# Servlet简介

## 认识Servlet

使用JSP可以完成动态Web的开发，但是从开发出来的代码中可以发现，一个页面上会存在大量的Java代码，造成编写及维护的困难，而要想让开发出来的页面更加干净、整洁，则可以使用Servlet。

**1）Servlet程序（服务端小程序）是使用Java语言编写的服务器端程序**，可以像JSP一 样完成动态Web的开发。Servlet主要运行在服务器端，由服务器调用执行，**是一 种按照Servlet标准开发的类。**

1. Servlet程序是Java对CGI程序的实现，但与传统的CGI的多进程处理操作相比， Servlet采用了多线程的处理方式，使得Servlet程序的运行效率比传统的CGI更高；

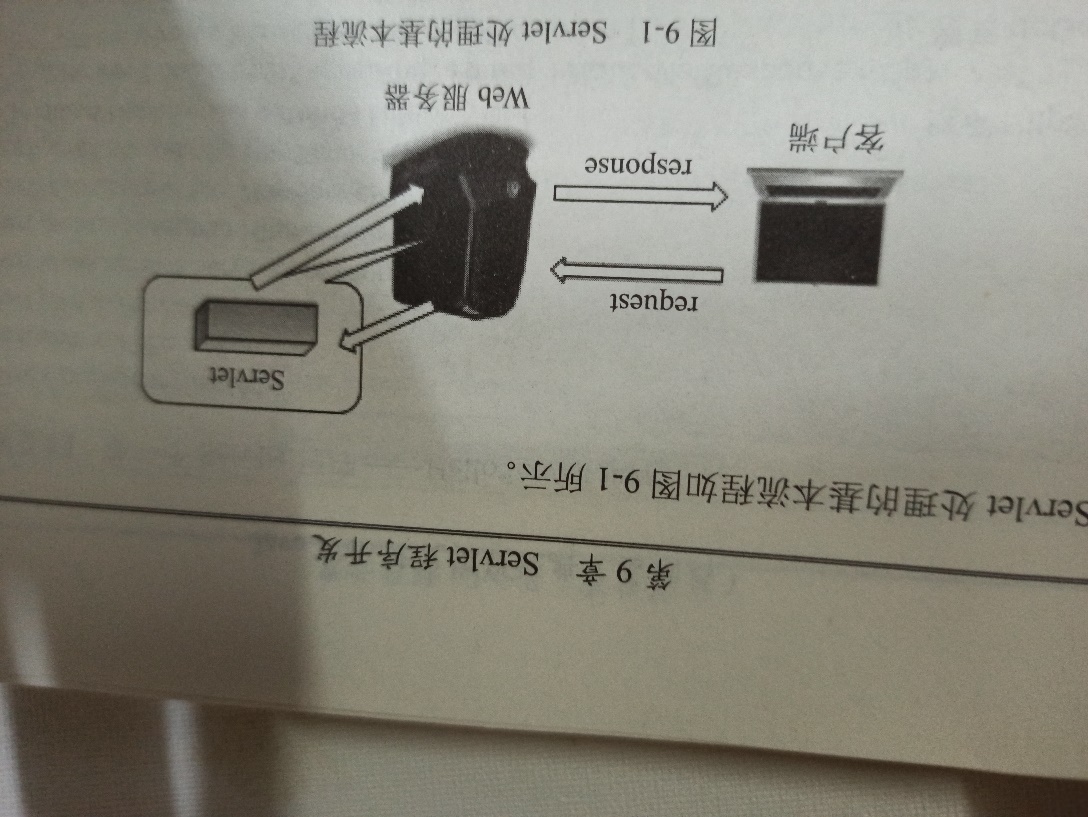
而且Servlet还拥有Java的可移植性的特点，使得Servlet更易使用，功能也更加强

大。

3）Servlet的最大好处是它可以处理客户端发送过来的HTTP请求，并返回一个响应。

## Servlet基本处理流程

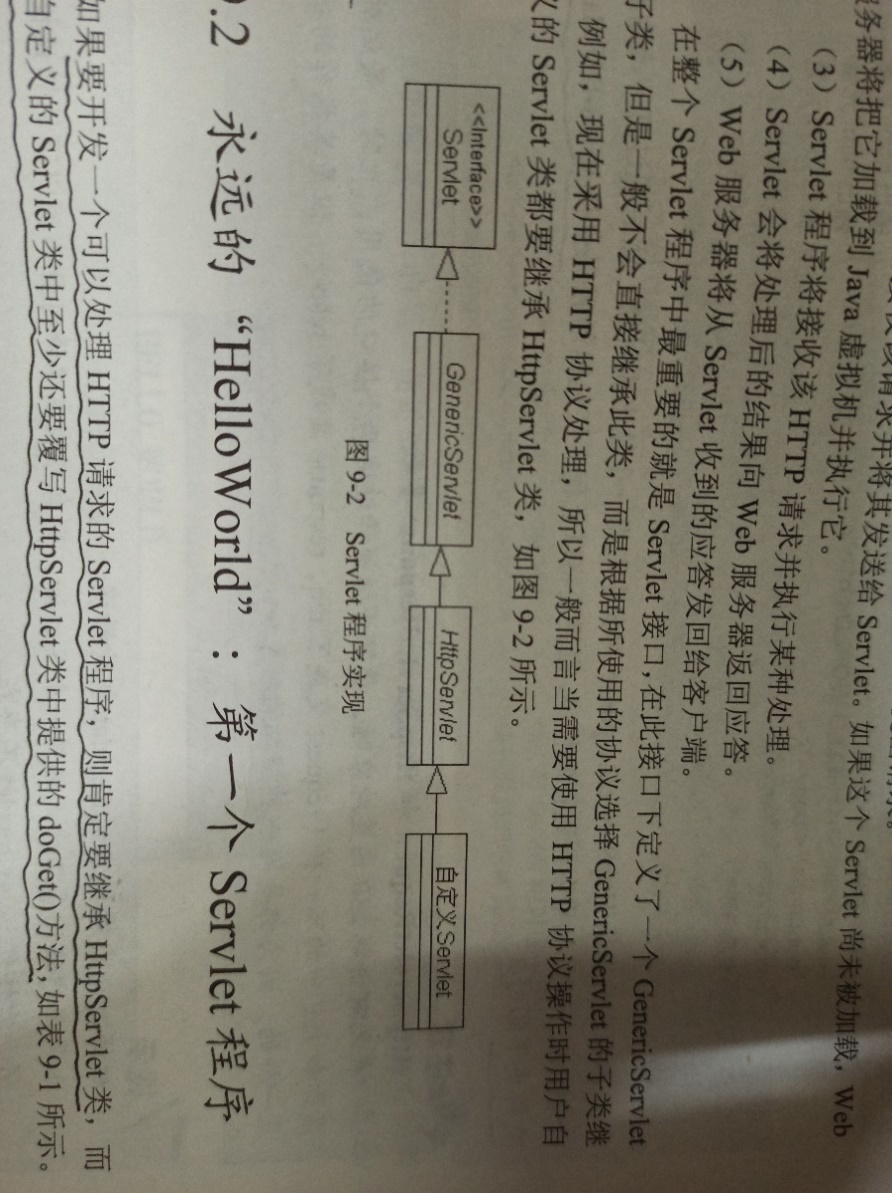
Servlet的最大好处是它可以处理客户端发送过来的HTTP请求，并返回一个响应。Servlet处理的基本流程如下：



