一、文件上传

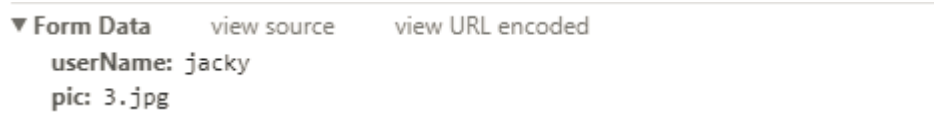
1.1 功能分析

实现文件上传需要明确的事情：
1） 也是需要使用表单来提交数据；
2） 表单的请求方式必须是 Post，而不能够 Get 方式；
3） 在表单标签中指定 enctype=”multipart/formdata”的属性；

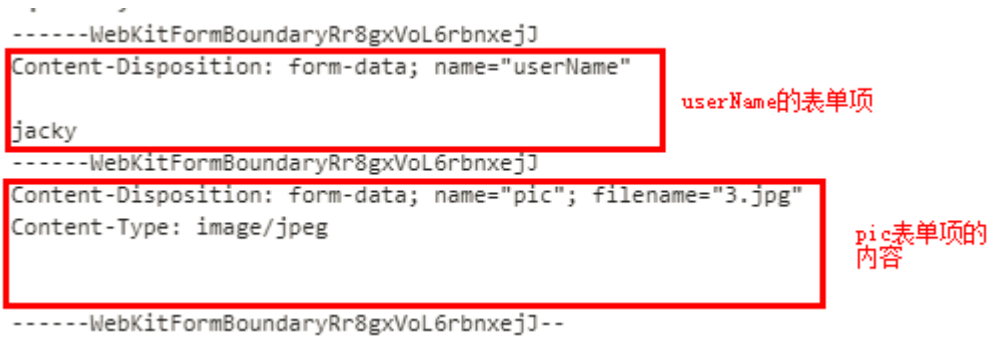
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <!-- 文件上传 -->
    <form action="${pageContext.request.contextPath}/upload.do"
method="post"
        enctype="multipart/form-data">
        用户名：<input type="text" name="userName"/><br/>
        大头照：<input type="file" name="pic"/><br/>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

文件上传表单与普通表单的区别：
1） 普通表单不需要指定 enctype=”multipart/form-data”属性；而文件上传表单就必须要指定该属性的值为“multipart/form-data”；
2） 普通表单只能够提交文本数据；而文件上传表单可以提交文本数据，也可以提交二进制数据；

如果是普通表单提交到服务器中数据的格式：



如果是文件上传表单，提交到服务器中的数据格式：



从上面可以看到，每一个表单项都包含自己的头信息。例如：

- ✓ Content-Diposition：表单项的配置相信
- ✓ name：表单项的名字；
- ✓ Content-Type：表单项的类型；
- ✓ filename：上传文件的名字；

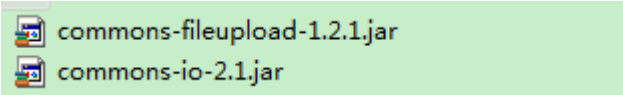
注意：如果是文件上传表单，就不能够再使用 request.getParameter 来获取表单项。如果要获取文件上传表单项，可以使用 request.getInputStream 方法来获取。该方法返回一个 ServletInputStream 的输入流对象。通过该对象来解析每一个表单项的数据。但是，在实际开发中，我们并不会自己来解析该对象的数据。而是使用第三方工具帮我们解析该对象的数据。

1.2 fileupload 介绍

fileupload 是 Apache 组织下的 commons 组件中的其中一个功能。commons-fileupload 的主要作用是帮我们解析 ServletInputStream 对象。

1.2.1 使用 fileupload 实现文件上传

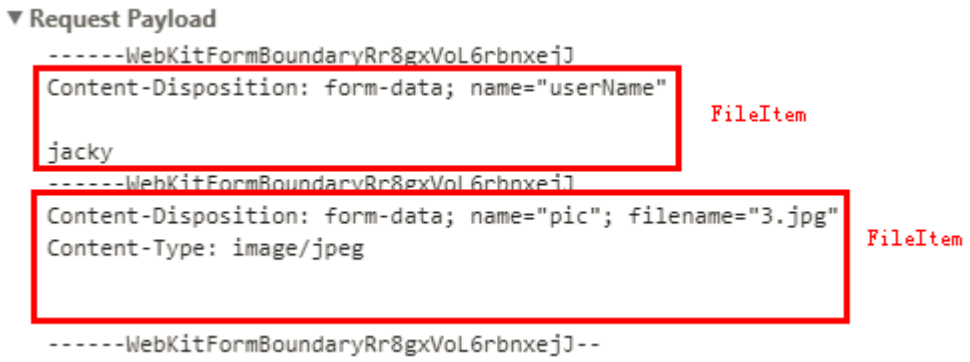
1.2.1.1 导入文件上传相关的 jar 包



1.2.1.2 解析 request 对象

fileupload 工具提供了一些解析 Request 对象的方法，比如说：

- ✓ FileItem：一个 FileItem 就代表一个表单项；



- ✓ DiskFileItemFactory： FileItem 的工厂类，负责产生 FileItem 对象；
- ✓ ServletFileUpload ： ServletInputStream 的解析器对象，负责解析 ServletInputStream 对象；

