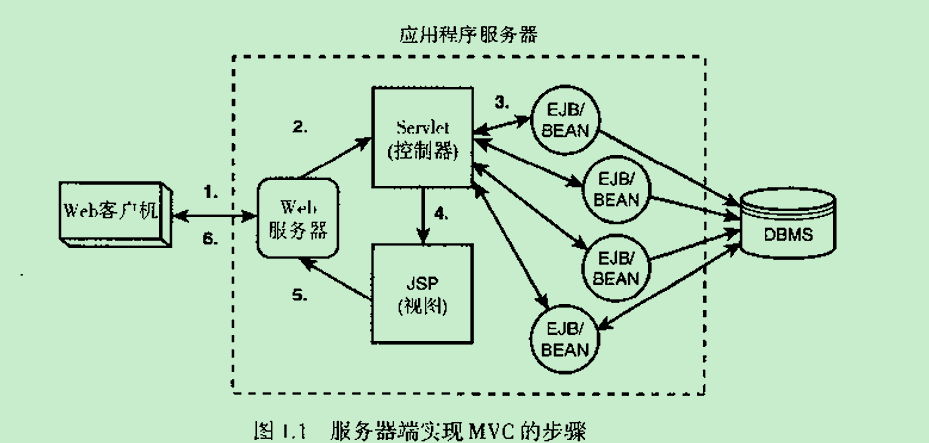
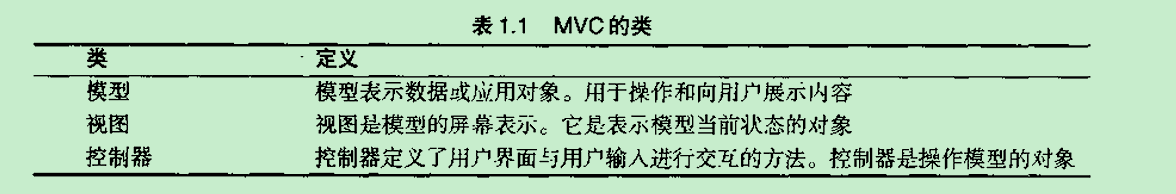
# MVC的模式



1. 客户端发送请求到web服务器
2. Web服务器把请求转送给servlet控制器
3. Servlet对javaBean/EJB模型进行必要的操作
4. 控制器把处理结果转换给jsp视图
5. Jsp视图对模型进行格式化显示，并把html结果返回给web服务器
6. Web服务器将信息返回给客户端



# servlet目前最新版本为3.0

## servlet是什么:

Servlet是一种服务器端的Java应用程序，具有独立于平台和协议的特性，可以生成动态的Web页面。 它担当客户请求（Web浏览器或其他HTTP客户程序）与服务器响应（HTTP服务器上的数据库或应用程序）的中间层。 Servlet是位于Web 服务器内部的服务器端的Java应用程序，与传统的从命令行启动的Java应用程序不同，Servlet由Web服务器进行加载，该Web服务器必须包含支持Servlet的Java虚拟机。

## Servlet的过程:

这个过程为：

客户端发送请求至服务器端；

服务器将请求信息发送至Servlet

Servlet生成响应内容并将其传给服务器。响应内容动态生成，通常取决于客户端的请求

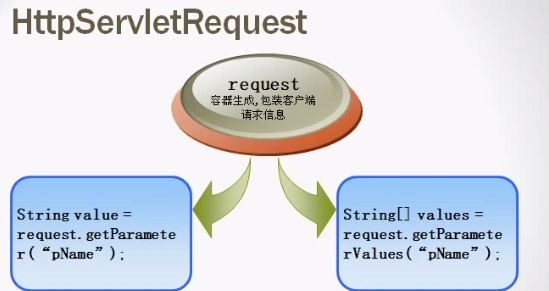
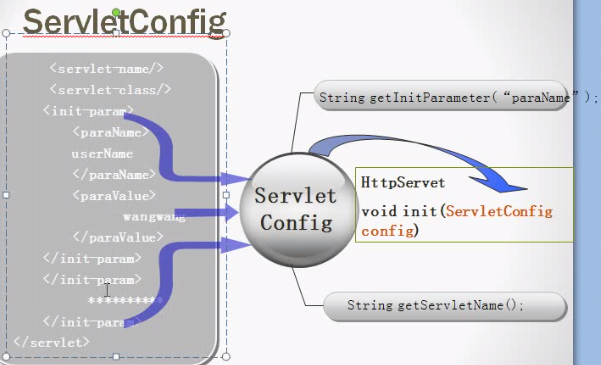
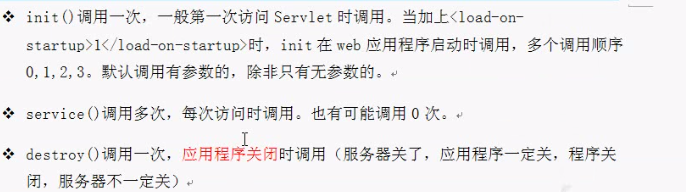
服务器将响应返回给客户端

## Servlet类与java类的区别:



## Servlet的生命周期:





加载和实例化Servlet。这项操作一般是动态执行的。然而，Server通常会提供一个管理的选项，用于在Server启动时强制装载和初始化特定的Servlet。

Server创建一个Servlet的实例

一个客户端的请求到达Server

Server调用Servlet的init()方法（可配置为Server创建servlet实例时调用）

Server创建一个请求对象，处理客户端请求

Server创建一个响应对象，响应客户端请求

Server激活Servlet的service()方法，传递请求和响应对象作为参数

service()方法获得关于请求对象的信息，处理请求，访问其他资源，获得需要的信息

service()方法使用响应对象的方法，将响应传回Server，最终到达客户端。service()方法可能激活其它方法以处理请求，如doGet()或doPost()或程序员自己开发的新的方法。

对于更多的客户端请求，Server创建新的请求和响应对象，仍然激活此Servlet的service()方法，将这两个对象作为参数传递给它。如此重复以上的循环，但无需再次调用init()方法。一般Servlet只初始化一次(只有一个对象),当Server不再需要Servlet时(一般当Server关闭时)，Server调用Servlet的Destroy()方法。客户端发送请求至[服务器](http://baike.baidu.com/view/899.htm)

