[置顶] Java面试题全集（中）

标签： Java面试题 面试题大全 Web服务 JSP Servlet

2015-04-09 22:05 11428人阅读 评论(0) 收藏 举报

分类： 面试笔试（19）

这部分主要是与Java Web和Web Service相关的面试题。

96、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。

97、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。

98、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）。

99、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() +
%>

<!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.HttpJspBas
        implements org.apache.jasper.runtime.JspSourceDependent {

    private static final javax.servlet.jsp.JspFactory _jspxFactory = java
            .getDefaultFactory();

    private static java.util.Map<java.lang.String, java.lang.Long> _jspx_

    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()).getExpressionFac
        _jsp_instancemanager = org.apache.jasper.runtime.InstanceManager
                .getInstanceManager(getServletConfig());
```

```
35      }
36
37      public void _jspDestroy() {
38      }
39
40      public void _jspService(
41              final javax.servlet.http.HttpServletRequest request,
42              final javax.servlet.http.HttpServletResponse response)
43              throws java.io.IOException, javax.servlet.ServletException {
44          // 内置对象就是在这里定义的
45          final javax.servlet.jsp.PageContext pageContext;
46          javax.servlet.http.HttpSession session = null;
47          final javax.servlet.ServletContext application;
48          final javax.servlet.ServletConfig config;
49          javax.servlet.jsp.JspWriter out = null;
50          final java.lang.Object page = this;
51          javax.servlet.jsp.JspWriter _jspx_out = null;
52          javax.servlet.jsp.PageContext _jspx_page_context = null;
53
54          try {
55              response.setContentType("text/html;charset=UTF-8");
56              pageContext = _jspxFactory.getPageContext(this, request, resp
57                      null, true, 8192, true);
58              _jspx_page_context = pageContext;
59              application = pageContext.getServletContext();
60              config = pageContext.getServletConfig();
61              session = pageContext.getSession();
62              out = pageContext.getOut();
63              _jspx_out = out;
64
65              out.write('\r');
66              out.write('\n');
67
68              String path = request.getContextPath();
69              String basePath = request.getScheme() + "://"
70                      + request.getServerName() + ":" + request.getServerPo
71                      + path + "/";
72  // 以下代码通过输出流将HTML标签输出到浏览器中
73              out.write("\r\n");
74              out.write("\r\n");
75              out.write("<!DOCTYPE html>\r\n");
76              out.write("<html>\r\n");
77              out.write("  <head>\r\n");
78              out.write("    <base href=\"");
79              out.print(basePath);
80              out.write("\">\r\n");
81              out.write("    <title>首页</title>\r\n");
82              out.write("    <style type=\"text/css\">\r\n");
83              out.write("    \t* { font-family: \"Arial\"; }\r\n");
```

```
84              out.write("    </style>\r\n");
85              out.write("  </head>\r\n");
86              out.write("  \r\n");
87              out.write("  <body>\r\n");
88              out.write("    <h1>Hello, World!</h1>\r\n");
89              out.write("    <hr/>\r\n");
90              out.write("    <h2>Current time is: ");
91              out.print(new java.util.Date().toString());
92              out.write("</h2>\r\n");
93              out.write("  </body>\r\n");
94              out.write("</html>\r\n");
95          } catch (java.lang.Throwable t) {
96              if (!(t instanceof javax.servlet.jsp.SkipPageException)) {
97                  out = _jspx_out;
98                  if (out != null && out.getBufferSize() != 0)
99                      try {
100                         out.clearBuffer();
101                     } catch (java.io.IOException e) {
102                     }
103                 if (_jspx_page_context != null)
104                     _jspx_page_context.handlePageException(t);
105                 else
106                     throw new ServletException(t);
107             }
108         } finally {
109             _jspxFactory.releasePageContext(_jspx_page_context);
110         }
111     }
112 }
```

100、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"。

101、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%。

102、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）。

103、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。

104、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。

