Web应用开发
之
高级应用

本章内容

- 8.1 Web监听器
- 8.2 Web过滤器
- 8.3 Servlet的多线程问题
- 8.4 Servlet的异步处理

8.1 Web监听器

- Web应用程序中的事件主要发生在三个对象上：ServletContext、HttpSession和ServletRequest对象。
- 事件的类型主要包括对象的生命周期事件和属性改变事件。
- 例如，对于ServletContext对象，当它初始化和销毁时会发生ServletContextEvent事件，当在该对象上添加属性、删除属性或替换属性时会发生ServletContextAttributeEvent事件。
- 为了处理这些事件，Servlet容器采用了监听器模型，即需要实现有关的监听器接口。
  - ServletContext事件监听器
  - HttpSession事件监听器
  - ServletRequest事件监听器

8.2 Web过滤器

- 8.2.1 什么是过滤器
- 8.2.2 过滤器API
- 8.2.3 一个简单的过滤器
- 8.2.4 @WebFilter注解
- 8.2.5 在DD中配置过滤器

8.2.1 什么是过滤器

- 过滤器（Filter）是Web服务器上的组件，它拦截客户对某个资源的请求和响应，对其进行过滤。
- 图8-3说明了过滤器的一般概念，其中F1是一个过滤器。它显示了请求经过滤器F1到达Servlet，Servlet产生响应再经过滤器F1到达客户。这样，过滤器就可以在请求和响应到达目的地之前对它们进行监视。

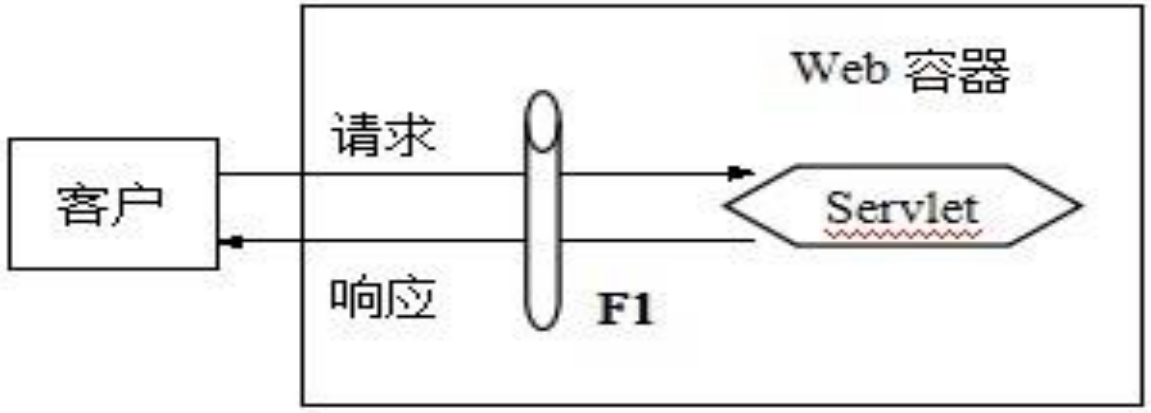


图 8-3 单个的过滤器

8.2.1 什么是过滤器

- 可以在客户和资源之间建立多个过滤器，从而形成过滤器链（filter chain）。在过滤器链中每个过滤器都对请求处理，然后将请求发送给链中的下一个过滤器。类似地，在响应到达客户之前，每个过滤器以相反的顺序对响应处理。
- 图8-4中，请求是按下列顺序处理的：过滤器F1、过滤器F2、过滤器F3，而响应的处理顺序是过滤器F3、过滤器F2、过滤器F1。