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //创建一个FileItem的工厂对象
        DiskFileItemFactory factory = new DiskFileItemFactory();
        //创建解析器对象
        ServletFileUpload servletFileUpload = new
ServletFileUpload(factory);
        try {
            //使用解析器解析请求
            List<FileItem> fileItems =
servletFileUpload.parseRequest(request);
        } catch (FileUploadException e) {
            e.printStackTrace();
        }
    }
```

1.2.1.3 获取表单项的数据

通过 FileItem 对象提供了一些方法用于获取表单项的数据，例如：

- ✓ getName()：获取上传文件的名称；
- ✓ getFieldName()：获取表单项的 name 属性值；
- ✓ isFormField()：判断该表单项是否是普通表单项，如果该方法返回 true，那么就是普通的表单项；否则就是文件上传表单项；
- ✓ getString()：获取普通表单项的内容；
- ✓ getInputStream()：获取文件上传的输入流对象；
- ✓ getContentType()：获取表单项的类型；例如：image/jpeg。
- ✓ getSize()：获取上传文件的大小，以字节为单位；

1.2.1.4 代码实现

```
@WebServlet("/upload.do")
public class UploadServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //解决响应的中文乱码
        response.setContentType("text/html;charset=utf-8");
        PrintWriter writer = response.getWriter();
        //创建一个FileItem的工厂对象
        DiskFileItemFactory factory = new DiskFileItemFactory();
        //创建解析器对象
        ServletFileUpload servletFileUpload = new
ServletFileUpload(factory);
        try {
            //使用解析器解析请求
            List<FileItem> fileItems =
servletFileUpload.parseRequest(request);
            //遍历所有的FileItem
            for (FileItem fi : fileItems) {
                /*//System.out.println(fi.getName());
                //System.out.println(fi.getFieldName());
                //System.out.println(fi.getFieldName() + ": " +
(fi.isFormField() ? "普通表单项" : "文件上传表单项"));
                //System.out.println(fi.getString());
                System.out.println(fi.getContentType());*/

                if (fi.isFormField()) { //普通表单项
                    //就直接获取内容
                    String content = fi.getString();
                    writer.write("普通表单项的内容: " + content + "<br/>");
                } else {
                    //获取输入流
                    InputStream inputStream = fi.getInputStream();
                    //获取uploads文件夹在磁盘上的路径
                    String savePath =
this.getServletContext().getRealPath("/uploads");
                    //创建数据的输出通道
                    FileOutputStream fos = new FileOutputStream(savePath +
"/" + fi.getName());
                    int len = -1;
                    byte[] buf = new byte[1024];
```

```
                    while ((len = inputStream.read(buf)) != -1) {
                        fos.write(buf, 0, len);
                    }
                    writer.write("文件“" + fi.getName() + "”上传成功!<br/>");
                    writer.write("上传文件的类型： " + fi.getContentType() +
"<br/>");
                    writer.write("上传文件的大小：" + fi.getSize() + "<br/>");
                    //关闭资源
                    fos.close();
                }
            }
        } catch (FileUploadException e) {
            e.printStackTrace();
        }
    }

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

}
```

1.2.2 文件上传的细节问题

1.2.2.1 保存路径问题

如果把文件保存到项目的根路径下，那么用户就可以直接访问该文件。这样就有可能会产生安全问题。

➢ 解决办法：

1）在实际开发中，不会把文件保存到项目的根路径下。而是保存到 WEB-INF 目录下。

```
//获取输入流
InputStream inputStream = fi.getInputStream();
//获取uploads文件夹在磁盘上的路径
String savePath = this.getServletContext().getRealPath("/WEB-INF/uploads");
//创建数据的输出通道
FileOutputStream fos = new FileOutputStream(savePath + "/" + fi.getName());
int len = -1;
byte[] buf = new byte[1024];
while ((len = inputStream.read(buf)) != -1) {
    fos.write(buf, 0, len);
}
writer.write("文件“" + fi.getName() + "”上传成功！<br/>");
writer.write("上传文件的类型：" + fi.getContentType() + "<br/>");
writer.write("上传文件的大小：" + fi.getSize() + "<br/>");
//关闭资源
fos.close();
```

2）限制上传文件的类型；

```
/**
     * 判断FileItem文件表单项是否是图片，如果是就返回true，否则返回false。
     * @param fi
     * @return
     */
    public boolean isValid(FileItem fi) {
        String contentType = fi.getContentType(); //例如：image/jpeg
        String regex = "image/[a-zA-Z]{3,4}"; //验证图片的正则
        return contentType.matches(regex);
    }
```

1.2.2.2 文件名中文乱码

对于文件上传表单项，如果普通表单项的内容出现中文，那么在服务器中就可能会出现中文乱码。

request.setCharacterEncoding 方法不能够解决普通表单项的中文乱码问题。这时候可以在调用 FileItem 对象的 getString 方法的时候，指定使用的字符集。

```
if (fi.isFormField()) { //普通表单项
    //使用指定的字符集获取普通表单项的内容
    String content = fi.getString("utf-8");
    writer.write("普通表单项的内容：" + content + "<br/>");
} else {
```

如果是文件上传表单项的文件名出现中文，那么可以通过 request.setCharacterEncoding 方法解决文件名的中文乱码问题。

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //处理Post请求的中文乱码
    request.setCharacterEncoding("utf-8");
    doGet(request, response);
}
```