```
1 <%@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。

105、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。

106、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：

```
1  import java.io.IOException;
2
3  import javax.servlet.Filter;
4  import javax.servlet.FilterChain;
5  import javax.servlet.FilterConfig;
6  import javax.servlet.ServletException;
7  import javax.servlet.ServletRequest;
8  import javax.servlet.ServletResponse;
9  import javax.servlet.annotation.WebFilter;
10 import javax.servlet.annotation.WebInitParam;
```

```
11
12 @WebFilter(urlPatterns = { "*" },
13         initParams = {@WebInitParam(name="encoding", value="utf-8")})
14 public class CodingFilter implements Filter {
15     private String defaultEncoding = "utf-8";
16
17     @Override
18     public void destroy() {
19     }
20
21     @Override
22     public void doFilter(ServletRequest req, ServletResponse resp,
23             FilterChain chain) throws IOException, ServletException {
24         req.setCharacterEncoding(defaultEncoding);
25         resp.setCharacterEncoding(defaultEncoding);
26         chain.doFilter(req, resp);
27     }
28
29     @Override
30     public void init(FilterConfig config) throws ServletException {
31         String encoding = config.getInitParameter("encoding");
32         if (encoding != null) {
33             defaultEncoding = encoding;
34         }
35     }
36 }
```

下载计数过滤器的例子：

```
1  import java.io.File;
2  import java.io.FileReader;
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.util.Properties;
6  import java.util.concurrent.ExecutorService;
7  import java.util.concurrent.Executors;
8
9  import javax.servlet.Filter;
10 import javax.servlet.FilterChain;
11 import javax.servlet.FilterConfig;
12 import javax.servlet.ServletException;
13 import javax.servlet.ServletRequest;
14 import javax.servlet.ServletResponse;
15 import javax.servlet.annotation.WebFilter;
16 import javax.servlet.http.HttpServletRequest;
17
18 @WebFilter(urlPatterns = {"/*"})
19 public class DownloadCounterFilter implements Filter {
20
```

```
21      private ExecutorService executorService = Executors.newSingleThreadE
22      private Properties downloadLog;
23      private File logFile;
24
25      @Override
26      public void destroy() {
27          executorService.shutdown();
28      }
29
30      @Override
31      public void doFilter(ServletRequest req, ServletResponse resp,
32              FilterChain chain) throws IOException, ServletException {
33          HttpServletRequest request = (HttpServletRequest) req;
34          final String uri = request.getRequestURI();
35          executorService.execute(new Runnable() {
36
37              @Override
38              public void run() {
39                  String value = downloadLog.getProperty(uri);
40                  if(value == null) {
41                      downloadLog.setProperty(uri, "1");
42                  }
43                  else {
44                      int count = Integer.parseInt(value);
45                      downloadLog.setProperty(uri, String.valueOf(++count)
46                  }
47                  try {
48                      downloadLog.store(new FileWriter(logFile), "");
49                  }
50                  catch (IOException e) {
51                      e.printStackTrace();
52                  }
53              }
54          });
55          chain.doFilter(req, resp);
56      }
57
58      @Override
59      public void init(FilterConfig config) throws ServletException {
60          String appPath = config.getServletContext().getRealPath("/");
61          logFile = new File(appPath, "downloadLog.txt");
62          if(!logFile.exists()) {
63              try {
64                  logFile.createNewFile();
65              }
66              catch(IOException e) {
67                  e.printStackTrace();
68              }
69          }
```

```
        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。

107、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) {
    }
```

```
15
16      @Override
17      public void contextInitialized(ServletContextEvent evt) {
18          evt.getServletContext().setAttribute("onLineCount", 0);
19          evt.getServletContext().setAttribute("maxOnLineCount", 0);
20      }
21
22  }
```

```
1   import java.text.DateFormat;
2   import java.text.SimpleDateFormat;
3   import java.util.Date;
4
5   import javax.servlet.ServletContext;
6   import javax.servlet.annotation.WebListener;
7   import javax.servlet.http.HttpSessionEvent;
8   import javax.servlet.http.HttpSessionListener;
9
10  /**
11   会话监听器，在用户会话创建和销毁的时候根据情况
12   修改onLineCount和maxOnLineCount的值
13  */
14  @WebListener
15  public class MaxCountListener implements HttpSessionListener {
16
17      @Override
18      public void sessionCreated(HttpSessionEvent event) {
19          ServletContext ctx = event.getSession().getServletContext();
20          int count = Integer.parseInt(ctx.getAttribute("onLineCount").toS
21          count++;
22          ctx.setAttribute("onLineCount", count);
23          int maxOnLineCount = Integer.parseInt(ctx.getAttribute("maxOnLin
24          if (count > maxOnLineCount) {
25              ctx.setAttribute("maxOnLineCount", count);
26              DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
27              ctx.setAttribute("date", df.format(new Date()));
28          }
29      }
30
31      @Override
32      public void sessionDestroyed(HttpSessionEvent event) {
33          ServletContext app = event.getSession().getServletContext();
34          int count = Integer.parseInt(app.getAttribute("onLineCount").toS
35          count--;
36          app.setAttribute("onLineCount", count);
37      }
38  }
```

