**Servlet总结**

**Part 1     Servlet运行原理**

web服务器收到一个http请求后，判断请求内容，若是静态页面数据，自行处理，若为动态数据，交给Servlet容器，Servlet容器找到相应Servlet实例处理；处理结果交给web服务器，再转交给客户端。

针对同一个Servlet，Servlet容器会在第一次收到HTTP请求时建立一个Servlet实例，然后启动一个线程，第二次收到http请求后，Servlet容器无需创建相同Servlet ,仅开启第二个线程来处理请求。

----->多线程的方式有效提高执行效率，降低服务器负担。

**Part 2 Servlet 优势**

   Servlet具有优良的跨平台性；

     可移植性良好：java语言编写，Servlet API标准完善，企业编写的Servlet程序可轻松移植到其他服务器中;

     执行效率高：Servlet请求到来时激活Servlet，处理完成后等待新请求；新请求产生新线程，而不是进程；

     使用方便：可轻松处理HTML表单数据，并读取和设置HTTP头，处理cookie,跟踪会话；

**Part 3 基础知识**

     HttpServlet作为一个抽象类来创建用户自己的Http Servlet. Http Servlet类扩展了GenericServlet类。HttpServlet类的子类必须至少重写doGet()和doPost()方法其中之一。HttpServlet类提供doGet()方法处理GET请求，doPost()处理POST请求：

      doGet() ：通过GenericServlet 类的service()方法来调用此方法；

      doPost():  通过GenericServlet 类的service()方法来调用此方法；

**Part 4 Servlet生命周期**

      Servlet生命周期由Servlet容器控制，该容器创建Servlet的实例。Servlet的生命周期是指Servlet实例在创建后，响应客户端请求直至销毁的全过程；

      其创建取决于Servlet的首次调用。

       Servlet生命周期定义了它如何被加载，初始化，以及它如何接受请求，响应请求，提供服务；

       代码中，Servlet生命周期由接口javax.servlet.Servlet定义，所有的Servlet必须直接或间接的实现该接口，这样才能在Servlet容器中运行。

       Servlet接口定义了生命周期三种方法：

       1    init():创建实例后进行初始化。实现ServletConfig接口的对象作为参数进行传递。在初始化过程中，Servlet容器使用ServletConfig接口信息(如Servlet的初始化参数的名称，初始化参数的值，以及Servlet实例名称等)传递给Servlet。

public void init(ServletConfig config)throws ServletException

       2    service():响应客户端发出的请求。service()方法接受ServletRequest接口和Servletresponse接口的对象来处理请求和发送响应；

public void service(ServleRequest request,ServletResponse response) throws ServletException,IOException

       3    destroy():如果不再有需要请求的对象，则释放Servlet对象；

public void destroy()

Servlet生命周期各阶段：

1.实例化：Servlet容器创建Servlet类的实例对象；

2.初始化：容器调用Servlet的init()方法，通常会申请资源以便后续使用;

3.服务：由容器使用以响应客户对Servlet的请求;

4.破坏:在释放Servlet实例前调用，通常会释放资源;

5.不可用:释放内存中的容器；

[要实例化一个Servlet，容器必须先找到Servlet类，加载servlet类并创建Servlet对象，然后通过调用Servlet的init()方法来初始化Servlet。ServletConfig接口对象作为参数传给init()方法，该接口对象为Servlet提供对ServletContext接口的访问。Servlet容器使用ServletContext接口与Servlet容器进行通信；

如果初始化Servlet失败,则抛出UnavailableException或ServletException异常,并再次尝试对Servlet进行实例化和初始化。然后将ServletRequest和ServletResponse接口作为参数传递给service方法，该方法将处理请求并返回响应。

如果响应请求时发生异常，则容器通过调用destroy()方法卸载实例]

**Part 5    Servlet API**

 在javax.servlet和javax.servlet.http包中的各类和接口如下：

（1）ServletInputStream类

    从java.io.InputStream类扩展而来的抽象类，该类创建的对象用于读取客户端请求中的二进制数据，而该类的readLine()方法用于每次读取一行数据；