1.2.2.3 限制上传文件的大小

ServletFileUpload 对象中提供了 setFileSizeMax 方法来限制上传文件的大小。如果上传文件超过了指定的大小，JVM 就会抛出一个异常。

```
org.apache.commons.fileupload.FileUploadBase$FileSizeLimitExceededException: The field pic exceeds its maximum permitted  size
        at org.apache.commons.fileupload.FileUploadBase$FileItemIteratorImpl$FileItemStreamImpl$1.raiseError(FileUploadBase.jav
        at org.apache.commons.fileupload.util.LimitedInputStream.checkLimit(LimitedInputStream.java:71)
        at org.apache.commons.fileupload.util.LimitedInputStream.read(LimitedInputStream.java:128)
        at java.io.FilterInputStream.read(FilterInputStream.java:107)
        at org.apache.commons.fileupload.util.Streams.copy(Streams.java:94)
        at org.apache.commons.fileupload.util.Streams.copy(Streams.java:64)
        at org.apache.commons.fileupload.FileUploadBase.parseRequest(FileUploadBase.java:362)
        at org.apache.commons.fileupload.servlet.ServletFileUpload.parseRequest(ServletFileUpload.java:126)
        at com.entor.servlet.UploadServlet.doGet(UploadServlet.java:39)
        at com.entor.servlet.UploadServlet.doPost(UploadServlet.java:96)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:641)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:722)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:304)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:210)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:224)
```

【代码】

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //解决响应的中文乱码
        response.setContentType("text/html;charset=utf-8");
        PrintWriter writer = response.getWriter();
        //创建一个FileItem的工厂对象
        DiskFileItemFactory factory = new DiskFileItemFactory();
        //创建解析器对象
        ServletFileUpload servletFileUpload = new
ServletFileUpload(factory);
        //限制上传文件不能够超过10Kb
        servletFileUpload.setFileSizeMax(1024 * 10);
        try {
            //使用解析器解析请求
            List<FileItem> fileItems =
servletFileUpload.parseRequest(request);
            //遍历所有的FileItem
            for (FileItem fi : fileItems) {
                /*//System.out.println(fi.getName());
                //System.out.println(fi.getFieldName());
                //System.out.println(fi.getFieldName() + ": " +
(fi.isFormField() ? "普通表单项" : "文件上传表单项"));
                //System.out.println(fi.getString());
                System.out.println(fi.getContentType());*/

                if (fi.isFormField()) { //普通表单项
                    //使用指定的字符集获取普通表单项的内容
```

```
                    String content = fi.getString("utf-8");
                    writer.write("普通表单项的内容：" + content + "<br/>");
                } else {
                    if (isValid(fi)) { //如果是有效的图片，那么就执行上传。
                        //获取输入流
                        InputStream inputStream = fi.getInputStream();
                        //获取uploads文件夹在磁盘上的路径
                        String savePath =
this.getServletContext().getRealPath("/WEB-INF/uploads");
                        //创建数据的输出通道
                        FileOutputStream fos = new FileOutputStream(savePath
+ "/" + fi.getName());
                        int len = -1;
                        byte[] buf = new byte[1024];
                        while ((len = inputStream.read(buf)) != -1) {
                            fos.write(buf, 0, len);
                        }
                        writer.write("文件“" + fi.getName() + "”上传成功!
<br/>");
                        writer.write("上传文件的类型：" + fi.getContentType()
+ "<br/>");
                        writer.write("上传文件的大小：" + fi.getSize() +
"<br/>");
                        //关闭资源
                        fos.close();
                    } else {
                        request.setAttribute("msg", "只能上传图片");

    request.getRequestDispatcher("/upload.jsp").forward(request,
response);
                    }

                }
            }
        } catch (FileSizeLimitExceededException e) {
            request.setAttribute("msg", "上传文件太大了！");
            request.getRequestDispatcher("/upload.jsp").forward(request,
response);
        } catch (FileUploadException e) {
            e.printStackTrace();
        }
    }
```

二、Servlet 过滤器

2.1 过滤器的介绍

过滤器是 Servlet 三大对象之一，它是用来拦截浏览器发送过来的请求。
当浏览器访问 servlet 的时候，如果配置了过滤器，那么就会先经过过滤器。如果过滤器不放行，那么就不会进入 Servlet 中。只有在过滤器中执行放行操作，才有访问请求的 Servlet。

2.2 使用过滤器的步骤

第一步：定义一个类，并实现 Filter 的接口，并重写过滤器的方法。

```
/*
    自定义过滤器
*/
public class MyFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("执行过滤器...");
    }

    @Override
    public void destroy() {

    }

}
```

第二步：在 web.xml 文件中配置该 Filter；

```
<!-- 配置过滤器 -->
  <filter>
    <!-- Filter的名字 -->
    <filter-name>MyFilter</filter-name>
    <!-- Filter的完整类名 -->
```

```
    <filter-class>com.chinasofti.filter.MyFilter</filter-class>
  </filter>

  <filter-mapping>
    <!-- 注意：该名字必须要与上面的Filter的名字相同 -->
    <filter-name>MyFilter</filter-name>
    <!-- 配置拦截的路径 -->
    <url-pattern>/download.jsp</url-pattern>
  </filter-mapping>
```

与 Servlet 不同，过滤器定义完成后不需要客户端主动调用，它会根据拦截路径自动执行的。

2.3 过滤器的生命周期方法

init()：服务器启动时候，创建过滤器对象，然后再执行 init 方法。在 init 方法中，一般会执行一些初始化的操作。
doFilter()：每次拦截请求后执行的方法。
destroy()：过滤器被销毁前，服务器会自动调用该方法。在 destroy 方法中，可以执行一些资源回收的操作。

2.4 FilterConfig 对象

作用：获取过滤器的配置参数。