> **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。

108、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.OpenSessionInViewFil
    </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>
```

```
5
6   <error-page>
7       <exception-type>java.lang.Exception</exception-type>
8       <location>/error.jsp</location>
9   </error-page>
```

⑤配置安全认证方式：

```
1   <security-constraint>
2       <web-resource-collection>
3           <web-resource-name>ProtectedArea</web-resource-name>
4           <url-pattern>/admin/*</url-pattern>
5           <http-method>GET</http-method>
6           <http-method>POST</http-method>
7       </web-resource-collection>
8       <auth-constraint>
9           <role-name>admin</role-name>
10      </auth-constraint>
11  </security-constraint>
12
13  <login-config>
14      <auth-method>BASIC</auth-method>
15  </login-config>
16
17  <security-role>
18      <role-name>admin</role-name>
19  </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。

109、你的项目中使用过哪些JSTL标签？

答：项目中主要使用了JSTL的核心标签库，包括<c:if>、<c:choose>、<c: when>、<c: otherwise>、<c:forEach>等，主要用于构造循环和分支结构以控制显示逻辑。

> **说明：**虽然JSTL标签库提供了core、sql、fmt、xml等标签库，但是实际开发中建议只使用核心标签库（core），而且最好只使用分支和循环标签并辅以表达式语言（EL），这样才能真正做到数据显示和业务逻辑的分离，这才是最佳实践。

110、使用标签库有什么好处？如何自定义JSP标签？

答：使用标签库的好处包括以下几个方面：

- 分离JSP页面的内容和逻辑，简化了Web开发；
- 开发者可以创建自定义标签来封装业务逻辑和显示逻辑；
- 标签具有很好的可移植性、可维护性和可重用性；
- 避免了对Scriptlet（小脚本）的使用（很多公司的项目开发都不允许在JSP中书写小脚本）

自定义JSP标签包括以下几个步骤：

- 编写一个Java类实现实现Tag/BodyTag/IterationTag接口（开发中通常不直接实现这些接口而是继承TagSupport/BodyTagSupport/SimpleTagSupport类，这是对缺省适配模式的应用），重写doStartTag()、doEndTag()等方法，定义标签要完成的功能
- 编写扩展名为tld的标签描述文件对自定义标签进行部署，tld文件通常放在WEB-INF文件夹下或其子目录中
- 在JSP页面中使用taglib指令引用该标签库

下面是一个自定义标签库的例子。

步骤1 - 标签类源代码TimeTag.java：

```
1  package com.jackfrued.tags;
2
3  import java.io.IOException;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  import javax.servlet.jsp.JspException;
8  import javax.servlet.jsp.JspWriter;
9  import javax.servlet.jsp.tagext.TagSupport;
10
11 public class TimeTag extends TagSupport {
12     private static final long serialVersionUID = 1L;
13
14     private String format = "yyyy-MM-dd hh:mm:ss";
15     private String foreColor = "black";
16     private String backColor = "white";
17
18     public int doStartTag() throws JspException {
```

```
            SimpleDateFormat sdf = new SimpleDateFormat(format);
            JspWriter writer = pageContext.getOut();
            StringBuilder sb = new StringBuilder();
            sb.append(String.format("<span style='color:%s;background-color
                foreColor, backColor, sdf.format(new Date())));
            try {
              writer.print(sb.toString());
            } catch(IOException e) {
              e.printStackTrace();
            }
            return SKIP_BODY;
        }

    public void setFormat(String format) {
        this.format = format;
    }

    public void setForeColor(String foreColor) {
        this.foreColor = foreColor;
    }

    public void setBackColor(String backColor) {
        this.backColor = backColor;
    }
}
```

步骤2 - 编写标签库描述文件my.tld：

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <description>定义标签库</description>
    <tlib-version>1.0</tlib-version>
    <short-name>MyTag</short-name>
    <tag>
        <name>time</name>
        <tag-class>com.jackfrued.tags.TimeTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>format</name>
            <required>false</required>
        </attribute>
        <attribute>
            <name>foreColor</name>
```