1. 客户端（很可能是Web浏览器）通过HTTP发出请求。
2. Web服务器接收该请求并将其发送给Servlet。如果这个Servlet尚未被加载，

Web浏览器将把它加载到Java虚拟并执行它。

1. Servlet程序将接收该HTTP请求并执行某种处理。
2. Servlet会将处理后的结果向Web服务器返回应答。
3. Web服务器将从Servlet收到的应答发回给客户端。



## Servlet接口的继承结构

在Servlet程序中最重要的就是Servlet接口，在此接口下定义了一个GenericServlet的子类，但是一般不会直接继承此类，而是根据所使用的协议选择GenericServlet的子类继承。例如，现在Web采用的HTTP协议处理，所以一般而言当需要使用HTTP协议操作时用户定义的Servlet类都要继承HttpServlet类。如下图：

# 第一个Servlet程序

实现一个Servlet需要分为两步：定义Servlet、在web.xml文件中配置Servlet对其进行映射。

## 定义Servlet

开发一个可以处理HTTP请求的Servlet程序，需要继承HttpServlet类，并且在开发的Servlet类中至少要覆写HttpServlet类中的doGet( )方法（因为这样这个Servlet才有意义）其他方法可以根据需要进行覆写。doGe( )方法在HttpServlet类中的定义如下：

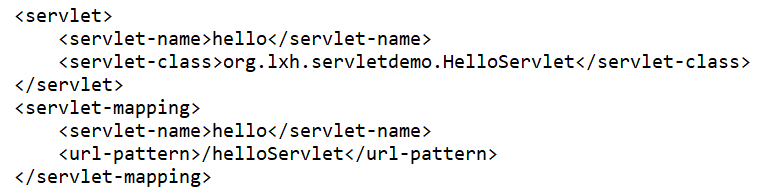
**protected void doGet( HttpServletRequest request,HttpServletResponse response ) throws ServletException , IOException**

Servlet程序本身也是按照请求和应答的方式进行的，所以在doGet( )方法中定义了两个参数，即HttpServletRequest对象和HttpServletResponse对象，用来接收和回应用户的请求。

除了可以可以覆写doGet( )方法外，还可以覆写doPost、doTrace( )、doDelete( )等方法（可以从HttpServlet类中查看），但是从实际开发来看，最常用的就是doGet( )和doPost( )两种方法。

## 配置Servlet

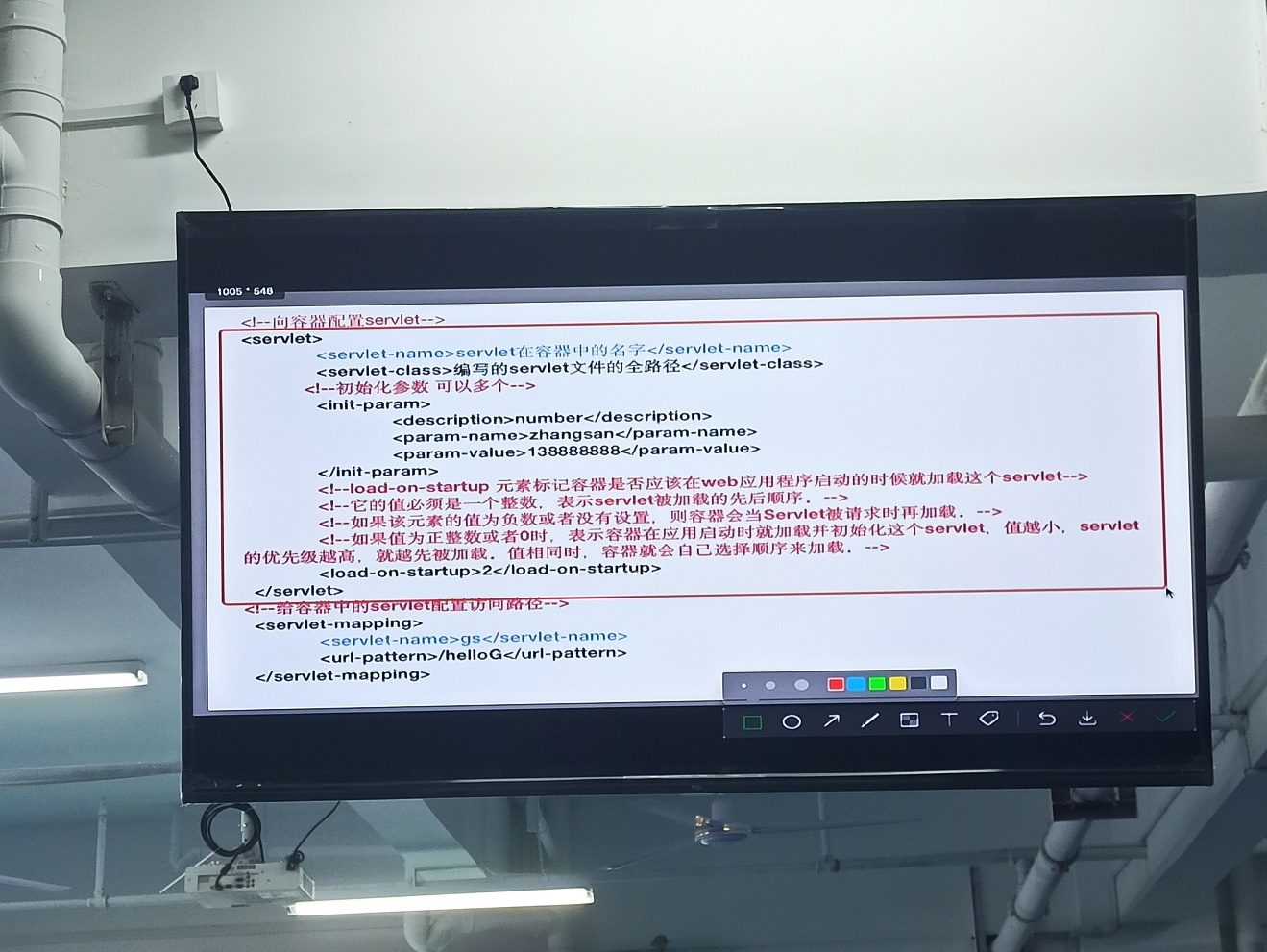
Servlet程序编写完成后，将其打包编译后保存在WEB-INF/classes文件夹中。但一个Servlet程序编译完成后，实际上是无法立即访问的，因为所有的Servlet程序都是以\*.class的形式存在的，所以还必须该项目的WEB-INF\web.xml文件中配置Servlet，对Servlet进行映射（配置完成后记得重启服务器）：



