**过滤器**

# 授课内容

授课科目：JavaEE

授课内容：过滤器

授课类型：讲授+实践

授课时间：3+1学时

主讲教师：解卫静

# 教学目标要求

## 能力目标：

1. 培养学生掌握编写、配置过滤器的技能，并能够使用过滤器解决实际问题；

## 知识目标：

1. 培养学生了解过滤器的作用及应用场景；
2. 培养学生掌握过滤器的编写、使用过程和配置方法。

# 课件分析

概 述：首先阐述什么是过滤器及过滤器的功能、用途，介绍过滤器相关的接口和类，重点介绍如何编写、配置过滤器，要求学生了解过滤器链的执行过程。

教学重点：过滤器的编写和配置。

教学难点：过滤器链的执行过程。

# 教学方法

讲授法、案例法、实践教学法、类比法等。

# 教学过程

## 课前补充

（0分钟）

## 上节回顾

（2分钟）

上一章介绍了监听器的作用、监听接口、事件，及编写监听器的步骤，重点介绍了与ServletContext、HttpSession、ServletRequest相关监听器的使用原理和编程实现。监听器其实就是一个实现特定接口的Java类，这个类专门用于监听另一个Java类的方法调用或属性改变。开发一个监听器，首先要编写它的实现类，这个类必须实现相应功能的监听接口，然后在web.xml中进行部署。

## 作业点评

（0分钟）

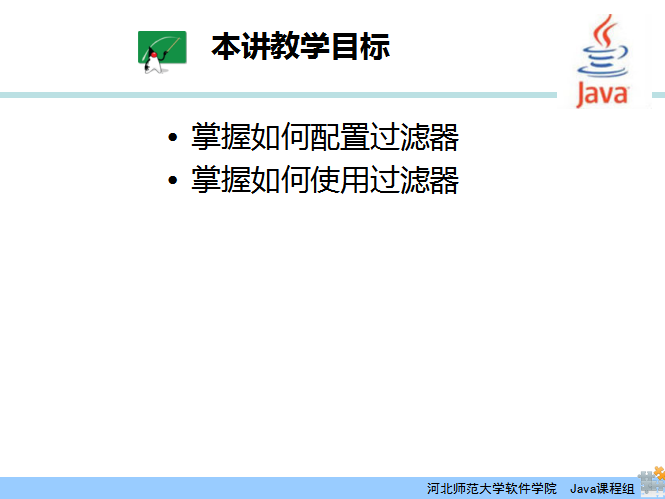
## 导入新课

（2分钟）

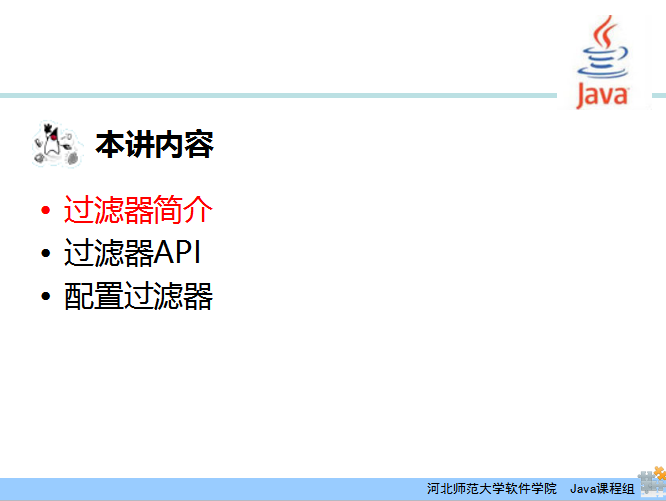
在生活中，访问一个视频网站时，对于同一个请求地址，登录过的用户可以直接观看，而未登录的用户则跳转到登录页面；访问一个页面时，当该页面无法加载或不存在时会返回一个错误页，这些功能的实现都会用到本章的内容——过滤器。本章将介绍什么是过滤器、过滤器的使用场景，及过滤器相关接口和类，重点介绍如何编写、配置过滤器，要求学生理解过滤器链的执行过程。

