目录

[BS架构 1](#_Toc500907209)

[URL 2](#_Toc500907210)

[TCP， IP和HTTP 2](#_Toc500907211)

[HTTP协议 2](#_Toc500907212)

[HTTP数据包 2](#_Toc500907213)

[HTTP状态码 3](#_Toc500907214)

[组件、容器 3](#_Toc500907215)

[Servlet和JSP 3](#_Toc500907216)

[Servlet如何处理Http协议 3](#_Toc500907217)

[GET与POST请求方式的区别 3](#_Toc500907218)

[Servlet运行步骤 4](#_Toc500907219)

[重定向、转发 4](#_Toc500907220)

[绝对路径、相对路径 4](#_Toc500907221)

[Servlet的生命周期 4](#_Toc500907222)

[Servlet核心接口和抽象类 5](#_Toc500907223)

[Servlet线程安全问题 5](#_Toc500907224)

[Scriptlet 5](#_Toc500907225)

[JSP转换为Servlet过程中的编码 5](#_Toc500907226)

[JSP指令 5](#_Toc500907227)

[JSP九大内置对象（隐含对象） 6](#_Toc500907228)

[Cookie和Session 6](#_Toc500907229)

[web.xml 6](#_Toc500907230)

[Servlet过滤器 7](#_Toc500907231)

[Servlet监听器 7](#_Toc500907232)

[EL表达式 8](#_Toc500907233)

[JSTL 8](#_Toc500907234)

[自定义JSP标签 8](#_Toc500907235)

# BS架构

|  |  |
| --- | --- |
| Browser（提供操作页面） + Web Server（负责业务逻辑处理） + DB（负责数据的管理） | |
| 优点 | 1. 不需要单独安装客户端 2. 开发相对于CS简单，客户端和服务端的通信模块都是使用标准的HTTP协议进行通信   注：三层CS架构需要自定义协议，通信模块 |

# URL

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| http://www.mywebsite.com/sj/test/test.aspx?name=sviergn&x=true#stuff | | | | | |
| schema | host | port | path | query-string | anchor |
| 指定底层使用的协议 | IP地址或域名 | HTTP服务器的默认端口是80 | 访问资源的路径 | 发送给http服务器的数据 | 锚 |
| http | www.mywebsite.com |  | /sj/test/test.aspx | name=sviergn&x=true | stuff |
| Snipaste_2018-07-21_08-47-05 | | | | | |

# TCP， IP和HTTP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IP | | TCP | | HTTP |
| 网络层协议 | | 传输层协议 | | 应用层协议 |
| 解决计算机网络相互连接并进行通信 | | 解决数据如何在网络中传输 | | 解决如何包装数据 |
| 关系 | 把IP想象成一种高速公路，它允许其它协议在上面行驶并找到其它电脑的出口。TCP和UDP是高速公路上的“卡车”，它们携带的货物就是HTTP、FTP等包装出来的数据包 | |  | |
| TCP/IP | 包含了一系列用于处理数据通信的协议，定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准协议。 | |

# HTTP协议

|  |  |
| --- | --- |
| 定义 | HTTP协议是HyperText Transfer Protocol的缩写，即超文本传输协议。是由w3c（万维网联盟）制定的一种应用层协议，用来定义浏览器与web服务器之间如何通信以及通信的数据格式。BS架构中的通信模块就是以HTTP这个协议作为标准协议的 |
| 步骤 | 1. 浏览器根据IP地址和端口号与服务器建立连接 2. 向Web服务器发送请求数据包 3. Web服务器接收请求数据包后，发送相应的响应数据包 4. 浏览器接收响应数据后关闭连接 |

# HTTP数据包

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 请求数据包 | | | | |  |
| 请求行(request line) | 请求方式 | 请求路径 | 协议类型/版本 | |
| GET | …/sayHi | HTTP/1.1 | |
| 消息头（若干）(header) | 一些键值对，一般由w3c定义，大部分自动生成 | | | |
| 实体内容(body) | 只有当请求方式为post时，才有实体内容? | | | |
| 响应数据包 | | | | |  |
| 状态行(status line) | 协议类型/版本 | 状态码 | | 状态描述 |
| HTTP/1.1 | 200 | | OK |
| 消息头（若干）(header) | Web服务器返回消息 | | | |
| 实体内容 | 程序处理的结果 | | | |