[服务器](http://baike.baidu.com/view/899.htm)启动并调用Servlet，Servlet根据[客户端](http://baike.baidu.com/view/930.htm)请求生成响应内容并将其传给服务器

服务器将响应返回客户端

## Servlet与传统CGI的比较:

Java Servlet 与 CGI(Common Gateway Interface公共网关接口) 的比较:

与传统的CGI和许多其他类似CGI的技术相比，Java Servlet具有更高的效率，更容易使用，功能更强大，具有更好的可移植性，更节省投资。在未来的技术发展过程中，Servlet有可能彻底取代CGI。

在传统的CGI中，每个请求都要启动一个新的进程，如果CGI程序本身的执行时间较短，启动进程所需要的开销很可能反而超过实际执行时间。而在Servlet中，每个请求由一个轻量级的Java线程处理(而不是重量级的操作系统进程)。

在传统CGI中，如果有N个并发的对同一CGI程序的请求，则该CGI程序的代码在内存中重复装载了N次；而对于Servlet，处理请求的是N个线程，只需要一份Servlet类代码。在性能优化方面，Servlet也比CGI有着更多的选择。

# 接口(重点):

HTTPServlet 使用一个 HTML 表单来发送和接收数据。要创建一个 HTTPServlet，请扩展 HttpServlet 类， 该类是用专门的方法来处理 HTML 表单的 GenericServlet 的一个子类。 HTML 表单是由 <FORM> 和 </FORM> 标记定义的。表单中典型地包含输入字段(如文本输入字段、复选框、单选按钮和选择列表)和用于提交数据的按钮。当提交信息时，它们还指定服务器应执行哪一个Servlet(或其它的程序)。 HttpServlet 类包含 init()、destroy()、service() 等方法。其中 init() 和 destroy() 方法是继承的。

(1) init() 方法

在 Servlet 的生命期中，仅执行一次 init() 方法。它是在服务器装入 Servlet 时执行的。 可以配置服务器，以在启动服务器或客户机首次访问 Servlet 时装入 Servlet。 无论有多少客户机访问 Servlet，都不会重复执行 init() 。

缺省的 init() 方法通常是符合要求的，但也可以用定制 init() 方法来覆盖它，典型的是管理服务器端资源。 例如，可能编写一个定制 init() 来只用于一次装入 GIF 图像，改进 Servlet 返回 GIF 图像和含有多个客户机请求的性能。另一个示例是初始化数据库连接。缺省的 init() 方法设置了 Servlet 的初始化参数，并用它的 ServletConfig 对象参数来启动配置， 因此所有覆盖 init() 方法的 Servlet 应调用 super.init() 以确保仍然执行这些任务。在调用 service() 方法之前，应确保已完成了 init() 方法。

(2) service() 方法

service() 方法是 Servlet 的核心。每当一个客户请求一个HttpServlet 对象，该对象的service() 方法就要被调用，而且传递给这个方法一个"请求"(ServletRequest)对象和一个"响应"(ServletResponse)对象作为参数。 在 HttpServlet 中已存在 service() 方法。缺省的服务功能是调用与 HTTP 请求的方法相应的 do 功能。例如， 如果 HTTP 请求方法为 GET，则缺省情况下就调用 doGet() 。Servlet 应该为 Servlet 支持的 HTTP 方法覆盖 do 功能。因为 HttpServlet.service() 方法会检查请求方法是否调用了适当的处理方法，不必要覆盖 service() 方法。只需覆盖相应的 do 方法就可以了。

Servlet的响应可以是下列几种类型：

一个输出流，浏览器根据它的内容类型(如text/HTML)进行解释。

一个HTTP错误响应, 重定向到另一个URL、servlet、JSP。

(3)doGet()方法

当一个客户通过HTML 表单发出一个HTTP GET请求或直接请求一个URL时，doGet()方法被调用。与GET请求相关的参数添加到URL的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用doGet()方法。

(4)doPost()方法

当一个客户通过HTML 表单发出一个HTTP POST请求时，doPost()方法被调用。与POST请求相关的参数作为一个单独的HTTP 请求从浏览器发送到服务器。当需要修改服务器端的数据时，应该使用doPost()方法。

(5) destroy() 方法

destroy() 方法仅执行一次，即在服务器停止且卸装Servlet 时执行该方法。典型的，将 Servlet 作为服务器进程的一部分来关闭。缺省的 destroy() 方法通常是符合要求的，但也可以覆盖它，典型的是管理服务器端资源。例如，如果 Servlet 在运行时会累计统计数据，则可以编写一个 destroy() 方法，该方法用于在未装入 Servlet 时将统计数字保存在文件中。另一个示例是关闭数据库连接。

当服务器卸装 Servlet 时，将在所有 service() 方法调用完成后，或在指定的时间间隔过后调用 destroy() 方法。一个Servlet 在运行service() 方法时可能会产生其它的线程，因此请确认在调用 destroy() 方法时，这些线程已终止或完成。

(6) GetServletConfig()方法

GetServletConfig()方法返回一个 ServletConfig 对象，该对象用来返回初始化参数和ServletContext。ServletContext 接口提供有关servlet 的环境信息。

(7) GetServletInfo()方法

GetServletInfo()方法是一个可选的方法，它提供有关servlet 的信息，如作者、版本、版权。

当服务器调用sevlet 的Service()、doGet()和doPost()这三个方法时，均需要 "请求"和"响应"对象作为参数。"请求"对象提供有关请求的信息，而"响应"对象提供了一个将响应信息返回给浏览器的一个通信途径。

