# Tomcat

Tomcat 是一个符合 JavaEE WEB 标准的最小的 WEB 容器，所有的 JSP 程序一定要有 WEB 容器的支持才能运行，而且在给定的 WEB 容器里面都会支持事务处理操作。

Tomcat 简单的说就是一个运行 Java 的网络服务器，底层是 Socket 的一个程序，它也是 JSP 和 Serlvet 的一个容器。

**Tomcat目录**

bin:启动和关闭tomcat的bat文件

conf：存放Tomcat的各种配置文件（serverxml,webxml）

lib:存放Tomcat支撑的jar包

logs:存放日志文件（catalina）

temp:存放Tomcat临时文件

webapps:web应用所在目录（供外界访问的web资源存放的目录）

work:Tomcat工作目录（存放jsp被访问后生成对应的server和class文件）

配置临时域名步骤：

1. C:\Windows\System32\drivers\etc,hosts文件
2. 在hosts文件下配置临时域名

ecplise关联Tomcat

Window—> preferences—>Server—>Runtime Environment—>add—>apache-tomcat-8.0.23

# 2. Http

HTTP 协议（Hypertext Transfer Protocol, 超文本传输协议），是一个客户端请求和回应的标准协议，这个协议详细规定了浏览器和万维网服务器之间互相通信的规则。

HTTP URL的格式: http://host[":"port][abs\_path]

http表示要通过HTTP协议来定位网络资源；host表示合法的Internet主机域名或者IP地址；port 指定一个端口号，为空则使用缺省端口80；abs\_path指定请求资源的URL；如果URL中没有给出abs\_path， 那么当它作为请求URL时，必须以“/”的形式给出，通常这个工作浏览器自动帮我们完成。

# Servlet

1.Servlet 是 Server 与 Applet 的缩写，是服务端小程序的意思。

2.要实现 web 开发，需要实现 Servlet 标准

3.Servlet 本质上也是Java类，但要遵循 Servlet 规范进行编写，没有 main()方法，它的创建、使用、销毁都由 Servlet 容器进行管理(写自己的类，不用写main方法，别人自动调用)

4.提供了 Servlet 功能的服务器，叫做 Servlet 容器，其常见容器有很多，如 Tomcat, Jetty, WebLogic Server, Websphere, JBoss 等等。

作用：接收请求数据，处理请求，完成响应

规定：能够处理请求的类必须实现（规范）Servlet接口，该接口中已经定义好了一个方法，service方法 （Request, Response）

**实现自己的Servlet类**

1.继承 HttpServelt类（实现Servlet接口, 继承GenericServlet类）

2.重写service方法（doGet， doPost）

两个参数（HttpServletRequest request,HttpServletResponse response）

Request : 请求对象🡪 请求参数（请求数据）

Response: 响应对象 🡪 响应数据

3. 在service中对请求进行处理

获取请求参数

设定编码格式

1. 配置web.xml文件

<servlet>

<servlet-name>给当前servlet取的名字</servlet-name>

<servlet-class>Servlet类的真实路径</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>给当前servlet取的名字</servlet-name>

<url-pattern>/url访问的路径名</url-pattern>

</servlet-mapping>

说明： 1、url-pattern 可以配多个（一个servlet可以通过多个url-pattern访问）

2、当多个 servlet 配置成了同一个 url-pattern，报错

5. 实现步骤：

1、 new Dynamic web Project (新建动态 web 项目)

2、 new class (新建自己的类)

3、 extends HttpServlet (满足 Servlet 规范)

4、 重写 service() (增加处理程序)

5、 配置 web.xml(配置映射文件)

6、 发布项目

7、 启动服务器

8、 访问

6.生命周期

1.init()初始化：第一次访问servlet执行此方法，并且只初始化一次（单例）

2.service()服务：每次请求servlet都会执行

3.destory()销毁：关闭服务器会自动执行

# 4. HttpServletRequest对象

HttpServletRequest对象：主要作用是用来接收客户端发送过来的请求信息

常用的方法：

request.getParameter(name); 获取指定名称的参数值。

getParameterValues(String name); 获取指定名称参数的所有值数组

getParameterNames();返回一个包含请求消息中的所有参数名的 Enumeration 对象。 通过遍 历这个 Enumeration 对象，就可以获取请求消息中所有的参数名。

getParameterMap():返回一个保存了请求消息中的所有参数名和值的 Map 对象。Map 对象 的 key 是字符串类型的参数名，value 是这个参数所对应的 Object 类型的值数组

request.setAttribute();设置域对象内容

request.getAttribute();获取域对象内容

request.setCharacterEncoding(“UTF-8”);解决乱码，这种方式只针对 POST有效

request.getRequestDispatcher().forword(request,response); 重定向

request. getRequestURI();返回请求行中的资源名部分(项目名称开始)