```
<!-- 配置过滤器 -->
<filter>
  <!-- Filter的名字 -->
  <filter-name>MyFilter</filter-name>
  <!-- Filter的完整类名 -->
  <filter-class>com.entor.filter.MyFilter</filter-class>
  <!-- 配置过滤器参数 -->
  <init-param>
      <param-name>name</param-name>
      <param-value>小宝</param-value>
  </init-param>
</filter>
```

```
@Override
public void init(FilterConfig filterConfig) throws ServletException {
    System.out.println("初始化过滤器...");

    //获取过滤器的配置参数
    String name = filterConfig.getInitParameter("name");
    System.out.println("name = " + name);
}
```

2.5 配置路径

过滤器的路径与 Servlet 的路径的配置相同的。配置路径的要求：
1） 要么以“/”开头，要么以“*”开头；
2） 如果在路径中使用星号，那么星号就不是通配符，而是一个普通的字符；
3） 可以使用多个 url-pattern 配置多个路径；

2.6 拦截方式

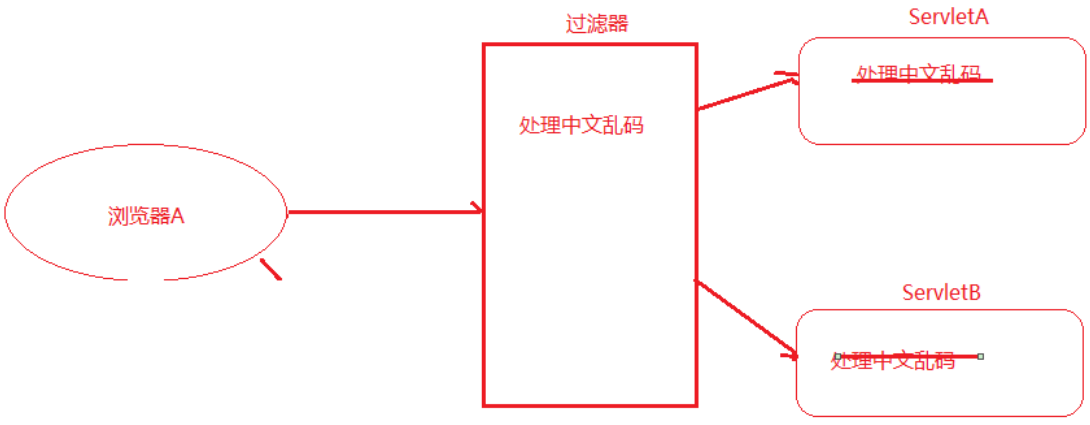
REQUEST：默认拦截方式。它只会对浏览器发起的请求进行拦截；
FORWARD：当执行请求转发之前会执行过滤器；
INCLUDE：当一个页面包含另外一个页面的时候执行过滤器；
ERROR：如果在 web.xml 文件中配置 error-page 节点，那么当程序发生异常的时候，就会先执行过滤器，然后再跳转到 location 节点指定的页面；

2.7 过滤器链

如果在同一个资源上配置了多个过滤器，那么他们的执行顺序：先配置的过滤器就会先启动，然后再执行放行，最后按照配置顺序的相反方向执行放行后的代码。

2.8 过滤器的应用

2.8.1 解决中文乱码问题



如果浏览器提交数据给 Servlet 的时候包含中文参数，那么在 Servlet 中就要处理中文乱码。但是，如果有多个 Servlet 都要同时接收中文参数，那么在 Servlet 中处理中文乱码就比较麻烦。

解决办法：使用过滤器来处理中文乱码。

```
/*
    处理中文乱码
*/
public class CharacterEncodingFilter implements Filter {
    private String charset;
```

```
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        charset = filterConfig.getInitParameter("charset");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        //放行
        chain.doFilter(req, response);
    }

    @Override
    public void destroy() {

    }

}
```

如果要处理 Get 请求的中文乱码，这时候需要对 request 对象进行增强处理。

➢ 使用装饰着模式增强 Request 的步骤：

第一步：创建一个类，继承 HttpServletRequestWrapper，并重写 getParameter 方法；

```
/*
    RequestFacade：HttpServletRequest的装饰类

*/
public class RequestFacade extends HttpServletRequestWrapper {
    //被增强的类
    private HttpServletRequest request;

    public RequestFacade(HttpServletRequest request) {
        super(request);
        this.request = request;
    }