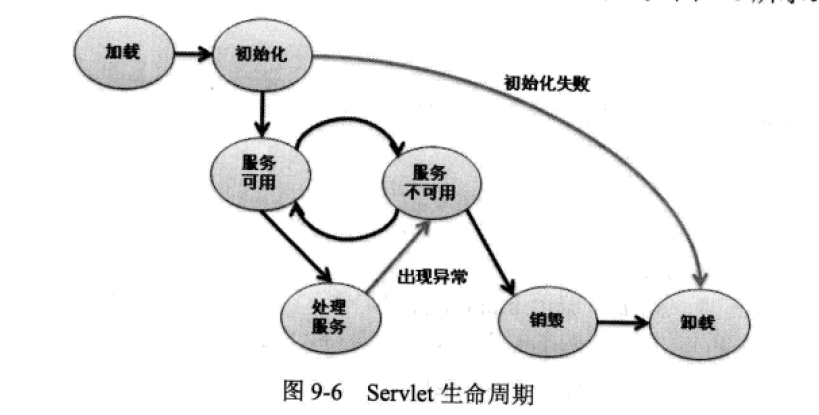
javax.servlet 软件包中的相关类为ServletResponse和ServletRequest，而javax.servlet.http 软件包中的相关类为HttpServletRequest 和 HttpServletResponse。Servlet 通过这些对象与服务器通信并最终与客户机通信。Servlet 能通过调用"请求"对象的方法获知客户机环境，服务器环境的信息和所有由客户机提供的信息。Servlet 可以调用"响应"对象的方法发送响应，该响应是准备发回客户机的。

开发servlet的三步走：

1. 包2.类3.superclass（httpservlet-javax.servlet.http包下面的）

配置web.xml文件

# Servlet生命周期：



## Servlet生命周期方法：



### 1.加载servlet

web容器负责加载servlet，成功加载后通过映射来实例化一个servlet

### 2.初始化

成功实例化之后调用servlet的init（）方法来初始化，初始化是为了在处理客户端的请求之前做一些准备工作：建立数据库连接，读取资源文件信息等。（如果初始化失败则会直接卸载）

### 3.处理服务

当有请求提交时servlet会调用service（）方法来处理在service方法中可以同过servletrequest来接受客户端的请求信息，同过servletresponse返回响应信息给客户端

### 4.销毁

当web容器关闭时或者检测到servlet要从容器中删除时会自动调用destroy方法

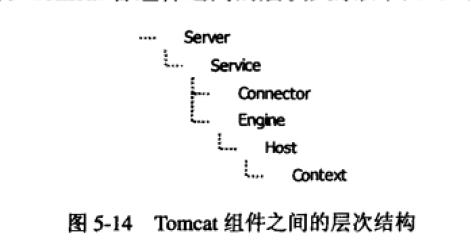
### 5.卸载

当servlet调用完destroy（）方法后此实例就等待着垃圾回收机制进行回收如果需要再次使用此servlet时会自动调用init（）方法进行初始化。

生命周期的特点：一个servlet只掉用一次init（）和初始化方法，和destroy（）。多次的servces（）方法

# tomcat的体系结构

tomcat是有一系列可配置的组件构成的其中核心组件是catalina servlet容器他是所有其他tomcat组件的顶层容器，tomcat各个组件之间的关系：



1. Server

Server表示整个的catalina servlet容器，tomcat提供了一个servlet接口的默认实现着通常不要用户自己去实现，servlet容器中可以包含一个或多个service组件

1. service

service是存在server中的内部的组件，他将一个或多个连接器（Connector）组件绑定到一个单独的引擎上（Engine）

1. Connector

连接器（Connector）处理与客户端的通信，他负责向客户端接受请求，以及向客户端返回相应结果，在tomcat中有多个连接器可以使用

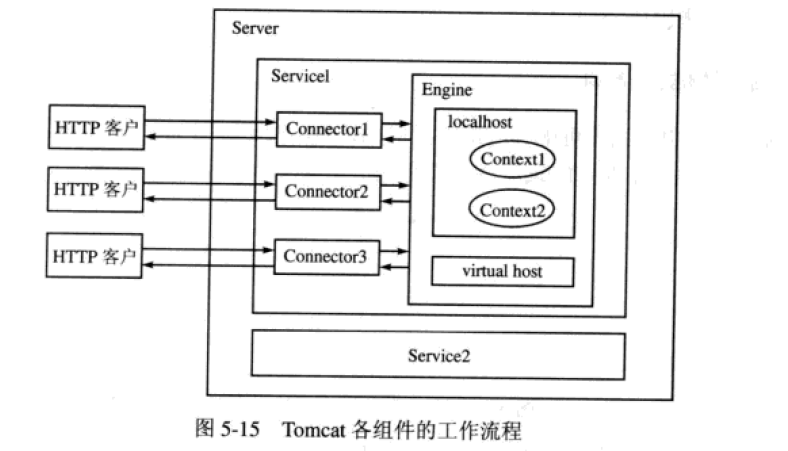
1. Engine

在tomcat中每个service都只包含一个engine。Engine（引擎）表示一个特定的service的请求处理路线，作为一个service通常有多个servlet引擎，引擎表示一个特定service请求流水线作为一个service可以有多个连接器，引擎将从连接器接受的和处理的所有的请求，将响应返回给适合的连接器。通过连接器传输给用户。用户可以通过实现engine接口提供自定义的引擎（但通常不需要这样做）

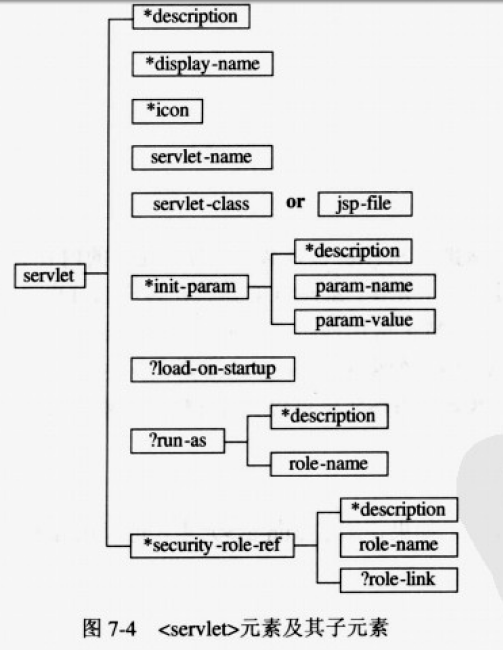
1. Host

Host表示的是一个虚拟的主机，一个engine通常有多个host

1. Context

一个context表示一个web应用程序运行在特定的虚拟主机中

# Web.xml中servlet的子元素：



## Description：

为servlet指定一个文本描述

<display-name>：

为servlet指定一个简短的名字

<icon>

为servlet指定一个图标（可以在一些图形界面中用于表示该servlet）

<servlet-name>

指定servlet的名字，这个名字在同一个web应用程序中必须是唯一的，

<servlet-class>

指定servlet类的完整权限名

<jsp-file>

指定在web应用程序中的jsp文件的完整路径，该路径以斜杠（/）开始

<init-param>

定义servlet的初始化参数，如果使用<init-param>元素就必须包含<param-name>和<param-value>元素可以包含零个或者多个 <description>元素