```
        </attribute>
        <attribute>
            <name>backColor</name>
        </attribute>
    </tag>
</taglib>
```

步骤3 - 在JSP页面中使用自定义标签：

```
<%@ page pageEncoding="UTF-8"%>
<%@ taglib prefix="my" uri="/WEB-INF/tld/my.tld" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme() + "://" + request.getServerName() +
%>

<!DOCTYPE html>
<html>
  <head>
    <base href="<%=basePath%>">
    <title>首页</title>
    <style type="text/css">
        * { font-family: "Arial"; font-size:72px; }
    </style>
  </head>

  <body>
    <my:time format="yyyy-MM-dd" backColor="blue" foreColor="yellow"/>
  </body>
</html>
```

> **提示：**如果要将自定义的标签库发布成JAR文件，需要将标签库描述文件（tld文件）放在JAR文件的META-INF目录下，可以JDK中的jar工具完成JAR文件的生成，如果不清楚如何操作，可以请教谷老师和百老师。

111、说一下表达式语言（EL）的隐式对象及其作用。

答：EL的隐式对象包括：pageContext、initParam（访问上下文参数）、param（访问请求参数）、paramValues、header（访问请求头）、headerValues、cookie（访问cookie）、applicationScope（访问application作用域）、sessionScope（访问session作用域）、requestScope（访问request作用域）、pageScope（访问page作用域）。

用法如下所示：

```
${pageContext.request.method}
${pageContext["request"]["method"]}
${pageContext.request["method"]}
```

```
${pageContext["request"].method}
${initParam.defaultEncoding}
${header["accept-language"]}
${headerValues["accept-language"][0]}
${cookie.jsessionid.value}
${sessionScope.loginUser.username}
```

> **补充：**表达式语言的.和[]运算作用是一致的，唯一的差别在于如果访问的属性名不符合Java标识符命名规则，例如上面的accept-language就不是一个有效的Java标识符，那么这时候就只能用[]运算符而不能使用.运算符获取它的值

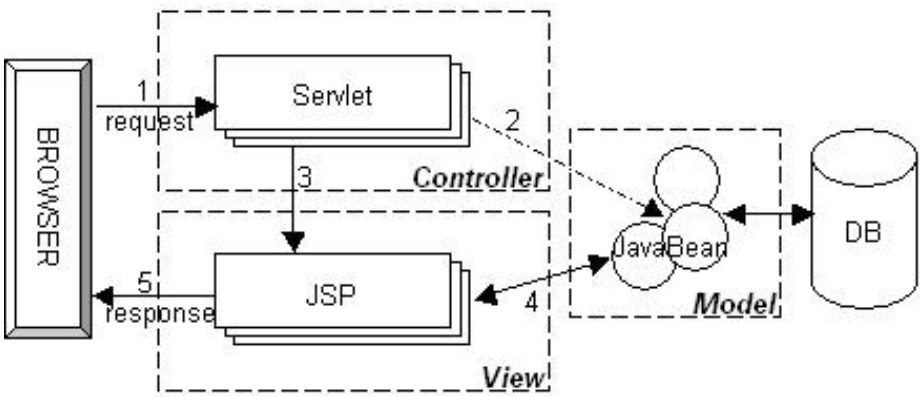
112、表达式语言（EL）支持哪些运算符？

答：除了.和[]运算符，EL还提供了：

- 算术运算符：+、-、*、/或div、%或mod
- 关系运算符：==或eq、!=或ne、>或gt、>=或ge、<或lt、<=或le
- 逻辑运算符：&&或and、||或or、!或not
- 条件运算符：${statement? A : B}（跟Java的条件运算符类似）
- empty运算符：检查一个值是否为null或者空（数组长度为0或集合中没有元素也返回true）

113、Java Web开发的Model 1和Model 2分别指的是什么？

答：Model 1是以页面为中心的Java Web开发，使用JSP+JavaBean技术将页面显示逻辑和业务逻辑处理分开，JSP实现页面显示，JavaBean对象用来保存数据和实现业务逻辑。Model 2是基于MVC（模型-视图-控制器，Model-View-Controller）架构模式的开发模型，实现了模型和视图的彻底分离，利于团队开发和代码复用，如下图所示。



114、Servlet 3中的异步处理指的是什么？

答：在Servlet 3中引入了一项新的技术可以让Servlet异步处理请求。有人可能会质疑，既然都有多线程了，还需要异步处理请求吗？答案是肯定的，因为如果一个任务处理时间相当长，那么Servlet或Filter会一直占用着请求处理线程直到任务结束，随着并发用户的增加，容器将会遭遇线程超出的风险，这这种情况下很多的请求将会被堆积起来而后续的请求可能会遭遇拒绝服务，直到有资源可以处理请求为止。异步特性可以帮助应用节省容器中的线程，特别适合执行时间长而且用户需要得到结果的任务，如果用户不需要得到结果则直接将一个Runnable对象交给Executor并立即返回即可。（如果不清楚多线程和线程池的相关内容，请查看《Java面试题全集（上）》关于多线程和线程池的部分或阅读我的另一篇文章《关于Java并发编程的总结和

思考》）

> **补充：**多线程在Java诞生初期无疑是一个亮点，而Servlet单实例多线程的工作方式也曾为其赢得美名，然而技术的发展往往会颠覆我们很多的认知，就如同当年爱因斯坦的相对论颠覆了牛顿的经典力学一般。事实上，异步处理绝不是Serlvet 3首创，如果你了解Node.js的话，对Servlet 3的这个重要改进就不以为奇了。

下面是一个支持异步处理请求的Servlet的例子。

