# web开发入门

## 引入

软件结构分类：

CS结构：客户端和服务器端

特点：

1．必须安装特点的客户端程序

2．服务器端升级，客户端同步升级

BS结构：浏览器和服务器端

特点：

1．不需要安装特点的客户端（只需要浏览器）

2．服务器端升级，客户端不需要升级的。

## 服务器

从物理上一台PC机器，配置比较高。

web服务器：在PC机器安装web服务软件，提供web服务

数据库服务器：在PC机器安装了数据库软件，提供数据管理服务

邮件服务器：在PC机器上安装了可以收发邮件服务软件

## web服务软件

web服务软件作用： 把本地资源文件共享给外部访问

|  |
| --- |
| /\*\*  \* socket服务器端程序  \* **@author** APPle  \*  \*/  **public** **class** Server {  **public** **static** **void** main(String[] args) **throws** Exception {  //1)创建ServerSocket对象,监听某个端口  ServerSocket server = **new** ServerSocket(8888);    System.*out*.println("服务器正在启动....");    **while**(**true**){  //2)接收客户端连接  Socket socket = server.accept();    //3)把本地的hello.html文件内容发送给socket客户端  //3.1 构建数据发送通道  OutputStream out = socket.getOutputStream();    //3.2 读取本地hello.html文件内容  FileInputStream fis = **new** FileInputStream("e:/web/hello.html");  /\*\*  \* 输入流  \* 输入字节流： InputStream  \* FileInputStream  \* BufferedInputStream  \*  \* 方法：  \* read(): 读取一个字节  \* read(byte[] data): 读取多个字节  \* 如何判断文件结束？ 返回的长度等于-1  \*  \* 输入字符流: Reader  \* FileReader  \* BufferedReader  \*  \* 方法：  \* read(): 读取一个字符  \* read(char[] data): 读取多个字符  \*/  //边读边写  /\*int len=0;  while( (len = fis.read())!=-1 ){  out.write(len);  }\*/    //使用字节数组形式  **byte**[] buf = **new** **byte**[1024];  **int** len = 0;  **while**( (len=fis.read(buf))!=-1 ){  out.write(buf, 0, len);//写出指定的buf数组内容，从0开始写，写到len位置  }    //4.关闭资源  fis.close();  out.close();  }  }  } |

## 市面上常见的web服务软件

WebSphere： 是IBM公司。是支持JavaEE规范的。收费的。

IIS：是微软公司。收费的。和net语言兼容性最好。

WebLogic：是BEA公司。收费的。支持JavaEE规范的。

**Tomcat**： 免费的。和java语言兼容性最好。不完全支持javaEE规范的。支持servlet，jsp.jdbc，javamail，但是不支持ejb

# tomcat服务器基本使用

## 基本使用

1．到apache开源组织下的官网下载

2．下载的版本：

**推荐下载压缩版**

3．启动软件

3.1 找到%tomcat%/bin/startup.bat。双击。

验证是否成功，打开浏览器：

http://localhost:8080

3.2 看到tomcat的页面，就证明启动了！

4．关闭软件

4.1 找打%tomcat%/bin/shutdown.bat.双击

## 体验tomcat服务器如何共享文件

规则：把文件拷贝到webappps目录下，不能单独共享文件，必须以目录（网站）形式共享

## URL

http://localhost:8080/web/hello.html 统一资源定位符

http:// 协议：http协议

执行流程：

1.本地hosts文件， ip地址 域名 （c:/windows/System32/drivers/etc/hosts）

2．在hosts文件中找不到，到网络运营商的DNS服务器中找域名对应的IP地址。

3．找到，访问对应的ip地址的PC机器.

localhost 域名：

本地域名： localhost

外部域名： www.baidu.com等

8080 端口：软件监听端口

qq: 5555

tomcat: 8080

mysql: 3306

oracle: 1521

系统默认端口： 80

注意： 一个软件监听了默认端口，那么访问这个软件不需要写80.

web 网站名称。（web应用的名称）

hello.html 网站中的资源名称。

## tomcat启动常见的问题

1．闪退

原因： tomcat软件使用java语言开发，需要jvm运行。tomcat软件在启动时寻找JAVA\_HOME环境变量，JAVA\_HOME配置的是jdk的目录。

解决办法：配置JAVA\_HOME环境变量

2．端口占用的问题

原样：其他软件在占用端口

解决办法：

a）关闭其他软件

b)修改tomcat软件监听的端口

3．CATALINA\_HOME变量的问题

# Tomcat服务器目录结构

|-bin目录： 存在tomcat软件操作指令。

启动：startup.bat catalina.bat start

关闭：shutdown.bat catalina.bat stop

|-conf目录： 存在tomcat软件的配置文件

核心配置文件：server.xml

|-lib目录： 支持tomcat软件运行的jar包，因为tomcat使用java语言开发的。

|-logs目录： 存放tomcat运行的日志信息

|-temp目录： 存放临时文件

|-webapps目录： 存放web应用(网站)的目录

|-work目录： tomcat运行目录，存放jsp页面运行过程中产生的临时文件。

# 网站（web应用）目录结构

|-WebRoot 根目录

|- 普通文件（html,css,js,images）

|- WEB-INF 目录。固定写法

|-classes 目录。固定写法。存放class文件

|-lib 目录。固定写法。存放jar包。

|-web.xml 文件。用于存储网站配置信息

注意 ：

1．WEB-INF外面的文件是可以直接通过浏览器访问的。

2．但是WEB-INF里面的文件是不可以通过浏览器直接访问的。如果想让WEB-INF里面的内容提供给外部访问，那么需要在web.xml文件中配置。

# 如何发布网站（web应用）

## 把整个网站拷贝到webapps目录下

**ROOT是默认网站，访问默认网站是不需要名称的。**

配置首页

**写在WEB-INF 文件夹的web.xml文件中**

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  version="2.5">  <!-- 配置网站的首页 -->  <welcome-file-list>  <welcome-file>hello.html</welcome-file>  </welcome-file-list>  </web-app> |

访问首页不需要名称的！

弊端： tomcat软件和发布的网站是在一起的。

## 配置虚拟网站

|  |
| --- |
| <!-- 配置虚拟网站  docBase: 网站绝对路径  path: 访问网站的名称  -->  <Context docBase="C:\web" path="/itcast"/> |

**好处：软件和网站是分离的。**

注意： path可以不写内容，表示当前网站是默认网站，优先于ROOT的网站。

**弊端：修改server.xml文件，风险比较大**

## 通过添加xml文件的方式配置虚拟网站[推荐]

**1．在conf/Catalina/localhost 添加news.xml文件**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <Context docBase="C:\web"/> |

