内容

- 域对象
- 文件上传
- 文件下载
- 过滤器

一、 域对象

HttpRequest→同一个请求，一次请求(可能存在服务器端跳转，请求转发)一次响应

HttpSession→ 同一次会话，浏览器开始访问服务器；浏览器关闭；服务也可以终止此次会话；关闭服务（正常，中止；非正常关闭，会话被迫终止）；

ServletContext→ 整个应用程序中都有效

作为数据传递的一个范围

二、 文件上传

Upload.html

```
<formaction="upload"method="post" enctype="multipart/form-data">
        <inputtype="file"name="file">
        <inputtype="submit"value="上传">
</form>
```

UploadServlet.java

使用 apache 提供的 commons-io 和 commons-fileupload 包

```
        //获取上下文路径
        String path = request.getServletContext().getRealPath("/");
//      System.out.println(path);
        //1、创建文件条目工厂
```

```
        DiskFileItemFactoryfactory=newDiskFileItemFactory();
        //2、设置缓存大小
        factory.setSizeThreshold(3*1024);
        //3、设置临时目录,该步骤可以省略
        File temp = newFile(path, "temp");
        if(!temp.exists()) {
            temp.mkdirs();
        }
        factory.setRepository(temp);
        //4、上传处理
        ServletFileUploadupload = newServletFileUpload(factory);
        //5、最大限制
        upload.setFileSizeMax(3*1024*1024);
        upload.setHeaderEncoding("utf-8");
        try {
            List<FileItem>items=upload.parseRequest(request);
            if(null!=items){
                for(FileItemitem:items){
                    //如果不是普通表单并且已选择上传内容则处理

    if(!item.isFormField()&&item.getName()!=null&&!item.getName().equals
("")){
                        String fileName=item.getName();
                        //String filePath = path + "/upload/"+fileName;
                        File uploader = newFile(path + "/upload/");
                        if(!uploader.exists()) {
                            uploader.mkdirs();
                        }
                        item.write(new File(uploader, fileName));
                    }
                }
            }
        } catch (FileUploadExceptione) {
            // TODO Auto-generated catch block
        } catch (Exception e) {
            // TODO Auto-generated catch block
        }
    }
```

三、 文件下载

文件的下载有两种方式:

(1)利用超链接的方式下载

缺点： 只有不认识的内容即浏览器不能直接打开的内容浏览器会自动下载；当浏览器碰见自己能认识的/能打开的内容时，就会直接显示出来

(2)利用第三方下载方式

①获取要下载的文件名

②相对路径转绝对路径

③向响应的头部添加信息

```
response.setHeader("content-disposition","attachment;filename="+filenam
e);
```

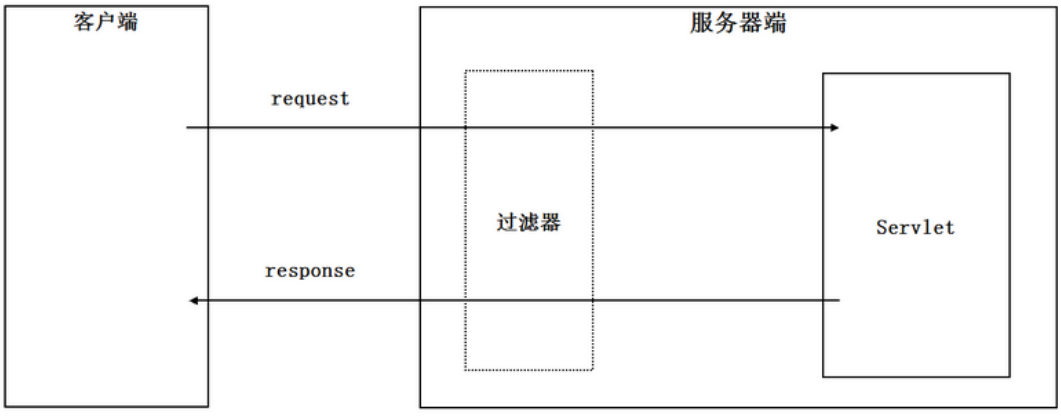
```
response.setCharacterEncoding("utf-8");
        //获取要下载的文件名称
        String filename=request.getParameter("filename");//下载的文件名称
        //下载文件的存放路径
        String
path=request.getServletContext().getRealPath("/upload/"+filename);
        System.out.println(path);
        //设置下载弹出框
        response.setHeader("content-disposition",
"attachment;filename="+filename);
        /**
         * 1.流的输入
         * 2.流的输出
         */
        File file=new File(path);
        if(file.exists()){//下载的路径存在再去执行相关操作
            FileInputStreamis=newFileInputStream(file);
            ServletOutputStreamos=response.getOutputStream();
            byte[] car=newbyte[1024];
            intlen=0;
            while(-1!=(len=is.read(car))){
```

```
            os.write(car,0,len);
        }
        os.flush();
        os.close();
        is.close();
    }else{
        System.out.println("路径不存在");
    }
```