上面的配置表示的是，通过/helloSevlet路径即可访问对应的Servlet；

一个Servlet还可以配置多个访问路径，只需要增加相应的<url-pattern>节点即可，或者再添加<servlet-mapping>元素；

另外还可以通过<init-param>节点给一个Servlet配置初始化参数。



# Servlet与表单

由于Servlet中存在HttpServletRequest和HttpServletResponse对象的声明，所以可以使用Servlet接收用户所提交的内容。

1. 在进行表单提交时，也可以将表单提交的路径（action属性）设置为一个Servlet 程序路径，并根据表单的提交方式调用Servlet程序中的处理方法。例如一个表 单 提交给DemoServlet这个Servlet程序，并且提交的方式post，则提交表单后会自动

调用DemoServlet类中doPost( )方法。

1. 在将表单内容提交给Servlet程序时，需要特别注意映射路径的配置，否则会出现4

04错误。

1. 地址提交属于get提交方式，即如果直接通过浏览器输入地址的方式访问一个Servlet

程序，对于服务器来讲就相当于客户端发出了一个get请求，会自动调用该Servlet程序中的doGet( )方法。因为地址提交属于get提交方式。

（具体示例请参考教材书）

# Servlet生命周期

## 认识Servlet生命周期

Servlet程序是运行在服务器端的一段Java程序，其生命周期将受到Web容器的控制，Servlet的生命周期包括加载、初始化、服务、销毁、卸载5个部分。对于Servlet的各个生命周期，都可以在HttpServlet和GenericServlet类中发现对应的方法。如下：

|  |  |
| --- | --- |
| 方法 | 描述 |
| public void init( ) throws ServletException | Servlet初始化时调用 |
| public void init( ServletConfig config )  throws ServletException | Servlet初始化时调用，可以通过ServletConfig读取配置信息 |
| public abstract void service(ServletRequest request , ServletResponse response ) throws ServletException , IOException | Servlet服务，一般不会直接覆写此方法，而是使用doGet( )或doPost( )方法 |
| publib void destroy( ) | Servlet销毁时调用 |

Servlet各个生命周期的作用如下：

1. **加载Servlet**

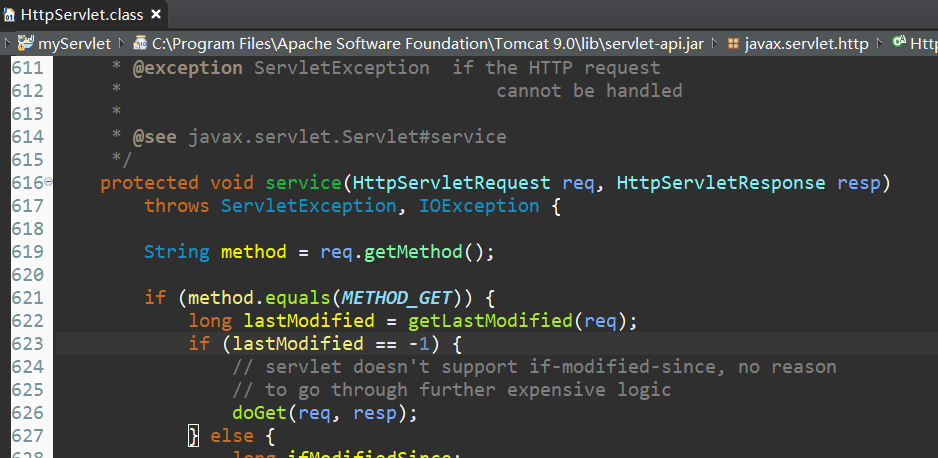
Web容器负责加载Servlet，当Web容器启动时或者是在第一次使用这个Servlet时，容器会负责创建Servlet实例，但是用户必须通过描述部署符web.xml指定Servlet的位置（Servlet所在的包.类名称），成功加载后，Web容器会通过反射的方式对Servlet进行实例化。

1. **初始化**

当一个Servle被加载实例化后，容器将调用init( )方法初始化这个对象，初始化的目的是为了让Servlet对象在处理客户端请求前完成一些初始化的工作，如建立数据库连接、读取资源文件信息等，如果初始化失败，则此Servlet将直接被卸载。

1. **处理服务**

当有请求提交时，Servlet将调用service( )方法进行处理，在service( )方法中，Servlet可以通过ServletRequest接收客户端的请求，也可以利用ServletReponse设置响应信息，在该方法中判断请求方式的类型，然后调用对应的doXXX( )方法。一般不覆写此方法，而是直接使用doGet( )或doPost( )方法。HttpServlet中service( )方法部分源码如下：



可以看出，在service( )方法中判断当前请求方式然后根据请求方式调用对应的doXXX( )方法。

1. **销毁**

当Web容器关闭或者检测到一个Servlet要从容器中被删除时，会自动调用destory( )方法，以便让该示例释放所占用的资源。

1. **卸载**

当一个Servlet调用完destory( )方法后，此实例将等待被垃圾收集器所回收，如果需要再次使用此Servlet时，会重新调用init( )方法初始化。

在正常情况下，Servlet只会初始化一次，而处理服务器会调用多次。但是如果一个Servlet长时间不使用的话，也会被容器自动销毁，而如果需要再次使用时会重新进行初始化的操作，即在特殊情况下初始化可能会进行多次，销毁也可能进行多次。

