# 这部分主要是与Java Web和Web Service相关的面试题。

## ****阐述Servlet和CGI的区别?****

答：Servlet与CGI的区别在于Servlet处于服务器进程中，它通过多线程方式运行其service()方法，一个实例可以服务于多个请求，并且其实例一般不会销毁，而CGI对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于Servlet。

补充：Sun Microsystems公司在1996年发布Servlet技术就是为了和CGI进行竞争，Servlet是一个特殊的Java程序，一个基于Java的Web应用通常包含一个或多个Servlet类。Servlet不能够自行创建并执行，它是在Servlet容器中运行的，容器将用户的请求传递给Servlet程序，并将Servlet的响应回传给用户。通常一个Servlet会关联一个或多个JSP页面。以前CGI经常因为性能开销上的问题被诟病，然而Fast CGI早就已经解决了CGI效率上的问题，所以面试的时候大可不必信口开河的诟病CGI，事实上有很多你熟悉的网站都使用了CGI技术。

## **Servlet接口中有哪些方法？**

答：Servlet接口定义了5个方法，其中前三个方法与Servlet生命周期相关：   
- void init(ServletConfig config) throws ServletException   
- void service(ServletRequest req, ServletResponse resp) throws ServletException, java.io.IOException   
- void destory()   
- java.lang.String getServletInfo()   
- ServletConfig getServletConfig()

Web容器加载Servlet并将其实例化后，Servlet生命周期开始，容器运行其init()方法进行Servlet的初始化；请求到达时调用Servlet的service()方法，service()方法会根据需要调用与请求对应的doGet或doPost等方法；当服务器关闭或项目被卸载时服务器会将Servlet实例销毁，此时会调用Servlet的destroy()方法。

## **转发（forward）和重定向（redirect）的区别？**

答：forward是容器中控制权的转向，是服务器请求资源，服务器直接访问目标地址的URL，把那个URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。redirect就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址，很明显redirect无法访问到服务器保护起来资源，但是可以从一个网站redirect到其他网站。forward更加高效，所以在满足需要时尽量使用forward（通过调用RequestDispatcher对象的forward()方法，该对象可以通过ServletRequest对象的getRequestDispatcher()方法获得），并且这样也有助于隐藏实际的链接；在有些情况下，比如需要访问一个其它服务器上的资源，则必须使用重定向（通过HttpServletResponse对象调用其sendRedirect()方法实现）。

## **JSP有哪些内置对象？作用分别是什么？**

答：JSP有9个内置对象：   
- request：封装客户端的请求，其中包含来自GET或POST请求的参数；   
- response：封装服务器对客户端的响应；   
- pageContext：通过该对象可以获取其他对象；   
- session：封装用户会话的对象；   
- application：封装服务器运行环境的对象；   
- out：输出服务器响应的输出流对象；   
- config：Web应用的配置对象；   
- page：JSP页面本身（相当于Java程序中的this）；   
- exception：封装页面抛出异常的对象。

**补充：**如果用Servlet来生成网页中的动态内容无疑是非常繁琐的工作，另一方面，所有的文本和HTML标签都是硬编码，即使做出微小的修改，都需要进行重新编译。JSP解决了Servlet的这些问题，它是Servlet很好的补充，可以专门用作为用户呈现视图（View），而Servlet作为控制器（Controller）专门负责处理用户请求并转发或重定向到某个页面。基于Java的Web开发很多都同时使用了Servlet和JSP。JSP页面其实是一个Servlet，能够运行Servlet的服务器（Servlet容器）通常也是JSP容器，可以提供JSP页面的运行环境，Tomcat就是一个Servlet/JSP容器。第一次请求一个JSP页面时，Servlet/JSP容器首先将JSP页面转换成一个JSP页面的实现类，这是一个实现了JspPage接口或其子接口HttpJspPage的Java类。JspPage接口是Servlet的子接口，因此每个JSP页面都是一个Servlet。转换成功后，容器会编译Servlet类，之后容器加载和实例化Java字节码，并执行它通常对Servlet所做的生命周期操作。对同一个JSP页面的后续请求，容器会查看这个JSP页面是否被修改过，如果修改过就会重新转换并重新编译并执行。如果没有则执行内存中已经存在的Servlet实例。我们可以看一段JSP代码对应的Java程序就知道一切了，而且9个内置对象的神秘面纱也会被揭开。