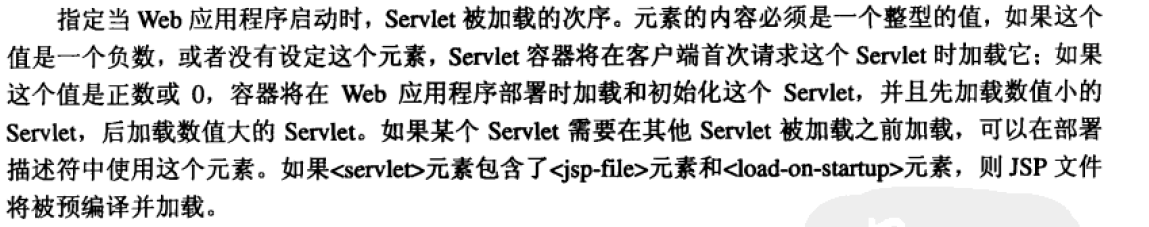
<param-name>

定义初始化参数的名字

<param-value>

定义初始化的值

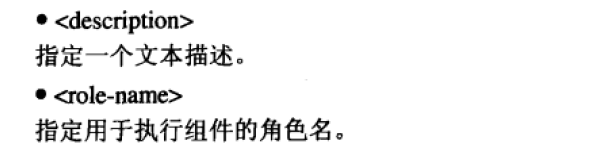
<load-on-startup>



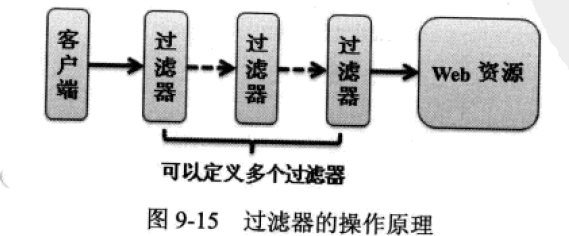
<run-as>

指定用于执行该组件的角色如果使用了<run-as>元素可以包含零个或者多个<descrption>元素

必须包含<role-name>元素



# Servlet过滤器（重点）



在Servlet规范2.3中定义了过滤器，它能够对Servlet容器的请求和响应对象进行检查和修改。

Servlet过滤器本身并不生成请求和响应对象，只是提供过滤功能。

Servlet过滤器能够在Servlet被调用之前检查Request对象，并修改Request Header和Request内容；

在Servlet被调用之后检查Response对象，修改Response Header和Response的内容。

Servlet过滤器可以过滤的Web组件包括Servlet，JSP和HTML等文件。

过滤器位于客户端和web应用程序之间，用于检查和修改两者之间流过的请求和响应。

在请求到达Servlet/JSP之前，过滤器截获请求。

在响应送给客户端之前，过滤器截获响应。

多个过滤器形成一个过滤器链，过滤器链中不同过滤器的先后顺序由部署文件web.xml中过滤器映射<filter-mapping>的顺序决定。

最先截获客户端请求的过滤器将最后截获Servlet/JSP的响应信息。

### 针对一个Servlet做过滤

<filter-mapping>  
    <filter-name>MyFilter</filter-name>  
    <servlet-name>MyServlet</servlet-name>  
</filter-mapping>

### 针对URL Pattern做过滤

<filter-mapping>  
    <filter-name>MyFilter</filter-name>  
    <url-pattern>/book/\*</url-pattern>  
</filter-mapping>

<filter-mapping>标记是有先后顺序的，它的声明顺序说明容器是如何形成过滤器链的。过滤器应当设计为在部署时很容易配置的形式。通过认真计划和使用初始化参数，可以得到复用性很高的过滤器。 过滤器逻辑与Servlet逻辑不同，它不依赖于任何用户状态信息，因为一个过滤器实例可能同时处理多个完全不同的请求。

4.过滤指定的servlet

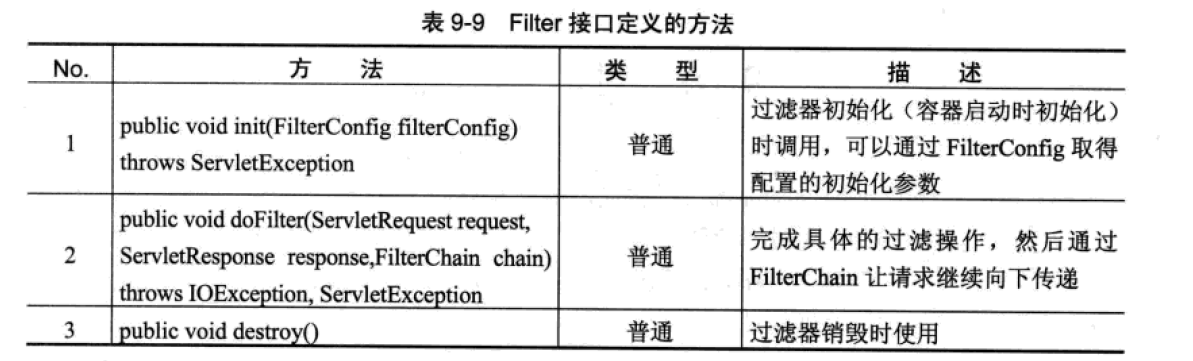
<filter>  
    <filter-name>loggerfilter</filter-name>  
    <filter-class>myfilter.LoggerFilter</filter-class>  
</filter>  
<filter-mapping>  
    <filter-name>loggerfilter</filter-name>  
   **<servlet-name>loggerservlet</servlet-name>**</filter-mapping>  
<servlet>  
    <servlet-name>**loggerservlet**</servlet-name>  
    <servlet-class>myfilter.LoggerServlet</servlet-class>  
</servlet>