**该方法将从给定偏移处开始的每字节读取到数组中，直到该方法遇到换行符或者读取完一定量的字节数量，该方法返回一个整数来指定实际读取的字节数，到达流尾时返回-1**

public int readLine(byte b[],int offset,int length)throws java.io.IOException  
//b为用于存储读取的数据的字节数组；offset指定方法开始读取字符的起始位置；length读取的最大字节数

 （2）ServletOutputStream类

     该类创建的对象用于将二进制数据从服务器发送到客户端。具体实现的方法如下：

       +   print():将字符串写入客户端。如果发生任何输入或输出异常，则方法print()会引发IOException异常，print()方法接受参数,如char,float,double,int,long,String。

public void print(String str)throws java.io.IOException  
//str为发送到客户端的字符串

       +     println()：将字符串写入客户端，紧跟后面输出回车。如果发生任何I/O异常，则会引发IOException

 （3）ServletRequest接口

     使用ServletRequest接口创建对象，用于使客户端请求信息对Servlet可用。创建的对象作为参数传递至Servlet的service()。

     该类实现方法如下：

       +     getInputStream():返回客户端请求中的二进制数据，并将其保存在getInputStream对象中

public ServletInputStream getInputStream()throws IOException

       +     getParameter():用于获取请求消息一起发送的附加信息-----请求参数

public String getParameter(String str)

       +     getContentLength():返回客户端发送的请求的实际长度，如果长度未知，则返回-1

public int getContentLength()

       +     getServerName():返回请求发至的服务器名称

public String getServerName()

   （4）ServletResponse接口

       使用该接口创建的对象用于向客户端提供响应。创建的对象作为参数传递至Servlet的service()方法中。该接口实现的方法如下：

       +     getOutpouStream():返回一个ServletOutputStream对象，它被用来发送对客户端的响应

public ServletOutputStream getOutputStream()throws IOException

       +     getWriter():返回将字符文本发送到客户端的PrintWriter对象

public PrintWriter getWriter()throws IOException

       +      setContentLength():允许用户设置将作为响应放的数据的长度

public void setContentLength(int length)

       +      getBufferSize():检索实际的以响应客户端的缓存区大小。若没有使用缓冲区则返回0

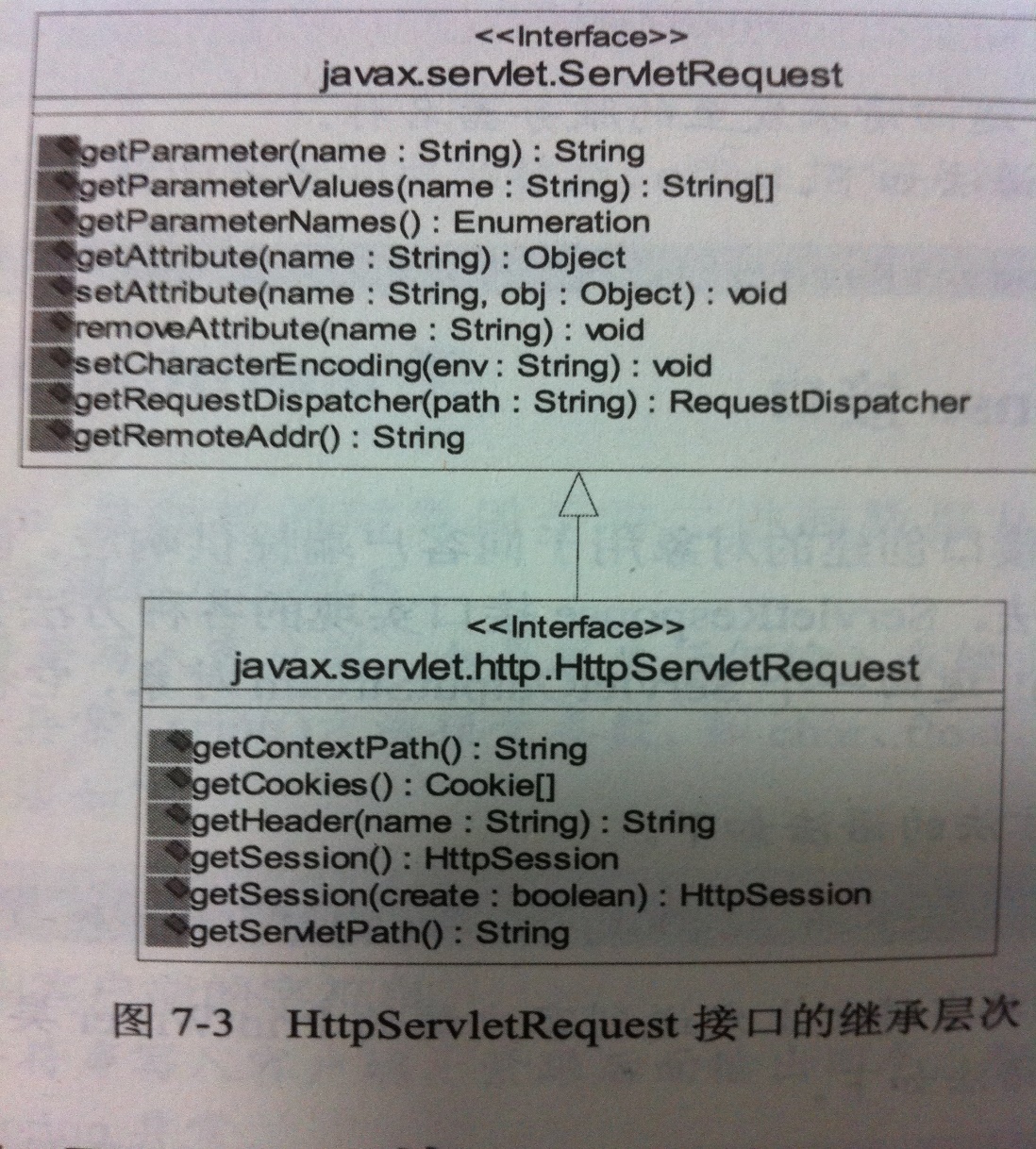
public int getBufferSize()