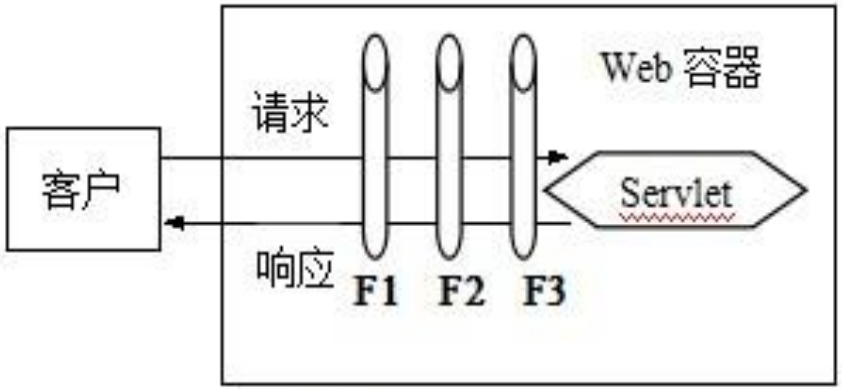


图 8-4 使用多个过滤器

1. 过滤器是如何工作的

- 当容器接收到对某个资源的请求时，它首先检查是否有过滤器与该资源关联。如果有过滤器与该资源关联，容器先把该请求发送给过滤器，而不是直接发送给资源。
- 在过滤器处理完请求后，它将做下面三件事：

  （1）将请求发送到目标资源。

  （2）如果有过滤器链，它将把请求（修改过或没有修改过）发送给下一个过滤器。

  （3）直接产生响应并将其返回给客户。
- 当请求返回到客户时，它将以相反的方向经过同一组过滤器。过滤器链中的每个过滤器都可能修改响应。

2. 过滤器的用途

- Servlet规范中提到的过滤器的一些常见应用包括：

  验证过滤器

  登录和审计过滤器

  数据压缩过滤器

  加密过滤器

  XSLT过滤器（eXtensible Stylesheet Language Transformation，可扩展样式表语言转换）

8.2.2 过滤器API

- 过滤器的类和接口定义在javax.servlet和javax.servlet.http包中。

- javax.servlet包中的3个接口
  - Filter：所有的过滤器都需要实现该接口
  - FilterConfig：过滤器配置对象。容器提供了该对象，其中包含了该过滤器的初始化参数
  - FilterChain：过滤器链对象

1. Filter接口

- Filter接口是过滤器API的核心，所有的过滤器都必须实现该接口。该接口声明了三个方法，分别是init()、doFilter()和destroy()，它们是过滤器的生命周期方法。
- public void init(FilterConfig filterConfig)
- public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)
  throws IOException, ServletException;
- public void destroy();

2. FilterConfig接口

- FilterConfig对象是过滤器配置对象，通过该对象可以获得过滤器名、过滤器运行的上下文对象以及过滤器的初始化参数。它声明了如下4个方法：
- public String getFilterName()
- public ServletContext getServletContext()
- public String getInitParameter(String name)
- public Enumeration getInitParameterNames()
- 容器提供了FilterConfig接口的一个具体实现类，容器创建该类的一个实例、使用初始化参数值对它初始化，然后将它作为一个参数传递给过滤器的init()。

3. FilterChain接口

- FilterChain接口只有一个方法，如下所示。

```
public void doFilter(ServletRequest request,
                     ServletResponse response)
throws IOException, ServletException
```

- 在Filter对象的doFilter()中调用该方法使过滤器继续执行，它将控制转到过滤器链的下一个过滤器或实际的资源。
- 容器提供了该接口的一个实现并将它的一个实例作为参数传递给Filter接口的doFilter()。在doFilter()内，可以使用该接口将请求传递给链中的下一个组件，它可能是另一个过滤器或实际的资源。该方法的两个参数将被链中下一个过滤器的doFilter()或Servlet的service()接收。

8.2.3　一个简单的过滤器

- 程序8.9 LogFilter.java是一个简单的日志过滤器，这个过滤器拦截所有的请求并将请求有关信息记录到日志文件中。程序声明的LogFilter类实现了Filter接口，覆盖了其中的init()、doFilter()和destroy()。

- 程序在doFilter()中首先将请求对象（request）转换成HttpServletRequest的类型，然后获得当前时间、客户请求的URI和客户地址，并将其写到日志文件中。之后将请求转发到资源，当请求返回到过滤器后再得到当前时间，计算请求资源的时间并写到日志文件中。