## 讲授新课

（120分钟）



本章的教学目标是了解过滤器的作用、应用场景，及过滤器链的执行过程，重点掌握如何配置、使用过滤器。

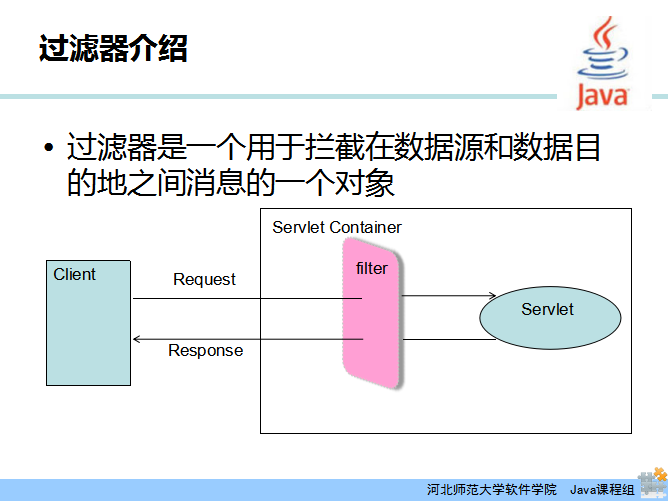


首先对过滤器进行简单介绍。

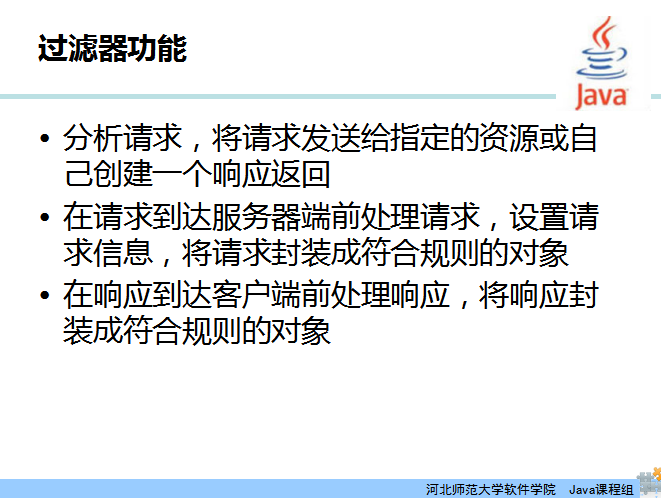


生活中，如果一个人对公司前台说，“来两张动物园门票”，她会回绝，因为公司不卖动物园门票；另一个人过来说“我是小丸子，来你们公司面试的”，前台则通知经理面试的人到了，经理回复前台把人安排在会议室，那前台就会把经理的意思传达给小丸子，让他在会议室等候。

这里前台的作用就是处理外来的访问，拒绝不合理的访问，把合理的访问向经理传达，再把经理的回复传给访问者。这个前台，就类似于本章要讲的过滤器。



过滤器，也称Filter，是一个服务器端的组件。用户在访问某个目标资源时，过滤器可以对访问的请求和响应进行拦截处理，从而实现一些特殊的功能，例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等。Servlet API中提供了一个接口——Filter，开发Web应用时，如果编写的Java类实现了此接口，则把这个Java类称为过滤器。过滤器的实现过程如图中所示，当客户端浏览器请求服务器资源时，先要经过过滤器的拦截处理，再将过滤后的请求转发到下一资源，而服务器返回响应时，也是先经过过滤器，再响应给客户端浏览器。



过滤器拦截到请求后，会根据具体分析结果将请求发送给指定的资源或自己创建一个响应返回。比如访问一个视频网站时，对于同一个请求网址，登录过的用户可以直接观看，而未登录的用户则跳转到登录页面；访问一个页面时，如果该页面无法加载或不存在，可以通过过滤器返回一个错误页。