5.过滤指定文件

<filter>  
    <filter-name>loggerfilter</filter-name>  
    <filter-class>myfilter.LoggerFilter</filter-class>  
</filter>  
<filter-mapping>  
    <filter-name>loggerfilter</filter-name>  
    <url-pattern>/simplefilter.html</url-pattern>  
</filter-mapping>

以上都要注意是否有斜杠“/”

Filter类似于IO中的过滤流，实现也类似于Servlet。



## ．Servlet过滤器开发步骤

①编写实现Filter接口的Servlet类。

②在web.xml中配置Filter。

开发一个过滤器需要实现Filter接口，Filter接口定义了以下方法：

①destory（）由Web容器调用，初始化此Filter。

②init（FilterConfig filterConfig）由Web容器调用，初始化此Filter。

③doFilter（ServletRequest request,ServletResponse response,FilterChain chain）具体过滤处理代码。

1. **过滤器的生命周期**

过滤器的生命周期：（一定要实现javax.servlet包的Filter接口的三个方法init()、doFilter()、destroy()，空实现也行）   
(1)、启动服务器时加载过滤器的实例，并调用init()方法来初始化实例；   
(2)、每一次请求时都只调用方法doFilter()进行处理；   
(3)、停止服务器时调用destroy()方法，销毁实例。

案例：在三个方法中编写打印语句，然后启动、关闭服务器演示生命周期。

1. **Servlet过滤器API**

Servlet过滤器API包含了3个接口，它们都在javax.servlet包中，分别是Filter接口、FilterChain接口和FilterConfig接口。

**1  public Interface Filter**  
    所有的过滤器都必须实现Filter接口。该接口定义了init,doFilter0，destory()三个方法：  
    (1) public void init (FilterConfig filterConfig) throws   ServletException．  
    当开始使用servlet过滤器服务时，Web容器调用此方法一次，为服务准备过滤器；然后在需要使用过滤器的时候调用doFilter()，传送给此方法的FilterConfig对象，包含servlet过滤器的初始化参数。  
    (2)public void doFilter(ServletRequest request，ServletResponse response，FilterChain chain) throws java.io.IOException，ServletException．

每个过滤器都接受当前的请求和响应，且FilterChain过滤器链中的过滤器（应该都是符合条件的）都会被执行。doFilter方 法中，过滤器可以对请求和响应做它想做的一切，通过调用他们的方法收集数据，或者给对象添加新的行为。过滤器通过传送至此方法的FilterChain参数，调用chain．doFilterO将控制权传送给下一个过滤器。当这个调用返回后，过滤器可以在它的 Filter方法的最后对响应做些其他的工作。如果过滤器想要终止请求的处理或得到对响应的完全控制，则可以不调用下一个过滤器，而将其重定向至其它一些页面。当链中的最后一个过滤器调用chain．doFilterO方法时，将运行最初请求的Servlet。

(3)public void destroy()  
     一旦doFilterO方法里的所有线程退出或已超时，容器调用此方法。服务器调用destoryO以指出过滤器已结束服务，用于释放过滤器占用的资源。

**2 public interface FilterChain**

public void doFilter(ServletRequest request，ServletResponse response)

thlows java．io．IOException，ServletException

此方法是由Servlet容器提供给开发者的，用于对资源请求过滤链的依次调用，通过FilterChain调用过滤链中的下一个过滤   器，如果是最后一个过滤器，则下一个就调用目标资源。

**3 public interface FilterConfig**

     FilterConfig接口检索过滤器名、初始化参数以及活动的Servlet上下文。该接口提供了以下4个方法：

     (1)public java．1ang．String getFilterName0  
           返回web．xml部署文件中定义的该过滤器的名称。  
     (2)public ServletContext getServletContextO  
          返回调用者所处的servlet上下文。  
     (3)public java.1ang.String getlnitParameter(java.1ang.String name)  
          返回过滤器初始化参数值的字符串形式，当参数不存在时，返回nul1．name是初始化参数名。  
     (4)public java.util.Enumeration getlnitParameterNames()  
          以Enumeration形式返回过滤器所有初始化参数值，如果没有初始化参数，返回为空。

**过滤器相关接口工作流程**

从编程的角度看，过滤器类将实现Filter接口，然后使用这个过滤器类中的FilterChain和FilterConfig接口。该过滤器类的

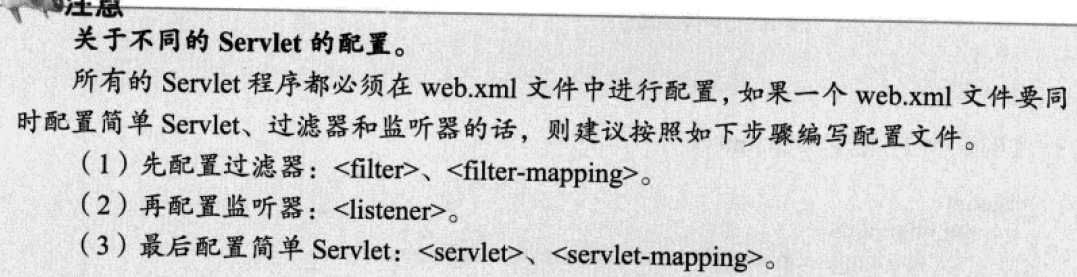
— 个引用将传递给FilterChain对象，以允许过滤器把控制权传递给链中的下一个资源。FilterConfig对象将由容器提供给过滤  
器，以允许访问该过滤器的初始化数据。详细流程如下图所示:

**过滤器配置**

   过滤器通过Web应用程序中的配置描述符web.xml文件中的明，包括部分：

过滤器定义，由<filter>元素表示，主要包括<filter-name>和<filter-class>两个必须的子元素和<icon>、<init-param>,<display-name>,<description>这4个可选的子元素。<filter-name>子元素定义了—个过滤器的名字，<filter-class>指定了由容器载入的实际类，<init-param>子元素为过滤器提供初始化参数。