# HTTP状态码

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Informational |  |  |  |
| 2 | Success | 200 | OK | 请求成功。一般用于GET与POST请求 |
| 3 | Redirection | 301 | Moved Permanently | 请求的页面已被移动到新的URL |
| 302 | Found | 请求的页面暂时移动到新的URL |
|  |  | 304 | Not Modified | 请求的页面未修改，服务器告诉客户端读取缓存 |
| 4 | Client Error | 400 | Bad Request | 服务器无法理解的请求 |
| 401 | Unauthorized | 请求的页面需要用户名和密码 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但是拒绝执行此请求 |
| 404 | Not Found | 服务器找不到请求的页面 |
| 405 | Method Not Allowed | 请求中的方法是被禁止 |
| 5 | Server Error | 500 | Internal Server Error | 服务器内部错误，无法完成请求 |

# 组件、容器

|  |  |
| --- | --- |
| 组件 component | 容器 container |
| 符合一定规范的类组合到一起就构成了组件，他可以提供某些特定的功能 | 符合一定规范，提供组件运行环境的一个程序，方便别人使用他来完成一系列功能 |
| 如：Servlet, JSP, JavaBean, EJB | 如：tomcat可以提供Servlet, JSP等服务，俗称叫Servlet容器 |

# Servlet和JSP

|  |  |
| --- | --- |
| Servlet | JSP |
| sun公司制定的一种用来扩展Web服务器功能的组件规范。早起的Web服务器只能处理静态资源请求，无法根据请求计算后生成相应的HTML内容 | sun公司制订的一种服务器端的动态页面技术规范，是Servlet技术的发展，本质上是Servlet的简易方式，更强调应用的外表表达。JSP编译后转换成一个Servlet类。 |
| 应用逻辑在Java文件中，并且完全从表示层中的HTML里分离开来 | Java和HTML可以组合成一个扩展名为.jsp的文件 |
| 主要用于控制逻辑 | 侧重于视图 |

开发Servlet的步骤

# Servlet如何处理Http协议

|  |  |
| --- | --- |
| Web容器(tomcat)收到一个Http请求时，通信数据由Web容器负责封装和提供，这些信息被解析成以下两个对象 | |
| HttpServletRequest | HttpServletResponse |
| 1. 输出请求数据包内容 getHeaderNames() … 2. 取得和设置Cookies getCookies() 3. 取得路径信息 4. 标识HTTP会话 getSession() 5. 实现请求转发 | 1. 设置请求数据包内容 addDateHeader(name, date) 2. 设置对客户端的输出内容 3. 响应的状态码 getStatus() 4. 设置Cookies addCookie(cookie) 5. 实现重定向 |

# GET与POST请求方式的区别

|  |  |
| --- | --- |
| get | post |
| 请求指定的资源 ??? | 向指定的资源提交需要处理的数据 |
| 将请求数据添加到请求资源的后面，因为各浏览器对url长度是有限制的，所有数据大小有限制 | 数据放在请求数据包的Body中，没有长度限制 |
| 请求数据显示在浏览器地址栏上，安全性较低 | 相对安全 |
| 1)在地址栏输入一个地址 2)点击链接 3)表单默认提交?? | 表单提交，向服务器进行数据添加或者更新的时候使用 |

**Servlet处理中文**

|  |  |  |
| --- | --- | --- |
| 获取请求中文 | | 响应输出中文 |
| post | request.setCharacterEncoding("utf-8") | response.setContentType(text/html;charset=utf-8") |
| 设置发往服务器的参数以何种编码。如没设置，服务器默认使用iso-8859-1来编码  注：表单所在页面也要设置charset="utf-8" | 生成消息头中Content-Type:text/html，并自动调response.setCharacterEncoding("utf-8")来通知浏览器解码 |
| get | get请求提交的参数在URI中，一开始服务器默认使用iso-8859-1解码，所以对get请求无效。 | 要在response.getWriter()执行之前或response提交之前使用 |
| tomcat修改config/server.xml文件中<Connector URIEncoding="utf-8"> | |
| username = new String(username.getBytes("iso-8859-1"), "utf-8") | |
| 注：表单所在页面也要设置charset="utf-8" | |

# Servlet运行步骤

|  |
| --- |
| 1. 浏览器依据ip, port建立与web服务器（Servlet容器）之间的连接 2. 浏览器将相关数据打包（按http协议创建一个请求数据报），发送给web服务器 3. web服务器拆包（按http协议将请求数据报中的相关数据解析出来），将解析到的数据添加到request对象里，同时，创建一个response对象 4. web服务器依据路径找到Servlet创建对象 5. web服务器调用service处理请求，处理结果写到response对象里 6. web服务器将response对象中的数据取出来，打包（按http协议创建数据包），发送给浏览器 7. 浏览器拆包（按http协议将数据包中的数据解析出来），生成相应的页面 |