过滤器还可以在请求到达目的地前处理请求，设置请求头信息，将请求封装成符合规则的对象。比如，可以在过滤器中设置字符编码方式，从而统一设置提交到某一个地址的数据格式。

过滤器拦截到服务器传给客户端浏览器的响应后，也可以处理响应，将其封装成符合规则的对象。



根据应用场景不同，过滤器有多种功能。例如以下几种。

1.身份验证过滤器，用来验证身份，如未登录的用户请求资源时，身份验证未通过会跳转到登录页面。

2.数据压缩过滤器，用来压缩数据。

3.加密过滤器，用来加密数据。

4.触发资源访问事件过滤器，当用户访问某个资源时，用来触发某个事件。

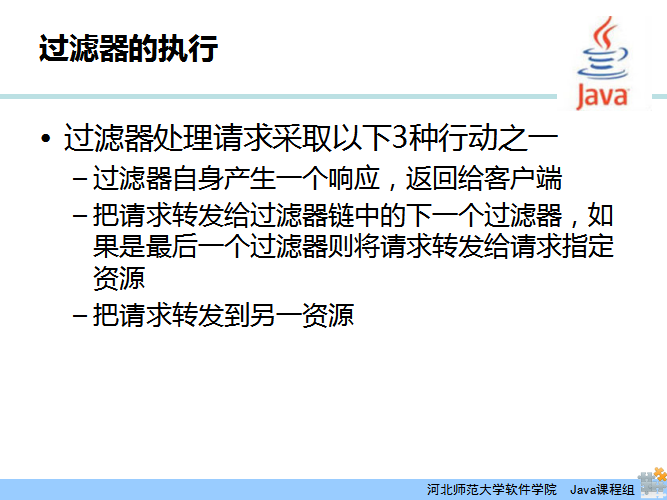
5.图像转换过滤器，用来转换图像格式。

6.日志记录和审核过滤器，用来记录并在控制台中输出运行时的日志信息。

7.MIME-TYPE链过滤器，用来过滤内容类型。

8.标记化过滤器，用于对分词器切分的最小单位进入索引前进行预处理，如：大写转小写，复数转单数等。

9.XSL/T过滤器，用来转换XML内容。



以上过滤器处理请求的基本行为都是一样的，主要采取以下三种行为之一。

1.过滤器拦截请求后，自身产生一个响应，返回给客户端。

2.把请求转发给过滤器链中的下一个过滤器，如果当前过滤器最后一个过滤器，则将请求转发给请求指定资源。

3.把请求转发到另一资源。



过滤器开发分为两个步骤：

1.编写Java类实现Filter接口，并实现接口中的doFilter()方法。

2.在web.xml文件中使用<filter>和<filter-mapping>元素对编写的filter类过滤器进行注册，并设置它所能拦截的资源或资源路径。

当我们编写好一个过滤器，并配置对哪个Web资源进行拦截后，Web服务器每次在调用Web资源的service()方法之前，都会先调用过滤器的doFilter()方法。在该方法内编写代码可达到如下目的：

1.调用目标资源之前，让一段代码执行，即处理请求；

2.是否调用目标资源，即判定请求是否通过过滤，是否允许用户访问Web资源；

3.调用目标资源之后，让一段代码执行，即处理响应内容。

Web服务器在调用doFilter()方法时，会传递一个FilterChain对象进来，FilterChain对象是Filter接口中最重要的一个对象，它也提供了一个doFilter()方法，开发人员可以根据需求决定是否调用此方法，调用该方法，则Web服务器就会调用Web资源的service()方法，即Web资源就会被访问，否则Web资源不会被访问。