四、 过滤器(Filter)

1、介绍

Filter 即为过滤，用于在 Servlet 之外对 Request 或者 Response 进行修改。例如，污水净化设备可以看做现实中的过滤器，它负责将污水中的杂质过滤，从而使进入的污水变成净水。对于 Web 应用程序来说，过滤器是一个驻留在服务器端的 Web 组件，它可以截取客户端和服务器端之间的请求与响应信息，对其修改或过滤。



作用如下:

①在客户访问后台资源之前拦截客户请求

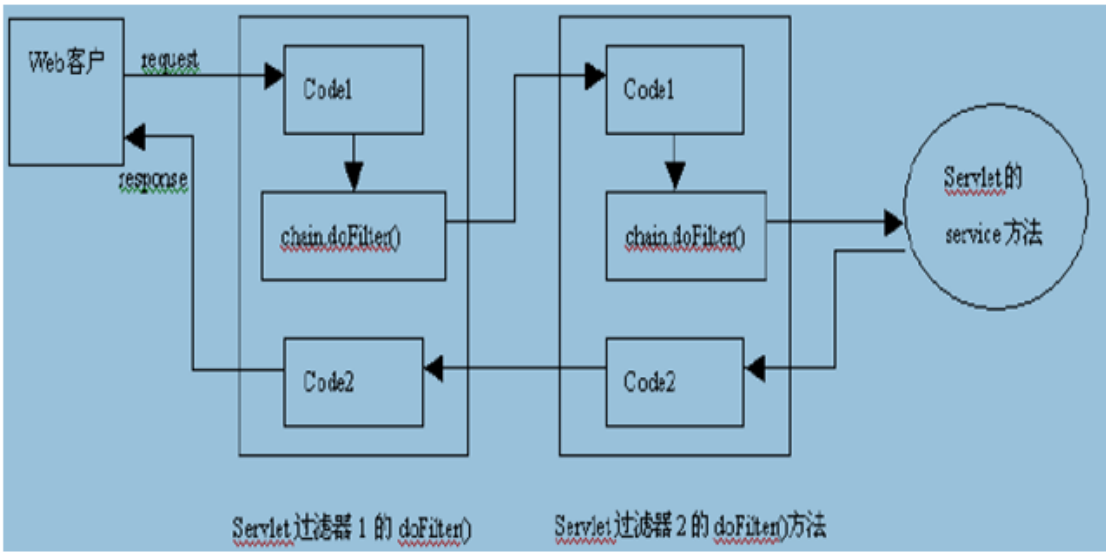
②在资源被送到客户端之前加以控制

③Servlet 过滤器负责过滤的 Web 组件可以是 Servlet、JSP 或 HTML 文件。

过滤器链：

在一个 Web 应用程序中，可以部署多个过滤器进行拦截，这些过滤器组成了一个过滤器链。过滤器链中的每个过滤器负责特定的操作和任务，客户端的请求在这些过滤器之间传递，直到服务器端的 Servlet。

若是一个过滤器链：先配置先执行(请求时的执行顺序)；响应时：以相反的顺序执行。



2、Filter 接口

接口 javax.servlet.Filter 定义了 3 个方法：

init()：初始化配置参数，在 doFilter()方法之前被调用

doFilter()：该方法在客户端请求及服务器端响应时自动调用(反复执行)

destroy()：结束过滤器，doFilter()方法完成后调用。(通常服务器关闭时销毁)

Filter 生命周期(饿汉单例)：创建、初始化、服务、销毁

3、编写步骤

①编写 java 类实现 java.servlet.Filter 接口

②web.xml 中配置(需要配置专门的拦截路径才生效)

③访问观察结果

整个流程：客户端发出请求后，先执行 filter 里面的 chain.doFilter(request,response)方法之前的代码，然后通过 chain.doFilter 调下一个 filter,若没有下一个，则调用客户端请求的 servlet/html/jsp，然后倒着执行每个 filter 里 chain.doFilter(request,response)之后的代码

web.xml 注意位置：在监听器及 servlet 之上

```
  <!—过滤器-->
<filter>
    <filter-name>helloFilter</filter-name>
    <filter-class>com.shsxt.servlet.filter.HelloFilter</filter-class>
<init-param>
        <param-name>school</param-name>
        <param-value>shsxt</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>helloFilter</filter-name>
  <!—需要配置专门的拦截路径才生效-->
    <url-pattern>/*</url-pattern>
 </filter-mapping>
  <!--监听器 -->
 <!--servlet -->
```

url-pattern 的配置

①配置具体路径/index.html/TestServlet.do

②带有通配符的配置*.do　/*　/user/* *.html　*.jsp

4、第一个实例：简单的 servlet

HelloFilter.java

```
publicclassHelloFilterimplementsFilter{
    //可以读取web.xml文件中Servlet过滤器的初始化参数
    publicvoidinit(FilterConfigconfig) throwsServletException {
        System.out.println("HelloFilter.init()");
        //获取filter的名字
```

```
        String filterName=config.getFilterName();
        System.out.println(filterName);
        //获取初始化参数
        String value=config.getInitParameter("school");
        System.out.println(value);
    }
    @Override
    publicvoiddoFilter(ServletRequestrequest, ServletResponseresponse,
FilterChainchain)
            throwsIOException, ServletException {
        System.out.println("HelloFilter.doFilter() start");
        //进去拦截，还得回来
        chain.doFilter(request, response);//放行
        System.out.println("HelloFilter.doFilter() end");
    }
    @Override
    publicvoiddestroy() {
        // TODO Auto-generated method stub
        System.out.println("HelloFilter.destroy(filter 驾崩了)");
    }
}
```