**注意：访问网站的使用xml的文件名。**

http://localhost:8080/news

# 站点配置

**在conf文件夹的server.xml文件中**

|  |
| --- |
| <Host name="localhost" appBase="webapps"  unpackWARs="true" autoDeploy="true"  xmlValidation="false" xmlNamespaceAware="false">  <!-- SingleSignOn valve, share authentication between web applications  Documentation at: /docs/config/valve.html -->  <!--  <Valve className="org.apache.catalina.authenticator.SingleSignOn" />  -->  <!-- Access log processes all example.  Documentation at: /docs/config/valve.html -->  <!--  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"  prefix="localhost\_access\_log." suffix=".txt" pattern="common" resolveHosts="false"/>  -->  </Host> |

name: 表示站点名称。访问站点就是使用站点名称。

appBases： 表示站点根目录

站点和网站的关系？

一个站点可以包含多个网站。

一个网站就在一个站点中。

## 添加站点：

1．在server.xml文件中添加host标签

|  |
| --- |
| <!-- 新建站点-->  <Host name="www.baidu.com" appBase="E:\baidu"  unpackWARs="true" autoDeploy="true"  xmlValidation="false" xmlNamespaceAware="false">  </Host> |

2．在本地的hosts文件，中添加一个站点和ip地址的映射

127.0.0.1 www.baidu.com

3．访问站点的百度音乐项目

http://www.baidu.com/music/index.html

# 动态网站的开发

html/css/javascript 前端（静态网页）

静态网站： 只包含静态网页

动态网站： 既有静态网页，可以有动态网页

静态网页：用户多个访问这个网页时，网页的源码是不会改变的。

动态网页：用户多个访问这个网页时，网页的源码是有可能改变的。

## 使用servlet技术开发动态网页

Servlet特点：

1．servlet就是一个普通java类，继承HttpServlet类

2．servlet交给tomcat服务器运行（web服务器软件）！！！

将**写在WEB-INF 文件夹的web.xml文件中**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <web-app xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  version="2.5">  <!--配置servlet -->  <servlet>  <!-- servlet名称，自定义-->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet类的全名：包名+类名 -->  <servlet-class>gz.itcast.b\_servlet.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <!-- servlet名称，和上面的名称保持一致 -->  <servlet-name>HelloServlet</servlet-name>  <!-- 公开给外部访问这个servlet的名称（路径） -->  <url-pattern>/hello</url-pattern>  </servlet-mapping>  </web-app> |

# Tomcat总结：

1．web服务软件作用：把本地资源共享给外部访问

2．tomcat服务软件基本使用：

2.1 启动： startup.bat

2.2 关闭： shutdown.bat

3．网站的目录结构

|-webRoot 根目录

|-文件

|-WEB-INF

|-classes

|-lib

web.xml 配置servlet,配置首页

## 部署网站的三种方式

1．直接拷贝到webapps目录

2．虚拟网站

server.xml

<Context docBase="网站的绝对路径" path="/访问网站名称"/>

3．添加xml文件方式配置虚拟网站

在conf/Catalina/locahost

添加bbs.xml

<Context docBase="网站的绝对路径"/>

访问网站名称就是xml的文件名称

## 增加站点

1．server.xml添加host标签

2．修改hosts文件

3．访问站点使用站点名称

## 手动开发一个servlet

**1．编写一个普通的java类，继承HttpServlet类，覆盖doGet方法**

注意： 到tomcat的lib目录下拷贝servlet-api.jar导入到你的项目中

public class HelloServlet extends HttpServlet{

//覆盖doGet

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

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

//向浏览器输出内容

resp.getWriter().write("当前时间为："+new Date());

}

}

2．把servlet交给tomcat服务器运行！！！！！！

2.1 把servlet对应的class文件拷贝到tomcat的某个项目（bbs）的WEB-INF/classes目录下

**2.2 在bbs项目的WEB-INF/web.xml文件配置servlet**

<?xml version="1.0" encoding="utf-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

version="2.5">

<!--配置servlet -->

<servlet>

<!-- servlet名称，自定义-->

<servlet-name>HelloServlet</servlet-name>

<!-- servlet类的全名：包名+类名 -->

<servlet-class>gz.itcast.b\_servlet.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<!-- servlet名称，和上面的名称保持一致 -->

<servlet-name>HelloServlet</servlet-name>

<!-- 公开给外部访问这个servlet的名称（路径） -->

<url-pattern>/hello</url-pattern>

</servlet-mapping>

</web-app>

3．通过URl访问这个servlet

http://localhost:8080/bbs/hello

## 工具开发一个servlet

1．创建一个bbs的web project

2．在WebRoot下编写静态网页

3．在src下编写动态网页

3.1 在src右键选择 “servlet”

3.2 写上servlet的包，名，选择覆盖doGet方法 -> next

3.3 修改servlet的名称，修改访问servlet的url -> finished

4．myecplise关联tomcat服务器

4.1 window-> preferences -》 搜索“tomcat” -> 找到“tomcat6.x”

4.2 选择 tomcat的根目录

4.3 把disable -> enable

5．部署网站

5.1 点击发布图标 -> 选择网站 -> 选择发布的tomcat服务器 -> ok

6．访问：

静态文件： http://localhost:8080/bbs/index.html

动态文件： http://locahost:8080/bbs/hello

回顾重点内容：

1．web服务软件：把本地资源文件共享给外部访问

2．tomcat基本使用

3．web应用（网站）目录结构

|-WebRoot 根目录

|- 普通静态网页文件（html、css。。。）

|-WEB-INF

|-classes 存放class文件。servlet类

|-lib 存放jar包

web.xml 文件。配置servlet

4．部署web应用三种方式

4.1）直接拷贝webapps

4.2）修改server.xml

在host标签中添加 <Context docBase="" path=""/>

4.3)直接在conf/Catalina/localhost目录添加xml文件

<Context docBase=""/>

5．开发动态网页

servlet技术：使用java语言来开发动态网页的技术

开发servlet的步骤：

5.1 编写java类，继承HttpServlet，覆盖doGet方法

5.2 把servlet的class文件拷贝到项目的WEB-INF/classes目录下

5.3 在项目的WEB-INF/web.xml文件配置

<servlet>

<servlet-name></servlet-name>

<servlet-class></servlet-class>

</servlet>

<servlet-mapping>

<servlet-name></servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

5.4 访问servlet

http://localhost:8080/web/hello

# http协议

## 概念

是对浏览器和服务器端数据传输格式的规范！

# http协议内容

## Http请求

|  |
| --- |
| GET /day08/first HTTP/1.1 **-- 请求行**  Host: localhost:8080 **-- 请求头（以键值对形式出现）**  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: en-GB,en-US;q=0.8,zh-CN;q=0.6,zh;q=0.4,en;q=0.2  Accept-Encoding: gzip, deflate  Connection: keep-alive  **-- 一个空行**  name=eric&password=123456 **-- 实体内容** |