示例：编写一个EncodeFilter类，实现Filter接口，实现所有请求编码方式的统一。

1.编写过滤器，部分代码如下：

|  |
| --- |
| public class EncodeFilter implements Filter {  private String DEFAULT\_ENCODING = "UTF-8";  private String encoding;  ...  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,  ServletException {  request.setCharacterEncoding(encoding);  response.setCharacterEncoding(encoding);  chain.doFilter(request, response);  }  public void init(FilterConfig fConfig) throws ServletException {  String enc = fConfig.getInitParameter("encoding");  if (enc == null){  this.encoding = DEFAULT\_ENCODING;  }else{  this.encoding = enc;  }  }  } |

1. 过滤器的部署分为两个步骤：注册过滤器和映射过滤器。

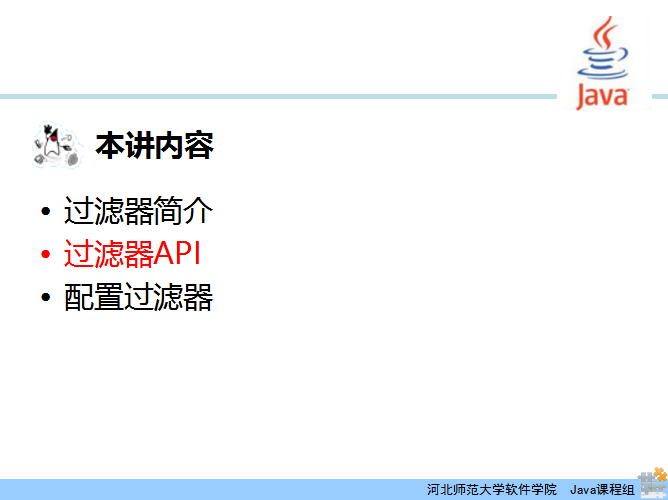
a.开发好过滤器之后，需要在web.xml文件中进行注册，这样才能够被Web服务器调用。

|  |
| --- |
| <filter>  <filter-name>EncodeFilter</filter-name>  <filter-class>onest.filter.EncodeFilter</filter-class>  </filter> |

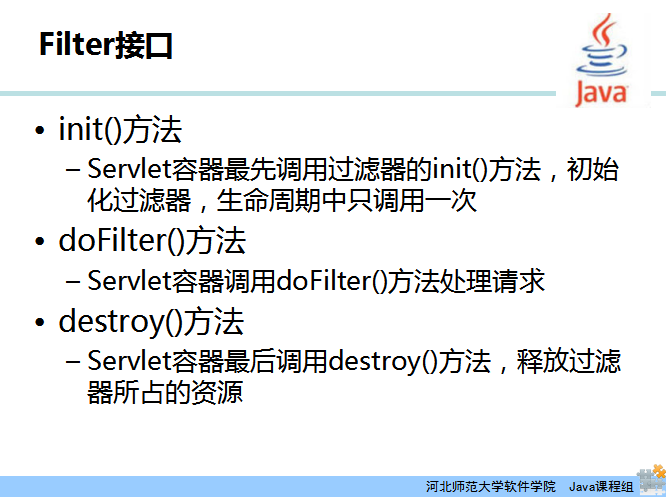
b.在web.xml文件中注册了过滤器之后，还要在web.xml文件中映射过滤器。

|  |
| --- |
| <filter-mapping>  <filter-name>EncodeFilter</filter-name>  <!-- 拦截所有的请求 -->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

经过以上两步，站点所有请求和响应都会经过此过滤器进行编码方式的转换。



下面详细介绍过滤器API。



所有过滤器都必须实现Filter接口。Filter接口中的init()、doFilter()、destroy()三个方法也代表了过滤器生命周期的三个阶段。过滤器的创建和销毁由Web容器负责。Web应用程序启动时，Web容器将创建过滤器的实例对象，并调用其init()方法，完成对象的初始化功能，从而为后续的用户请求做好拦截的准备工作，Web容器在卸载Web应用程序前回调destroy()方法，销毁过滤器。过滤器对象只创建一次，init()方法和destroy()方法也只执行一次。在destroy()方法中，可以释放过滤器使用的资源。