```
import java.io.IOException;
import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns = {"/async"}, asyncSupported = true)
public class AsyncServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // 开启Tomcat异步Servlet支持
        req.setAttribute("org.apache.catalina.ASYNC_SUPPORTED", true);

        final AsyncContext ctx = req.startAsync();  // 启动异步处理的上下文
        // ctx.setTimeout(30000);
        ctx.start(new Runnable() {

            @Override
            public void run() {
                // 在此处添加异步处理的代码

                ctx.complete();
            }
        });
    }
}
```

115、如何在基于Java的Web项目中实现文件上传和下载？

答：在Sevlet 3 以前，Servlet API中没有支持上传功能的API，因此要实现上传功能需要引入第三方工具从POST请求中获得上传的附件或者通过自行处理输入流来获得上传的文件，我们推荐使用Apache的commons-fileupload。

从Servlet 3开始，文件上传变得无比简单，相信看看下面的例子一切都清楚了。

上传页面index.jsp：

```
<%@ page pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Photo Upload</title>
</head>
<body>
<h1>Select your photo and upload</h1>
<hr/>
<div style="color:red;font-size:14px;">${hint}</div>
<form action="UploadServlet" method="post" enctype="multipart/form-data">
    Photo file: <input type="file" name="photo" />
    <input type="submit" value="Upload" />
</form>
</body>
</html>
```

支持上传的Servlet：

```
package com.jackfrued.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;

@WebServlet("/UploadServlet")
@MultipartConfig
public class UploadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOExce
        // 可以用request.getPart()方法获得名为photo的上传附件
        // 也可以用request.getParts()获得所有上传附件（多文件上传）
        // 然后通过循环分别处理每一个上传的文件
        Part part = request.getPart("photo");
        if (part != null && part.getSubmittedFileName().length() > 0) {
            // 用ServletContext对象的getRealPath()方法获得上传文件夹的绝对路径
            String savePath = request.getServletContext().getRealPath("/u
```

```
            // Servlet 3.1规范中可以用Part对象的getSubmittedFileName()方法获
            // 更好的做法是为上传的文件进行重命名（避免同名文件的相互覆盖）
            part.write(savePath + "/" + part.getSubmittedFileName());
            request.setAttribute("hint", "Upload Successfully!");
        } else {
            request.setAttribute("hint", "Upload failed!");
        }
        // 跳转回到上传页面
        request.getRequestDispatcher("index.jsp").forward(request, respo
    }