# 重定向、转发

|  |  |
| --- | --- |
| 重定向 | 转发 |
| 服务器向浏览器发送一个302状态码和一个Location消息头（地址），浏览器收到后立即向重定向地址发出请求 | 一个web组件（Servlet/JSP）将未完成的处理通过容器转交给另外一个web组件继续完成。 |
| response.sendRedirect("servlet-day07/aaa.jsp") | 1. 绑定数据到request对象：   request.setAttribute(String name, Object o)   1. 获得转发器:   request.getRequestDispatcher("aaa.jsp")   1. 转发：   rd.forward(request, response) |
| 重定向地址栏的地址发生改变 | 转发后地址栏地址不变 |
| 重定向地址可以是任意的 | 转发的地址必须是同一个应用内部的某个地址 |
| 重定向所涉及到的Web组件不共享… | 转发所涉及到的Web组件共享同一个request和response |

# 绝对路径、相对路径

|  |  |
| --- | --- |
| 绝对路径 | 相对路径 |
| 以"/"开头的路径，以一个固定位置作为起始位置，这个固定位置可能是应用名或应用名之后 | 不以"/"开头的路径 |
| 链接地址，表单提交，重定向：从应用名开始写  转发：从应用名后开始写 | 退回上一级目录："../"开头 |

# Servlet的生命周期

|  |  |
| --- | --- |
| 实例化 | 容器调用Servlet的构造器，创建一个Servlet对象   1. 默认：容器收到请求之后，开始创建 2. 配置<load-on-start-up>：容器启动之后立即创建 |
| 初始化 | 容器在创建好Servlet之后，立即调用init方法，只执行一次 |
| 处理请求 | 容器收到请求之后调用Servlet对象的service()，根据请求的不同调用不同的方法（如doGet， doPost） |
| 销毁 | 容器依据自身算法删除Servlet对象，删除前调用destroy方法 |

# Servlet核心接口和抽象类

|  |  |
| --- | --- |
| *Servlet* | 1)init(config) 2)service(req, res) 3)destroy() |
| GenericServlet | 实现了init和detroy等方法 |
| HttpServlet | 实现了service方法 |

# Servlet线程安全问题

|  |  |
| --- | --- |
| 线程安全问题 | Servlet工作机制是单实例提供多线程的服务，它是通过一个调度线程来维护多个工作线程的有序执行；但由于多个线程共享一份实例，在对公共资源进行访问时一定会出现线程不同步的问题 |
| 解决方法 | 避免使用全局变量或者静态变量，使用局部变量，局部变量在每个线程中都有各自的实例 |
| 采用同步锁来保证安全：synchronized(this) {} |

html out.write

# Scriptlet

|  |  |  |
| --- | --- | --- |
| <% %> | <%= %> | <%! %> |
| JSP小脚本 | JSP表达式 | JSP声明 |
|  | 变量、变量加运算符组合的表达式、有返回值的方法 | 成员属性或成员方法的声明 |
| 原封不动 | service()中的out.print | Servlet类中成员属性或成员方法 |

# JSP转换为Servlet过程中的编码

|  |  |
| --- | --- |
| <%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8"%> | |
| 容器将.jsp文件转换为.java文件(servlet) | JVM首先会获取pageEncoding的值，如果该值存在则采用它设定的编码来编译，否则则采用file.encoding编码来编译 |
| JVM将.java文件转换为.class文件 | 与任何编码的设置都没有关系，.java文件就转换成了统一的Unicode格式的.class文件 |
| 输出到浏览器 | 如果设置了charset则浏览器就会使用指定的编码格式进行解码，否则采用默认的ISO-8859-1 |

# JSP指令

|  |  |  |
| --- | --- | --- |
| page | contentType="text/html;  charset=utf-8" | 指定当前JSP页面的MIME类型和字符编码 |
| pageEncoding="utf-8" |  |
| import="java.util.Date" | 导入要使用的java类 |
| language="java" | 定义JSP页面锁用的脚本语言，目前只支持java，默认也是java |
| errorPage="error.jsp" | 指定当JSP页面发生异常时需要转向的错误处理页面 |
| isErrorPage="true" | 指定当前页面是否可以作为另一个JSP页面的错误处理页面 |
| buffer="10kb" | 指定out对象使用缓冲区的大小，默认8kb |
| autoFlush="true" | 控制out对象的缓存区 |
| extends | 指定servlet从哪一个类继承，一般不需要设置 |
| info="测试页面" | 定义JSP页面的描述信息，可通过servlet.getServletInfo()获得 |
| isThreadSafe="true" | 指定对JSP页面的访问是否为线程安全。设置为false，则以单线程运行 |
| session="true" | 指定JSP页面是否使用session |
| isELIgnored="ture" | 指定是否执行EL表达式。设置为true，则忽略"${}的计算" |
| isScriptingEnabled="true" | 默认值为true，它表示启用脚本，表达式和声明。 |
| include | file="header.jsp" | 用来包含其他外部文件（JSP, HTML），用于导航栏，版权声明，logo等 |
| taglib | 引入一个自定义标签集合的定义，包括库路径、自定义标签  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> | |