request. getRequestURL();返回客户端发出请求时的完整URL

request.getContextPath();获取 webapp 名字

request. getMethod();得到客户端请求方式

request. getHeader (String name); 获取单个请求头内容

现在遇到的客户端跳转： 超链接、form 表单、重定向

服务器端跳转： 请求转发

**相对路径**：相对于当前资源本身

**绝对路径**: 相对的位置（相对于8080，端口后面）/相当于http://localhost:8080

# HttpServletResponse 对象

HttpServletResponse 的主要功能用于服务器对客户端的请求进行响应，将 WEB 服务器处理后的结果返回给客户端

常用方法：

**addCookie(Cookie cookie)**：将指定的 Cookie 加入到当前的响应中

sendRedirect(String location)：发送一个临时响应到客户端，重定向

setContentType(String ContentType)：设置响应的 MIME 类型

**getWriter()：**获取输出字符流

**getOutputStream()：**获取输出的字节流

设置头信息

刷新页面：response.setHeader("refresh","2");

跳转页面：response.setHeader("refresh","3;URL=ok.html");

HTML 本身也可以 设置头信息。(客户端跳转)

<meta http-equiv="refresh" content="3;http://www.baidu.com">

返回状态码

sendError(int sc)方法设置错误状态代码

sendError(int sc,String msg)方法除了设置状态码，还向客户发出一条错误信息

添加 Cookie

response. addCookie()可以在 Web 服务器响应中加入 Cookie 对象，这个对象将被浏览器所 保存。Cookie 机制也被用来维护会话状态。

跳转

1)、请求转发(Dispatcher): 此种跳转从头到尾只有一个请求，即只有一个Request对象。request.getRequetsDispatcher(“url”).forward(req, resp)

2)、请求重定向(Redirect):客户浏览器发送请求，服务器处理不了，但是服务器给你返回一个新的地址，然后浏览器再次向新的地址发送请求

response.sendRedirect(“url”)

解决乱码问题

两种方式响应回数据：PrintWriter和ServletOutputStream [重定向 ;请求转发，间接响应]

getOutputStream 和 getWriter 这两个方法互相排斥，只能用一个

字节流响应：

response.setHeader("content-type", "text/html;charset=UTF-8"); // 告知浏览器，以什么格式来解析数据。前提：服务器传输的数据编码格式必须和告知浏览器的一致

字符流响应：

response.setCharacterEncoding("UTF-8"); // 告知服务器

response.setHeader("content-type", "text/html;charset=UTF-8");//告知客户端

一句顶两句：response.setContentType("text/html;charset=utf-8");

响应图片

resp.setHeader("Content-Type", "image/gif");

1）、找源头，找到图片的真实位置； 抽象成一个File对象； 用字节输入流对接

2）、定位目的地， 通过response获取OutputStream

3）、读取 写出 循环

# 6. HttpSession对象

HttpSession对象的获取

该对象 是由服务器产生

HttpSession s = request.getSession();

常用方法

setAttribute(String name,Object value)：将value对象以 name 名称绑定到会话 getAttribute(String name)：取得 name 的属性值，如果属性不存在则返回 null removeAttribute(String name)：从会话中删除 name 属性，若不存在不会执行，也 不会抛处错误。

invalidate()：使会话失效，同时删除属性对象

isNew()：用于检测当前客户是否为新的会话

setMaxInactiveInterval(int seconds)：允许客户客户请求的最长时间

String getId()：返回会话期间的识别号

session id

对于每一个用户而言，实际上都表示一个个不同的 session，服务器通过 session id 来 区分这些用户

session的产生和失效

session的产生：意味着一次会话的开始

打开浏览器，在服务器端 getSession， 去访问一下session

session的失效：意味着一次会话的结束（）

域对象

HttpServletRequest、ServletContext 、HttpSession 都是域对象，它们三个是 Servlet 中可以使用的域对象

HttpServletRequest：一个请求创建一个 request 对象，所以在同一个请求中可以共 享 request

ServletContext：一个应用只创建一个 ServletContext 对象，所以在 ServletContext 中的数据可以在整个应用中共享，只要不重启服务器。

HttpSession：一个会话创建一个 HttpSession 对象，同一会话中的多个请求中可以共 享 session 中的数据；

会话跟踪

①http 协议无状态

②为了在一个会话中多个请求间共享数据

③前面的请求转发，数据只能在一个请求

④重定向，不能共享数据

# Cookie

Cookie 是浏览器提供的一种技术，通过服务器的程序能将一些只须保存在客户端，或 者在客户端进行处理的数据，放在本地的计算机上，不需要通过网络传输，因而提高网 页处理的效率，并且能够减少服务器的负载，但是由于 Cookie 是服务器端保存在客户 端的信息，所以其安全性也是很差的。

Cookie 就是一个键和一个值构成的，由服务器创建，客户端保存

设置 Cookie：在服务器端 ,发送到客户端 response.addCookie();

取得 Cookie：request.getCookies();

获取 Cookie：Cookie.getName+Cookie.getValue()