}
```

116、服务器收到用户提交的表单数据，到底是调用Servlet的doGet()还是doPost()方法？

答：HTML的<form>元素有一个method属性，用来指定提交表单的方式，其值可以是get或post。我们自定义的Servlet一般情况下会重写doGet()或doPost()两个方法之一或全部，如果是GET请求就调用doGet()方法，如果是POST请求就调用doPost()方法，那为什么为什么这样呢？我们自定义的Servlet通常继承自HttpServlet，HttpServlet继承自GenericServlet并重写了其中的service()方法，这个方法是Servlet接口中定义的。HttpServlet重写的service()方法会先获取用户请求的方法，然后根据请求方法调用doGet()、doPost()、doPut()、doDelete()等方法，如果在自定义Servlet中重写了这些方法，那么显然会调用重写过的（自定义的）方法，这显然是对模板方法模式的应用（如果不理解，请参考阎宏博士的《Java与模式》一书的第37章）。当然，自定义Servlet中也可以直接重写service()方法，那么不管是哪种方式的请求，都可以通过自己的代码进行处理，这对于不区分请求方法的场景比较合适。

117、JSP中的静态包含和动态包含有什么区别？

答：静态包含是通过JSP的include指令包含页面，动态包含是通过JSP标准动作<jsp:forward>包含页面。静态包含是编译时包含，如果包含的页面不存在则会产生编译错误，而且两个页面的"contentType"属性应保持一致，因为两个页面会合二为一，只产生一个class文件，因此被包含页面发生的变动再包含它的页面更新前不会得到更新。动态包含是运行时包含，可以向被包含的页面传递参数，包含页面和被包含页面是独立的，会编译出两个class文件，如果被包含的页面不存在，不会产生编译错误，也不影响页面其他部分的执行。代码如下所示：