## 观察Servlet生命周期

编写程序，可以观察到Servlet的生命周期。

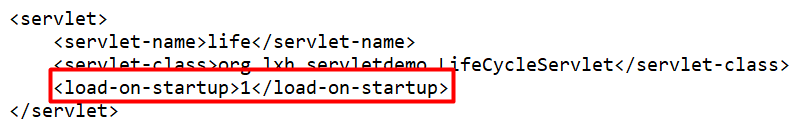


（完整示例请参考教材书）

1. 如果容器配置了动态加载（reloadable=”true”），每当服务器重新加载后，实际上原先的Servlet也会销毁。
2. 如果Servlet中覆写了service( )方法，则对应的doGet( )或doPost( )方法就不再起作用，而是直接使用servie( )方法进行处理。因为在HttpServlet类中已经将service( )方法覆写，方法的主要功能就是区分不同的请求类型，如果是get请求则自动调用doGet( )，如果是post请求则自动调用doPost( )（实际上Servlet本身就是Java模板设计模式的应用）。
3. 编写Servlet时用到最多的是doGet( )和doPost( )方法。当定义一个Servlet时可以不用将全部的声明周期方法写出，从实际开发来看，doGet( )和doPost( )两个方法的使用几率最高。

## Servlet自动加载

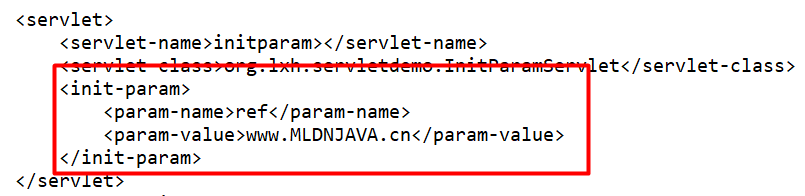
前面在讲Servlet生命周期时已经讲过，Servlet会在Web容器启动时或者在第一次使用这个Servlet时进行初始化操作（调用init( )方法），但实际上Servlet默认的是第一次使用时才会进行初始化操作，可以通过配置web.xml文件，让容器启动时就自动为Servlet初始化，只需要直接配置启动选项即可。



如果多个Servlet配置了启动选项，值越小的越先被容器加载，值相同则由容器进行选择。

# 取得初始化配置信息

每一个Servlet可以同时配置多个初始化参数，配置参数时直接使用<init-param>元素即可，其中的<param-name>表示参数名称，<param-value>表示参数的内容，Servlet读取时通过参数名称即可取得参数的内容。



在讲解JSP内置对象时讲解过config对象，通过该对象可以取得web.xml中配置的初始化参数，并且该对象是ServletConfig接口的实例，而刚好Servlet生命周期中有一个初始化方法可以取得config对象，由此可以取得该Servlet配置的初始化参数。



在Servlet中初始化方法一共有init( )和init(ServletConfig config)两个，如果同时覆写了两个方法，则调用的是init( ServletConfig config )方法。

# 取得其他内置对象

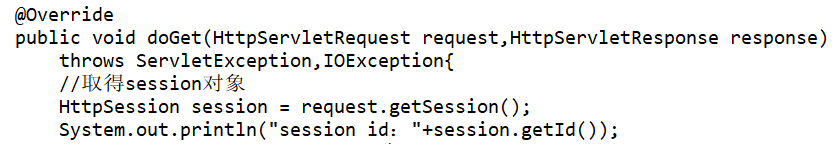
通过前面的学习发现，在Servlet中可以取得request、response、config对象，实际也可以取得session和application内置对象。

1. **取得session内置对象**

在Servlet程序中要想取得session对象，则可以通过HttpServletRequest接口（request内置对象）完成，在此接口中提供了以下操作方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| public HttpSession getSession( ) | 返回当前的session |
| public HttpSession getSession( boolean create ) | 返回当前的session，如果没有则创建一个新的session对象返回 |

示例关键代码如下：



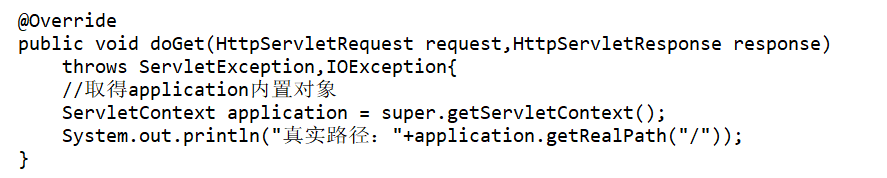
1. **取得ServletContext实例**

application对象时ServletContext接口的实例，表示的是Servlet上下文。如果要在一个Servlet中使用此对象，直接通过父类GenericServlet类中提供的方法即可：

**public ServletContext getServletContext( )**

通过调用此方法即可取得，因为GenericServlet是HttpServlet的父类，而HttpServlet类是Servlet的父类，所以直接通过super.getSerlvetContext( )即可获得application对象。

示例关键代码如下：



# Servlet跳转

从一个JSP或者是一个HTML页面可以通过表单或超链接跳转进Servlet，那么从Servlet也可以跳转到其他Servlet、JSP或其他页面。

1. **客户端跳转（地址栏会发生改变）**

在Servlet中如果想要进行客户端跳转，直接使用HttpServletResponse接口中（response对象）的sendRedirect( )方法即可。但是需要注意的是，客户端跳转只能传递session范围或application范围的属性，而无法传递request范围的属性。 （执行跳转语句后，后面的语句依然后执行）

示例关键代码如下：



**2、服务器端跳转（地址栏不会发生改变）**

在Servlet中没有像JSP中的<jsp:forward/>指令，所以，如果想要执行服务器端跳转，就必须依靠javax.servlet.RequestDispathcer接口完成，此接口中提供以下方法：

|  |
| --- |
| public void forward ( ServletRequet request , ServletResponse response ) throws ServletException , IOException |

但要想获取RequestDispathcer实例还需要使用ServletRequest接口中提供的以下方法进行实例化：

|  |
| --- |
| public RequestDispathcer getRequestDispathcer ( String path ) |

参数path用于指定跳转路径。（执行跳转语句后，后面的语句依然会执行）

示例关键代码如下：



# Web开发模式

在实际的Web开发中，主要有两种开发模式：模式一（Model1）和模式二（Model2）。

## Model1（模式一）

Model1是指在开发中将显示层、控制层、数据层的操作同一交给JSP或JavaBean类进行处理。Model1的处理情况分为两种：**一种是完全使用JSP进行开发，另外一种是使用JSP+JavaBean的模式进行开发。**