JSP页面：

<%@ page pageEncoding="UTF-8"%>

<%

String path = request.getContextPath();

String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + path + "/";

%>

<!DOCTYPE html>

<html>

<head>

<base href="<%=basePath%>">

<title>首页</title>

<style type="text/css">

\* { font-family: "Arial"; }

</style>

</head>

<body>

<h1>Hello, World!</h1>

<hr/>

<h2>Current time is: <%= new java.util.Date().toString() %></h2>

</body>

</html>

对应的Java代码：

/\*

\* Generated by the Jasper component of Apache Tomcat

\* Version: Apache Tomcat/7.0.52

\* Generated at: 2014-10-13 13:28:38 UTC

\* Note: The last modified time of this file was set to

\* the last modified time of the source file after

\* generation to assist with modification tracking.

\*/

package org.apache.jsp;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

public final class index\_jsp extends org.apache.jasper.runtime.HttpJspBase

implements org.apache.jasper.runtime.JspSourceDependent {

private static final javax.servlet.jsp.JspFactory \_jspxFactory = javax.servlet.jsp.JspFactory

.getDefaultFactory();

private static java.util.Map<java.lang.String, java.lang.Long> \_jspx\_dependants;

private javax.el.ExpressionFactory \_el\_expressionfactory;

private org.apache.tomcat.InstanceManager \_jsp\_instancemanager;

public java.util.Map<java.lang.String, java.lang.Long> getDependants() {

return \_jspx\_dependants;

}

public void \_jspInit() {

\_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(

getServletConfig().getServletContext()).getExpressionFactory();

\_jsp\_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory

.getInstanceManager(getServletConfig());

}

public void \_jspDestroy() {

}

public void \_jspService(

final javax.servlet.http.HttpServletRequest request,

final javax.servlet.http.HttpServletResponse response)

throws java.io.IOException, javax.servlet.ServletException {

// 内置对象就是在这里定义的

final javax.servlet.jsp.PageContext pageContext;

javax.servlet.http.HttpSession session = null;

final javax.servlet.ServletContext application;

final javax.servlet.ServletConfig config;

javax.servlet.jsp.JspWriter out = null;

final java.lang.Object page = this;

javax.servlet.jsp.JspWriter \_jspx\_out = null;

javax.servlet.jsp.PageContext \_jspx\_page\_context = null;

try {

response.setContentType("text/html;charset=UTF-8");

pageContext = \_jspxFactory.getPageContext(this, request, response,

null, true, 8192, true);

\_jspx\_page\_context = pageContext;

application = pageContext.getServletContext();

config = pageContext.getServletConfig();

session = pageContext.getSession();

out = pageContext.getOut();

\_jspx\_out = out;

out.write('\r');

out.write('\n');

String path = request.getContextPath();

String basePath = request.getScheme() + "://"

+ request.getServerName() + ":" + request.getServerPort()

+ path + "/";

// 以下代码通过输出流将HTML标签输出到浏览器中

out.write("\r\n");

out.write("\r\n");

out.write("<!DOCTYPE html>\r\n");

out.write("<html>\r\n");

out.write(" <head>\r\n");

out.write(" <base href=\"");

out.print(basePath);

out.write("\">\r\n");

out.write(" <title>首页</title>\r\n");

out.write(" <style type=\"text/css\">\r\n");

out.write(" \t\* { font-family: \"Arial\"; }\r\n");

out.write(" </style>\r\n");

out.write(" </head>\r\n");

out.write(" \r\n");

out.write(" <body>\r\n");

out.write(" <h1>Hello, World!</h1>\r\n");

out.write(" <hr/>\r\n");

out.write(" <h2>Current time is: ");

out.print(new java.util.Date().toString());

out.write("</h2>\r\n");

out.write(" </body>\r\n");

out.write("</html>\r\n");

} catch (java.lang.Throwable t) {

if (!(t instanceof javax.servlet.jsp.SkipPageException)) {

out = \_jspx\_out;

if (out != null && out.getBufferSize() != 0)

try {

out.clearBuffer();

} catch (java.io.IOException e) {

}

if (\_jspx\_page\_context != null)

\_jspx\_page\_context.handlePageException(t);

else

throw new ServletException(t);

}

} finally {

\_jspxFactory.releasePageContext(\_jspx\_page\_context);