### 请求行

GET /day08/first HTTP/1.1

HTTP/1.1

http协议版本：

http1.0: 当浏览器和服务器连接之后，在一次连接中只能发出一个请求

http1.1:当浏览器和服务器连接之后，在一次连接中可以发出多次请求。（效率比1.0更高）

**常见问题：**

**在一个html页面，在页面上有三张图片，访问这个html页面，问一共发出几次请求？**

**答案：4次**

/day08/first

请求资源：请求服务器的哪个资源文件

/day08/first URI：统一资源描述符。可以描述任何文件，本地，互联网

http://localhsot:8080/day08/first URL：统一资源定位符（基于http协议） ,是URI的子集。

GET

请求方式（提交方式）：

http协议中的请求方式：GET、POST、HEAD、PUT、CONNECT……

最常用的的请求方式： **GET和 POST**

#### GET和 POST的区别：

**使用上都是form表单的method属性**

<form action="提交的地址" method="get/post">

</form>

##### GET：

**1．提交的参数显示到地址栏**

原理：get方式提交的参数会跟在请求行内容的URI后面。以？开始，每个参数使用=号分开,name=jacky,第二个参数之后使用&符合连接

|  |
| --- |
| GET /day08/testMethod.html?name=jacky&password=123456 HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: en-GB,en-US;q=0.8,zh-CN;q=0.6,zh;q=0.4,en;q=0.2  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day08/testMethod.html  Connection: keep-alive |

**2**．**不适合存放敏感数据**

**3**．**不超过1kb**

##### POST：

**1．提交的参数不会显示到地址栏**

原理：post提交的参数跟在请求的实体内容中。第二个参数用&连接

|  |
| --- |
| POST /day08/testMethod.html HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:37.0) Gecko/20100101 Firefox/37.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: en-GB,en-US;q=0.8,zh-CN;q=0.6,zh;q=0.4,en;q=0.2  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day08/testMethod.html  Connection: keep-alive  name=jacky&password=123456 -实体内容 |

**2．适合敏感数据**

**3．提交的大小没有限制**

### 请求头

|  |
| --- |
| Accept: text/html,image/\* **-- 浏览器接受的数据类型**  Accept-Charset: ISO-8859-1 **-- 浏览器接受数据编码格式**  Accept-Encoding: gzip,compress **-- 浏览器接受的数据压缩格式**  Accept-Language: en-us,zh-cn **-- 浏览器接受的语言**  Host: www.it315.org:80 **-- 请求发出的主机和端口（必须）**  If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT **-- 浏览器缓存的最后修改时间**  Referer: http://www.it315.org/index.jsp **-- 当前请求来自于哪里 （防止非法链接）**  User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) **-- 浏览器的类型**  Cookie:name=eric **-- 浏览器保存的cookie数据**  Connection: close/Keep-Alive **-- 浏览器和服务器的连接状态。close：关闭。keep-alive:保持连接**  Date: Tue, 11 Jul 2000 18:23:51 GMT **-- 请求发出的时间** |

### Http请求的原理：

1．tomcat服务器接收到浏览器发送的请求数据

2．tomcat服务器把请求数据封装成HttpServletRequest对象

3．tomcat服务器调用doGet方法，把request对象传入servlet

4．通过request对象获取请求数据

### 获取参数数据

#### service方法和doXXX方法的关系？

1．service方法是程序的入口。我们的代码逻辑就在这个方法被调用到。

2．在HttpServlet的service方法源码中，根据不同请求方式调用了不同的doXX方法，

所以我们在开发servlet的时候，就不需要去覆盖service方法，而是去doXX方法。

因为get和post是最常用的的两种请求方式，所以只需要覆盖doGet和doPost方法即可！

//doGet方法只能接受get提交的请求

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

**//使用get提交的表单参数放在请求行的URI后面**

}

//doPost接受post提交的请求

protected void doPost(HttpServletRequest request, HttpServletResponse resp)

throws ServletException, IOException {

**//使用post提交的表单参数放在请求的实体内容**

}

#### 使用HttpServletRequest获取请求信息

HttpServletRequest对象：获取请求数据

##### 获取请求行信息：

请求方式： request.getMethod()

请求资源： request.getRequestURI() / request.getRequestURL()

http协议版本： request.getProtocol()

**例：**

System.out.println("请求方式："+request.getMethod()); //请求方式

System.out.println("URI:"+request.getRequestURI()); //请求URI资源

System.out.println("URL:"+request.getRequestURL()); //请求URL资源

System.out.println("Http协议："+request.getProtocol()); // http协议版本

##### 获取请求头信息

根据头名称获取头值：request.getHeader("name")

获取所有头名称：request.getHeaderNames()

**例：**

System.out.println("host:"+ request.getHeader("host")); //根据头名称获取头值

**//使用Enumeration遍历所有头和值**

Enumeration<String> enums = request.getHeaderNames(); //获取所有头名称

while(enums.hasMoreElements()){ // **hasMoreElements()判断是否还有元素**

String headerName = enums.nextElement(); // **nextElement()返回当前元素，并指向下一个元素**

String headerValue = request.getHeader(headerName); //根据头名称获取头值

System.out.println(headerName+":"+headerValue); //输出所有头和值

}

##### 获取实体内容信息

返回实体内容：request.getInputStream();

**例：**

**//输出所以的实体内容**

**InputStream in = request.getInputStream();**

byte[] buf = new byte[1024];

int len = 0;

while( (len=in.read(buf))!=-1 ){

String str = new String(buf,0,len);

System.out.print(str);

}

##### 获取请求的参数数据

通用的获取参数的方法：

获取一个值的参数：reuqest.getParameter("name")

获取所有参数名称：request.getParameterNames()

获取所有参数对象：request.getParameterMap()

获取多个值的参数：request.getParameterValue("name")

**（参数名称就是表单的name属性值）**

不通用也不方便使用：

获取get提交的参数：request.getQueryString(); URI后面的参数数据

**例：**

String name = request.getParameter("name"); //根据参数名称获取参数值

**//遍历所有的参数**

Enumeration<String> enums = **request.getParameterNames()**; //获取所有参数名称列表

while(enums.hasMoreElements()){ // **hasMoreElements()判断是否还有元素**

String paramName = enums.nextElement(); // **nextElement()返回当前元素，并指向下一个元素**

String paramValue = **request.getParameter(paramName)**; //根据参数名获取参数值

System.out.println(paramName+"="+paramValue);