8.2.4 @WebFilter注解

表 8-5 @WebFilter 注解的常用元素

元素名	类 型	说 明
filterName	String	指定过滤器的名称，等价于 web.xml 中的<filter-name>元素。如果没有显式指定，则使用 Filter 的完全限定名作为名称
urlPatterns	String[]	指定一组过滤器的 URL 匹配模式，该元素等价于 web.xml 文件中的<url-pattern>元素
value	String[]	该元素等价于 urlPatterns 元素。两个元素不能同时使用
servletNames	String[]	指定过滤器应用于哪些 Servlet。取值是@WebServlet 中 name 属性值，或者是 web.xml 中<servlet-name>的取值
dispatcherTypes	DispatcherType	指定过滤器的转发类型。具体取值包括：ASYNC、ERROR、FORWARD、INCLUDE 和 REQUEST
initParams	WebInitParam[]	指定一组过滤器初始化参数，等价于<init-param>元素
asyncSupported	boolean	声明过滤器是否支持异步调用，等价于<async-supported>元素
description	String	指定该过滤器的描述信息，等价于<description>元素
dispalyName	String	指定该过滤器的显示名称，等价于<display-name>元素

8.2.4 @WebFilter注解

- @WebFilter中所有属性均为可选属性，但是 value、urlPatterns、servletNames 三者必须至少包含一个，且 value 和 urlPatterns 不能共存，如果同时指定，通常忽略 value 的取值。
- 过滤器接口Filter与Servlet非常相似，它们具有类似的生命周期行为，区别只是Filter的doFilter()中多了一个FilterChain的参数，通过该参数可以控制是否放行用户请求。
- 像Servlet一样，Filter也可以具有初始化参数，这些参数可以通过@WebFilter注解或部署描述文件定义。在过滤器中获得初始化参数使用FilterConfig实例的getInitParameter()。

8.2.4 @WebFilter注解

- 在实际应用中，使用Filter可以更好实现代码复用。

① 一个系统可能包含多个Servlet，这些Servlet都需要进行一些通用处理，比如权限控制、记录日志等，这将导致多个Servlet的service()中包含部分相同代码。

② 为解决这种代码重复问题，就可以考虑把这些通用处理提取到Filter中完成，这样在Servlet中就只剩下针对特定请求相关的处理代码。

8.2.4 @WebFilter注解

- 程序8.10 AuthorityFilter.java定义一个较为实用的Filter

① 对用户请求进行过滤，为请求设置编码字符集，从而可以避免为每个JSP页面、Servlet都设置字符集

② 实现验证用户是否登录，如果用户没有登录，系统直接跳转到登录页面。

8.2.4 @WebFilter注解

- 该过滤器通过@WebFilter注解的initParams元素指定了三个初始化参数，参数使用@WebInitParam注解指定，每个@WebInitParam指定一个初始化参数。在Filter的doFilter()中通过FilterConfig对象取出参数的值。
- 程序中设置了请求的字符编码，还通过Session对象验证用户是否登录，若没登录将请求直接转发到登录页面，若已登录则转发到请求的资源。

8.2.5　在web.xml中配置过滤器

- 除了可以通过注解配置过滤器外，还可以使用部署描述文件web.xml配置过滤器类并把请求URL映射到该过滤器上。
- 配置过滤器要用下面两个元素：<filter>和<filter-mapping>。
- 每个<filter>元素向Web应用程序引进一个过滤器，每个<filter-mapping>元素将一个过滤器与一组请求URI关联。两个元素都是<web-app>的子元素。

1. <filter>元素

- 该元素用来指定过滤器名和过滤器类，下面是<filter>元素的DTD定义：

  <!ELEMENT filter (description?, display-name?,icon?, filter-name, filter-class, init-param*)>

- 从上面定义可以看到，每个过滤器都需要一个<filter-name>元素和一个<filter-class>元素。其他元素是可选的。下面代码说明了<filter>元素的使用。