    @Override
    public String getParameter(String name) {
        //获取请求参数
        String content = request.getParameter(name);
        //如果请求参数不为null，而且是get请求，那么就处理中文乱码
```

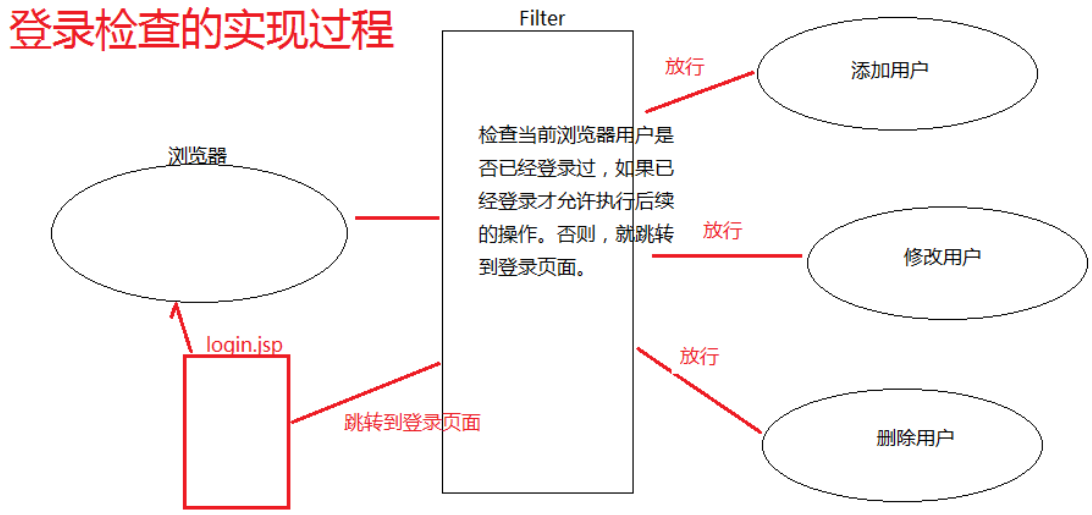
```
        if (content != null && "get".equalsIgnoreCase(request.getMethod()))
{
            try {
                //处理Get请求的中文乱码
                content = new String(content.getBytes("iso-8859-1"),
"utf-8");
            } catch (UnsupportedEncodingException e) {
                throw new RuntimeException();
            }
        }
        return content;
    }

}
```

在 getParameter 方法中获取请求参数，然后再进行中文乱码的处理。

第二步：在过滤器中对使用上面定义的类对 Request 对象进行封装，然后在传给 Servlet；

```
@Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        //处理Post中文乱码
        request.setCharacterEncoding(charset);
        //对Request对象的功能进行增强（装饰者模式）
        RequestFacade requestFacade = new RequestFacade(req);
        //放行
        chain.doFilter(requestFacade, response);
    }
```

2.8.2 实现登录认证



在实际开发中，执行后台操作之前都必须要登陆后才可以执行。所以，可以通过过滤器来实现登录拦截的功能。用户每次执行操作之前，都需要先进行过滤器中检查用户是否已经登录。如果登陆过就直接放行，否则，就跳转到登录页面。

➢ 实现登录检查的步骤：

第一步：创建登录的 JSP 页面；

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="${pageContext.request.contextPath}/login.do"
method="post">
        用户名：<input type="text" name="userName"/><br/>
        密码：<input type="password" name="userPass"/><br/>
        <input type="submit" value="登录"/><br/>
    </form>
</body>
</html>
```

第二步：创建一个 Servlet，实现登录功能；

```
@WebServlet("/login.do")
```

```
public class LoginServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //获取表单的参数
        String userName = request.getParameter("userName");
        String userPass = request.getParameter("userPass");

        if ("admin".equals(userName) && "123".equals(userPass)) {
            //记录用户的登录状态
            request.getSession().setAttribute("loginUser", userName);
            response.sendRedirect(request.getContextPath() + "/main.jsp");
        } else {
            request.setAttribute("msg", "用户名或密码不正确！");
            request.getRequestDispatcher("/login.jsp").forward(request,
response);
        }
    }


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

}
```

第三步：定义登录过滤器；

```
/*
    登录过滤器：负责登录检查
*/
public class LoginFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
```

```
        //获取请求的路径
        String requestURI = req.getRequestURI();
        if (requestURI.endsWith("login.jsp") ||
requestURI.endsWith("login.do")) {
            chain.doFilter(request, response);
        } else {
            Object o = req.getSession().getAttribute("loginUser");
            //如果o不为null就放行，否则就跳转到登录页面
            if (o != null) {
                chain.doFilter(request, response);
            } else {
                resp.sendRedirect(req.getContextPath() + "/login.jsp");
            }
        }
    }

    @Override
    public void destroy() {

    }

}
```

第四步：配置登录过滤器；

```
<filter>
        <!-- Filter的名字 -->
        <filter-name>LoginFilter</filter-name>
        <!-- Filter的完整类名 -->
        <filter-class>com.chinasofti.filter.LoginFilter</filter-class>
    </filter>

    <filter-mapping>
        <!-- 注意：该名字必须要与上面的Filter的名字相同 -->
        <filter-name>LoginFilter</filter-name>
        <!-- 配置拦截的路径 -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

2.8.3 实现七天免登录

实现七天免登录功能：cookie 和过滤器来实现。

➢ 实现思路：

1） 用户登录成功，把用户的信息保存 Session 和 Cookie。

2）定义一个过滤器，该过滤从 Cookie 获取用户的登录信息。如果用户没有登录，那么就从 cookie 中获取登录信息，然后把登录信息添加到 Session。

➢ 实现步骤：

第一步：在登录页面添加一个 checkbox 按钮。

```
<form action="${pageContext.request.contextPath}/login.do" method="post">
    用户名：<input type="text" name="userName"/><br/>
    密码：<input type="password" name="userPass"/><br/>
    <input type="checkbox" name="memberPass" value="1"/>下次自动登录<br/>
    <input type="submit" value="登录"/><br/>
</form>
```

第二步：修改登录的 Servlet。如果用户登录成功，而且用户勾选了“下次免登录”，那么就把用户信息保存到 Cookie 中。否则，就从 Cookie 中删除用户登录信息。