}

Map<String,String[]> map= **request.getParameterMap()**; //获取参数对象列名 （Map集合）

/\*\*

\* 每个map的对象就是一个参数（包含参数名称和参数值）

\* key: 参数名称

\* value: 参数值(默认情况下都是多个值的参数)

\*/

**//方法一：entrySet()方法：获取键值对象的Set集合（推荐）**

// Entry对象中包含一个键对象，和一个值对象

**Set<Entry<String,String[]>> entrySet = map.entrySet();**

for (Entry<String, String[]> entry : entrySet) {

//获取键对象

String key = entry.getKey();

//获取值对象(数组的第一个元素就是参数值)

String[] value = **entry.getValue()**;

System.out.println(key+"="+value[0]);

}

**//方法二：keySet()： 获取所有键对象的Set集合**

**Set<String> keySet = map.keySet();**

for (String key : keySet) {

//通过键对象获取值对象

String[] value = **map.get(key)**;

System.out.println(key+"="+value[0]);

}

**//方法三：values(): 获取所有值对象的Collection集合(只能获取值对象，不能获取键对象)**

**Collection<String[]> values = map.values();**

for (String[] value : values) {

System.out.println(value[0]);

}

String[] hobits = **request.getParameterValues("hobit")**; //根据参数名称获取多个参数值

for (String h : hobits) {

System.out.print(h+",");

}

##### 获取post提交的参数（实体内容中）

**//因为是通用的在doPost方法中调用doGet里面的逻辑代码即可实现和doGet一样的方法了**

doGet(request,response);

**//输出所以的实体内容（获取实体内容的例子中有）**

ServletInputStream in = request.getInputStream();

byte[] buf = new byte[1024];

int len = 0;

while( (len=in.read(buf))!=-1 ){

String str = new String(buf,0,len);

System.out.print(str);

}

**//遍历所有的参数（获取请求的参数数据例子中有）**

**Enumeration<String> enums = request.getParameterNames(); //获取所有参数名称列表**

while(enums.hasMoreElements()){

String paramName = enums.nextElement();

String paramValue = request.getParameter(paramName);

System.out.println(paramName+"="+paramValue);

}

#### 解决乱码问题

##### 解决get乱码问题：

**手动解码：在获取到参数值后手动转码**

|  |
| --- |
| String name = request.getParameter("name");  //根据参数名称获取参数值（参数名称就是表单的name属性值  **if**("GET".equals(request.getMethod())){  name = **new** String(name.getBytes("iso-8859-1"),"utf-8");  } |

##### 解决post乱码问题：

**设置参数解码时查询的码表**

注意：

只对post提交的参数有效，对get提交的参数无效的！

因为这个方法只能设置对请求实体内容的解码时查询的码表。post提交的参数时放在实体内容中，所以对post是有效的！

但是get提交参数时放在URI后面的，这个方法无法影响URI后面的内容。

|  |
| --- |
| request.setCharacterEncoding("utf-8"); |

### 案例1 user-agent头 -- 获取浏览器类型

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

**//以html去读取发送过去的文本，编码为utf-8**

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

String userAgent = **request.getHeader("user-agent"); //获取user-agent头的值**

if(**userAgent.contains("Firefox"**)){ **// contains()判断是否包含这些字符串**

response.getWriter().write("你正在使用火狐浏览器");

}else if(userAgent.contains("Chrome")){

response.getWriter().write("你正在使用谷歌浏览器");

}else if(userAgent.contains("Trident")){

response.getWriter().write("你正在是IE浏览器");

}else{

response.getWriter().write("识别不了的浏览器");

}

}

### 案例2 referer头 -- 防止非法链接

问题：

第一次：下载资源 - > 下载页面 -> 打开广告页面（下载链接） -> 开始下载

第二次：直接点击下载链接 -> 跳转广告页面(下载链接) -> 开始下载

非法请求：

1．直接访问下载资源

2．不是从广告页面过来的请求

**referer请求头只有从超链接过来的请求才有这个头**

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//设置编码

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

/\*\*

\* 代表下载资源文件:

\* **referer: 表示当前请求来自于哪里**

\*/

String referer = request.getHeader("referer");

System.out.println("referer="+referer); //输出referer请求头的值

/\*\*

\* 判断非法请求（链接）

\* **1．直接访问 referer==null**

\* **2．当前请求不是来自于广告页面 !referer.contains("adv.html")**

\*/

if(referer==null || !referer.contains("/adv.html")){ **//判断是否是指定的页面过来的**

response.getWriter().write("非法请求，请转到<a href='/day08/adv.html'>首页</a>");

}else{

response.getWriter().write("资源正在下载.....");

}

## Http响应

|  |
| --- |
| HTTP/1.1 200 OK **-- 响应行**  Server: Apache-Coyote/1.1 **-- 响应头**  Content-Length: 24  Date: Mon, 08 Jun 2015 01:51:04 GMT  -- 一个空行  **-- 实体内容** |

### 响应行

#### http协议版本

HTTP/1.1

http协议版本：

http1.0: 当浏览器和服务器连接之后，在一次连接中只能发出一个请求

http1.1:当浏览器和服务器连接之后，在一次连接中可以发出多次请求。（效率比1.0更高）

#### 状态码：当前服务器处理请求的状态（结果）

常见的状态码：

200： 请求处理完成，成功返回！

302： 需要浏览器进一步请求，才能完成。

404： 浏览器端的错误。没有找到资源。

500： 服务器端的错误。

#### 状态描述：对状态码的文字补充

### 响应头

|  |
| --- |
| Location: http://www.it315.org/index.jsp -- 重定向的地址。结合302状态使用完成重定向的效果  Server:apache tomcat -- 服务器的类型  Content-Encoding: gzip -- 服务器发送给浏览器的数据压缩格式  Content-Length: 80 -- 服务器发送给浏览器的数据长度  Content-Language: zh-cn -- 服务器支持语言  Content-Type: text/html; charset=GB2312 -- 服务器发送给浏览器的数据类型和数据编码格式  Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT -- 服务器资源的最后修改时间  Refresh: 1;url=http://www.it315.org -- 定时刷新或每隔n秒跳转资源  Content-Disposition: attachment; filename=aaa.zip -- 以下载方式打开资源  Transfer-Encoding: chunked  Set-Cookie:SS=Q0=5Lb\_nQ; path=/search -- 服务器发送给浏览器的cookie数据  Expires: -1 -- 通知浏览器不使用缓存（3个都是）  Cache-Control: no-cache  Pragma: no-cache  Connection: close/Keep-Alive -- 连接状态  Date: Tue, 11 Jul 2000 18:23:51 GMT -- 响应发出的时间 |