}

}

}

## **get和post请求的区别？**

答：   
①get请求用来从服务器上获得资源，而post是用来向服务器提交数据；   
②get将表单中数据按照name=value的形式，添加到action 所指向的URL 后面，并且两者使用"?"连接，而各个变量之间使用"&"连接；post是将表单中的数据放在HTTP协议的请求头或消息体中，传递到action所指向URL；   
③get传输的数据要受到URL长度限制（1024字节）；而post可以传输大量的数据，上传文件通常要使用post方式；   
④使用get时参数会显示在地址栏上，如果这些数据不是敏感数据，那么可以使用get；对于敏感数据还是应用使用post；   
⑤get使用MIME类型application/x-www-form-urlencoded的URL编码（也叫百分号编码）文本的格式传递参数，保证被传送的参数由遵循规范的文本组成，例如一个空格的编码是"%20"。

## **常用的Web服务器有哪些？**

答：Unix和Linux平台下使用最广泛的免费HTTP服务器是Apache服务器，而Windows平台的服务器通常使用IIS作为Web服务器。选择Web服务器应考虑的因素有：性能、安全性、日志和统计、虚拟主机、代理服务器、缓冲服务和集成应用程序等。下面是对常见服务器的简介：   
- **IIS**：Microsoft的Web服务器产品，全称是Internet Information Services。IIS是允许在公共Intranet或Internet上发布信息的Web服务器。IIS是目前最流行的Web服务器产品之一，很多著名的网站都是建立在IIS的平台上。IIS提供了一个图形界面的管理工具，称为Internet服务管理器，可用于监视配置和控制Internet服务。IIS是一种Web服务组件，其中包括Web服务器、FTP服务器、NNTP服务器和SMTP服务器，分别用于网页浏览、文件传输、新闻服务和邮件发送等方面，它使得在网络（包括互联网和局域网）上发布信息成了一件很容易的事。它提供ISAPI(Intranet Server API）作为扩展Web服务器功能的编程接口；同时，它还提供一个Internet数据库连接器，可以实现对数据库的查询和更新。   
- **Kangle**：Kangle Web服务器是一款跨平台、功能强大、安全稳定、易操作的高性能Web服务器和反向代理服务器软件。此外，Kangle也是一款专为做虚拟主机研发的Web服务器。实现虚拟主机独立进程、独立身份运行。用户之间安全隔离，一个用户出问题不影响其他用户。支持PHP、ASP、ASP.NET、Java、Ruby等多种动态开发语言。   
- WebSphere：WebSphere Application Server是功能完善、开放的Web应用程序服务器，是IBM电子商务计划的核心部分，它是基于Java的应用环境，用于建立、部署和管理Internet和Intranet Web应用程序，适应各种Web应用程序服务器的需要。   
- **WebLogic**：WebLogic Server是一款多功能、基于标准的Web应用服务器，为企业构建企业应用提供了坚实的基础。针对各种应用开发、关键性任务的部署，各种系统和数据库的集成、跨Internet协作等Weblogic都提供了相应的支持。由于它具有全面的功能、对开放标准的遵从性、多层架构、支持基于组件的开发等优势，很多公司的企业级应用都选择它来作为开发和部署的环境。WebLogic Server在使应用服务器成为企业应用架构的基础方面一直处于领先地位，为构建集成化的企业级应用提供了稳固的基础。   
- **Apache**：目前Apache仍然是世界上用得最多的Web服务器，其市场占有率很长时间都保持在60%以上（目前的市场份额约40%左右）。世界上很多著名的网站都是Apache的产物，它的成功之处主要在于它的源代码开放、有一支强大的开发团队、支持跨平台的应用（可以运行在几乎所有的Unix、Windows、Linux系统平台上）以及它的可移植性等方面。   
- **Tomcat**：Tomcat是一个开放源代码、运行Servlet和JSP的容器。Tomcat实现了Servlet和JSP规范。此外，Tomcat还实现了Apache-Jakarta规范而且比绝大多数商业应用软件服务器要好，因此目前也有不少的Web服务器都选择了Tomcat。   
- **Nginx**：读作"engine x"，是一个高性能的HTTP和反向代理服务器，也是一个IMAP/POP3/SMTP代理服务器。 Nginx是由Igor Sysoev为俄罗斯访问量第二的[Rambler](http://www.rambler.ru/)站点开发的，第一个公开版本0.1.0发布于2004年10月4日。其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。在2014年下半年，Nginx的市场份额达到了14%。