       +      setBufferSize():设置将发送到客户端的数据的缓冲区的大小

public void setBufferSize(int size)

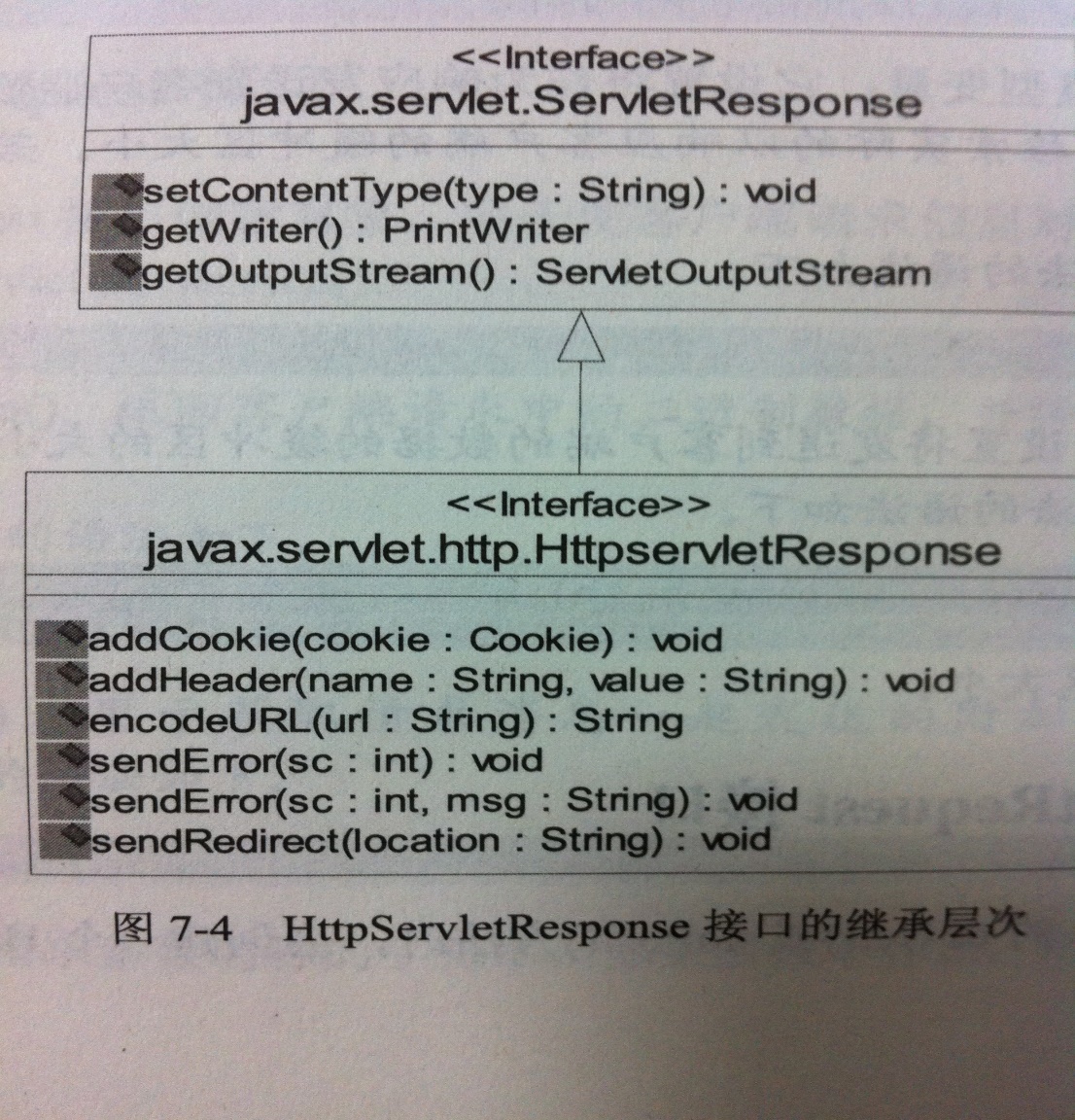
    （5）HttpServletRequest接口

         容器在调用Servlet的doGet()或doPost()方法时，会创建一个HttpServletRequest接口的实例和一个HttpServletResponse接口的实例，作为参数传递给doGet和doPost()方法。该接口代表客户请求，它提供了多种获取请求数据的方法，具体继承层次如图：



    （6）HttpServletResponse接口

         该接口代表返回给客户端的响应；具体继承层次如图:



     （7）ServletConfig接口

           在初始化过程中，Servlet容器使用ServletConfig接口的对象作为参数来传递Servlet的配置信息。方法如下：

              +    getServletName():用于获取Servlet实例名称

public String getServletName()

              +     getInitParameter():检索初始化参数的值，如果参数不存在，则getInitParameter()方法返回null

public String getInitParameter(String name)  
//name为初始化参数的名称字符串

              +     getServletContext():返回Servlet用来与其容器交互的ServletContext对象

public ServletContext getServletContext()

       （8）ServletContext接口

             该接口定义了一组方法，Servlet使用这些方法与容器进行交互并获取信息（如读写文件等）

               +     getContext():返回允许Servlet访问服务器上下文的ServletContext类对象

public ServletContext getServletContext(String uripath)

//uripath是Web容器上的另外一个Web程序的上下文路径名称字符串

              +     getMimeType():返回文件的MIME类型。MIME定义了一种协议，允许用户通过Internet交换非ASCII消息。不同的MIME类型分为"text/html"和"image/gif"

public String getMimeType(String file)  
//file是文件名称

              +     getResource():返回与路径名相对应的资源的URL

public java.net.URL getResource(String path) throws MalFormedURLexception  
//path是资源对应的路径名称字符串

         （9）获取请求中的数据

                在Servlet类的请求处理方法中（如doGet()，doPost()方法），要想获得客户端请求中提交的数据，需要使用HttpServletRequest提供的以下方法：

public String getParameter(String name)

//获取指定名称的参数值

public String[] getParameterValues(String name)

//获取指定名称参数的所有值数组。它适用于一个参数名对应多个值的情况，如页面表单中的复选框，多选列表提交的值

public java.util.Enumeration getParameterNames()