### Http响应的原理:

1．tomcat服务器提供了一个HttpServletResponse对象，用于给开发者修改响应数据

2．通过service方法把response对象传入servlet中

3．通过response对象修改响应数据

4．tomcat服务器把response对象转换成响应格式的字符串，发送给浏览器

### 使用HttpServletResponse修改响应数据

response.setStatus(404) 设置状态码

response.setHeader("name","value") 修改响应头

response.getWriter().write() 以字符形式发送实体内容

response.getOutputStream().write() 以字节形式发送实体内容

#### 例：

response.setStatus(404); //设置状态码为404

response.sendError(404); // 404+404错误页面

response.setHeader("server", "webLogic"); //设置响应头server为webLogic，隐藏本来的Tomcat

response.getWriter().write("this is content!"); //以字符流将文本输出到浏览器

response.getOutputStream().write("this is content!!!".getBytes()); //以字节流将文本输出到浏览器

### 案例1 location+302 -- 请求重定向

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

/\*\*

\* 请求重定向（ 跳转到其他页面 ）

\*/

//设置302状态码

response.setStatus(302);

//设置location响应头，当出现302时就会执行location

response.setHeader("location", "/day08/adv.html");

/\*\*

\* **简化版本（推荐使用）**

\*/

**response.sendRedirect("/day08/adv.html");**

}

重定向时，在跳转页面地址加？可以写参数，参数以&链接。以get方式发送过去。

**response.sendRedirect("/day08/adv.html?** name=eric&password=123456**");**

### 案例2 refresh -- 定时刷新或每隔n秒跳转页面

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

//设置"refresh响应头"，"秒数"

//response.setHeader("refresh", "2");

//每隔n秒跳转页面

response.getWriter().write("注册成功！3秒之后会跳转到主页");

**//设置"refresh响应头"，"秒数;跳转页面"**

**response.setHeader("refresh", "3;/day08/adv.html");**

}

### 案例3 content-type -- 设置发送给浏览器的数据类型及编码格式

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

/\*

//设置content-type响应，及编码

response.setHeader("content-type", "text/html;charset=utf-8");

**response.setContentType("text/html;charset=utf-8"); //等价于上面的代码**

1．设置数据类型

response.setContentType("text/html"); //告诉浏览器以什么样的格式来解析实体内容

response.setContentType("image/jpg"); //告诉浏览器以什么样的格式来解析实体内容

**注意： 一定要写服务器支持的数据类型，如果写了服务器不支持的类型，就会报错**

2．设置数据编码格式

两个作用：

1．设置输出数据的编码

2．告诉浏览器自动适应输出数据的编码

\*/

response.setContentType("text/html;charset=gbk"); //和下面的代码是效果是一样的。

//response.setCharacterEncoding("utf-8"); //不会告诉浏览器自动跳转解码的码表

response.getWriter().write("<html><head><title>this is tille</title></head><body>主题</body></html>");

}

### 案例4 Content-Disposition -- 以下载方式打开资源

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

File file = new File("e:/mm.jpg");

/\*\*

\* 告诉浏览器以下载的方法打开

\* response.setHeader("content-disposition", "attachment;filename="+"**文件名**");

\*/

**response.setHeader("content-disposition", "attachment;filename="+file.getName());**

//文件下载

//1．读取本地文件

FileInputStream in = new FileInputStream(file);

//2．写出给浏览器(字节内容)

OutputStream out = response.getOutputStream();

byte[] buf = new byte[1024];

int len = 0;

//边读边写

while( (len=in.read(buf))!=-1 ){

out.write(buf, 0, len);

}

//4．关闭

in.close();

out.close();

}

# Http总结：

1．http协议： 对浏览器和服务器之间数据传输格式规定。

2．http请求：（浏览器 -> 服务器）

2.1 请求格式

请求行（请求方式+请求资源+http协议）

请求头（user-agent/referer）

空行

实体内容

2.2 对开发获取请求数据：

HttpServletRequest对象：

获取的方法

3．http响应（服务器 -> 浏览器）

3.1 响应格式

响应行（http协议+状态码+描述）

响应头（location+refresh+content-type+content-diposition）

空行

实体内容

3.2 对开发修改响应数据

修改方法

# Servlet概念

## servlet的特点

1．sevlet是一个普通的java类，继承HttpServlet类。（HttpServlet实现了Servlet接口）

2．其实实现了Servlet接口的java类，才是一个Servlet类。

3．servlet程序需要交给tomcat服务器运行！

## Servlet开发过程

1．编写java类，继承HttpServlet

2．servlet程序交给tomcat服务器运行

2.1 把servlet的class字节码拷贝到WEB-INF/classes目录下

2.2 在web.xml文件中配置

|  |
| --- |
| <!-- 浏览器访问： http://localhost:8080/day10/hello -->  <!-- 配置一个servlet -->  <!-- servlet的配置 -->  <servlet>  <!-- servlet内部名称。可以自定义 -->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet类的全名：包名+简单类名 -->  <servlet-class>gz.itcast.a\_servlet.HelloServlet</servlet-class>  </servlet>    <!-- servlet的映射配置 -->  <servlet-mapping>  <!-- sevlet内部名称。和servlet配置的内部名称保持一致！！ -->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet的路径映射。访问servlet的名称 -->  <url-pattern>/hello</url-pattern>  </servlet-mapping> |

## Servlet执行过程

浏览器： http://localhost:8080/day09/hello（/hello是映射在web.xml中的一个Servlet类）

### 执行步骤

**前提： tomcat服务器启动时会把每个网站的web.xml文件加载进内存。**

1．在浏览器输入；http://localhost:8080/day09/hello

2．截取URL的字符串，得到需要访问的资源名称： /hello

3．在day09网站下的web.xml信息中搜索是否存在 /hello 匹配的url-pattern。

4．匹配到对应的url-pattern之后，取出当前映射信息中的servlet-name的名称，然后在web.xml信息中搜索是否存在一个相同名称的servlet-name的servlet配置信息。

5．找到对应的servlet配置信息之后，取出当前配置信息的servlet-class的内容

字符串： gz.itcast.a\_servlet.HelloServlet

6．创建HelloServlet对象，然后调用HelloServlet类中的方法。

7．Servlet就把内容返回给浏览器用户。

# servlet映射路径

|  |
| --- |
| <!-- servlet映射信息 -->  <servlet-mapping>  <!-- servlet内部名称，和servlet配置的内容名称保持一致！ -->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet的映射路径：访问servlet的名称 -->  <url-pattern>/hello</url-pattern>  </servlet-mapping> |