## **JSP和Servlet是什么关系？**

答：其实这个问题在上面已经阐述过了，Servlet是一个特殊的Java程序，它运行于服务器的JVM中，能够依靠服务器的支持向浏览器提供显示内容。JSP本质上是Servlet的一种简易形式，JSP会被服务器处理成一个类似于Servlet的Java程序，可以简化页面内容的生成。Servlet和JSP最主要的不同点在于，Servlet的应用逻辑是在Java文件中，并且完全从表示层中的HTML分离开来。而JSP的情况是Java和HTML可以组合成一个扩展名为.jsp的文件。有人说，Servlet就是在Java中写HTML，而JSP就是在HTML中写Java代码，当然这个说法是很片面且不够准确的。JSP侧重于视图，Servlet更侧重于控制逻辑，在MVC架构模式中，JSP适合充当视图（view）而Servlet适合充当控制器（controller）。

## **讲解JSP中的四种作用域。**

答：JSP中的四种作用域包括page、request、session和application，具体来说：   
- **page**代表与一个页面相关的对象和属性。   
- **request**代表与Web客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个Web组件；需要在页面显示的临时数据可以置于此作用域。   
- **session**代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的session中。   
- **application**代表与整个Web应用程序相关的对象和属性，它实质上是跨越整个Web应用程序，包括多个页面、请求和会话的一个全局作用域。

## **如何实现JSP或Servlet的单线程模式？**

答：  
对于JSP页面，可以通过page指令进行设置。  
<%@page isThreadSafe=”false”%>  
对于Servlet，可以让自定义的Servlet实现SingleThreadModel标识接口。

**说明：** 如果将JSP或Servlet设置成单线程工作模式，会导致每个请求创建一个Servlet实例，这种实践将导致严重的性能问题（服务器的内存压力很大，还会导致频繁的垃圾回收），所以通常情况下并不会这么做。

## **实现会话跟踪的技术有哪些？**

答：由于HTTP协议本身是无状态的，服务器为了区分不同的用户，就需要对用户会话进行跟踪，简单的说就是为用户进行登记，为用户分配唯一的ID，下一次用户在请求中包含此ID，服务器据此判断到底是哪一个用户。   
①URL 重写：在URL中添加用户会话的信息作为请求的参数，或者将唯一的会话ID添加到URL结尾以标识一个会话。   
②设置表单隐藏域：将和会话跟踪相关的字段添加到隐式表单域中，这些信息不会在浏览器中显示但是提交表单时会提交给服务器。   
这两种方式很难处理跨越多个页面的信息传递，因为如果每次都要修改URL或在页面中添加隐式表单域来存储用户会话相关信息，事情将变得非常麻烦。   
③cookie：cookie有两种，一种是基于窗口的，浏览器窗口关闭后，cookie就没有了；另一种是将信息存储在一个临时文件中，并设置存在的时间。当用户通过浏览器和服务器建立一次会话后，会话ID就会随响应信息返回存储在基于窗口的cookie中，那就意味着只要浏览器没有关闭，会话没有超时，下一次请求时这个会话ID又会提交给服务器让服务器识别用户身份。会话中可以为用户保存信息。会话对象是在服务器内存中的，而基于窗口的cookie是在客户端内存中的。如果浏览器禁用了cookie，那么就需要通过下面两种方式进行会话跟踪。当然，在使用cookie时要注意几点：首先不要在cookie中存放敏感信息；其次cookie存储的数据量有限（4k），不能将过多的内容存储cookie中；再者浏览器通常只允许一个站点最多存放20个cookie。当然，和用户会话相关的其他信息（除了会话ID）也可以存在cookie方便进行会话跟踪。   
④HttpSession：在所有会话跟踪技术中，HttpSession对象是最强大也是功能最多的。当一个用户第一次访问某个网站时会自动创建HttpSession，每个用户可以访问他自己的HttpSession。可以通过HttpServletRequest对象的getSession方法获得HttpSession，通过HttpSession的setAttribute方法可以将一个值放在HttpSession中，通过调用HttpSession对象的getAttribute方法，同时传入属性名就可以获取保存在HttpSession中的对象。与上面三种方式不同的是，HttpSession放在服务器的内存中，因此不要将过大的对象放在里面，即使目前的Servlet容器可以在内存将满时将HttpSession中的对象移到其他存储设备中，但是这样势必影响性能。添加到HttpSession中的值可以是任意Java对象，这个对象最好实现了Serializable接口，这样Servlet容器在必要的时候可以将其序列化到文件中，否则在序列化时就会出现异常。