# JSP九大内置对象（隐含对象）

|  |  |  |
| --- | --- | --- |
| 隐含对象 | 类型 | 说明 |
| request | HttpServletRequest | 请求信息，在一JSP网页发出请求到另一个JSP网页之间有效 |
| response | HttpServletResponse | 响应信息 |
| out | JSPWriter | 输出的数据流 |
| pageContext | PageContext | JSP页面上下文，适用于当前页面范围 |
| session | HttpSession | 会话，用户持续和服务器所连接的时间内有效 |
| application | ServletContext | 全局的上下文对象，从服务器开始执行服务到服务器关闭为止 |
| page | Object | JSP页面本身 |
| config | ServletConfig | Servlet配置对象 |
| exception | Throwable | 捕获网页异常 |
| 容器会依次从pageContext, request, session, application中查找，调用"getxxx"方法输出 | | |

pageContext

JSP是如何运行的

# Cookie和Session

|  |  |
| --- | --- |
| Cookie | Session |
| 状态管理：HTTP是无状态的，无状态是指协议对于事务处理没有记忆能力。将浏览器与web服务器之间多次交互当做一个整体来看待（即为了完成某个业务，需要多次交互，比如购物），并且将多次交互所涉及的数据（即状态）保存下来 | |
| 服务器临时存放在浏览器端的少量的数据，当浏览器再次访问服务器时，会将这些数据发送给服务器 | 服务器端为了保存状态而创建的一个特殊对象，服务器会将这个对象唯一的sessionId以cookie的形式发送给浏览器 |
| new [Cookie](https://www.cnblogs.com/xdp-gacl/p/3803033.html" \l "Cookie(java.lang.String, java.lang.String))([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang) name, [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang) value) | request.getSession() |
| cookie.setMaxAge(0) | sesson.invalidate() |
| 将状态保存在浏览器，安全性低，可以被改写 | 将状态保存在服务器的内存中，安全性高；也可以持久化到file，数据库，memcache，redis等 |
| 只能保存4kb左右的数据，且个数在20个左右 | 能保存更多的数据 |
| 只能保存ASCII字符串  new Cookie("city", URLEncoder.encode( "北京", "UTF-8"); | 能存放对象，可设置生命周期，但当存放对象过多时，会影响服务器的性能 |
| 可被用户禁止，如Cookie被完全禁用，Session也将失效 | 当用户第一次请求生成Session对象时会生成SessionID，SessionID保存在Cookie中 |
|  |  |

# web.xml

|  |  |  |  |
| --- | --- | --- | --- |
| 部署描述符文件 | | | |
| web.xml加载过程 | | 启动一个WEB项目的时候，WEB容器(JBoss, Tomcat等)会去读取它的配置文件web.xml，读取<listener>和<context-param>两个节点 | |
| 紧接着，容器创建一个ServletContext(application)，这个web项目的所有部分都将共享这个上下文 | |
| 容器将<context-param>转换为键值对，存入servletContext | |
| 容器创建<listener>中的类实例，根据配置的class类路径<listener-class>来创建监听 | |
| 容器会读取<filter>，根据指定的类路径来实例化过滤器 | |
| 加载顺序 | | ServletContext -> context-param -> listener -> filter -> servlet | |
| 标签 | <web-app> | | 根元素 |
| <icon> | | 指出IDE和GUI工具用来表示Web应用的大图标和小图标；  <large-icon> <small-icon> |
| <display-name> | | 定义web应用的名称 |
| <disciption> | | Web应用描述，给出于此相关的说明性文本 |
| <context-param> | | 声明应用范围内的初始化参数。它用于向ServletContext提供键值对，我们的listener, filter等在初始化时会用到这些上下文中的信息。在servlet里面可以通过getServletContext().getInitParameter("param-name")得到。  <param-name> <param-value> |
| <filter>  <filter-mapping> | | 过滤器，将一个名字与一个实现javaxs.servlet.Filter接口的类相关联  <filter-name> <filter-class> <init-param> (<param-name> <param-value>)  <filter-name> <url-pattern> |
| <listener> | | 监听器 <listerner-class> |
| <servlet>  <servlet-mapping> | | <servlet-name> <servlet-class> <init-param> (<param-name> <param-value>)  <load-on-startup> <jsp-file>  <servlet-name> <url-pattern> |
| <session-config> | | 会话超时配置，单位分钟 <session-timeout> |
| <mime-mapping> | | [mime-mapping元素将mime类型映射到扩展名，用于规定下载格式](http://baike.baidu.com/view/1002818.htm" \t "_blank)  <extension> <mime-type> |
| <welcome-file-list> | | 设置欢迎文件页 <welcome-file> |
| <error-page> | | 错误页面 <exception-type> <location> |
| <jsp-config> | | 设置jsp |