//返回一个包含请求消息中的所有参数名的Enumeration对象。通过遍历这个Enumeration对象，就可以获取请求消息中所有的参数名

public java.util.Map getParameterMap()

//返回一个保留了请求消息中所有参数名和值的Map对象。Map对象的key是字符串类型的参数名，value是这个参数所对应的Object类型的值数组。

        [提示：在此说明如何处理客户端提交给服务器的数据的乱码问题

                  若客户端以POST方式提交请求，请求消息主体中的参数数据是按HTML页面中指定的编码方式进行编码的，在Servlet类的请求处理方法中需要先调用         HttpServletRequest接口的setCharacterEncoding(String enc)方法对请求消息主体中的数据按参数指定的编码方式进行编码，然后才能使用上述介绍的方法正 确获得参数值

                  若客户端使用GET方法请求，上述方法无效，此时，最好的解决方案是在URL中不使用中文等非ASCII字符]

        （9）重定向和请求分配

              1.重定向：HttpServletRequest接口提供的sendRedirect()方法用于生成302响应码和Location响应头从而通知客户端去重新访问Location响应头中指定的URL

public void sendRedirect(String location)throws IOException  
 //其中location参数指定了重定向的URL，它可以使用绝对或相对URL，Servlet会进行转化

             2.请求分派：RequestDispatcher接口，（分派器），定义了下面两个方法

public void forward(ServletRequest request,ServletResponse response)throws ServletException,IOException;

//forward()用于将请求转发到RequestDispatcher实例封装的资源

public void include(ServletRequest request,ServletResponse response)throws ServletException,IOException;

//include()用于将RequestDispatcher实例封装的资源作为当前响应内容的一部分包含进来

RequestDispatcher dispatcher=request.getRequestDispatcher("servlet1");

//参数为当前servlet名称

dispatcher.forward(request,response);

     [重定向与请求分派的区别：

             请求分派只能将请求转发到同Web应用中的其他组件；而重定向也可以发送到其他Web；

             请求分派过程结束后，浏览器内网址不改变；重定向完成后浏览器内原网址变为重定向目标网址；

             ]

    （11）利用请求域属性传递对象数据

          HttpServletRequest接口中提供了几个方法来操作请求实例中的存储对象

public void setAttribute(String name,Object obj)

//将对象存储进HttpServletRequest实例中

public Object getAttribute(String name)

//检索存储在HttpServletRequest实例中的对象

public Enumeration getAttributeNames()

//返回HttpServletRequest实例中所有属性名的Enumeration对象

public void removeAtribute(String name)

//从指定HttpServletRequest实例中删除指定名称的属性

        这种存储在HttpServletRequest中的对象称之为请求域属性，属于同一请求过程的多个处理模块之间可以通过请求域属性来传递对象数据；

//Servlet1.java

package test

import java.io.IOException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.\*;

public class Servlet1 extends HttpServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException{

this.doPost(request,response);

}

public void doPost(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException{

String str="Job done";

request.setAttribute("string",str);

RequestDispatcher dispatcher=request.getRequestDispatcher("Servlet2");

dispatcher.forward(request,response);

}

}

//Servlet2.java

package test

import java.io.IOException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.\*;

public class Servlet2 extends HttpServlet{

public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException{

this.doPost(request,response);

}

public void doPost(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException{

response.setContentType("text/html;charset=utf-8");

PrintWriter out=response.getWriter();

out.println("<html>");

out.println("<head><title>test</title><head>");

out.println("<body>");

//获取名为"string"的请求域属性的值

String str=(String)request.getAttribute("string");

out.println(str);

out.println("</body>");

out.println("</html>");

}

}

**Part 6   Servlet的线程安全问题**

**Servlet默认是多线程模式执行的，当有多个用户同时并发请求一个Servlet时，容器将启动多个线程调用相应的请求处理方法，此时，请求处理方法中的局部变量是安全的，但对于成员变量和共享数据是不安全的，因此这多个线程有可能同时都操作这些数据，这是需要同步处理。所以，编写代码时需要非常细致的考虑多线程的安全性问题。多数人编写Servlet时不注意多线程问题，导致少量用户访问时没有问题，但并发量大则出现莫名其妙的问题。**