1. <filter>元素

```
<filter>
    <!--指定过滤器名和过滤器类-->
    <filter-name>validatorFilter</filter-name>
    <filter-class>filter.ValidatorFilter</filter-class>
    <init-param>
        <param-name>locale</param-name>
        <param-value>Hangzhou</param-value>
    </init-param>
</filter>
```

2. <filter-mapping>元素

- 该元素的作用定义过滤器映射，<filter-mapping>元素的DTD定义如下：

  <!ELEMENT filter-mapping (filter-name, (url-pattern | servlet-name),dispatcher)>

- <filter-name>元素是在<filter>元素中定义的过滤器名，<url-pattern>用来将过滤器应用到一组通过URI标识的请求，<servlet-name>用来将过滤器应用到通过该名标识的Servlet提供服务的所有请求。在使用<servlet-name>情况下，模式匹配遵循与Servlet映射同样的规则。

2. <filter-mapping>元素

下面代码说明了<filter-mapping>元素的使用：

```
<filter-mapping>
        <filter-name>validatorFilter</filter-name>
        <url-pattern>*.jsp</url-pattern>
</filter-mapping>

<filter-mapping>
        <filter-name>validatorFilter</filter-name>
        <servlet-name>reportServlet</servlet-name>
</filter-mapping>
```

3. 配置过滤器链

- 在某些情况下，对一个请求可能需要应用多个过滤器，这样的过滤器链可以使用多个<filter-mapping>元素配置。
- 当容器接收到一个请求，它将查找所有与请求URI匹配的过滤器映射的URL模式，这是过滤器链中的第一组过滤器。
- 接下来，它将查找与请求URI匹配的Servlet名，这是过滤器链中的第二组过滤器。
- 在这两组过滤器中，过滤器的顺序是它们在DD文件中的顺序。

3. 配置过滤器链

- 为了理解这个过程，考虑下面对过滤器和Servlet映射代码。

```
<servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
<filter-mapping>
    <filter-name>perfFilter</filter-name>
    <servlet-name>FrontController</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>auditFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>transformFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

- 如果一个请求URI为**/admin/addCustomer.do**，将以下面的顺序应用过滤器：**auditFilter**、**transformFilter**。
- 如果一个请求URI为***.do**，将以下面的顺序应用过滤器：**auditFilter**、**transformFilter**、**perfFilter** 。

4. 为转发的请求配置过滤器

- 过滤器还可以应用在从组件内部转发的请求上，这包括使用RequestDispatcher的include()和forward()转发的请求以及对错误处理调用的资源的请求。
- 要为转发的请求配置过滤器，可以使用<filter-mapping>元素的子元素<dispatcher>实现，该元素的取值包括下面4个：REQUEST、INCLUDE、FORWARD和ERROR。

4. 为转发的请求配置过滤器

- REQUEST表示过滤器应用在直接来自客户的请求上。
- INCLUDE表示过滤器应用在与调用RequestDispatcher的include()匹配的请求。
- FORWARD表示过滤器应用在与调用RequestDispatcher的forward()匹配的请求。
- ERROR表示过滤器应用于由在发生错误而引起转发的请求上。

4. 为转发的请求配置过滤器

- 在<filter-mapping>元素中可以使用多个<dispatcher>元素使过滤器应用在多种情况下，例如：

```
<filter-mapping>
    <filter-name>transferFilter</filter-name>
    <url-pattern>*.do</url-pattern>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

- 上述过滤器映射将只应用在从内部转发的且其URL与*.do匹配的请求上，任何直接来自客户的请求，即使其URL与*.do匹配也将不应用transferFilter过滤器。

8.3 Servlet的多线程问题

- 在Web应用程序中，一个Servlet在一个时刻可能被多个用户同时访问。这时Web容器将为每个用户创建一个线程。
- 如果Servlet不涉及共享资源的问题，不必关心多线程问题。但如果Servlet需要共享资源，需要保证Servlet是线程安全的。