\*\*补充：\*\*HTML5中可以使用Web Storage技术通过JavaScript来保存数据，例如可以使用localStorage和sessionStorage来保存用户会话的信息，也能够实现会话跟踪。

## **过滤器有哪些作用和用法？**

答： Java Web开发中的过滤器（filter）是从Servlet 2.3规范开始增加的功能，并在Servlet 2.4规范中得到增强。对Web应用来说，过滤器是一个驻留在服务器端的Web组件，它可以截取客户端和服务器之间的请求与响应信息，并对这些信息进行过滤。当Web容器接受到一个对资源的请求时，它将判断是否有过滤器与这个资源相关联。如果有，那么容器将把请求交给过滤器进行处理。在过滤器中，你可以改变请求的内容，或者重新设置请求的报头信息，然后再将请求发送给目标资源。当目标资源对请求作出响应时候，容器同样会将响应先转发给过滤器，在过滤器中你可以对响应的内容进行转换，然后再将响应发送到客户端。

常见的过滤器用途主要包括：对用户请求进行统一认证、对用户的访问请求进行记录和审核、对用户发送的数据进行过滤或替换、转换图象格式、对响应内容进行压缩以减少传输量、对请求或响应进行加解密处理、触发资源访问事件、对XML的输出应用XSLT等。

和过滤器相关的接口主要有：Filter、FilterConfig和FilterChain。

编码过滤器的例子：

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.annotation.WebFilter;

import javax.servlet.annotation.WebInitParam;

@WebFilter(urlPatterns = { "\*" },

initParams = {@WebInitParam(name="encoding", value="utf-8")})

public class CodingFilter implements Filter {

private String defaultEncoding = "utf-8";

@Override

public void destroy() {

}

@Override

public void doFilter(ServletRequest req, ServletResponse resp,

FilterChain chain) throws IOException, ServletException {

req.setCharacterEncoding(defaultEncoding);

resp.setCharacterEncoding(defaultEncoding);

chain.doFilter(req, resp);

}

@Override

public void init(FilterConfig config) throws ServletException {

String encoding = config.getInitParameter("encoding");

if (encoding != null) {

defaultEncoding = encoding;

}

}

}

下载计数过滤器的例子：

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Properties;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.annotation.WebFilter;

import javax.servlet.http.HttpServletRequest;

@WebFilter(urlPatterns = {"/\*"})

public class DownloadCounterFilter implements Filter {

private ExecutorService executorService = Executors.newSingleThreadExecutor();

private Properties downloadLog;

private File logFile;

@Override

public void destroy() {

executorService.shutdown();

}

@Override

public void doFilter(ServletRequest req, ServletResponse resp,

FilterChain chain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

final String uri = request.getRequestURI();

executorService.execute(new Runnable() {

@Override

public void run() {

String value = downloadLog.getProperty(uri);

if(value == null) {

downloadLog.setProperty(uri, "1");

}

else {

int count = Integer.parseInt(value);

downloadLog.setProperty(uri, String.valueOf(++count));

}

try {

downloadLog.store(new FileWriter(logFile), "");

}

catch (IOException e) {

e.printStackTrace();

}

}

});

chain.doFilter(req, resp);

}

@Override

public void init(FilterConfig config) throws ServletException {

String appPath = config.getServletContext().getRealPath("/");

logFile = new File(appPath, "downloadLog.txt");

if(!logFile.exists()) {

try {

logFile.createNewFile();

}

catch(IOException e) {

e.printStackTrace();

}

}

downloadLog = new Properties();

try {

downloadLog.load(new FileReader(logFile));

} catch (IOException e) {

e.printStackTrace();

}

}

}

**说明：**这里使用了Servlet 3规范中的注解来部署过滤器，当然也可以在web.xml中使用<filter>和<filter-mapping>标签部署过滤器，如108题中所示。

## **监听器有哪些作用和用法？**

答：Java Web开发中的监听器（listener）就是application、session、request三个对象创建、销毁或者往其中添加修改删除属性时自动执行代码的功能组件，如下所示：   
①ServletContextListener：对Servlet上下文的创建和销毁进行监听。   
②ServletContextAttributeListener：监听Servlet上下文属性的添加、删除和替换。   
③HttpSessionListener：对Session的创建和销毁进行监听。