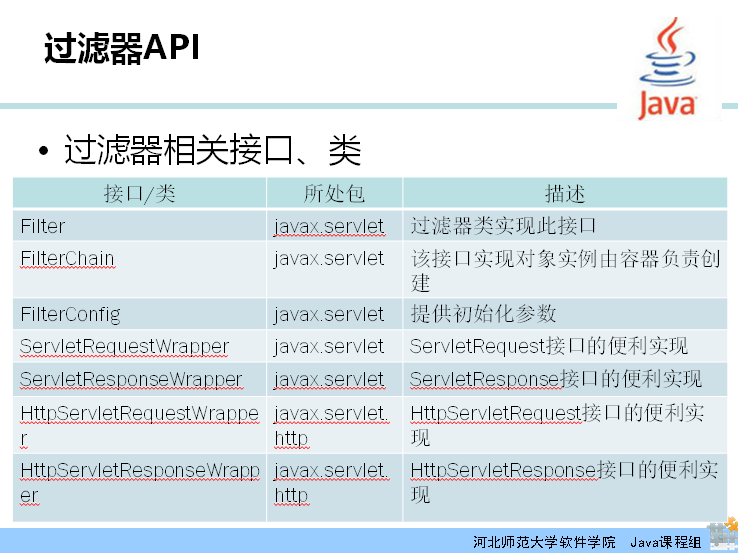
总结过滤器的生命周期，一共四个阶段：

1.实例化：Web容器在部署Web应用程序时对所有过滤器进行实例化；

2.初始化：Web容器回调init()方法；

3.过滤：当请求路径匹配过滤器的URL映射，Web容器回调doFilter()方法；

4.Web容器在卸载Web应用程序前回调destroy()方法，回收资源。



Filter接口的init()方法接收FilterConfig对象作为参数，FilterConfig对象代表当前过滤器的配置信息，用来提供初始化参数。

Filter接口的doFilter()方法除了接收Request、Response两个参数外，还接收一个参数FilterChain，FilterChain接口中只有一个方法doFilter(),与Filter接口中doFilter()不同的是，调用FilterChain中的doFilter()方法，表示被拦截的对象已经通过此过滤器的过滤，可以执行下一步了。

ServletRequestWrapper类实现了ServletRequest接口，用于更便利地使用ServletRequest对象。

ServletResponseWrapper类实现了ServletResponse接口，用于更便利地使用ServletResponse对象。

在过滤器中，虽然可以使用HttpServletRequest的getParameter()取得请求参数值，但是没有一个像setParameter()的方法，可以将处理后的请求参数重新设置给HttpServlet。于是产生了请求封装器HttpServletRequestWrapper，它继承了ServletRequestWrapper类，并且实现了HttpServletRequest接口，可以让getParameter()返回过滤后的请求参数值。下面通过一个示例来演示请求封装器的使用。

示例：有些用户会在留言中输入HTML的超链接标签，利用留言功能打广告，现要使用过滤器过滤用户请求参数中的HTML来杜绝这种现象。

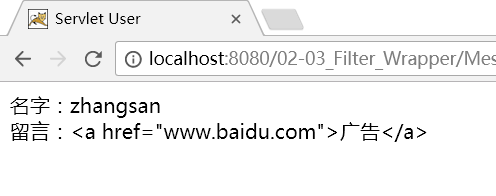
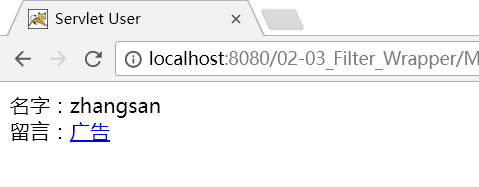
1.编写一个EscapeWrapper类，继承HttpServletRequestWrapper类，重写getParameter()方法。

|  |
| --- |
| @Override  public String getParameter(String name){  String value = getRequest().getParameter(name);  return StringEscapeUtils.escapeHtml(value);  } |