### 精确匹配：

**<url-pattern>**  **浏览器访问**

/hello http://localhost:8080/day09/hello

/itcast/hello http://localhost:8080/day09/itcast/hello

### 模糊匹配：

**<url-pattern>**  **浏览器访问**

/\* http://localhost:8080/day09/任意路径

/itcast/\* http://localhost:8080/day09/itcast/任意路径

\*.后缀名（\*.action, \*.html…） http://localhost:8080/day09/任意路径.后缀名

## 注意事项：

1．要么以斜杠开头，要么\*号开头

例如：<url-pattern> **itcast/demo1**</url-pattern> 错误写法

2．当多个url-pattern同时被匹配的情况下

2.1 **精确匹配优先（长的最像的那个优先被匹配）**

* 1. **以后缀名结尾的url-pattern的优先级最低的**

3．不能同时使用两种模糊匹配。

例如：<url-pattern>**/itcast/\*.html** </url-pattern> 错误写法

# servlet缺省路径

**/ : 表示servlet的缺省路径。**

## 什么是缺省路径

在tomcat服务器内部配置了一个缺省路径 <url-pattern>/</url-pattern>, 这个缺省路径在tomcat服务器内部被一个缺省Servlet（DefaultServlet）匹配。缺省的这个Servlet的作用是专门用于处理所有网站中的静态网页。

例如浏览器访问： http://localhost:8080/day09/hello.html

一个路径在tomcat服务器的执行过程：

1．搜索day09下面的web.xml文件信息。搜索是否存在匹配的<url-pattern>

2．匹配到对应的url-pattern,执行对应的Servlet程序。

3．如果没有匹配的url-pattern，就把这个请求交给tomcat服务器的DefaultServlet处理。

4．DefaultServlet会在day09的根目录下搜索是否存在一个名称叫hello.html的文件。

5．如果找到次文件， 那么DefaultServlet读取该文件内容返回给用户。

6．如果找不到次文件， 那么就给用户返回一个404状态码+404错误页面。

**结论： 先找动态网页，再找静态网页。**

# Sevlet的生命周期（重点）

## 生命周期

以前：以前的程序，java类，自己去new对象，自己去使用对象调用。

现在：sevlet程序。servlet的生命周期由tomcat服务器控制的。

servlet什么时候对象创建，什么销毁，什么时候调用什么方法？？？

## Sevlet的四个重要的生命周期方法

LifeDemo方法：在创建servlet对象时调用。只调用1次。证明servlet对象在tomcat服务器中是单实例的。

init方法：创建完对象之后调用。只调用1次。该方法用于初始化对象。

service方法：在每次请求时调用，调用n次。该方法入口方法，我们的逻辑代码在这里被调用。

destroy方法：servlet对象销毁时调用。只调用1次。tomcat服务器停止或web应用重新部署时调用。

## 用伪代码演示tomcat服务器如何调用四个方法

浏览器输入： http://localhost:8080/day10/hello

得到字符串： gz.itcast.a\_servlet.HelloServlet

1．tomcat服务器通过反射，创建HelloServlet的对象

1.1 得到HelloServlet的CLass对象

Class clazz = Class.forName("gz.itcast.a\_servlet.HelloServlet ")

1.2 通过class对象调用构造方法

//调用无参的构造方法。

Object obj = clazz.newInstance(); --1．servlet无参的构造方法被调用

2．tomcat服务器通过反射，调用init方法

2.1 得到init方法对象

Method m = clazz.getDeclaraeMethod("init",ServletConfig.class);

2.2 调用方法

m.invoke(obj,config); --2．servlet的init方法被调用

3．tomcat服务器通过反射，调用service方法

3.1 得到service方法对象

Method m = clazz.getDeclareMethod("service",HttpServletRequest.class,HttpServletResponse.class);

3.2 调用方法

m.invoke(obj,request,response); --3．servlet的service方法被调用

4．tomcat服务器通过反射，调用destroy方法

4.1 得到destroy方法对象

Method m= clazz.getDeclareMethod('destroy",null);

4.2调用方法

m.invoke(obj,null); --4．servlet的destroy方法被调用

## 生命周期流程图



# Servlet的线程并发问题

servlet对象特点： 在tomcat服务器中是单实例多线程的！！！

## 引发Servlet多线程问题

多个线程同时操作了Servlet的成员变量（共享数据）。

## 避免Servlet并发问题建议

1．尽量不要在servlet类中使用成员变量。

2．如果要使用成员变量，那么就要给使用到成员变量的代码块加上代码锁，尽量缩小同步锁的范围，以避免因为同步产生代码并发执行效率降低的问题。

## 实例：

|  |
| --- |
| /\*\*  \* 1．同步代码块  \* 给使用到共享数据的代码块添加同步锁  \* 注意:同步锁必须多个线程唯一的  \*/  **synchronized** (ThreadDemo.**class**) {  response.getWriter().write("你当前是第"+*count*+"个访客！");  // A线程刚刚执行完这句代码，被B线程抢去了执行时机。  *count*++;  }    /\*\*  \* 2 ）同步方法  \* **@param** response  \*/  **public** **synchronized** **static** **void** method(HttpServletResponse response){  **try** {  response.getWriter().write("你当前是第"+*count*+"个访客！");  } **catch** (IOException e1) {  e1.printStackTrace();  } // A线程刚刚执行完这句代码，被B线程抢去了执行时机。    *count*++;  } |

# Servlet的自动加载机制

## 引入

默认情况servlet对象是在第一次访问的时候创建。如果在这个servlet的构造方法或者init方法执行比较多的逻辑，那么第一次访问servlet的用户体验就不好！

那么能不能改变创建servlet的机制？ 可以的，可以让servlet在服务器启动的时候自动加载。

## 自动加载的配置

|  |
| --- |
| <servlet>  <servlet-name>LifeDemo</servlet-name>  <servlet-class>gz.itcast.c\_life.LifeDemo</servlet-class>  <!-- 这个配置可以让servlet在tomcat服务器启动的时候自动创建servlet对象 -->  <load-on-startup>1</load-on-startup>  </servlet> |

**注意：load-on-startup中的整数值越大，创建的优先级越低！！！**

# Servlet的init方法

## 两个init方法作用

init(ServletConfig) 这个有参数的init方法的作用就是Servlet的其中一个重要的生命周期方法。这个方法一定会被tomcat服务器调用。

init(): 这个无参的init方法的作用就是为了提供给开发者方便去覆盖，覆盖这个方法，可以在此方法编写初始化的逻辑。这个方法才是真的给开发者做初始化逻辑的方法。

Servlet中重要对象：

1．HttpServletRequest对象：封装请求信息

2．HttpServletResponse对象：封装响应信息

3．ServletConfig对象： 封装一个servlet配置参数信息

4．ServletContext对象： 封装web应用环境信息

# ServletConfig对象

## 引入

servletconfig是配置对象，主要把servlet的初始化参数封装到这个对象中。一个网站中可能会存在多个ServletConfig对象，一个ServletConfig对象就封装了一个servlet的配置信息。