补充：session的销毁有两种情况：1). session超时（可以在web.xml中通过<session-config>/<session-timeout>标签配置超时时间）；2). 通过调用session对象的invalidate()方法使session失效。

④HttpSessionAttributeListener：对Session对象中属性的添加、删除和替换进行监听。   
⑤ServletRequestListener：对请求对象的初始化和销毁进行监听。   
⑥ServletRequestAttributeListener：对请求对象属性的添加、删除和替换进行监听。

下面是一个统计网站最多在线人数监听器的例子。

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

import javax.servlet.annotation.WebListener;

/\*\*

上下文监听器，在服务器启动时初始化onLineCount和maxOnLineCount两个变量

并将其置于服务器上下文（ServletContext）中，其初始值都是0

\*/

@WebListener

public class InitListener implements ServletContextListener {

@Override

public void contextDestroyed(ServletContextEvent evt) {

}

@Override

public void contextInitialized(ServletContextEvent evt) {

evt.getServletContext().setAttribute("onLineCount", 0);

evt.getServletContext().setAttribute("maxOnLineCount", 0);

}

}

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.servlet.ServletContext;

import javax.servlet.annotation.WebListener;

import javax.servlet.http.HttpSessionEvent;

import javax.servlet.http.HttpSessionListener;

/\*\*

会话监听器，在用户会话创建和销毁的时候根据情况

修改onLineCount和maxOnLineCount的值

\*/

@WebListener

public class MaxCountListener implements HttpSessionListener {

@Override

public void sessionCreated(HttpSessionEvent event) {

ServletContext ctx = event.getSession().getServletContext();

int count = Integer.parseInt(ctx.getAttribute("onLineCount").toString());

count++;

ctx.setAttribute("onLineCount", count);

int maxOnLineCount = Integer.parseInt(ctx.getAttribute("maxOnLineCount").toString());

if (count > maxOnLineCount) {

ctx.setAttribute("maxOnLineCount", count);

DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

ctx.setAttribute("date", df.format(new Date()));

}

}

@Override

public void sessionDestroyed(HttpSessionEvent event) {

ServletContext app = event.getSession().getServletContext();

int count = Integer.parseInt(app.getAttribute("onLineCount").toString());

count--;

app.setAttribute("onLineCount", count);

}

}

**说明：**这里使用了Servlet 3规范中的@WebListener注解配置监听器，当然你可以在web.xml文件中用<listener>标签配置监听器，如108题中所示。

## **web.xml文件中可以配置哪些内容？**

答：web.xml用于配置Web应用的相关信息，如：监听器（listener）、过滤器（filter）、 Servlet、相关参数、会话超时时间、安全验证方式、错误页面等，下面是一些开发中常见的配置：

①配置Spring上下文加载监听器加载Spring配置文件并创建IoC容器：

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

②配置Spring的OpenSessionInView过滤器来解决延迟加载和Hibernate会话关闭的矛盾：

<filter>

<filter-name>openSessionInView</filter-name>

<filter-class>

org.springframework.orm.hibernate3.support.OpenSessionInViewFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>openSessionInView</filter-name>

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

</filter-mapping>

③配置会话超时时间为10分钟：

<session-config>

<session-timeout>10</session-timeout>

</session-config>

④配置404和Exception的错误页面：

<error-page>

<error-code>404</error-code>

<location>/error.jsp</location>

</error-page>

<error-page>

<exception-type>java.lang.Exception</exception-type>

<location>/error.jsp</location>

</error-page>

⑤配置安全认证方式：

<security-constraint>

<web-resource-collection>

<web-resource-name>ProtectedArea</web-resource-name>

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

<http-method>GET</http-method>

<http-method>POST</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

**说明：**对Servlet（小服务）、Listener（监听器）和Filter（过滤器）等Web组件的配置，Servlet 3规范提供了基于注解的配置方式，可以分别使用@WebServlet、@WebListener、@WebFilter注解进行配置。

**补充：**如果Web提供了有价值的商业信息或者是敏感数据，那么站点的安全性就是必须考虑的问题。安全认证是实现安全性的重要手段，认证就是要解决“Are you who you say you are?”的问题。认证的方式非常多，简单说来可以分为三类：   
A. What you know? — 口令   
B. What you have? — 数字证书（U盾、密保卡）   
C. Who you are? — 指纹识别、虹膜识别   
在Tomcat中可以通过建立安全套接字层（Secure Socket Layer, SSL）以及通过基本验证或表单验证来实现对安全性的支持。