```
//获取表单的参数
        String userName = request.getParameter("userName");
        String userPass = request.getParameter("userPass");

        if ("admin".equals(userName) && "123".equals(userPass)) {
            //记录用户的登录状态
            request.getSession().setAttribute("loginUser", userName);

            String memberPass = request.getParameter("memberPass");
            if (memberPass != null) {
                //保存用户的登录信息到Cookie
                Cookie c = new Cookie("loginUser", userName);
                c.setMaxAge(60 * 60 * 24 * 7);
                response.addCookie(c);
            } else {
                //删除cookie
                Cookie c = new Cookie("loginUser", null);
                c.setMaxAge(0);
                response.addCookie(c);
            }

            response.sendRedirect(request.getContextPath() + "/main.jsp");
        } else {
            request.setAttribute("msg", "用户名或密码不正确！");
            request.getRequestDispatcher("/login.jsp").forward(request,
response);
        }
```

第三步：新建一个过滤器，该过滤器用来判断用户是否登录，如果没有登录，那么就从 cookie 查找是否包含该用户的登录信息，如果有就把用户的登录信息设置到 Session 中。

```
/*
    cookie过滤器：实现自动登录的功能
*/
public class CookieFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;

        //如果用户已经登录了，就直接放行
        Object o = req.getSession().getAttribute("loginUser");
        if (o == null) {
            Cookie[] cookies = req.getCookies();
            if (cookies != null) {
                for (Cookie c : cookies) {
                    //获取Cookie的名字
                    String name = c.getName();
                    if ("loginUser".equals(name)) {
                        String value = c.getValue();
                        //把用户登录信息保存到Session
                        req.getSession().setAttribute("loginUser", value);
                        break;
                    }
                }
            }
        }
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {

    }
```

```
}
```

第二步：在 web.xml 文件中配置该过滤器。

```
<filter>
        <!-- Filter的名字 -->
        <filter-name>CookieFilter</filter-name>
        <!-- Filter的完整类名 -->
        <filter-class>com.chinasofti.filter.CookieFilter</filter-class>
    </filter>

    <filter-mapping>
        <!-- 注意：该名字必须要与上面的Filter的名字相同 -->
        <filter-name>CookieFilter</filter-name>
        <!-- 配置拦截的路径 -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <!-- Filter的名字 -->
        <filter-name>LoginFilter</filter-name>
        <!-- Filter的完整类名 -->
        <filter-class>com.chinasofti.filter.LoginFilter</filter-class>
    </filter>

    <filter-mapping>
        <!-- 注意：该名字必须要与上面的Filter的名字相同 -->
        <filter-name>LoginFilter</filter-name>
        <!-- 配置拦截的路径 -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

注意：该过滤器必须要放在登录过滤器的前面定义。

三、Servlet 监听器

3.1 Servlet 监听器介绍

监听器也是 Servlet 组件之一，用来监听对象或对象属性状态的变化。当对象或对象属性状态发生变化的时候，会自动执行该对象监听器对应的方法。

JavaWeb 中的事件：

- ✓ 事件源：ServletContext、ServletRequest、HttpSession 对象。
- ✓ 事件类型：ServletContextEvent、ServletRequestEvent、HttpSessionEvent、ServletContextAttributeEvent、ServletRequestAttributeEvent、HttpSessionAttributeEvent。
- ✓ 监听器：ServletContextListener、ServletRequestListener、HttpSessionListener、ServletContextAttributeListener、ServletRequestAttributeListener、HttpSessionAttributeListener。
- ✓ 响应动作：监听器对象中的方法。

3.1 使用监听器的步骤

第一步：定义一个类，继承 XxxListener 接口，并重写它的所有抽象方法；

```
/*
 * MyServletContextListener: 负责监听ServletContext对象的创建和销毁
 *
 *      ServletContext的创建：服务器启动的时候自动创建。
 *      ServletContext的销毁：关闭服务器或重启服务器的时候销毁。
 */
public class MyServletContextListener implements ServletContextListener {

    //创建ServletContext对象的时候自动调用该方法
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("ServletContext对象被创建了...");

    }

    //ServletContext对象被销毁的时候自动执行该方法
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("ServletContext对象被销毁了...");

    }

}
```

第二步：在 web.xml 文件中配置监听器；

```
<!-- 配置监听器 -->
  <listener>
    <!-- 监听器的完整类名 -->

    <listener-class>com.chinasofti.listener.MyServletContextListener</li
```

```
stener-class>
  </listener>
```

3.2 常见的监听器

3.2.1 ServletContextListener

ServletContextListener 用于监听 ServletContext 对象的创建和销毁。

在实际开发中，我们可以使用 ServletContextListener 实现如下需求：
1）在服务器启动的时候把磁盘上的一些不会经常改变的数据读取到 ServletContext 对象中保存起来；
2）服务器启动的时候可以执行一些初始化的操作，比如创建表。在服务器关闭的时候，执行一些资源销毁的操作，比如删除表。

