**过滤器**

**概述**

过滤器(Filter)，用来截获客户端向服务器发送的请求进行过滤操作，或者截获服务器对客户端作出的响应进行过滤操作。如果某些Servlet对象和页面只希望特定的用户访问，很明显这里涉及到访问权限的问题，因此可以通过滤器对Servlet对象或页面的访问权限进行控制。

**过滤器的作用**

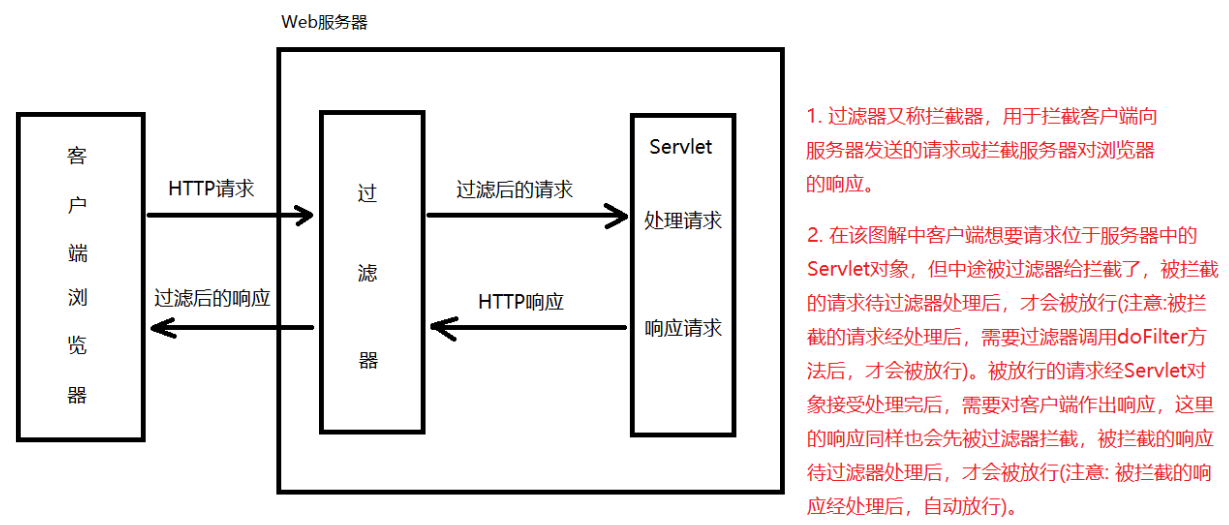
过滤器可以再请求和响应之前做一部分预处理，有效的过滤掉不需要的内容，而且过滤器可以被复用，节省了大量的复用代码，提高了java的代码执行效率

**过滤器的应用场景**

防止未登录就进入界面

控制应用编码

过滤敏感词汇等场景

**过滤器拦截请求和响应的过程图解**

**如何声明一个Filter类**

声明一个类实现javax.servlet包中的Filter接口(注意包名别弄错)，重写接口的doFilter方法即可。我们习惯的把实现Filter接口的类称为Filter类(过滤器类)，该类创建的对象，也就是filter对象(过滤器对象)，简称过滤器。doFilter方法如下所示:

publicvoid doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException; //客户端请求Setvlet对象，Servlet对象在调用service方法响应请求前，如果发现该Servlet对象应用了某个过滤器，Web服务器就会自动调用该过滤器的doFilter方法对该请求执行过滤操作。

**如何部署过滤器**

同Servlet类一样，服务器想要使用Filter类编译后的得到的字节码文件创建filter对象(过滤器)，就必须先在服务器中部署filter。部署的方法同Servlet一样，也有两种，一种是在web.xml中部署filter，一种基于注解的方式部署filter。这里不详述，具体查书P163.

**过滤器通配符**

/\* 表示客户端对任何页面或Servlet对象的请求，都会被指定的过滤器过截获进行过滤。

**过滤器链**

过滤器链实际上就是一个请求被多个过滤器截获进行过滤。至于该请求是先被哪个过滤器截获过滤，取决于这些过滤器部署的先后顺序。过滤器调用doFilter方法对截获的请求进行过滤，过滤完毕一定要在该方法中使用FilterChain对象调用doFilter将截获的请求放行，否则被截获的请求一直无法到底目的地。如果一个请求需要经过多个过滤器过滤，该请求会先被第一个过滤器截获进行过滤，当第一个过滤器过滤完毕将其放行后，又接着被第二个过滤器截获进行过滤，依次类推，直至该请求被所有的过滤器都过滤为止。如过一个请求只需要经过1个过滤器过滤，当该过滤器过滤完毕并放行后，该请求直接抵达目的地。

注1: 过滤器的doFilter方法和FilterChain对象的doFilter方法不是同一个方法，注意不要弄混淆。过滤器的doFilter方法来源于Filter接口，而FilterChain对象的doFilter方法来源于FilterChain接口。

注2: 一个Servlet对象/页面可能应用了多个过滤器，所以客户端对该Servlet对象/页面的请求就会被多个过滤器截获进行过滤。

注3: 如果一个请求需要被多个过滤器过滤，那到底由哪个过滤器先过滤。

如果这些过滤器都是在web.xml中部署的

过滤器过滤的的顺序取决于部署的先后顺序( <filter-mapping>的先后顺序)