非线程安全的Servlet示例

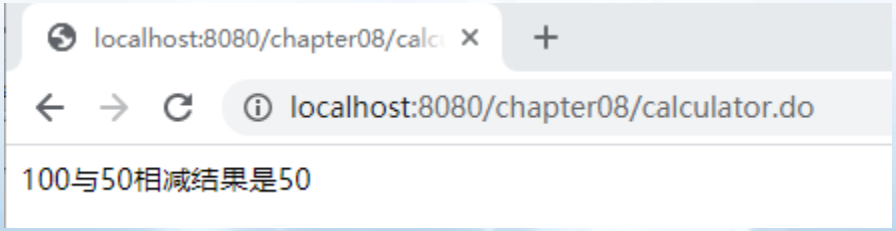
- 程序8.11CalculatorServlet.java从客户接受两个整数，然后计算它们的和或差
- 该程序将计算结果存放在变量result中，它根据用户在页面中单击的是“相加”按钮或“相减”按钮决定求和还是求差。
- result被声明为一个成员变量。为了演示多个用户请求时出现的问题，程序中调用Thread类的sleep()在计算出result后睡眠一段时间（假设2秒种），睡眠的时间通过Servlet初始化参数sleepTime得到。getNumber()实现字符串到int数据的转换。最后输出计算的结果。

非线程安全的Servlet示例

- 程序8.12 calculator.jsp 页面中的表单包含两个文本框用来接受两个整数，两个提交按钮，一个做加法、一个做减法。
- 测试该Servlet的执行。打开两个浏览器窗口，每个窗口都载入calculator.jsp页面，在两个页面的文本框中都输入100和50，然后单击第一个页面中的“相加”按钮，在2秒钟内单击第二个页面的“相减”按钮

非线程安全的Servlet示例

- Servlet的执行过程如下：
- 当两个用户同时访问该Servlet时，服务器创建两个线程来提供服务。当第一个用户提交表单后，它执行其所在线程的doPost()，计算100与50的和并将结果150存放在result变量中，然后在输出前睡眠2秒钟。
- 在这个时间内，当第二个窗口提交时，它将计算100与50的差，将结果50写到result变量中，此时第一个线程的计算结果被覆盖，当第一个线程恢复执行后输出结果也为50。



Servlet非线程安全原因和解决方法

- 出现这种错误的原因是在Servlet中使用成员变量result来保存请求计算结果，成员变量在多个线程（请求）中只有一份拷贝，而这里的result应该是请求的专有数据。
- Servlet还经常要共享外部资源，如使用一个数据库连接对象。如果将连接对象声明为Servlet的成员变量，则当多个并发的请求在同一个连接上写入数据时，数据库将产生错误的数据
- 解决这个问题的办法是用方法的局部变量来保存请求的专有数据。这样，进入方法的每个线程都有自己的一份方法变量的拷贝，任何线程都不会修改其他线程的局部变量。

编写线程安全的Servlet的建议

- （1）用方法的局部变量保存请求中的专有数据。对方法中定义的局部变量，进入方法的每个线程都有自己的一份方法变量拷贝。如果要在不同的请求之间共享数据，应该使用会话来共享这类数据。
- （2）只用Servlet的成员变量来存放那些不会改变的数据。有些数据在Servlet生命周期中不发生任何变化，通常是在初始化时确定的，这些数据可以使用成员变量保存。如，数据库连接名称、其他资源的路径等。在上述例子中sleepTime的值是在初始化时设定的并在Servlet的生命期内不发生改变，所以可以把它定义为一个成员变量。

编写线程安全的Servlet的建议

（3）对可能被请求修改的成员变量同步（使用synchronized关键字）。有时数据成员变量或者环境属性可能被请求修改。当访问这些数据时应该对它们同步，以避免多个线程同时修改这些数据。

（4）如果Servlet访问外部资源，那么需要对这些资源同步。例如假设Servlet要从文件中读写数据。当一个线程读写一个文件时，其他线程也可能正在读写这个文件。文件访问本身不是线程安全的，所以必须编写同步代码访问这些资源。

编写线程安全的Servlet的建议

- 在编写线程安全的Servlet时，下面两种方法是不应该使用的：

  （1）在Servlet API中提供了一个SingleThreadModel接口，实现这个接口的Servlet在被多个客户请求时一个时刻只有一个线程运行。这个接口已被标记不推荐使用。