3.2.2 ServletContextAttributeListener

ServletContextAttributeListener 用于监听 ServletContext 属性状态的变化。

```
/*
 * MyServletContextAttributeListener: 负责监听ServletContext对象属性的操作
 */
public class MyServletContextAttributeListener implements
ServletContextAttributeListener {

    //往ServletContext对象中添加属性时候自动执行该方法
    @Override
    public void attributeAdded(ServletContextAttributeEvent scab) {
        System.out.println("属性名: " + scab.getName());
        System.out.println("属性值: " + scab.getValue());
    }

    //删除ServletContext对象中属性时候自动执行该方法
    @Override
    public void attributeRemoved(ServletContextAttributeEvent scab) {
        System.out.println("删除属性: " + scab.getName());
    }

    //修改ServletContext对象中属性时候自动执行该方法
    @Override
    public void attributeReplaced(ServletContextAttributeEvent scab) {
        System.out.println("属性名: " + scab.getName());
```

```
        System.out.println("修改前的属性值：" + scab.getValue());
        System.out.println("修改后的属性值：" +
scab.getServletContext().getAttribute(scab.getName()));
    }

}
```

配置监听器：

```
<listener>

    <listener-class>com.chinasofti.listener.MyServletContextAttributeLis
tener</listener-class>
  </listener>
```

3.2.3 ServletRequestListener

ServletRequestListener 用于监听 Request 对象的创建和销毁。

```
/*
 * MyServletRequestListener: 负责监听ServletRequest对象的创建和销毁
 *
 */
public class MyServletRequestListener implements ServletRequestListener {

    //请求结束的时候自动调用
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("request对象被销毁了！");
    }

    //创建REquest对象的时候自动创建的
    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        //获取客户端的ip地址
        String ip = sre.getServletRequest().getLocalAddr();
        System.out.println("客户端的IP地址：" + ip);

    }

}
```

配置监听器：

```
<listener>
```

```
    <listener-class>com.chinasofti.listener.MyServletRequestListener</li
stener-class>
  </listener>
```

3.2.4 ServletRequestAttributeListener

ServletRequestAttributeListener 用于监听 Request 对象属性状态的变化。

```
/*
 * MyServletRequestAttributeListener: 负责监听
ServletRequestAttributeListener对象属性的操作
 *
 */
public class MyServletRequestAttributeListener implements
ServletRequestAttributeListener {

    //添加属性的时候自动调用该方法
    @Override
    public void attributeAdded(ServletRequestAttributeEvent srae) {
        System.out.println("属性名: " + srae.getName());
        System.out.println("属性值: " + srae.getValue());
    }

    //删除属性的时候自动调用该方法
    @Override
    public void attributeRemoved(ServletRequestAttributeEvent srae) {
        System.out.println("删除属性: " + srae.getName());
    }

    //修改属性的时候自动调用该方法
    @Override
    public void attributeReplaced(ServletRequestAttributeEvent srae) {
        System.out.println("属性名: " + srae.getName());
        System.out.println("修改前的属性值: " + srae.getValue());
        System.out.println("修改后的属性值: " +
srae.getServletRequest().getAttribute(srae.getName()));
    }

}

<listener>
```

```
    <listener-class>com.chinasofti.listener.MyServletRequestAttributeLis
tener</listener-class>
  </listener>
```

3.2.5 HttpSessionListener

HttpSessionListener 用于监听 Session 对象的创建和销毁。

例如：统计当前网站的访问人数。

```
/*
 * HttpSessionListener: 负责监听session对象的创建和销毁
 *
 */
public class MySessionListener implements HttpSessionListener {
    private int count = 0; //记录当前访问人数

    //创建Session时候自动调用该方法
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        count++;
        System.out.println("有用户进来了，当前网站的人数：" + count);
    }

    //session被销毁时候自动调用
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        count--;
        System.out.println("有用户离线了，当前网站的人数：" + count);

    }

}
```

配置监听器：

```
<listener>

    <listener-class>com.chinasofti.listener.MySessionListener</listener-
class>
  </listener>
```

3.2.6 HttpSessionAttributeListener

HttpSessionAttributeListener 监听器用于监听 session 属性状态的变化。

例如：显示当前网站的所有登录用户。

```
/*
 * HttpSessionAttributeListener: 负责监听session对象属性。
 *
 */
public class MySessionAttributeListener implements
HttpSessionAttributeListener {
    private ArrayList list = new ArrayList(); //保存登录用户的名字

    //添加属性的时候自动调用
    @Override
    public void attributeAdded(HttpSessionBindingEvent se) {
        if ("loginUser".equals(se.getName())) {
            String name = (String)
se.getSession().getAttribute("loginUser");
            list.add(name);
            System.out.println("当前在线用户：" + list);
        }
    }

    //删除属性的时候自动调用
    @Override
    public void attributeRemoved(HttpSessionBindingEvent se) {
        //获取删除前的session属性的值
        String name = (String) se.getValue();
        System.out.println("删除前的属性值：" + name);
        list.remove(name);
        System.out.println("当前在线用户：" + list);
    }

    //修改属性的时候自动调用
    @Override
    public void attributeReplaced(HttpSessionBindingEvent se) {
        // TODO Auto-generated method stub

    }

}
```

配置监听器：

```
<listener>

    <listener-class>com.chinasofti.listener.MySessionAttributeListener</
listener-class>
  </listener>
```