1. **完全使用JSP进行开发**

完全使用JSP开发，将显示层、控制层、数据层操作都写在JSP页面中。

优点：开发速度快。不需要编写额外的JavaBean或Servlet，只需要专注开发 JSP页面；小幅度修改代码较为方便。直接修改JSP后，再交给Web容器重 新编译执行即可，不用像写JavaBean或Servlet先将Java源文件编译成类 文件。

缺点：程序可读性低，代码繁杂，难以维护，可重复利用性低。

1. **使用JSP+JavaBean模式进行开发**

使用JSP和JavaBean进行开发，将显示操作写入JSP页面，业务层都写成JavaBean的形式，JavaBean负责大部分的数据处理，包括DAO操作等，再将处理后的结果返回到JSP页面进行显示。

优点：程序可读性较高，后期维护较为轻松，JavaBean组件可重复使用，

可重用性高。

缺点：没有流程控制，每一个JSP页面都需要检查请求的参数是否正确、

条件判断、异常发生时的处理；并且所有的显示操作都与具体的业务代 码耦合在一起，日后维护困难。

总得来说，Model1模式适合小型Web项目或复杂度较低的Web项目开发，优点是开 发速度较快，缺点是维护代价大。考虑到代码的维护在开发中的地位，所以使用最多的 还是Model2，也就是MVC设计模式。

## Model2（模式二）

在Model2（MVC设计模式）中所有的开发都是以Servlet为主体展开的，由Servlet接收所有的客户端请求，然后根据Servlet请求调用相应JavaBean进行处理，最后将处理的结果交给JSP完 成显示。

Model2也称为MVC设计模式，它强制地使应用程序的输入、处理和输出分开。MVC设计模式被分为3个核心层，即模型层、显示层和控制层。

1）显示层（View）：主要负责接收Servlet传递的内容，并且可以调用JavaBean，将内 容显示给用户。

2）控制层（Controller）：主要负责所有的用户请求参数，判断请求参数是否合法，根 据请求的类型调用JavaBean执行操作并将最终的处理结构交 给显示层进行显示。

3）模型层（Model）：完成一个独立的业务操作组件，一般都是以JavaBean或EJB的形 式进行定义。

在整个MVC设计模式中，最关键的部分是使用RequestDispatcher接口，因为内容都是 通过此接口完成服务端跳转利用request保存到JSP页面上进行显示的。当有用户请求 提交时，所有请求都会都会交给Servlet进行处理，然后由Servlet调用JavaBean，并将 JavaBean的操作返回的结果通过RequestDispatcher接口传递到JSP页面上。由于这些要 显示的内容只是在一次请求-回应中有效，所以在MVC设计模式中，主要使用request 属性范围进行传递，提升代码性能。

在实际开发中，一个JSP页面最好只包含3种类型的代码：

1. 接收属性：通常使用request接收从Servlet传递过来的属性
2. 判断语句：判断传递到该JSP中的属性是否存在
3. 输出内容：使用迭代或者VO将内容进行输出

在一个JSP页面中最好只导入java.util包，如果可能，什么包都不导入。只有这样，才 可开发出一个简洁、清晰的JSP页面。

## 简单Java类的标准

简单Java类对应着数据库中的表，在DAO设计模式中称为VO，在Hibernate中称为POJO、在最早使用EJB开发时，称为TO，在JPA中称为PO。其开发原则如下：

1. 类名称要和表名保持一致；
2. 为了日后的操作方便，所有的简单Java类必须实现java.io.Serializable接口；
3. 类中属性的类型不可以定义为基本数据类型，只能使用包装类；
4. 类中所有的属性都必须封装，必须编写setter和getter方法；
5. 类中一定要有一个无参构造方法。

# 过滤器

JSP可以完成的功能Servlet都可以完成，但是Servlet具备了很多的功能是JSP所不具备的（JSP是从Servlet中诞生的）。从使用上看Servlet可以分为简单Servlet、过滤Servlet（过滤器）和监听Servlet（监听器）三种。JSP只能完成简单Servlet的一些功能，而对于过滤Servlet和监听Servlet，JSP是无法完成的。

## 过滤器的基本概念

1. 当需要限制用户访问某些资源或者在处理请求时提前进行某些处理，即可使

用过滤器完成。

2）过滤器是以一种组件的形式绑定到Web应用程序当中，与其他Web应用程序组件 不同是，过滤器是采用“链”的方式进行处理的。如下：

客户端

过滤器

过滤器

过滤器

Web资源

可以定义多个过滤器

1. 在没有使用过滤器以前，客户端都是直接请求Web资源的，但是一旦加入了过滤器，所有的请求都是先交给了过滤器处理，然后再访问相应的Web资源，以此可以达到对某些资源的访问限制。

## 实现过滤器

要想实现过滤器，则直接让一个类实现javax.servlet.Filter接口并覆写其中的三个方法即 可，该接口中有如下三个方法：

|  |  |
| --- | --- |
| public void init( FilterConfig filterConfig )  throws ServletException | 过滤器初始化时（容器启动时）调用，可以通过FilterConfig取得该过滤器配置的初始化参数 |
| public void doFilter( ServletRequest requet ,  ServletResponse response , FilterChain chain)  Throws IOException,ServletException | 完成具体的过滤操作，然后通过FilterChain让请求继续向下传递 |
| public void destroy( ) | 过滤器销毁时（容器关闭时）调用 |

在doFilter( )方法中，javax.servlet.FilterChain接口对象的主要作用是将用户的请求向下 传递给其他的过滤器或者Servlet，在FilterChain接口中定义如下方法：

|  |  |
| --- | --- |
| public void doFilter(ServletRequest request ,  ServletResponse response )throws IOException ,  ServletExcepiton | 将请求向下继续传递 |

可以看出在FilterChain接口中也定义了doFileter( )方法，这是因为在一个过滤器后面可 能存在另外一个过滤器，也可以是请求的最终目标（Serlvet），这样就通过FileterChain 形成了一个“过滤链”的操作。

