Tomcat文件conf：配置文件，server.xml：用于配置server相关的信息如tomact启动端口，配置Host，配置Context即web应用。

Web.xml：配置与web应用

Tomcat-users.xml：用户配置tomcat的用户密码和权限

<http://localhost:8080/web/index.html：URL>（统一资源定位）

http:协议，localhost：主机域名，8080：端口，web：web应用，index.html：资源文件

web/hello.html：URI

work：存放jsp被方问后生产的对应的servlet文件class文件

web应用下的web.xml文件中配置默认首页面：

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

配置虚拟目录在tomcat的conf目录下的server.xml的<Host>节点中添加<Context path=”/web应用名” docBase=”文件绝对路径” />

Context元素reloadable设置有文件更新tomcat会自动重新加载，unpackWAR设置运行前解压所有web应用

tomcat体系架构从大到小：tomcat（server），service，engine（引擎），host（主机），context（web应用）

开发servlet的三种方法：实现servlet接口，继承GenericServlet类，继承HttpServlet类

Servlet的init（ServletConfig config）初始化servlet，函数只会调用一次

将Servlet部署到web.xml文件

<servlet>

<servlet-name>servlet的注册名</servlet-name>

<servlet-class>servlet的绝对路径</servlet-class>

</servlet>

<servlet>

<servlet-name>servlet的注册名</servlet-name>

< url-pattern >servlet的访问路径（格式为：/名字）</url-pattern>

</servlet>

Servlet的访问路径使用通配符的格式为：/\*表示匹配所有，\*.扩展名

匹配标准：匹配度高优先选择，\*.扩展名优先级最低

使用javac编译有自定义包名的java文件，需要带命令参数格式为：javac –d .java文件

提交方式get与post的区别：get会将提交的信息显示到地址栏，get提交内容一般不大于2k，post理论无限制，get提交会立即处理响应。

Servlet配置<load-on-startup>启动优先级（数字）</load-on-startup>指定servlet的创建顺序

<init-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

<init-param>通过getServletConfig()函数读取

setContentType("text/html;charset=utf-8");设置浏览器默认解析的编码格式

setCharacterEncoding("utf-8");设置程序编码格式

Servlet请求重定向函数sendRedirect(“跳转地址?参数1名=参数1值&参数2名=参数2值”)

超文本传输协议（HTTP，HyperText Transfer Protocol）

http1.0称为短连接，http1.1称为长连接，是指请求时间间隔的长短

http的请求基本结构：

请求行（请求方式，请求的资源名称，http协议版本）

消息头（格式 消息名：内容）

空行

内容（格式：内容名字=内容体）

http请求消息头：

Accept：浏览器接受的格式

Accept-Charset：浏览器接受字符编码

Accept-Encoding：浏览器接受数据的格式

Accept-Language：浏览器支持的语言

Host：浏览器访问的主机和端口

If-Wodified-Since：浏览器缓存文件及时间

Referer：浏览器访问主机方式

Use-Agent：浏览器内核信息

Cookie：网页信息

Connection： 连接方式

Date：浏览器发送http请求时间

http的响应基本结构：

状态行（格式 http版本号 状态码 原因）

消息头（格式 消息名：内容）

空行

内容（格式：内容名字=内容体）

http响应消息头：

Location：浏览器重新定位地址

Server：服务器信息

Content-Encoding：数据格式

Content-Length：数据大小

Content-Language：数据语言

Content-Type：内容格式

Last-Modified：资源最近更新的时间

Refresh：多少时间后跳转到指定地址

Content-Disposition：attachment;filename=文件名称

Transfer-Encoding：传输编码，校验和

Set-Cookie：cookie信息

Expires：缓存数据方式（为了支持不同浏览器内核设置3个缓存数据方式）

Cache-Control：缓存数据方式

Pragma：缓存数据方式

Connection： 连接方式

Date：浏览器收到http响应时间

getWriter()用于回送字符数据

getOutputStream()用于回送字节数据（二进制数据）

由于web服务器会检测数据流是否关闭，未关闭服务器会关闭，所以getWriter()与getOutputStream()同时使用会第二个无法正常传输数据。

中文乱码原因：浏览器根据不同的请求提交方式有不同编码格式，tomcat服务器默认编码格式为iso-8859-1

解决方法：

浏览器显示编码格式设置：

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

response.setCharacterEncoding("utf-8");

post请求提交：request.setCharacterEncoding(“utf-8”);

get请求提交：new String(string.getBytes(“iso-8859-1”),”utf-8);

下载文件提示框编码格式设置：java.net.URLEncoder.encode(“中文名”,”utf-8”);

getHeader(“参数名”)返回请求头信息中参数名的值

服务器内部转发request.getRequestDispatcher("/转向的Servlet名").forward(request, response);转发后的request和response与转发前是同一个request和response