编写测试的Servlet类：

```
/**
 * 用户登录
 */
@WebServlet("/login.do")
public class LoginServlet extends HttpServlet {



    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        //验证用户输入的验证码是否正确
        String verCode = request.getParameter("verCode");
        String vc = (String) request.getSession().getAttribute("verCode");
        if (!verCode.equalsIgnoreCase(vc)) {
            request.setAttribute("msg", "验证码不正确！");
            request.getRequestDispatcher("/index.jsp").forward(request,
response);
        } else {
            //获取用户名和密码
            String userName = request.getParameter("userName");
            String userPass = request.getParameter("userPass");

            if ("admin".equals(userName) && "123".equals(userPass)) {
                request.getSession().setAttribute("loginUser", userName);
                response.getWriter().write("<h1>登录成功！</h1><a href='" +
request.getContextPath() + "/logout.do'>注销</a> ");
            } else {
                request.setAttribute("msg", "用户名或密码不正确！");

    request.getRequestDispatcher("/index.jsp").forward(request,
response);
            }
        }
    }
```

```
}


@WebServlet("/logout.do")
public class LogoutServlet extends HttpServlet {


    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        HttpSession session = request.getSession();
        session.removeAttribute("loginUser");
        response.sendRedirect(request.getContextPath() + "/index.jsp");
    }

}
```

【index.jsp】

```
%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
<script>
    //刷新验证码
    function refresh() {
        //设置图片的src属性
        var imgNode = document.getElementsByTagName("img")[0];
        imgNode.src =
"${pageContext.request.contextPath}/getVerCode.do?r=" + Math.random();
    }
</script>
</head>
<body>
    <form action="${pageContext.request.contextPath}/login.do"
method="post">
        用户名：<input type="text" name="userName"/><br/>
        密码：<input type="password" name="userPass"/><br/>
        验证码：<input type="text" name="verCode"/>
        <img src="${pageContext.request.contextPath}/getVerCode.do"
title="看不清，点击刷新..."
```

```
            onclick="refresh()"/><br/>
        <input type="submit" value="登录"/>
        <span style="color:red">${msg}</span>
    </form>
</body>
</html>
```

注意：监听器的执行顺序是按照它们配置的先后顺序执行。

四、Servlet 零配置

从 Servlet3.0 以后，Servlet 支持使用注解的配置方式。使用配置可以大大简化 Servlet 的配置工作，提高开发效率。

4.1 @WebServlet

@WebServlet 注解用于描述一个 Servlet 类。该注解将会在部署时被容器处理，容器将根据具体的属性配置将相应的类部署为 Servlet。

注解属性包含有：

属性名	类型	属性描述
name	String	指定servlet的name属性,等价于<Servlet-name>.如果没有显示指定，则该servlet的取值即为类的全限定名.
value	String[]	等价于urlPatterns,二者不能共存.
urlPatterns	String[]	指定一组servlet的url的匹配模式,等价于<url-pattern>标签.
loadOnStartup	int	指定servlet的加载顺序,等价于<load-on-startup>标签.
initParams	WebInitParam[]	指定一组初始化参数,等价于<init-param>标签.

```
/*
 @WebServlet：描述一个Servlet类
    name：Servlet的名字
    value：指定请求路径。如果只有一个value属性的时候，可以写value，也可以不写。
    urlPatterns：指定多个url路径，多个路径之间使用英文逗号隔开
    load-on-startup：指定创建Servlet对象的顺序。如果指定了该属性，那么服务器启动的时候就会自动创建Servlet对象。
    initParams：指定Servlet的初始化参数
 @WebInitParam：相对web.xml中的init-param节点。每一个@WebInitParam注解配置一个参数。
*/

//@WebServlet("/hello.do")
```

```
@WebServlet(urlPatterns={"/a.do", "/b.do"}, loadOnStartup=1, initParams={
        @WebInitParam(name = "charset", value = "utf-8"),
@WebInitParam(name = "username", value = "jacky")})
public class HelloServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        System.out.println("init...");
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        String charset = config.getInitParameter("charset");
        System.out.println("charset = " + charset);
        String username = config.getInitParameter("username");
        System.out.println("username = " + username);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        resp.getWriter().write("hello servlet...");
    }

}
```

如果@WebServlet 注解只有 value 属性，那么 value 可以不写，例如：@WebServlet(“/a.do”)

4.2 @WebFilter

@WebFilter 注解用于将一个类声明为过滤器。该注解将会在部署时被容器处理，容器将根据具体的属性配置将相应的类部署为过滤器。该注解具有以下一些常用属性：

属性名	类型	描述
filterName	String	指定过滤器的 name 属性，等价于 <filter-name>
value	String[]	该属性等价于 urlPatterns 属性。但是两者不应该同时使用。
urlPatterns	String[]	指定一组过滤器的 URL 匹配模式。等价于 <url-pattern> 标签。
servletNames	String[]	指定过滤器将应用于哪些 Servlet。取值是 @WebServlet 中的 name 属性的取值，或者是 web.xml 中 <servlet-name> 的取值。

```
/*
    @WebFilter：配置一个过滤器
        value：指定过滤器的路径；
```

```
        urlPatterns: 指定多个路径;
*/
//@WebFilter(urlPatterns={"/a.do", "/b.do"})
@WebFilter(servletNames={"HelloServlet"})
public class LoginFilter implements Filter {

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

    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("指定LoginFilter过滤器...");
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {

    }

}
```

4.3 @WebListener

@WebListener 注解用于将一个类声明为监听器。这样我们在 web 应用中使用监听器时，也不再需要在 web.xml 文件中配置监听器的相关描述信息了。