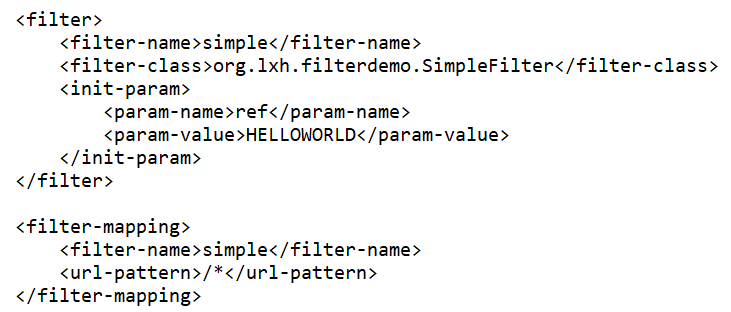
1）过滤器编写完成之后，需要在web.xml文件中进行配置，与配置servlet程序基本类 似，只是标签名不同，在<url-pattern>中配置的表示过滤器要过滤的路径，如果是

“/\*”则表示对于任何路径都需要过滤，包括静态资源；配置过滤器一般都放在 servlet配置之前；配置过滤器时也可以通过<init-param>为其配置初始化参数。

1. 在web.xml文件中配置过滤器时，<url-pattern>用于配置过滤路径，/\*表示过滤所有路径。但是在开发中，有可能只对某一个或某几个目录过滤，此时就可以明确地写出是对哪个目录进行过滤或者是多增加过滤的映射路径（即配置多个<filter-mapping>）。
2. 过滤器中的初始化方法是在容器启动时自动加载的，并且通过FilterConfig的getInitParameter( )方法可以取出配置的初始化参数。

一个完整的过滤器示例如下：





记得观察Tomcat后台打印。

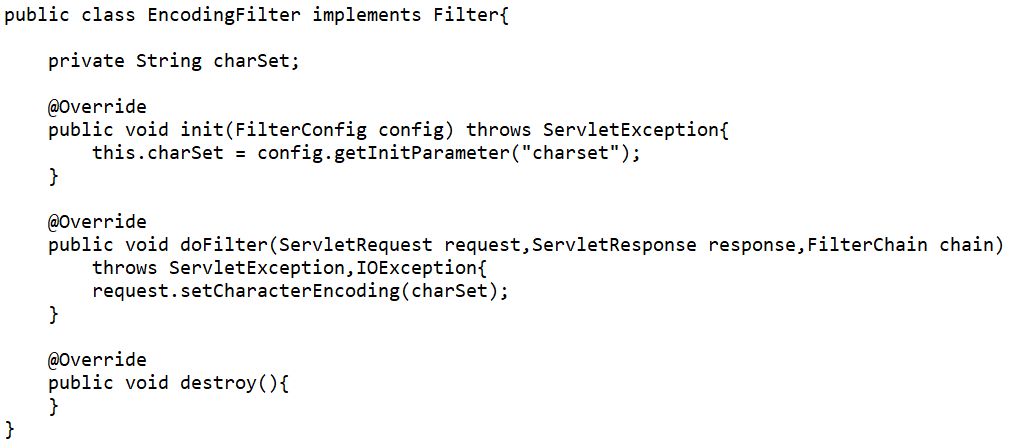
## 过滤器的应用

过滤器本身是以一个组件的形式加入到应用程序之中的。过滤器常用的应用有：使用过 滤器完成编码的过滤操作、用户的登录验证操作。

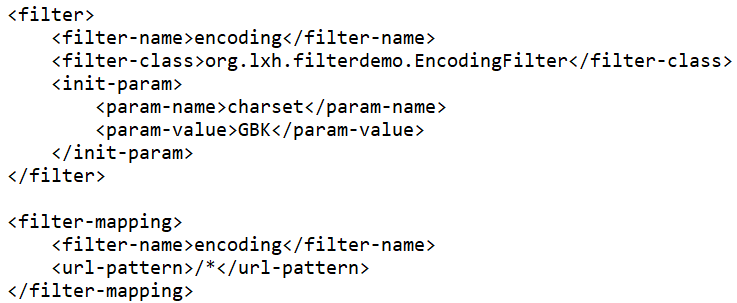
**1、编码过滤器**

在Web开发中，编码过滤是必不可少的操作。为了解决接收请求参数时出现乱码的问 题，在每一个JSP页面都需要使用 “request.setCharacterEncoding( “utf-8 “)”统一编 码，但为每一个JSP页面都添加编码处理的代码肯定是不可取的，会造成大量的代码 重复，那么此时即可通过过滤器完成编码过滤的操作。

·定义编码过滤器：EncodingFilter.java



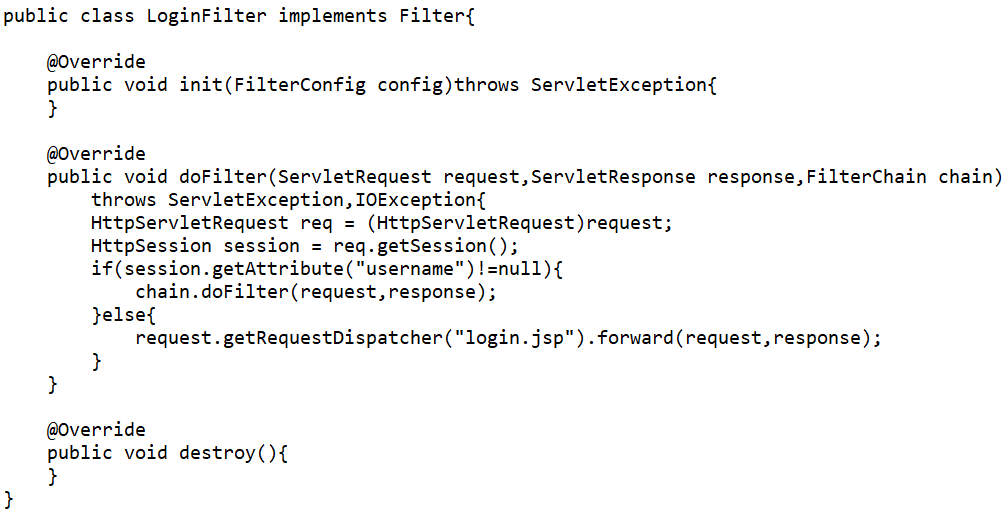
·在web.xml文件中配置编码过滤器：



**2、登录验证过滤器**

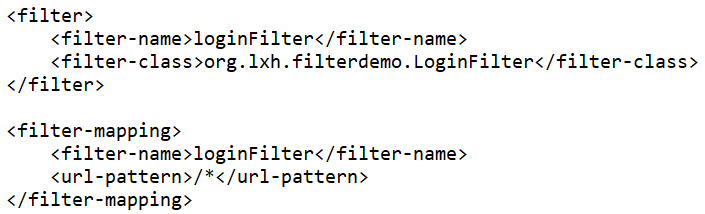
登录验证是所有Web开发中不可缺少的部分，最早的做法是通过验证session是否 包含了登录成功时设置的属性，但是如果每个页面都这样做的话，则肯定会造成大 量的代码重复，而通过过滤器的方式即可避免这种重复的操作。