浏览器转发response.sendRedirect(“/web应用名/资源名（servlet）”)转发是在，request和response对象与转发之前不是同一个

分页变量

pageNow：现在显示第几页

pageSize：每页显示多少条记录

pageCount：共有多少页

rowCount：共有多少条记录

java web读取文件使用类加载器，类加载器读取资源默认是主目录是src目录

inputStream=类名.class.getClassLoader().getResourceAsStream("文件名");

浏览器的Cookie是在服务器创建会保存到浏览器

Cookie 名字=new Cookie(属性名, 属性值);cookie只能存放String类型数据

名字.setMaxAge(正数);设置cookie存放时间（单位是秒）

名字.setMaxAge(0);在浏览器退出后删除cookie

名字.setMaxAge(负数);负数与默认不设置一样在关闭浏览器时清除

response.addCookie(名字);cookie通过http响应返回给浏览器并保存

cookie可以多个浏览器共享

一个浏览器最多存放300个cookie，一个web站点最多存放20个cookie，每个cookie大小最大为4k

浏览器的cookie读取

Cookie 名字[]=request.getCookies();通过request类读取返回所有cookie

名字.getName()获得cookie的属性名，重名的cookie值会被替换

名字.getValue()获得cookie的属性值

一个web应用可以保存多个cookie，有最大数量限制

一个web应用的所有cookie都存放在同一个文件里

Cookie存放中文时可能会出现乱码或者错误，因为cookie中文编码格式默认为base64

cookie存放中文通过java.net.URLEncoder.encode(需要编码的值, 编码格式);进行编码

cookie取出中文通过java.net.URLDecoder.decode(需要解码的值, 解码格式);进行解码

cookie默认是明文存放不安全重要信息需要加密

密码加密存放，一般是用户输入密码进行加密后再存放到数据库，验证时对用户输入的密码加密后到数据库对比验证

Session是存在服务器端的内存中，服务器为每个用户的浏览器创建各自独享的session对象

服务器未发现session会自动创建一个，一般都是在浏览器第一次访问时创建

Session 名字=request.getSession(参数)获得服务器的session，参数为false表示服务器未找到session不去创建新的session，参数为true与默认不填表示服务器未找到session会创建新的session

session默认存在时间30分钟（单位是分钟），服务器（tomcat）下所有web应用的所有session默认时间修改在服务器（tomcat）的web.xml文件中，一个web应用下的所有session默认时间修改在相应web应用下的web.xml文件中，一个session的默认时间设置是名字.setMaxInactiveInterval(时间);设置session名为名字的存在时间（单位是秒），默认时间的设置优先级从大到小为session，web应用，服务器

名字.setAttribute(属性名（字符串）, 属性值（object）);设置session内容，属性名一样的值将会被替换

名字.getAttribute(参数);返回session中属性名为参数的值，返回类型为object

名字.removeAttribute(参数);删除session中属性名为参数的值，让session中属性名为参数值失效

重启服务器（tomcat），重新加载（reload）web应用，关机等操作会导致session失效

名字.invalidate();设置session立即失效（包含session中所有属性），一般用于安全退出

List集合有序，map集合无序

用户禁用浏览器的cookie后的session处理方法

response. encodeRedirectURL(url地址);用于对sendRedirect方法后的url地址进行重写

response.encodeURL(url地址);用于对表单提交和超链接的url地址进行重写

ServletContext是在服务器端web应用启动时创建的，一个ServletContext被改web应用的所有客户端共享，

ServletContext可以通过ServletContext 名字=this.getServletContext();方法获得也可以通过

ServletContext名字=this.getServletConfig().getServletContext();方法获得

ServletContext的生命周期从创建开始到web应用服务器关闭为止

web应用下的web.xml文件servlet标签中配置

<init-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

</init-param>只会被该servlet访问

访问使用String 名字=this.getServletConfig().getInitParameter("参数名");

web应用下的web.xml文件中配置

<context-param>

<param-name>参数名</param-name>

<param-value>参数值</param-value>

</context-param>会被该web应用所有servlet访问

访问使用String 名字=this.getServletContext().getInitParameter("参数名");

ServletContext服务器web应用内部实现跳转this.getServletContext().getRequestDispatcher("跳转地址");

ServletContext读取文件

InputStream 名称=this.getServletContext().getResourceAsStream("资源名称及位置");

属性读取对象Properties

properties.load(inputStream流对象);加载文件流

properties.getProperty("属性名");获得属性值

获取文件的绝对路径String 名字=this.getServletContext().getRealPath("资源文件名称");

ServletContext能读取web目录和WEB-INF目录下的文件

其他文件下需要使用类加载器this.getClass().getClassLoader().getResourceAsStream("资源文件名称");

统计页面访问次数不包含网页刷新的方法

将页面跳转方式服务器内部跳转request.getRequestDispatcher("页面地址").forward(request, response);换成浏览器跳转response.sendRedirect("页面地址");