<filter-mapping> 主要由<filter-name>,<servlet-name>和<url-pattem>子元素组成。<servlet-name>将过滤器映射到一个或多个Servlet上，<url-pattem>将过滤器映射到—个或多个任意特征的URL的JSP页面。



**一个简单过滤器的实现：**

编写Filter过滤器：

**public** **class** LoginFilter **implements** Filter {

@Override

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

System.out.println("init LoginFilter");

}

@Override

**public** **void** doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) **throws** IOException, ServletException {

//把ServletRequest和ServletResponse转换成真正的类型

HttpServletRequest req = (HttpServletRequest)request;

HttpSession session = req.getSession();

//由于web.xml中设置Filter过滤全部请求，可以排除不需要过滤的url

String requestURI = req.getRequestURI();

**if**(requestURI.endsWith("login.jsp")){

chain.doFilter(request, response);

**return**;

}

//判断用户是否登录，进行页面的处理

**if**(null == session.getAttribute("user")){

//未登录用户，重定向到登录页面

((HttpServletResponse)response).sendRedirect("login.jsp");

**return**;

} **else** {

//已登录用户，允许访问

chain.doFilter(request, response);

}

}

@Override

**public** **void** destroy() {

System.out.println("destroy!!!");

}

}

在web.xml中配置Filter：

<filter>

<filter-name>LoginFilter</filter-name>

<filter-**class**>com.itzhai.login.LoginFilter</filter-**class**>

</filter>

<filter-mapping>

<filter-name>LoginFilter</filter-name>

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

</filter-mapping>

注意：一般Filter配置在所有的Servlet之前。

**4、过滤敏感词汇的Filter简单实现：**

@Override

**public** **void** doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) **throws** IOException, ServletException {

//转换成实例的请求和响应对象

HttpServletRequest req = (HttpServletRequest)request;

HttpServletResponse resp = (HttpServletResponse)response;

//获取评论并屏蔽关键字

String comment = req.getParameter("comment");

comment = comment.replace("A", "\*\*\*");

//重新设置参数

req.setAttribute("comment", comment);

//继续执行

chain.doFilter(request, response);

}

**5、Filter的执行顺序**

Filter的执行顺序与在web.xml配置文件中的配置顺序一致，一般把Filter配置在所有的Servlet之前。

# 请求转发：requestDispatcher接口：

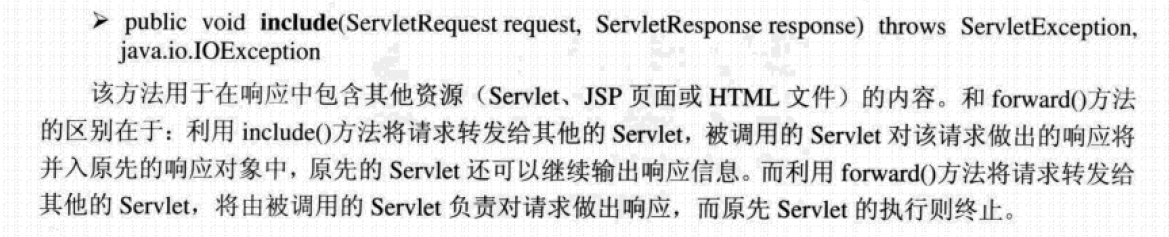
requestDispatcher接口由servlet容器创建封装了一个由路径标识的服务器资源，利用requestDispatcher对象可以将请求转发给其他的servlet或者jsp

request接口中的两个方法是：

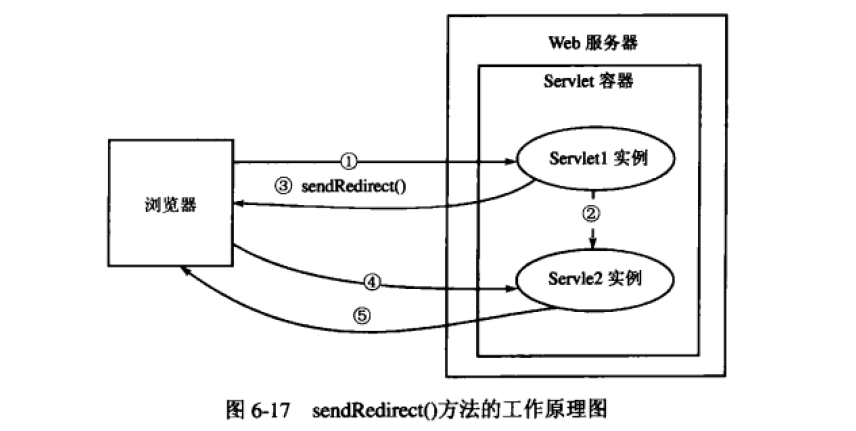
## requestDispatcher.forward(request,response);

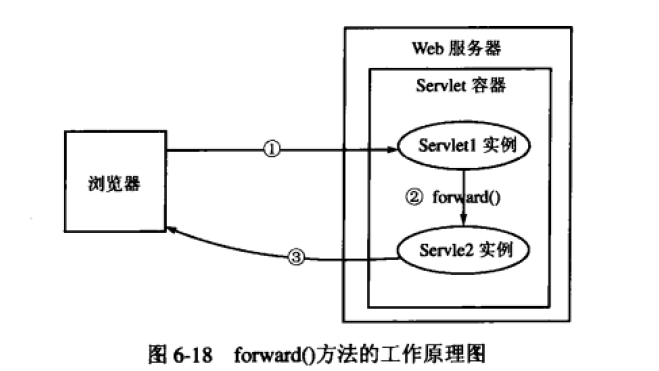
public void forward(ServletRequest request,Servletresponse response)throws ServletException,java.io.IOException

这个方法必须在请求传递给客户端之前调用，而且在forward方法调用之后（下面的内容会被清除）

RequestDispatcher.include();