**解决：**

**1.使用synchronized：使用synchonized关键字同步操作成员变量和共享数据的代码，就可以防止出现线程安全性问题，但这也意味着线程需要排队处理。因此，在使用同步语句时要尽可能缩小同步代码范围，不能直接在请求处理方法（如doGet(),doPost()方法）使用同步，这样会严重影响效率。**

**2.尽量少使用成员变量和共享数据：对于集合，使用Vector代替非线程安全的ArrayList，使用Hashtable代替HashMap;不能在Servlet内创建自己的线程，导致复杂化。**

**Part 7   Servlet  过滤器**

过滤器技术（Filter）是Servlet2.3以上版本新增的功能，2.5对其进一步增强。

Filter是一个程序，它先于相关的Servlet或JSP页面运行在服务器上。过滤器可附加到多个或一个Servlet或JSP上，并且可以检查进入这些资源的请求消息。

Filter在Request到达Servlet之前预处理，也可以在离开Servlet时处理Response；其实本质就是一个Servlet Chaining.

 一个Filter必须实现javax.servlet.Filter接口并定义三个方法：

public void init(FilterConfig config)

//Filter实例化后进行初始化的回调方法

public void doFilter(ServletRequest req,ServletResponse res,FilterChain chain)

//处理过滤业务的方法

public void destroy()

//Filter释放时回调的方法

每一个Filter从doFilter()方法中得到当前的request和response。在该方法内，可进行任何针对request和response的操作。Filter调用chain.doFilter()方法把控制权交给下一个Filter。

实例：Filter处理中文乱码问题：

import java.io.IOException;

import javax.servlet.\*;

public class CharacterEncodingFilter implements Filter{

private FilterConfig config;

public void destroy(){

}

public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)throws IOException,ServletException{

String encoding=config.getInitParameter("encoding");

if(null!=encoding&&!"".equals(encoding)){

request.setCharacterEncoding(encoding);

}

chain.doFilter(request,response);

//将修改过的请求和响应传递给下一个过滤器或者Servlet

}

//Filter初始化时的回调方法

//FilterConfig接口实例中封装了这个Filter的初始化参数

public void init(FilterConfig config){

this.config=config;

}

}

在web.xml中注册

......

<filter>

<filter-name>characterEncodingFilter</filter-name>

<filter-class>(路径).characterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

.........

此处/\*是过滤所有页面请求，如果需要，可以指定页面 或文件夹下的所有页面。

**过滤链**：为web添加按顺序执行的多个Filter，进入时执行顺序A-B，离开时执行顺序B-A

排列方法是在web.xml中，按顺序注册即可；

如下示例：（在上一示例基础上添加一个可以自动压缩响应输出流的Filter，提高传输效率）

......

<filter>

<filter-name>characterEncodingFilter</filter-name>

<filter-class>(路径).characterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>Compression Filter</filter-name>

<filter-class>compressionFilters.CompressionFilter</filter-class>

<!--设置缓冲区大小-->

<init-param>

<param-name>compressionThreshold</param-name>

<param-value>512</param-value>

</init-param>

<!--调试级别-->

<init-param>

<param-name>debug</param-name>

<param-value>0</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>Compression Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

.........

即可实现Filter Chain。

**Part 8  Servlet 监听器**

**监听器可使应用对某些事件做出反应:**

**Servlet2.3以上版本提供了一下几个接口：**

**+   ServletContextListener:应用上下文生命周期监听器，用于监听Web应用的启动和销毁事件;**

**+   ServletContextAttributeListener:应用上下文属性事件监听器，用于监听Web应用上下文中的属性改变事件;**

**+   ServletRequestListener:请求生命周期监听器，监听请求的创建和销毁;**

**+   ServletRequestAttributeListener:请求属性事件监听器，用于监听请求中的属性改变事件;**

**+   HttpSessionListener:会话生命周期监听器，用于监听会话的创建和销毁;**

**+   HttpSessionActivationListener:会话激活和钝化事件监听器;**

**+   HttpSessionAtributeListener:会话属性事件监听器;**

**+   HttpSessionBindingListener:会话值绑定事件监听器;**