## 配置初始化参数

**写在WEB-INF 文件夹的web.xml文件中**

|  |
| --- |
| <servlet>  <servlet-name>ConfigDemo</servlet-name>  <servlet-class>gz.itcast.f\_config.ConfigDemo</servlet-class>  <!-- servlet的初始化参数 -->  <init-param>  <param-name>path</param-name>  <param-value>e:/aaa.txt</param-value>  </init-param>  </servlet> |

## 获取初始化参数的方法

config.getInitParameter("name"); 根据参数名称获取参数值

config.getInitParameterNames(); 获取所有参数名称（返回Enumeration<String>）

### 实例：

|  |
| --- |
| //读取到servlet的初始化参数（init-param）  String path = **this**.getServletConfig().**getInitParameter**("path");  //获取所有头名称  Enumeration<String> enums =  **this**.getServletConfig().getInitParameterNames();  **while**(enums.hasMoreElements()){  String path = request.getHeader(headerName);  System.***out***.println(path);  }  //读取文件内容  BufferedReader br = **new** BufferedReader(**new** FileReader(path));  String str = **null**;  **while**( (str=br.readLine())!=**null**){  System.***out***.println(str);  } |

config.getServletName() 得到servlet名称

config.getServletContext() 得到servlet上下文对象

# ServletContext对象（重点）

## 引入

ServletConfig对象叫servlet上下文对象。 一个网站只会创建一个ServletContext对象。代表的是整个网站的环境信息。

## 获取ServletContext对象

**this.getServletConfig().getServletContext(); 通过ServletConfig对象来获取到ServletContext对象**

**简写： this.getServletContext();**

ServletContext对象：启动的时候创建

ServletConfig对象：调用init方法之前创建的，在ServletContext对象创建之前

## ServletContext创建时机

ServletContext对象是在tomcat服务器加载完当前web应用后创建出来。ServletContext对象是作为ServletConfig对象成员变量传入servlet中。通过ServletConfig的getServletContext()方法得到ServletContext对象。

class ServletConfig{

ServletContext context;

getInitParameter()

getInitParameterNames();

getServletContext(){

return contex;

}

}

init(ServletConfig config){

context.getServletContext()

}

## ServletContext的5大作用及方法

**this.getServletConfig().getServletContext(); 通过ServletConfig对象来获取到ServletContext对象**

**简写： this.getServletContext();**

### 1．获取web的上下文路径

java.lang.String getContextPath()

**简化版本： request.getContextPath();**

|  |
| --- |
| // 获取  String path = context.getContextPath();  System.***out***.println(path); // /day09  /\*\*  \* web上下文路径就是项目在tomcat服务器中运行的路径。注意不是开发目录的项目名称。  \*/  //请求重定向  //response.sendRedirect("/day09/hello.html");    //作用：可以让这个获取文件的路径更加灵活  response.sendRedirect(context.getContextPath()+"/hello.html"); |

### 2．获取全局参数

java.lang.String getInitParameter(java.lang.String name) 根据参数名称获取参数值

java.util.Enumeration getInitParameterNames() 获取所有参数名称（返回Enumeration<String>）

**写在WEB-INF 文件夹的web.xml文件中**

|  |
| --- |
| <!-- 全局参数配置 -->  <context-param>  <param-name>AAA</param-name>  <param-value>AAA'value</param-value>  </context-param>  <context-param>  <param-name>BBB</param-name>  <param-value>BBB'value</param-value>  </context-param> |

|  |
| --- |
| /\*\*  \* 得到全局参数  \*/  ServletContext context = **this**.getServletContext();    System.***out***.println(**context.getInitParameter("AAA")**);    //遍历所有全局参数  Enumeration<String> enums = **context.getInitParameterNames()**;  **while**(enums.hasMoreElements()){  String paramName = enums.nextElement();  String paramValue = context.getInitParameter(paramName);  System.***out***.println(paramName+"="+paramValue);  } |

### 3．和域对象相关的

void setAttribute(java.lang.String name, java.lang.Object object) //将对象数据保存进域

java.lang.Object getAttribute(java.lang.String name) //以指定name从域中获取对象数据

void removeAttribute(java.lang.String name) //清除指定name的对象数据

什么是域对象？

域对象在不同的资源之间来共享数据。保存数据，获取数据。

ServletContext就是我们学习的第一个域对象。

Servlet三个域对象：

ServletContext

HttpServletRequest

HttpSession

|  |
| --- |
| /\*\*  \* 把数据存储到ServletContext域对象  \*/  ServletContext context = **this**.getServletContext();  //context.setAttribute("name", "eric");  List list = **new** ArrayList();  list.add("eric");  list.add("jacky");  list.add("rose");  **context.setAttribute("list", list);**  System.***out***.println("保存成功"); |

|  |
| --- |
| /\*\*  \* 从ServletContext域对象中取出数据  \*/  ServletContext context = **this**.getServletContext();  //String name = (String)**context.getAttribute("name")**;  List list = (List)**context.getAttribute("list")**;  System.***out***.println(list); |

### 4．请求转发的

RquestDispatcher getRequestDispatcher(java.lang.String path)

#### 使用格式：

ServletContext context = this.getServletContext();

RequestDispatcher rd = context.getRequestDispatcher("/hello.html");

rd.forward(request, response);

**简写方式：request.getRequestDispatcher("path").forward(request, response);**

在servlet跳转页面：

请求重定向：

1．地址栏会发送改变，变成重定向到的地址。

2．可以跳转到项目内的资源，也可以跳转项目外的资源

3．浏览器向服务器发出两次请求，那么就不能使用请求来作为域对象来共享数据。

请求转发：

1．地址栏不会改变。

2．只能跳转到项目内的资源，不能跳转到项目外的资源

3．浏览器向服务器发出一次请求，那么可以使用请求作为域对象共享数据。

#### 注意事项：

转发可将数据写入request域中

如：request.setAttribute("name", "jacky");

在有接收转发的文件去获取request域中的数据

如：String name = (String)request.getAttribute("name");

#### 路径问题

服务器行为 / 代表从当前项目的根目标开始

浏览器行为 / 代表从当前站点的根目录开始

//**目标地址： hello.html**

// 转发

request.getRequestDispatcher("**/hello.html**").forward(request, response);

// 重定向

//response.sendRedirect("**/day09/hello.html**");

// html页面的超链接

response.getWriter().write("<a href='**/day09/hello.html**'>link</a>");

// html页面的表单的action

response.getWriter().write("<form action='**/day09/hello.html**'><input type='submit'/></form>");