# Servlet过滤器

|  |  |
| --- | --- |
| 定义 | 是一个小型的web组件，它们通过拦截请求和响应，以便查看、提取或以某种方式操作客户端和服务器之间交换的数据，实现过滤的功能。 拦截响应??? |
| 编写步骤 | 1. 写一个java类，实现javax.servlet.Filter接口 或 直接新建Filter文件 |
| 1. 在doFilter()里编写处理逻辑   如有需要，可以在init()设置一个FilterConfig实例接收FilterConfig参数  FilterChain:过滤器链，如调用了该对象的doFilter()，则容器会继续向后调用过滤器或处理器 |
| 1. 在web.xml配置Filter，如有多个过滤器，容器会依据<filter-mapping>配置的先后顺序来执行 |
| 优点 | 1. 实现代码的"可插拔性"，即增加或减少某个功能模块，不会影响程序的正常执行 2. 可以将多个相同处理逻辑的模块集中写在过滤器里面，方便代码的维护 |

# Servlet监听器

|  |  |
| --- | --- |
| 定义 | Servlet规范中定义的一种特殊的组件，用来监听Servlet容器产生的事件并进行相应的处理 |
| 生命周期相关 | ServletRequestListener |
| HttpSessionListener: 统计网站在线人数 |
| ServletContextListener: 统计网站历史访问次数，系统启动时初始化信息 |
| 绑定数据相关 | ServletRequestAttributeListener |
| HttpSessionAttributeListener |
| ServletContextAttributeListener |
| 编写步骤 | 写一个java类，实现XxxListener接口 或 直接新建Listener文件 |
| 在监听器接口方法中编写处理逻辑 |
| 在web.xml配置Listener |

Servlet3.0 web.xml

# EL表达式

|  |  |  |  |
| --- | --- | --- | --- |
| 定义 | 一套简单的运算规则，用于给JSP标签的属性赋值，也可以直接用来输出 | | |
| 作用 | 访问bean属性 | 1. ${sessionScope.user.name}:   容器会依次从pageContext, request, session, application当中取绑订名为user的对象(getAttribute方法)，接下来，调用该对象的"getXXX"方法输出 | |
| 1. ${user['name']}, ${user.interest[0]} | |
| 进行简单计算 | ${"1"+"2"} | 3，+运算只能求和 |
| ${empty 引用} | 判断是否为空集合，空字符串，是否为null，是否找到某值 |
| 获取请求参数 | ${param.name} | request.getParameter("name") |
| ${param.Values.name[0] | 相当于request.getParameterValues("name") |

# JSTL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 定义 | apache开发的一套jsp标签，后来捐献给了sun，sun将其命名为jstl。简化了JSP和[WEB应用程序](https://baike.baidu.com/item/WEB%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F" \t "_blank)的开发 | | | |
| 使用步骤 | 导包 | jstl | | |
| 引入标签 | <%@ taglib url="http://xxx.xxx" prefix="c" %> | | |
| 几个常用标签 | if | <c:if test="" var="" scope="">  标签体  </c:if> | test | 表达式为true时，执行标签体内容 |
| var | 指定一个绑定名 |
| scope | 指定绑定范围 |
| choose | <c:choose>  <c:when test="">标签体</c:when> // 可以出现多次  <c:otherwise>标签体</c:otherwise> // 可以出现0次或1次 | | |
| forEach | <c:forEach …>  标签体  </c:forEach> | items | 表示要遍历的集合或数组 |
| var | 指定绑定名，绑定范围pageContext |
| varStatus | 指定绑定名，绑定范围pageContext |

# 自定义JSP标签

|  |
| --- |
| 写一个java类，继承SimpleTagSupport类 |
| 在doTag()里编写处理逻辑 |
| 在tld文件描述标签 详细查看-markTag.tld |