查看 filter 对象的创建时机，及销毁时机

doFilter 中需要放行，FilterChain.doFilter(request, response);

放行到下一个 filter （具有相应的拦截功能）没有 filter 了则放行到请求的目的地，过滤器链

5、过滤器链实例

过滤器可以被串联在一起，形成管道效应，协同修改请求和响应对象。多个过滤器同时生效，组成链式结构,底层为栈 123-->321

原则：大过滤在前，小过滤在后(权限、字符集等全局性过滤置前，特殊过滤放后)

顺序：拦截顺序有要求

参考 filter-mapping 在 web.xml 配置顺序

FirstFilter.java、SecondFilter.java、ThirdFilter.java

注意 web.xml 中的配置顺序，与执行代码时的输出顺序

6、过滤器绑定具体 servlet

BindServlet.java

```
publicclassBindServletextendsHttpServlet{
    @Override
    protectedvoidservice(HttpServletRequestrequest,
HttpServletResponseresponse) throwsServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriterout=response.getWriter();
        out.println("success");
        out.flush();
        out.close();
    }
}
```

BindServletFilter.java

```
publicclassBindServletFilterimplementsFilter{
    @Override
    publicvoidinit(FilterConfigfilterConfig) throwsServletException {
        // TODO Auto-generated method stub
    }
    @Override
    publicvoiddoFilter(ServletRequestrequest, ServletResponseresponse,
FilterChainchain)
            throwsIOException, ServletException {
        // TODO Auto-generated method stub
        System.out.println("BindServletFilter------------->");
        chain.doFilter(request, response);
        System.out.println("BindServletFilter------------->");
    }
    @Override
    publicvoiddestroy() {
        // TODO Auto-generated method stub
    }
}
```

web.xml

```
<!--过滤器 -->
<filter>
```

```
    <filter-name>BindServletFilter</filter-name>
<filter-class>com.shsxt.servlet.filter.BindServletFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>BindServletFilter</filter-name>
    <url-pattern>/test</url-pattern>
</filter-mapping>
<!--监听器 -->
<!--servlet -->
<servlet>
<servlet-name>BindServlet</servlet-name>
<servlet-class>com.shsxt.servlet.filter.web.BindServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>BindServlet</servlet-name>
    <url-pattern>/test</url-pattern>
</servlet-mapping>
```

过滤器配置的另一种方式

```
<filter>
    <filter-name>BindServletFilter</filter-name>
<filter-class>com.shsxt.servlet.filter.BindServletFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>BindServletFilter</filter-name>
    <!--<url-pattern>/test</url-pattern>-->
    <!—填写servlet的name-->
    <servlet-name>BindServlet</servlet-name>
   <dispatcher><!--设置过滤器过滤什么 -->
    <!-- REQUEST  (默认)过滤请求 -->
    <!-- FORWARD       过滤转发 -->
    <!-- INCLUDE       过滤包含 -->
    <!-- ERROR 过滤报错页面 -->
    </dispatcher>
</filter-mapping>
```

url-pattern 可以用 servlet-name 和 dispatcher 替代

7、使用

①字符集处理

POST

```
request.setCharacterEncoding("utf-8");
```

GET

```
String uname=request.getParameter("uname");
uname=new String(uname.getBytes("ISO-8859-1"),"utf-8");
request.setAttribute("uname", uname);
```

注意：在 Tomcat8 以后 GET 提交方式不会出现乱码。

Tomcat7 及以下的版本：上述解决 GET 乱码的方式不好，不能解决所有的问题，可以借助 HttpServletRequestWrapper 类和过滤器来完成转换乱码的问题：

①在过滤器中判断哪些需要解决

②定义 HttpServletRequestWrapper 的子类，并且重写 getParameter(String name)方法，在该方法中实现乱码的处理

③chain.doFilter(自定义类(HttpServletRequestWrapper 的子类),response)

8、案例

案例：分 IP 统计访问次数

一个 IP 对应一个次数

①创建一个 Map 用来存 IP 和访问次数

②把这个 Map 存到 servletContext 中，在过滤器当中放*.jsp

③在 doFilter 方法中取出 map

获得访问的 IP 地址：判断当前访问的 IP 在 map 中是否存在，如果不存在则把 IP 和 1 存入 map；如果存在，就把 IP 和次数+1 放入 map 中