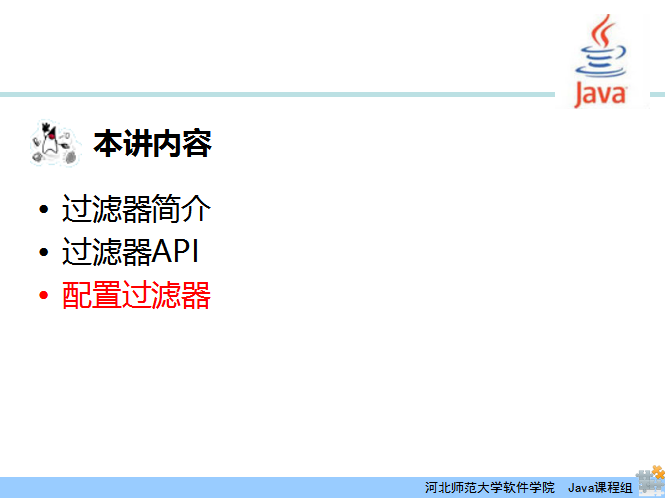
2.编写一个过滤器EscapeFilter，使用EscapeWrapper类转换请求。

|  |
| --- |
| public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  HttpServletRequest requestWrapper = new EscapeWrapper((HttpServletRequest)request);  chain.doFilter(requestWrapper, response);  } |

转换前的运行结果，与转换后的结果对比，如下图所示。



HttpServletResponseWrapper类继承了ServletResponseWrapper类，并且实现了HttpServletResponse接口，用于更便利地使用HttpServletResponse对象。因为要对浏览器进行输出响应，必须通过getWriter()取得PrintWriter，或是通过getOutputStream()取得ServletOutputStream，所以如果要对响应的内容进行压缩处理，就可以通过继承HttpServletResponseWrapper类，重写这两个方法来对HttpServletResponse对象进行封装。



使用过滤器API完成过滤器的编写以后，必须对过滤器进行配置部署。



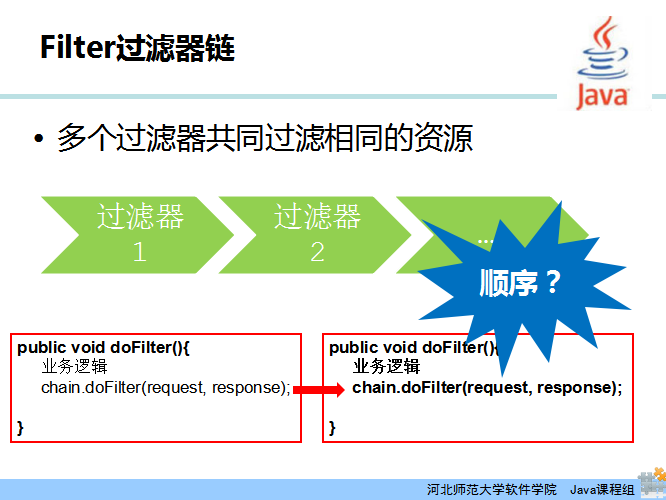
刚才介绍了配置过滤器有两步，第一步是注册过滤器，第二步是映射过滤器。

注册过滤器时的一些标签，<filter-name>表示过滤器的逻辑名称，<filter-class>表示过滤器的实现类，<description>描述信息，除此之外，还可以使用<init-param>为Filter配置一些初始化参数，<param-name>指参数名称，<param-value>指参数值。当Web容器实例化过滤器对象，调用其init()方法时，会将封装过滤器初始化参数的FilterConfig对象传递进来，通过FilterConfig对象的getInitParameter()方法可以获取到这些初始化参数信息。