  （2）对doGet()或doPost()同步。如果必须在Servlet中使用同步代码，应尽量在最小的代码块范围上进行同步。同步代码越少，Servlet执行效率越高。

8.4 Servlet的异步处理

- Servlet 3.0之前，Servlet的执行过程大致如下：Web容器接收到用户对某个Servlet请求之后启动一个线程，在该线程中对请求的数据进行预处理；接着，调用业务接口的某些方法，以完成业务处理；最后，根据处理的结果提交或转发响应，Servlet线程结束。
- 其中第二步的业务处理通常是最耗时的，如访问数据库操作、跨网络调用等。此时，Servlet线程一直处于阻塞状态，直到业务执行完毕。在此过程中，线程资源一直被占用而得不到释放，对于并发用户较多的应用，这有可能造成性能的瓶颈 。

8.4.1 异步处理概述

- Servlet 3.0增加了异步处理支持，Servlet的执行过程调整如下：

➢ Web容器接收到用户对某个Servlet请求之后启动一个线程，在该线程中对请求的数据进行预处理；

➢ 接着，Servlet线程将请求转交给一个异步线程来执行业务处理，Servlet线程本身返回至容器，此时Servlet还没有生成响应数据。

➢ 异步线程处理完业务以后，可以直接生成响应数据（异步线程拥有ServletRequest和ServletResponse对象的引用），或者将请求转发给其他Servlet或JSP页面。

➢ 这样，Servlet线程不再是一直处于阻塞状态以等待业务逻辑的处理，而是启动异步线程之后可以立即返回。

8.4.1 异步处理概述

- 异步线程处理可应用于Servlet和过滤器两种组件，由于异步处理的工作模式和普通工作模式在实现上有着本质的区别，因此默认情况下，Servlet和过滤器并没有开启异步处理特性。
- Servlet 3.0的异步处理是通过AsyncContext类来实现的，Servlet可以通过ServletRequest的startAsync方法创建AsyncContext对象，开启异步调用。

8.4.1 异步处理概述

- public AsyncContext startAsync()：开始异步调用并返回AsyncContext对象，其中包含最初的请求和响应对象。
- public AsyncContext startAsync(ServletRequest request,ServletResponse response)：开启异步调用，并传递经过包装的请求和响应对象。
- AsyncContext表示异步处理的上下文，该类提供了一些工具方法，可完成启动后台线程、转发请求、设置异步调用的超时时长、获取request和response对象等功能。

8.4.2 异步调用Servlet的开发

- 如果一个任务需要花费较长时间完成，就应该通过异步Servlet实现。
- 开发支持异步线程调用的Servlet的一般步骤：

① 调用ServletRequest对象的startAsync()，该方法返回AsyncContext对象，它是异步处理的上下文对象。

② 调用AsyncContext对象的setTimeout()，传递一个毫秒时间设置容器等待指定任务完成的时间。如果没有设置超时时间，容器将使用默认时间。在指定的时间内任务不能完成将抛出异常。

③ 调用AsyncContext对象的start()，为其传递一个要用异步线程执行的Runnable对象。

④ 当任务结束时在线程对象中调用AsyncContext对象的complete()或dispatch()。

8.4.2　异步调用Servlet的开发

- 程序8.13 AsyncDemoServlet.java是一个简单的模拟异步处理的Servlet。
- 该类中创建了一个AsyncContext对象，并通过该对象以异步的方式启动了一个后台线程。该线程执行体模拟调用耗时的业务方法，下面的Executor类就是线程体类。
- 程序8.14 Executor.java 主线程结束时异步线程还没有结束，主线程已经返回给容器。