## sendRedice和requestDispatcher不同：





# 会话跟踪：

会话跟踪

2012-08-03 00:27:14| 分类： 总结 | 标签：技术 |字号 订阅

1. 什么是会话

客户端打开与服务器的连接发出请求到服务器响应客户端请求的全过程称之为会话

2. 什么是会话跟踪

对同一个用户对服务器的连续的请求和接受响应的监视

3. 为什么需要会话跟踪

浏览器与服务器之间的通信是通过HTTP协议进行通信的，而HTTP协议是”无状态”的协议，它不能保存客户的信息，即一次响应完成之后连接就断开了，下一次的请求需要重新连接，这样就需要判断是否是同一个用户，所以才应会话跟踪技术来实现这种要求

4. 介绍

当服务器响应客户端的第一次请求时，将会创建一个新的session对象(该对象实现了HttpSession接口)和一个唯一的ID分配给该请求，以后客户将此会话ID与请求一起传给服务器，此会话ID在后续的请求中会将用户与session对象进行匹配，用于识别不同的客户。

5. 会话跟踪常用的方法:

a) URL重写:URL(统一资源定位符)是Web上特定页面的地址，URL重写的技术就是在URL结尾添加一个附加数据以标识该会话,把会话ID通过URL的信息传递过去，以便在服务器端进行识别不同的用户

b) 隐藏表单域:将会话ID添加到HTML表单元素中提交到服务器,此表单元素并不在客户端显示

c) Cookie:Cookie是Web服务器发送给客户端的一小段信息，客户端请求时可以读取该信息发送到服务器端，进而进行用户的识别。对于客户端的每次请求，服务器都会将Cookie发送到客户端,在客户端可以进行保存,以便下次使用。

客户端可以采用两种方式来保存这个Cookie对象，一种方式是 保存在 客户端内存中，称为临时Cookie，浏览器关闭后 这个Cookie对象将消失。另外一种方式是保存在 客户机的磁盘上，称为永久Cookie。以后客户端只要访问该网站，就会将这个Cookie再次发送到服务器上，前提是 这个Cookie在有效期内。 这样就实现了对客户的跟踪。

Cookie是可以被禁止的。

d) session:

每一个用户都有一个不同的session，各个用户之间是不能共享的，是每个用户所独享的，在session中可以存放信息。

在服务器端会创建一个session对象，产生一个sessionID来标识这个session对象，然后将这个sessionID放入到Cookie中发送到客户端，下一次访问时，sessionID会发送到服务器，在服务器端进行识别不同的用户

Session是依赖Cookie的，如果Cookie被禁用，那么session也将失效

创建Session对象

HttpSession session=request.getSession();

HttpSession session=request.getSession(true);

上面二种创建session的方法效果是一样的。首先获取传递进来的在Cookie中存放的sessionID，再根据sessionID查找session对象，如果没有找到将创建一个新的session对象，并保存到Cookie中发送到客户端。如果找到了，就将session对象赋予给引用

HttpSession session=request.getSession(false);

这种方式创建过程: 首先获取传递进来的在Cookie中存放的sessionID，再根据sessionID查找session对象，如果没有找到将返回null，如果找到了，就将session对象赋予给引用

**一，会话跟踪(session tracking)技术** 会话是客户端发送请求，服务器返回响应的连接时间段。  
 HTTP是无状态协议：每次都是单独连接，不能维持客户的上下文信息。  
 会话跟踪技术是用于维持客户端和服务器端通信信息的技术。  
   
 三种典型客户端会话跟踪解决方案：  
  1，Cookie；  
  2，URL重写；  
  3，隐藏表单域；

**二，会话Cookie** 用于会话跟踪的Cookie叫做会话Cookie。Servlet规范中会话跟踪的cookie名字必须是JSESSIONID,保存在浏览器的内存中。区别于保存在外部存储设备的cookie。   
 对于存储在内存中的cookie，是不能被不同的浏览器进程共享，共享只能发生在同一个浏览器进程的不同窗口（对个窗口共享一个进程）中。对于存储在外部设备的cookie，可以多浏览器共享。

**三，url重写** 在浏览器不支持或禁用cookie的情况下，使用url重写代替cookie来跟踪用户会话。  
 每个页面都必须使用servlet或jsp动态生成(动态页面)。因为附加在URL上的sessionID是动态产生，所以对于静态页面的跳转，URL重写机制无能为力。  
 即使使用动态页面，如果用户离开了会话并且通过书签或链接再次回来，会话信息也会丢失，因为存储下来的链接含有错误的标识信息。  
   
 必须将所有发送到客户端的url进行编码，调用HttpServletResponse接口中的encodeURL()方法和encodeRedirectURL()方法来实现。在调用sendRedirect()方法之前使用encodeRedirectURL()方法.  
   
**四，隐藏表单域**  
 只能用于特定操作中。仅当每个页面都由表单提交而动态生成的时候，才可以使用隐藏表单域，用来存储相关会话信息。

**五，servlet中的会话跟踪** javax.servlet.http.HttpSession接口是servlet提供会话跟踪解决方案。  
 HttpSession对象存放在服务器端，只是对cookie和url重写技术的封装应用，所以要求服务器支持cookie，可以全局切换到url重写。

 1，访问与当前请求相关联的会话对象：getSession(boolean value)方法

  在会话过程中，web容器负责为客户端维护一个唯一的HttpSession对象。  
  对于新会话，web容器创建使用实现了HttpSession接口的HttpSession对象封装当前请求会话的信息（散列表的方式存储）。  
    
  使用HttpServletRequest的getSession()方法访问HttpSession对象。  
   在后台，系统从cookie或URL重写附加的数据中提取出用户ID。以ID为key，遍历之前创建的HttpSession对象内建的散列表。  
   如果找不到匹配的会话ID,系统重新创建一个新的会话。默认情况下(不禁用Cookie)还会创建一个名为JSESSIONID，值为唯一标识用户表示会话ID的输出cookie。    
  因为调用getSession()方法会影响到后面的响应，所以只能在发送任何文档内容到客户端之前调用getSession()方法。