映射过滤器时，<filter-name>必须与过滤器注册时<filter-name>相同，<url-pattern>指拦截的资源。比如”/\*”指当前应用程序根下的所有文件，包括多级子目录下的所有文件；”\*.html”指当前应用程序根目录下的所有HTML文件；“/folder\_name/\*”指当前应用程序根目录下的folder\_name子目录(包括多级子目录)下所有文件。如果要同时过滤多种类型的文件，可以用分号间隔或写多个<filter-mapping>。



此过滤器示例在过滤器注册时，设置了初始化参数编码方式为GB2312。红框中的“CharseFilter”是过滤器的逻辑名称，<url-pattern>中的设置的“/MyServlet”表示拦截根目录下的MyServlet请求。

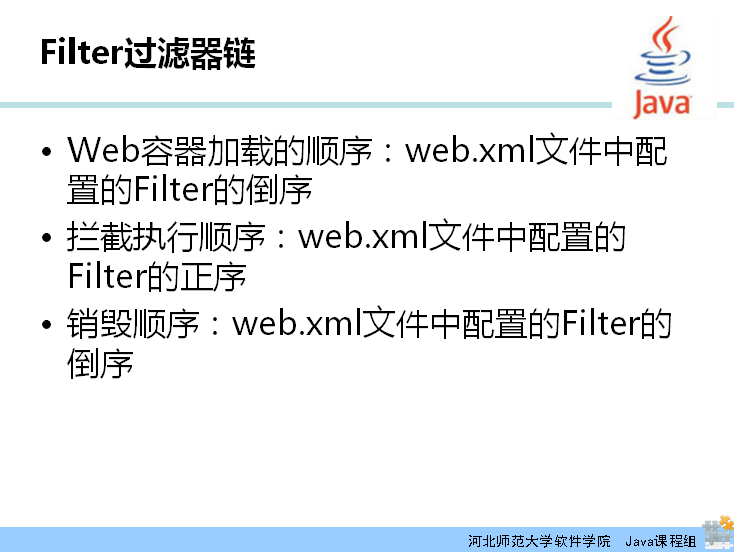


以上介绍的都是只有一个过滤器的情形。当只有一个过滤器时，过滤器拦截请求，执行chain.doFilter()之前的程序，然后执行chain.doFilter()，表示请求通过此过滤器的过滤。chain.doFilter()之后的程序，在服务器作出响应之后、响应的资源到达客户端浏览器之前执行。例如客户端浏览器要请求LoginServlet，运行过程是先执行chain.doFilter()之前的程序，再执行LoginServlet，完成以后，执行过滤器中chain.doFilter()之后的程序。