·定义身份验证过滤器：LoginFilter.java



但需要注意session本身属 于HTTP协议的范畴，但是过滤器中的doFilter( )方法形 参定义的是ServletRequest类型 的对象，那么要想取得session对象，则必须向下 转型，将ServletRequest对象变为 HttpServletRequest接口对象，才能通过 getSession( )方法取得session对象。

·在web.xml文件中配置



# 监听器

1）监听器是Servlet程序的第三种形式，也称为监听Servlet，主要功能是负责监听Web 的各种操作。当监听的事件触发后，将调用相关的方法对此事件进行处理。

1. 在Web中可以对application、session和request这三个内置对象的操作进行监听。
2. 所有的Servlet程序都必须在web.xml文件中进行配置，如果一个web.xml文件要同时配置简单Servlet、过滤器和监听器的话，则建议先配置过滤器、再配置监听器、最后配置简单Servlet。

## 对application监听

对application监听实际上就是对ServletContext（Servlet上下文）进行监听，主要使用 ServletContextListener接口ServletContextAttributeListener接口。

**1、上下文状态监听：javax.servlet.ServletContextListener接口**

上下文状态监听指的是对Web容器的启动和关闭进行监听，对上下文状态监听可 以使用ServletContextListener接口完成，此接口定义的方法如下：

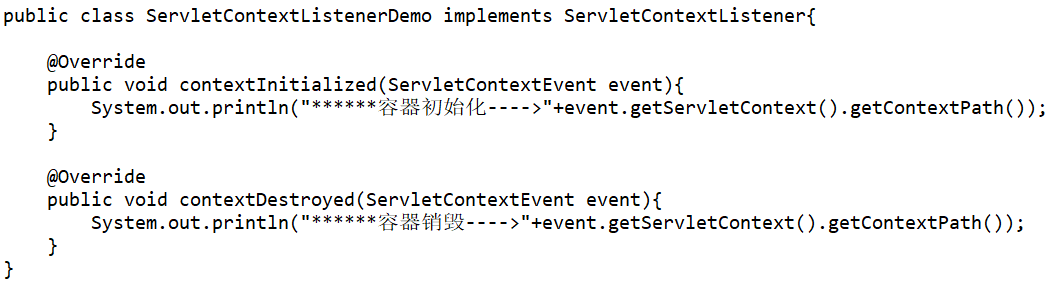
|  |  |
| --- | --- |
| public void contextInitialized ( ServletContextEvent e ) | **容器启动时触发** |
| public void contextDestroyed ( ServletContextEvent e ) | **容器关闭时触发** |

上下文状态监听操作中，一旦触发了ServletContextListener接口中定义的事件，则 可以通过javax.servlet.ServletContextEvent对象进行处理，此事件定义的方法如下：

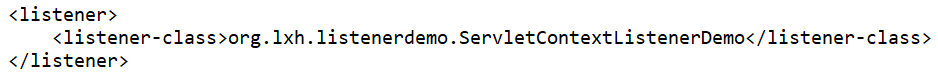
|  |  |
| --- | --- |
| public ServletContext getServletContext( ) | 取得ServletContext对象 |

在覆写的方法中可以通过ServletContextEvent对象得到ServletContext对象并进行 想要的操作。示例如下：

·编写上下文状态监听器：ServletContextListenerDemo.java



·配置监听器



可以发现，Listener的配置比较简单，只需指定监听器的包类路径即可。

**2、上下文属性监听：java.servlet.ServletContextAttributeListener接口**

对上下文属性操作监听，可以使用javax.servlet.ServletContextAttributeListener接口 完成，此接口中定义了如下方法：

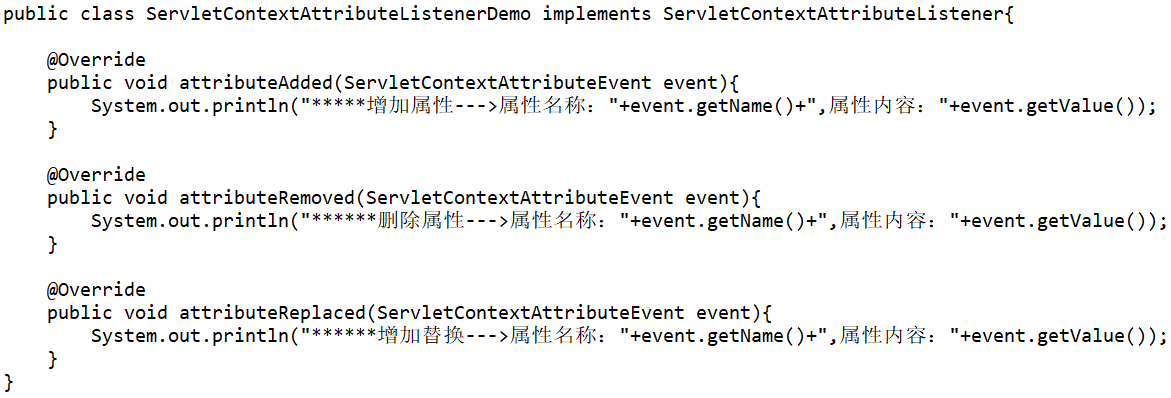
|  |  |
| --- | --- |
| public void attributeAdded ( ServletContextAttributeEvent e ) | **增加属性时触发** |
| public void attributeRemoved ( ServletContextAttributeEvent e ) | **删除属性时触发** |
| public void attributeReplaced ( ServletContextAttributeEvent e ) | **替换属性（重复设置）时触发** |

在上下文属性监听操作中，一旦触发了ServletContextAttributeListener接口中定义 的事件，则可以通过javax.servlet.ServletContextAttributeEvent对象进行事件的处理， 此事件定义的方法如下：