如果这些过滤器都是通过注解部署的

将过滤器类(filter类)的名字按字典顺序进行排序，类名小的过滤器优先进行过滤。

可能存在这么一种情况，一个过滤器链由多个过滤器组成，其中几个是在web.xml部署的，另外几个是通过注解部署，那么哪个过滤器先进行过滤呢？

这种情况下，在web.xml中的部署的过滤器先根据filter-mapping标记出现的先后顺序进行过滤，然后通过注解部署的过滤器再按filter类名进行字典顺序，按顺序进行过滤。

**监听器**

**监听器的分类**

根据监听的对象可将监听器分为3类

- ServletRequest对象的监听器。用于监听request对象本身，和request对象属性的变化。

- ServlelContext对象的监听器。用于监听application对象本身，和application对象属性的变化。

- HttpSession对象的监听器。用于监听session对象本身，session对象属性的变化，以及监听存储在session中的对象的状态变化。

根据监听的事件可将监听器分为3类

- 监听对象自身变化的监听器。对象自身的变化有: 创建和销毁

- 监听对象属性变化的监听器。对象属性的变化有: 添加属性，修改属性，删除

属性。

- 监听存储在session中的某个对象的状态变化的监听器。存储在session中的

对象的状态变化有: 绑定->解除绑定，钝化->活化。

**如何声明监听器类**

声明一个类实现某个监听接口，并重写相应方法即可，这样的类被为监听器类，由监听器类创建的对象被称为监听器。注意，如果想要创建的监听器具有特定的监听功能，只需要监听器类实现特定功能的监听接口即可。如果想要创建监听器具有多种监听功能，则监听器类需要实现多个具有不同功能的监听接口。

**监听接口**

以下三个监听接口，用于监听对象自身的变化:

- ServletRequestListener接口

由实现ServletRequestListener接口的监听器类创建的监听器，可以监听request对象的创建和销毁。具体做法是: 在实现ServletRequestListener接口的监听器类中重写该接口中的两个方法实现监听request对象的创建和销毁，分别是requestInitialized和requestDestroyed方法。

- ServletContextListener接口

由实现ServletContextListener接口的监听器类创建的监听器，可以监听application对象的创建和销毁。具体做法是: 在实现ServletContextListener接口的监听器类中重写该接口中的两个方法实现监听application对象的创建和销毁，分别是contextInitialized和contextDestroyed方法。

- HttpSessionListener接口

由实现ServletContextListener接口的监听器类创建的监听器，可以监听session对象的创建和销毁。具体做法是:在实现ServletContextListener接口的监听器类中重写该接口中的两个方法实现监听session对象的创建和销毁，分别是sessionCreated和sessionDestroyed方法。

以下三个监听接口，用于监听对象属性变化:

- ServletRequestAttributeListener接口

监听器类实现该接口中有以下三个方法需要实现:

- public void attributeAdded(ServletRequestAttributeEvent srae); //当往request对象中添加属性时，监听器会执行该方法。

- public void attributeRemoved(ServletRequestAttributeEvent srae); //当删除request对象中的属性时，监听器会执行该方法。

- public void attributeReplaced(ServletRequestAttributeEvent srae); //当修改request对象中的属性时，监听器会执行该方法。

- ServletContextAttributeListener接口

同理也有三个方法需要实现

- HttpSessionAttributeListener接口

同理也有三个方法需要实现

以下二个监听接口，用于监听session域中对象的变化:

- HttpSessionBindingListener接口，该接口中有2个方法需要实现

- public void valueBound(HttpSessionBindingEvent event); //当session对象调用setAttribute方法与某个对象进行绑定时，监听器会执行该方法。绑定就是将某个对象通过setAttribute方法存储到session对象中。

- public void valueUnbound(HttpSessionBindingEvent event); //当session对象调用removeAttribute方法与某个对象解除绑定时，监听器会执行该方法。解除绑定就是session对象调用removeAttribute方法删除其中的某个对象。

注: 添加到session中的对象，如果要监听其绑定和解除绑定，该对象对应的要实现这个监听接口。可以看出实现该接口的类的对象，既是一个普通对象(数据bean)，也是一个监听器。

- HttpSessionActivationListener接口

该接口用于监听存储在session中的对象的钝化和活化。

可以将在服务器中创建的session对象可以进行钝化，钝化实际就是使用序列化流将session对象写入到硬盘文件中存储。可想而知存储在session中对象，也会随着其钝化。

反之，也可以将存储在硬盘文件中的session对象，使用反序列流重新读取出来，这就是活化。可想而知存储在session中对象，也会随着其活化。

该接口中有两个方法需要实现:

public void sessionWillPassivate(HttpSessionEvent se); //某个对象将要随着seesion对象钝化前，该对象会执行这个方法。所以可已看出实现HttpSessionActivationListener接口的类的对象，即使一个普通对象，也是一个监听器。

- public void sessionDidActivate(HttpSessionEvent se); //某个对象随着seesion对象活化前，该对象会执行这个方法。

注: 想要监听存储在session中的某个对象的钝化和活化，该对象对应类必须实现该监听接口和Serializable接口。

拓展: 监听对象自身变化的监听器和监听对象属性变化的监听器需要部署才能

使用；用于监听session域中对象的变化的监听器不需要部署即可使用。