8.4.2 异步调用Servlet的开发

- 对于希望启用异步调用的Servlet而言，开发者必须显式指定开启异步调用，有两种方式指定异步调用：

① 为@WebServlet注解指定asyncSuppored=true。

② 在web.xml文件的<servlet>元素中增加<async-suppored>子元素。

- Servlet 3.0为<servlet>和<filter>标签增加了<async-supported>子标签，该标签的默认取值为false，要启用异步处理支持，则将其设为true即可。
- 以Servlet为例，其配置如下所示：

```
<servlet>
    <servlet-name>AsyncDemoServlet</servlet-name>
    <servlet-class>com.demo.AsyncDemoServlet</servlet-class>
    <async-supported>true</async-supported>
</servlet>
```

8.4.3 实现AsyncListener接口

- 在Servlet 3.0中增加了一个AsyncListener接口来处理异步操作事件。当Servlet启用异步调用时发生AsyncEvent事件。
- 要处理这类事件，需实现AsyncListener接口，该接口定义了如下4个方法：

- public void onStartAsync (AsyncEvent event)：当异步调用开始时触发该方法。
- public void onComplete (AsyncEvent event)：当异步调用完成时触发该方法。
- public void onError (AsyncEvent event)：当异步调用出错时触发该方法。
- public void onTimeout (AsyncEvent event)：当异步调用超时时触发该方法。

8.4.3 实现AsyncListener接口

- 上述方法的参数是AsyncEvent类的对象，通过该对象的getAsyncContext()、getSuppliedRequest()和getSuppliedResponse()可分别返回AsyncContext对象、ServletRequest对象及ServletResponse对象等。
- 程序8.15 MyAsyncListener.java 实现了AsyncListener接口，当发生异步操作事件时可由它处理。

8.4.3　实现AsyncListener接口

与其他Web监听器不同，AsyncListener监听器不需要使用@WebListener注册或在web.xml中注册，但需要使用AsyncContext对象的addListener()进行手工注册，该方法格式如下：

```
public void addListener(AsyncListener listener)
```

- 程序8.16　AsyncListenerServlet.java 是一个异步Servlet，它使用了上述监听器。

8.5 小 结

- 在Web应用程序运行过程中会发生某些事件，为了处理这些事件，容器也采用了事件监听器模型。根据事件的类型和范围，可以把事件监听器分为三类：ServletContext事件监听器、HttpSession事件监听器和ServletRequest事件监听器。
- 对Web应用来说，过滤器是Web服务器上的组件，它们对客户和资源之间的请求和响应进行过滤。可以定义多种类型的过滤器，如验证过滤器、审计过滤器、数据压缩过滤器、加密过滤器等。

8.5 小 结

- 在编写有多个用户同时访问的Servlet时，一定要保证Servlet是线程安全的。一般不使用成员变量共享请求的专有数据。如果必须要在多个请求之间共享数据，则应当对这些数据同步，并且尽量在最小的代码块范围上进行同步。
- Servlet 3.0新增的Servlet和过滤器的异步处理线程的目的是提高系统的性能。它通过AsyncContext对象的startAsync()和start()开始和启动一个异步执行的线程，使当前线程立即返回，使耗时的操作在一个异步线程中执行，从而提高系统的性能。

课后作业

1、修改教材P254的例子程序8.3和程序8.4，实现客户端访问onlineCount.jsp的次数，并将客户端的访问记录（包括ip、访问时间）保存至数据库中，要求同一个IP只能记录一次。

提示：用户访问的IP可存到List或数据库表中，若有用户访问onlineCount.jsp，则在监听器MyRequestListener中得到客户端IP，若该IP已在List对象或数据库中，则不增加访问次数，否则访问次数加1，并将该IP加到List对象或数据库中。测试时，用两台电脑或者手机与电脑组成一个网络，通过IP访问页面。

课后作业

2、编写测试的例子，实现教材P279第8题的过滤器，其中过滤器FilterOne 、FilterTwo和FilterThree类中的doFilter方法中分别输出“这是过滤器FilterOne ”、“这是过滤器FilterTwo ”、“这是过滤器FilterThree ”。另，如何能运行过滤器FilterOne，并写程序进行测试。

提示：

（1）编写3个过滤器类，doFilter中打印该过滤器名，并调用过滤器类

（2）在web.xml或者用@WebFilter中写这几个过滤器的映射

（3）编写/admin/index.jsp并允许测试