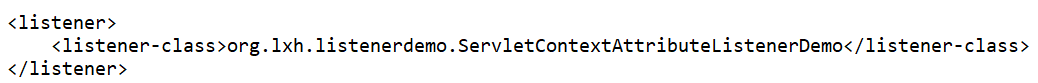
|  |  |
| --- | --- |
| public String getName ( ) | 取得设置的属性名称 |
| public Object getValue ( ) | 取得设置的属性内容 |

示例如下：

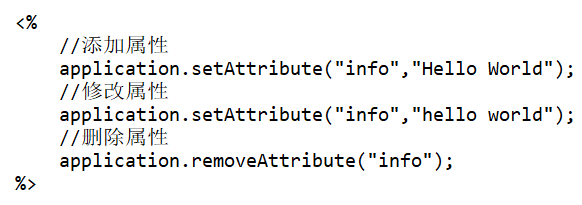
·上下文属性监听器：ServletContextAttributeListenerDemo.java



·在web.xml文件中配置监听器：



·编写JSP页面实现application范围的属性的添加、修改和删除：



访问该jsp页面，观察Tomcat控制台打印。

## 对session监听

针对于session监听主要使用HttpSessionListener接口、HttpSessionAttributeListener接口 和HttpSessionBindingListener接口。

1. **session状态监听：javax.servlet.http.HttpSessionListener接口**

如果需要对创建或销毁session的操作进行监听时，可以实现

javax.servlet.http.HttpSessionListener接口，此接口定义了如下方法：

|  |  |
| --- | --- |
| public void sessionCreated ( HttpSessionEvent e ) | **session创建时调用** |
| public void sessionDestroyed ( HttpSessionEvent e ) | **session销毁时调用** |

当session创建或销毁时，将产生javax.servlet.http.HttpSessionEvent事件，此事件 对象拥有以下方法：

|  |  |
| --- | --- |
| public HttpSession getSession ( ) | 取得当前的session |

当用户打开打开一个动态页面时，session创建，当用户注销或者session超时时， session被销毁。

1. **对session属性监听javax.servlet.http.HttpSessionAttributeListener接口**

对session属性进行监听可以使用javax.servlet.http.HttpSessionAttributeListener接口完成。此接口定义了如下方法：

|  |  |
| --- | --- |
| public void attributeAdded ( HttpSessionBindingEvent e ) | 增加属性时触发 |
| public void attributeRemoved ( HttpSessionBindingEvent e ) | 删除属性时触发 |
| public void attributReplaced ( HttpSessionBindingEvent e ) | 替换属性时触发 |

当session对象进行属性操作时，将根据属性的操作触发HttpSessionAttributeListener 接口中的对于的方法，并产生javax.servlet.http.HttpSessionBindingEvent事件，此事 件定义了如下方法：

|  |  |
| --- | --- |
| public HttpSession getSession ( ) | 取得session对象 |
| public String getName ( ) | 取得属性的名称 |
| public Object getValue ( ) | 取得属性的内容 |

**3、session属性监听：javax.servlet.http.HttpSessionBindingListener接口**

先前讲解的session属性监听器需要在web.xml文件中进行配置，但是在Web中也 提供了一个javax.servlet.http.HttpSessionBindingListener接口，通过此接口实现的监 听程序可以不用在web.xml中配置而直接使用。其功能是对指定类型对象是否被添 加为session属性进行监听，当session添加或删除实现了此接口的类的对象时才会 触发相应的事件。此接口中定义了如下方法：

|  |  |
| --- | --- |
| public void valueBound ( HttpSessionBindingEvent e ) | 绑定对象到session时触发 |
| public void valueUnbound ( HttpSessionBindingEvent e ) | 从session移除对象时触发 |

以上两个方法都会产生javax.servlet.http.HttpSessionBindingEvent事件，此事件定义 的方法如下所示。

|  |  |
| --- | --- |
| public HttpSession getSession ( ) | 取得session对象 |
| public String getName ( ) | 取得属性的名称 |
| public Object getValue ( ) | 取得属性的内容 |

## 对request监听

对request监听主要使用ServletRequestListener接口和ServletRequestAttributeListener 接口。

**1、请求状态监听：javax.servlet.ServletRequestListener接口**

当需要对用户的每次请求进行监听时，可以使用javax.servlet.ServletRequestListener 接口，此接口定义了如下方法：

|  |  |
| --- | --- |
| public void requestInitialized ( ServletRequestEvent e ) | 请求开始时调用 |
| public void requestDestroyed ( ServletRequestEvent e ) | 请求结束时调用 |

一旦监听到事件后，将产生javax.servlet.ServletRequestEvent事件，该事件定义的 方法如下：

|  |  |
| --- | --- |
| public ServletRequest getServletRequest ( ) | 取得ServletRequest对象 |
| public ServletContext getServletContext ( ) | 取得ServletContext对象 |

**2、对request属性监听：javax.servlet.ServletRequestAttributeListener**

对request范围属性监听可以使用javax.servlet.RequestAttributeListener接口， 此接口定义了如下方法：

|  |  |
| --- | --- |
| public void attributeAdded ( ServletRequestAttributeEvent e ) | 属性增加时触发 |
| public void attributeReplaced ( ServletRequestAttributeEvent e ) | 属性替换时触发 |
| public void attributeRemoved ( ServletRequestAttributeEvent e ) | 属性删除时触发 |

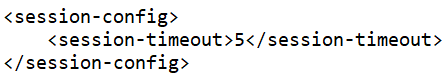
request属性的操作将产生ServletRequestAttributeEvent事件，此事件中定义了如下 方法：

|  |  |
| --- | --- |
| public String getName ( ) | 取得设置的属性名称 |
| public Object getValue ( ) | 取得设置的属性内容 |
| publicServletRequest getServletRequest ( ) | 取得ServletRequest对象 |
| public ServletContext getServletContext ( ) | 取得ServletContext对象 |

## session的创建与销毁

当用户访问任意动态页时，服务器会为该用户分配session，并且触发HttpSessionListener 接口中的sessionCreated( )事件，但是session销毁却有两种不同的方式来触发 sessionDestroyed( )事件。

1. 方式一：调用HttpSession接口中的invalidate( )方法，让一个sessoin失效
2. 方式二：在web.xml文件中配置session的超时时间，如下：



以上将一个session的超时时间配置成了5分钟，如果一个用户在5分钟之内没有 与服务器进行任何交互操作的话，那么服务器会认为此用户已将离开，会自动将其 注销。如果没有在项目中配置session超时时间，则tomcat中默认的超时时间为3 0分钟。