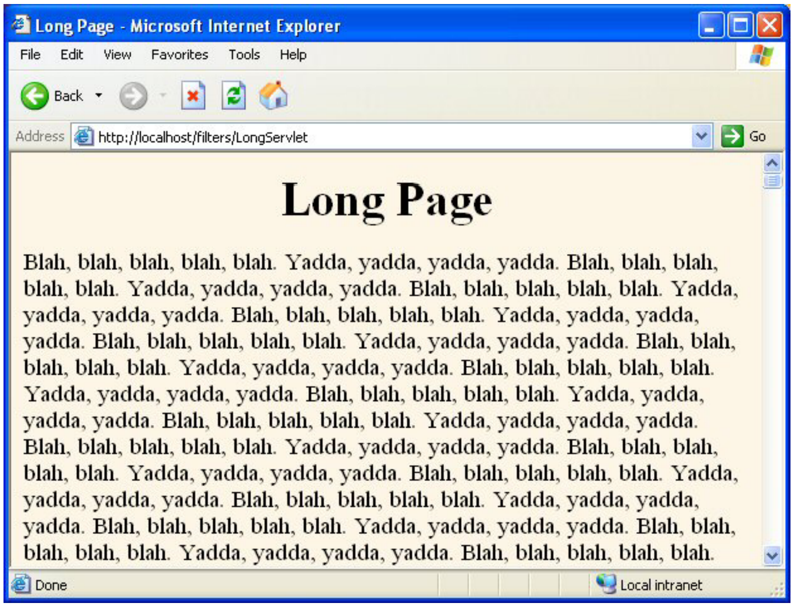
A Specific Replacement Filter (Results)



A Compression Filter

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws ServletException, IOException {
  HttpServletRequest req = (HttpServletRequest)request;
  HttpServletResponse res = (HttpServletResponse)response;
  if (!isGzipSupported(req)) {
    chain.doFilter(req,res);
  } else {
    res.setHeader("Content-Encoding", "gzip");
    StringWrapper responseWrapper =
      new StringWrapper(res);
    chain.doFilter(req,responseWrapper);
    ByteArrayOutputStream byteStream =
      new ByteArrayOutputStream();
    GZIPOutputStream zipOut =
      new GZIPOutputStream(byteStream);
    OutputStreamWriter tempOut =
      new OutputStreamWriter(zipOut);
    tempOut.write(responseWrapper.toString());.
    tempOut.close();
    OutputStream realOut = res.getOutputStream();
    byteStream.writeTo(realOut);
  }}
```

Long Page



Compression Filter: Results

- **Speedup: 12 fold on 28.8K modem connection**
- **Compression: 300 fold**
- **Use caution in generalizing benchmarks**

Dilbert used with permission of United Syndicates Inc.

Summary

- **Implement the Filter interface**
- **Override doFilter, init, and destroy**
  - init and destroy are often empty
- **Declare filter in web.xml**
  - Give it a name and designate URLs to which it applies
- **Accessing the servlet context**
  - Look it up in init, store it in a field
- **Filters have init parameters**
- **Blocking resources**
  - Simply omit call to FilterChain.doFilter
- **Modifying response**
  - Pass a wrapper to resource, invoke resource, extract output from wrapper, modify it, pass it to client.

Questions?