**Cookie的产生**

a、服务器自动产生 JSESSIONID

b、手动生成并添加给response的

**Cookie路径的设置**

一个servlet/jsp设置的cookie能够被同一个路径下或子路径下的servlet/jsp读到（这里的路径=URL）

**设定Cookie的过期时间：cookie.setMaxAge();**

正数： 可以存活从存放时间后的m秒

负数： 关闭浏览器

零 ： 立刻马上删除

**Cookie 保存中文**

URLEncoder.encode(name,"utf-8");

URLDecoder.decode(coo.getName(), "utf-8");

# ServletContext 对象

一个 web 应用只有一个 ServletContext 对象，所有的 servlet 都共享这个 ServletContext 对象，又称为 Application 对象。

**ServletContext的作用**

1）、作为域对象， 最大的一个域对象

2）、可以获取一些和web应用程序相关的内容

3）、获取和web应用程序相关的服务器的信息

ServletConfig对象中持有该对象（ServletContext）的引用

ServletConfig: 由服务器创建， 通过init方法赋值给 Config的成员变量，然后再通过一个getServletConfig的方法，返回出这个Config对象， 则外界可以获取该对象，并加以使用了。

**ServletContext对象的获取：**

1、通过ServletConfig对象获取

2、通过Servlet对象（类）的 getServletContext方法获取

3、通过request对象的 getServeltContext方法获取

# 9. 域对象

HttpRequest:同一次请求，一次请求（可能存在服务器请求转发）一次响应

HttpSession:同一次会话，浏览器开始访问服务器；浏览器关闭；服务也可以终止 此次会话；关闭服务（正常，中止；非正常关闭，会话被迫终止）；

ServletContext: 整个应用程序中都有效

文件上传：enctype=”nultipart/form-data

文件下载：response.setHeader("content-disposition","attachment;filename="+filename);

1. 利用超链接的方式下载

缺点： 只有不认识的内容即浏览器不能直接打开的内容浏览器会自动下载；当浏览器 碰见自己能认识的/能打开的内容时，就会直接显示出来

1. 利用第三方下载方式

①获取要下载的文件名

②相对路径转绝对路径

③向响应的头部添加信息

# 10. 过滤器

作用：①在客户访问后台资源之前拦截客户请求

②在资源被送到客户端之前加以控制

③Servlet 过滤器负责过滤的 Web 组件可以是 Servlet、JSP 或 HTML 文件。

1、过滤器链：在一个 Web 应用程序中，可以部署多个过滤器进行拦截，这些过滤器组成了一个过滤器链。

注意：若是一个过滤器链：先配置先执行(请求时的执行顺序)；响应时: 以相反的顺序执行。

2、Filter 接口: javax.servlet.Filter

init()：初始化配置参数，在 doFilter()方法之前被调用

doFilter()：该方法在客户端请求及服务器端响应时自动调用(反复执行)

destroy()：结束过滤器，doFilter()方法完成后调用。(通常服务器关闭时销毁)

Filter 生命周期(饿汉单例)：创建、初始化、服务、销毁

web.xml 注意位置：在监听器及 servlet 之上

url-pattern 的配置

①配置具体路径/index.html/TestServlet.do

②带有通配符的配置\*.do /\* /user/\* \*.html \*.jsp

url-pattern 可以用 servlet-name 和 dispatcher 替代

# 11. 监听器

监听器是一个用来监听其他对象的一个对象

servlet下的监听器主要控制**application(SrevletContext) session request**对象的发生的特定事件

监听器有三类 8 种：

1. 监听生命周期：实现接口 ServletRequestListener、 HttpSessionListener、ServletContextListener
2. 监听值的变化：实现接口 ServletRequestAttributeListener、HttpSessionAttributeListener、 ServletContextAttributeListener
3. 针对 session 中的对象:监听 session 中的 java 对象 (javaBean) 是 javaBean 直接实现监听器的接口

# 12. Session 中的 javaBean 对象

1、绑定监听对象

监听 Session 中的 java 对象(javaBean)：其不需要在 web.xml 中配置，直接在对象上实现监听器接口 HttpSessionBindingListener(监听对象绑定到 Session 中)。

2、Session 钝化和活化

①为了提高服务器端的效率，节约内存的消耗

②当 Web 服务器产生故障时，可以从文件系统或数据为中恢复 Session 的数据

钝化：当 Session 在一定的时间内没有被使用时，把 Session 序列化到磁盘上(从内存写 入硬盘)，不占用服务器的资源和内存。

活化：将钝化的 Session 重新加载到服务器端的内存中(从硬盘上读取到内存中)

若需要同时保存 Session 中的 JavaBean 则 JavaBean 也要**实现 Serializable 序列化接口**

注意：session 的钝化活化与 HttpSessionActivationListener 没有关系,只是可以通 过实现该接口**监控钝化与活化的过程**

# 13. Servlet3.0

@WebServlet:有了此注解，就不用配置web.xml

@WebInitParam 表示一个参数。

@WebFilter：标注过滤器

@MultipartConfig: 将一个 Servlet 标识为支持文件上传, Servlet3.0 将 multipart/form-data 的 POST 请求封装成 Part，通过 Part 对上传的文件进行操作。

@WebListene：将一个实现了特定监听器接口的类定义为监听器。