```
<%-- 静态包含 --%>
<%@ include file="..." %>

<%-- 动态包含 --%>
<jsp:include page="...">
    <jsp:param name="..." value="..." />
</jsp:include>
```

118、Servlet中如何获取用户提交的查询参数或表单数据？

答：可以通过请求对象（HttpServletRequest）的getParameter()方法通过参数名获得参数值。如果有包含多个值的参数（例如复选框），可以通过请求对象的getParameterValues()方法获得。当然也可以通过请求

对象的getParameterMap()获得一个参数名和参数值的映射（Map）。

119、Servlet中如何获取用户配置的初始化参数以及服务器上下文参数？

答：可以通过重写Servlet接口的init(ServletConfig)方法并通过ServletConfig对象的getInitParameter()方法来获取Servlet的初始化参数。可以通过ServletConfig对象的getServletContext()方法获取ServletContext对象，并通过该对象的getInitParameter()方法来获取服务器上下文参数。当然，ServletContext对象也在处理用户请求的方法（如doGet()方法）中通过请求对象的getServletContext()方法来获得。

120、如何设置请求的编码以及响应内容的类型？

答：通过请求对象（ServletRequest）的setCharacterEncoding(String)方法可以设置请求的编码，其实要彻底解决乱码问题就应该让页面、服务器、请求和响应、Java程序都使用统一的编码，最好的选择当然是UTF-8；通过响应对象（ServletResponse）的setContentType(String)方法可以设置响应内容的类型，当然也可以通过HttpServletResponsed对象的setHeader(String, String)方法来设置。

> **说明：**现在如果还有公司在面试的时候问JSP的声明标记、表达式标记、小脚本标记这些内容的话，这样的公司也不用去了，其实JSP内置对象、JSP指令这些东西基本上都可以忘却了，关于Java Web开发的相关知识，可以看一下我的《Servlet&JSP思维导图》，上面有完整的知识点的罗列。想了解如何实现自定义MVC框架的，可以看一下我的《Java Web自定义MVC框架详解》。

121、解释一下网络应用的模式及其特点。

答：典型的网络应用模式大致有三类：B/S、C/S、P2P。其中B代表浏览器（Browser）、C代表客户端（Client）、S代表服务器（Server），P2P是对等模式，不区分客户端和服务器。B/S应用模式中可以视为特殊的C/S应用模式，只是将C/S应用模式中的特殊的客户端换成了浏览器，因为几乎所有的系统上都有浏览器，那么只要打开浏览器就可以使用应用，没有安装、配置、升级客户端所带来的各种开销。P2P应用模式中，成千上万台彼此连接的计算机都处于对等的地位，整个网络一般来说不依赖专用的集中服务器。网络中的每一台计算机既能充当网络服务的请求者，又对其它计算机的请求作出响应，提供资源和服务。通常这些资源和服务包括：信息的共享和交换、计算资源（如CPU的共享）、存储共享（如缓存和磁盘空间的使用）等，这种应用模式最大的阻力安全性、版本等问题，目前有很多应用都混合使用了多种应用模型，最常见的网络视频应用，它几乎把三种模式都用上了。

> **补充：**此题要跟"电子商务模式"区分开，因为有很多人被问到这个问题的时候马上想到的是B2B（如阿里巴巴）、B2C（如当当、亚马逊、京东）、C2C（如淘宝、拍拍）、C2B（如威客）、O2O（如美团、饿了么）。对于这类问题，可以去百度上面科普一下。

122、什么是Web Service（Web服务）？

答：从表面上看，Web Service就是一个应用程序，它向外界暴露出一个能够通过Web进行调用的API。这就是说，你能够用编程的方法透明的调用这个应用程序，不需要了解它的任何细节，跟你使用的编程语言也没有关系。例如可以创建一个提供天气预报的Web Service，那么无论你用哪种编程语言开发的应用都可以通过调用它的API并传入城市信息来获得该城市的天气预报。之所以称之为Web Service，是因为它基于HTTP

协议传输数据，这使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件，就可相互交换数据或集成。

> **补充：**这里必须要提及的一个概念是SOA（Service-Oriented Architecture，面向服务的架构），SOA是一种思想，它将应用程序的不同功能单元通过中立的契约联系起来，独立于硬件平台、操作系统和编程语言，使得各种形式的功能单元能够更好的集成。显然，Web Service是SOA的一种较好的解决方案，它更多的是一种标准，而不是一种具体的技术。

123、概念解释：SOAP、WSDL、UDDI。

答：

- SOAP：简单对象访问协议（Simple Object Access Protocol），是Web Service中交换数据的一种协议规范。
- WSDL：Web服务描述语言（Web Service Description Language），它描述了Web服务的公共接口。这是一个基于XML的关于如何与Web服务通讯和使用的服务描述；也就是描述与目录中列出的Web服务进行交互时需要绑定的协议和信息格式。通常采用抽象语言描述该服务支持的操作和信息，使用的时候再将实际的网络协议和信息格式绑定给该服务。
- UDDI：统一描述、发现和集成（Universal Description, Discovery and Integration），它是一个基于XML的跨平台的描述规范，可以使世界范围内的企业在互联网上发布自己所提供的服务。简单的说，UDDI是访问各种WSDL的一个门面（可以参考设计模式中的门面模式）。

> **提示：**关于Web Service的相关概念和知识可以在W3CSchool上找到相关的资料。

124、Java规范中和Web Service相关的规范有哪些？

答：Java规范中和Web Service相关的有三个：

- JAX-WS(JSR 224)：这个规范是早期的基于SOAP的Web Service规范JAX-RPC的替代版本，它并不提供向下兼容性，因为RPC样式的WSDL以及相关的API已经在Java EE5中被移除了。WS-MetaData是JAX-WS的依赖规范，提供了基于注解配置Web Service和SOAP消息的相关API。
- JAXM(JSR 67)：定义了发送和接收消息所需的API,相当于Web Service的服务器端。
- JAX-RS(JSR 311 & JSR 339 & JSR 370)：是Java针对REST（Representation State Transfer）架构风格制定的一套Web Service规范。REST是一种软件架构模式，是一种风格，它不像SOAP那样本身承载着一种消息协议，（两种风格的Web Service均采用了HTTP做传输协议，因为HTTP协议能穿越防火墙，Java的远程方法调用（RMI）等是重量级协议，通常不能穿越防火墙），因此可以将REST视为基于HTTP协议的软件架构。REST中最重要的两个概念是资源定位和资源操作，而HTTP协议恰好完整的提供了这两个点。HTTP协议中的URI可以完成资源定位，而GET、POST、OPTION、DELETE方法可以完成资源操作。因此REST完全依赖HTTP协议就可以完成Web Service，而不像SOAP协议那样只利用了HTTP的传输特性，定位和操作都是由SOAP协议自身完成的，也正是由于SOAP消息的存在使得基于SOAP的Web Service显得笨重而逐渐被淘汰。

125、介绍一下你了解的Java领域的Web Service框架。

答：Java领域的Web Service框架很多，包括Axis2（Axis的升级版本）、Jersey（RESTful的Web Service框架）、CXF（XFire的延续版本）、Hessian、Turmeric、JBoss SOA等，其中绝大多数都是开源框架。

> **提示：**面试被问到这类问题的时候一定选择自己用过的最熟悉的作答，如果之前没有了解过就应该在面试前花一些时间了解其中的两个，并比较其优缺点，这样才能在面试时给出一个漂亮的答案。