在实际Web应用中，可以开发多个过滤器，多个过滤器同时过滤一个请求时，组成过滤器链，又称FilterChain。在过滤器链中的程序的执行顺序是怎样的呢？

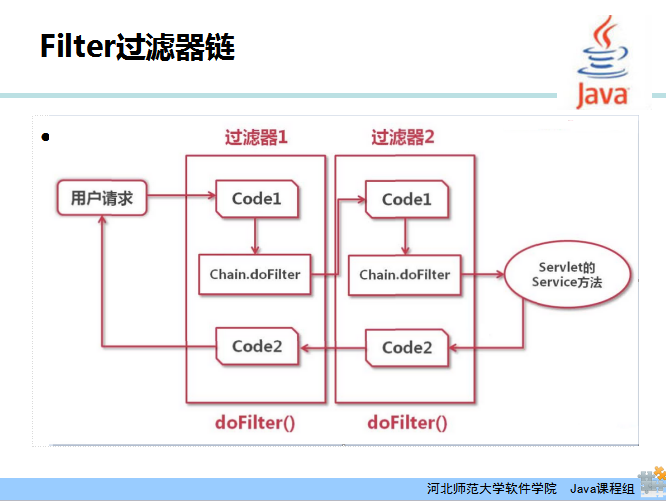


如示例中所示，FilterTest和CharsetFilter两个过滤器都要对”/MyServlet”请求进行过滤，哪个过滤器先执行呢？



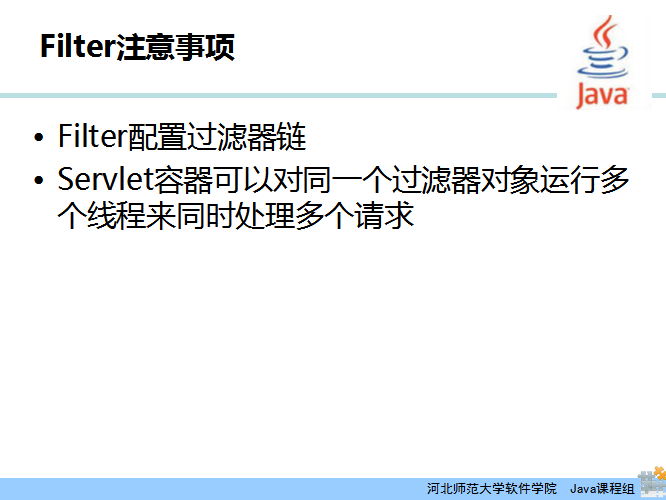
Web容器在部署Web应用程序时对所有过滤器进行加载并实例化，加载的顺序是web.xml文件中配置的过滤器的倒序。Web容器在卸载Web应用程序前回调destroy()方法销毁过滤器，销毁的顺序也是web.xml文件中配置的过滤器的倒序。

但过滤器执行的顺序是web.xml文件中配置的过滤器的正序。也就是Web服务器根据过滤器在web.xml文件中的注册顺序，决定先调用哪个过滤器，当第一个过滤器的doFilter()方法被调用时，Web服务器会创建一个代表过滤器链的FilterChain对象传递给该方法。在doFilter()方法中，开发人员如果调用了FilterChain对象的doFilter()方法，则Web服务器会检查FilterChain对象中是否还有过滤器，如果有，则调用第二个过滤器，如果没有，则调用目标资源。



过滤器链的具体执行流程如下：

客户端浏览器发出请求后，执行第一个过滤器的chain.doFilter()之前的代码—>第二个过滤器的chain.doFilter()之前的代码—>…….—>第n个过滤器的chain.doFilter()之前的代码—>所请求Servlet的service()方法中的代码—>所请求的doGet()或doPost()方法中的代码—>第n个过滤器的chain.doFilter()之后的代码—>……—>第二个过滤器的chain.doFilter()之后的代码—>第一个过滤器的chain.doFilter()之后的代码。



使用过滤器时注意，服务器会按web.xml中定义的过滤器先后顺序，组成一条链来执行doFilter()方法。Servlet容器对每一个过滤器只实例化一个对象，当有多个请求访问时，Servlet容器可以对同一个过滤器对象运行多个线程来同时处理多个请求。

过滤器也可以使用注解来替换web.xml中的配置。使用注解时，通过实践发现如果想要控制Filer的执行顺序可以通过控制Filter的文件名来控制。比如：UserLoginFilter.java 和 ApiLog.java 这两个文件里面分别是“用户登录检查过滤器”和“接口日志过滤器”，因为这两个文件的首字母A排U之前，导致每次执行的时候都是先执行“接口日志过滤器”再执行“用户登录检查过滤器”，所以修改两个文件的名称分别为Filter0\_UserLogin.java和Filter1\_ApiLog.java，这样就能先执行“用户登录检查过滤器”再执行“接口日志过滤器”。

## 课时小结

（10分钟）

1. 过滤器的作用及应用场景。
2. 过滤器API
3. 过滤器的开发和配置
4. 过滤器链的执行过程。

# 布置作业

（5分钟）

1. 完成过滤器的实验手册。
2. 实现登录验证过滤器和编码过滤器。