  区别：getSession(boolean value)方法：如果系统没有找到与请求关联的会话ID，true表示返回新会话。false表示方法返回null。

 2，访问和设置与会话相关联信息，维护会话的状态：HttpSession的getAttribute（）方法和setAttribute（）方法

  散列表：HttpSession对象内建数据结构，用于存储 当前请求会话的数据(会话的属性)。可以存储任意数量的key-value对。  
  使用HttpSession的getAttribute（）方法和setAttribute（String key，Object value）方法读取和设置当前请求会话数据(即对散列表的操作)，维护会话的状态。  
  setAttribute方法会替换任何之前的属性。如果不想被替换，则需要在设置之前使用removeArrtibute（String key）方法移除该属性。  
    
  setAttribute方法会触发所有实现了HttpSessionBindingListener接口的valueBound方法,做一些初始化状态的操作。  
  removeArrtibute方法会触发所有实现了HttpSessionBindingListener接口的valueUnbound方法，做一些消除状态的操作。

  对于分布式WEB应用程序，将web应用程序标记为可分布式执行，系统需要能够将会话对象在机器之间传递，这时需要将会话的属性实现Serializable接口。

 3，废弃会话数据  
  只移除自己编写的的servlet创建的数据：removeArrtibute(String key)方法  
  （web应用程序中删除）删除整个会话：invalidate()方法，可以用该方法注销用户。  
  （Web服务器中删除）将用户从系统中注销并且删除所有与该会话关联的会话：logout()方法。一定要与其他web应用程序协调loggout命令的使用。  
    
 4,会话超时时间间隔  
  getMaxInactiveInterval()方法和setMaxInactiveInterval()方法读取和设置在没有访问的情况下，会话保存的最长时间。秒为单位。负数表示会话从不超时。超时由服务器来维护。

 5，会话最后次被客户端访问的时间：getLastAccessedTime()方法  
  可以用来确定客户端在二次请求之间会话的非活动时间。

 6，获取会话被创建的时间：getCreationTime()方法 返回long类型数据

 7，返回分配给session的唯一标识ID,为字符串。getId()方法

**六，浏览器会话与服务器会话** 浏览器会话  
  默认情况下，会话跟踪基于存储在浏览器内存中的cookie，区别于存储在外围存储设备上的cookie。  
  需要servlet显示的读取JSESSION cookie，设置最大时效和路径并添加到客户端，否则退出浏览器就会中断会话。  
 服务器会话  
  服务器需要将会话保存在内存中，在会话处于非活动状态超过设定的间隔（会话超时）就移除会话。

**七，对绑定到session中的对象(属性)初始化和消除状态。HttpSessionBindingListener监听器接口和HttpSessionBindingEvent事件类。**

 Servlet容器通过实现HttpSessionBindingListener监听器接口的监听器来监听HttpSessionBindingEvent事件。  
   
 HttpSessionBindingListener监听器接口方法：  
 1，valueBound(HttpSessionBindingEvent event)：对象被绑定到session中时该方法通知对象。做初始化操作。  
 2，valueUnbound(HttpSessionBindingEvent event): 对象被移除出session时该方法通知对象。做消除状态操作。  
   
 HttpSessionBindingEvent事件方法：  
 1，getName()：获取触发事件的属性的名字。  
 2，getValue()：获取触发事件的属性的值。  
 3，getSession()：返回Session对象。

**八，在 Java Servlet API中有关的监听器接口。**

 与Session相关的监听器接口  
 javax.servlet.http.HttpSessionActivationListener:如果绑定到Session中，当Session被钝化或激活时，Servlet容器将通知该对象。  
 javax.servlet.http.HttpSessionAttributeListener:如果想要在Session中的属性列表发生改变时得到通知，可以实现这个接口。  
 javax.servlet.http.HttpSessionListener:如果需要在Session创建后或者Session无效前得到通知，可以实现这个接口，实现该接口的类必须在web应用程序的部署描述符中进行配置。  
   
 与Servlet上下文有关的监听器接口：  
 javax.servlet.ServletContextAttributeListener:在Servlet上下文中的属性列表发生改变时得到通知，可以实现这个接口，实现该接口的类必须在web应用程序的部署描述符中进行配置。  
 javax.servlet.ServletContextListener:如果需要在Servlet上下文对象初始化时或被销毁的时候得到通知，可以实现这个接口，实现该接口的类必须在web应用程序的部署描述符中进行配置。

 与请求相关的监听器接口：servlet2.4规范定义  
 javax.servlet.ServletRequestAttributeListener:Servlet请求对象中的属性发生改变时得到通知。  
 javax.servlet.ServletRequestListener:请求对象初始化或者被销毁时得到通知。

在servlet的规范中定义了三种会话跟踪：

* + - 1. SSL（secure Socket layer,安全套接字层）会话
      2. Cookies
      3. URL重写

一：Sll（secure Scoket layer，安全套接字层）会话：一种运行在tcp/ip之上

http之下的加密技术，他可以让使用ssl的服务器和使用ssl的客户端进行通信（会产生一种会话密钥的东西）https协议的服务器可以使用这个会话密钥来建立会话

## Cookies

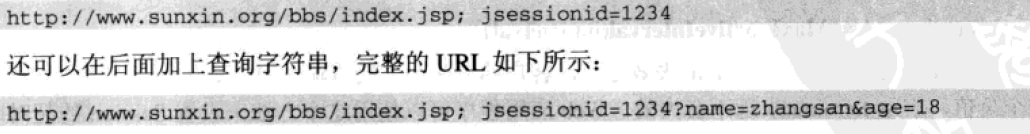
Cookies是利用键-值对的形式来记录会话跟踪的内容的。服务器利用响应报头（set-Cookie）来发送cookie信息：  
set-cookie的内容：

|  |
| --- |
| Set-Cookie: Name=Value;comment=value;domain=value;max-age=value;path=value;secure;version=1\*DIGIT |

Name是cookie的名字value是她的值（必须是键-值对的形式）（在servlet规范中使用的cookie的名字必须是JSESSIONID）

## URL重写：

由于cookie是服务器发送给客户端的在客户端保存，如果客户端禁止kookie可以使用URL重写的方式进行会话跟踪所谓的URL重写就是在URL中加入标识客户的Sessin iD然后servlet容器解析URL取出session id然后根据seesionid 将请求和特定的session id关联



要得到一个session对象可以调用httpservletrequest接口的getSession（）方法