### 5．读取web项目的资源文件

java.lang.String getRealPath(java.lang.String path) 得到资源文件的绝对路径

java.io.InputStream getResourceAsStream(java.lang.String path) 获取资源文件的输入流

java.net.URL getResource(java.lang.String path) 获取资源文件的URL

**// 1．getRealPath() 获取资源文件的真实路径**

String path = this.getServletContext().getRealPath("/WEB-INF/classes/news.properties");

**//2．getResourceAsStream() 获取资源文件的输入流**

InputStream in = this.getServletContext().getResourceAsStream("/WEB-INF/classes/news.properties");

**// 3．getResource() 获取资源文件的URL**

//URL url = this.getServletContext().getResource("/WEB-INF/classes/news.properties");

#### 实例：读取Properties文件？

/\*\*

\* 相对路径：

\* 当前路径相对于java命令运行的目录

\* 结论：

\* 在web项目中，java命令运行的目录就是在tomcat的bin目录。

\* 在web项目，不能使用相对路径。

\*/

**//使用getResourceAsStream() 获取资源文件的输入流**

InputStream in = this.getServletContext().getResourceAsStream("/WEB-INF/classes/news.properties");

//1．使用Properteis对象

Properties prop = new Properties();

//2．使用load方法加载properties文件

prop.load(in);

//3．通过getProperty（）获取内容

System.out.println(prop.getProperty("name"));

System.out.println(prop.getProperty("password"));

request.getRemoteHost() //获取访问的IP

# Sevlet总结：

Servlet编程：

1．servlet编写

2．servlet的映射路径（精确匹配 模糊匹配 缺省路径？？？）

3．servlet的生命周期（构造方法，init方法，service方法，destroy方法）

4．多线程并发问题（单实例多线程，servlet使用成员变量，同步锁解决）

5．自动加载（load-on-startup） + 两个init方法的区别

6．ServletConfig对象：读取初始化参数

<servlet>

<init-param>

<param-name></param-name>

<param-value></param-value>

<init-param>

<servlet>

7．ServetContext对象（重点）

5大作用：

1．读取web上下文路径（1）

context.getContextPath(); 简化版本： request.getContextPath();

2．读取全局参数（2）

context.getInitParameter("name");

context.getInitParameterNames();

3．作为域对象使用（3）

保存数据： context.setAttribute("name",Object);

获取数据： context.getAttribute("name")

移除数据： context.removeAttribute("name")

4．可以转发页面（1）

context.getRequestDispatcher("路径").forward(request,response);

//简化版本

request.getRequestDispatcher("路径").forward(request,response);

5．读取web资源文件（3）

context.getRealPath("路径"); 得到资源文件的绝对路径

context.getResourceAsStream("路径") 得到资源，返回输入流

context.getResource("路径") 得到资源，返回URL

# URL加密及解密

**URLEncoder类：URL的加密类**

**URLDecoder类：URL的解密**

**浏览器不能解读中文所以要加密**

## 使用格式：

String str = "传智";

str = URLEncoder.encode(str, "utf-8"); //加密成16进制（%等价于H）

System.out.println(str); //%E4%BC%A0%E6%99%BA

str = URLDecoder.decode(str, "utf-8"); //解密

System.out.println(str);

# 会话管理入门

## 生活中的会话

我： 你最近有坐船吗？

小明： 有啊，昨天才刚坐

我： 这样啊！！前几天有船沉了，你没事吧？

小明： 不是吧？

以上通话过程，会产生对话数据。

## 程序中的会话

一次会话： 打开浏览器 -> 访问服务器 -> 关闭浏览器

### 登录场景：

打开浏览器 - > 访问网站 -> 登录页面 -> 输入用户名和密码（保存会话信息） -> 提交到后台 -> 校验成功 -> 返回用户首页（看到当前登录的用户名信息。。。）（取出会话信息） -> 关闭浏览器

使用什么技术保存这些会话信息？？？？

### 购物场景：

打开浏览器 -> 浏览购物网站 -> 点击“购买” （把商品信息保存下来）-> 放入“购物车” -> 关闭浏览器

打开浏览器- > 浏览器购物网站- > 查看"购物车" -> 看到之前的商品信息- >付款购买-关闭浏览器

使用什么技术保存这些商品信息？

会话管理： 管理浏览器与服务器之间的会话过程中产生的会话数据！！！！

思考：

可以使用域对象来保存会话信息？？？？

context域：

登录： 登录页面( 张三 context.setAttribute("name","张三") )

-> 访问用户主页（context.getAttribute("name")）

使用context域会覆盖之前的数据（不合理）

request域：

登录： 登录页面( 张三 request.setAttribute("name","张三") )

-> 访问用户主页（request.getAttribute("name")）

使用request域之后，一定要使用转发技术来跳转页面。（不合理）

## 会话管理技术

Cookie技术：会话数据保存在浏览器端。

Session技术: 会话数据保存在服务器端。（域对象）

# 引入

## Cookie特点：

1．会话数据放在浏览器端

2．数据类型只能string，而且有大小限制的

3．相对数据存放不安全。

## Session特点:

1．会话数据放在服务器端（服务器内存），占用服务器资源

2．数据类型任意类型，没有大小限制的。

3．相对安全

# Cookie技术

## Cookie技术的使用

Cookie对象：

1．创建Cookie对象，用于存储会话数据

new Cookie(java.lang.String name, java.lang.String value)

2．修改Cookie对象

cookie.setPath(java.lang.String uri) 修改cookie所在的有效路径

cookie.setMaxAge(int expiry) 设置cookie的有效时间

cookie.setValue(java.lang.String newValue) 修改指定cookie对象的值

3．把cookie数据发送给浏览器保存

response.addCookie(cookie);

4．浏览器带着cookie访问服务器，服务器接收cookie信息

request.getCookies();

## Cookie技术原理

1．服务器创建Cookie对象，保存会话数据，把Cookie数据发送给浏览器

response.addCookie(cookie); (响应头：set-cookie: name=jacky)

2．浏览器获取cookie数据，保存在浏览器缓存区，然后在下次访问服务器时携带cookie数据

(请求头： cookie: name=jacky)

3．服务器获取浏览器发送的cookie数据

request.getCookies();

## Cookie细节

1. cookie的数据类型一定是字符串，如果要发送中文，必须先对中文进行URL加密才可以发送。
2. setPath（path）： 修改cookie所在的有效路径。什么是有效路径？ 如果把该cookie设置到某个有效路径下，然后当浏览器访问这个有效路径的时候，才会携带cookie数据给服务器。
3. setMaxAge(整数,单位：秒) ： 设置cookie的有效时间

正整数： 表示超过了正整数的数值的时间，cookie就会丢失！！（cookie保存浏览器的缓存目录）

负整数： 表示如果浏览器关闭了，cookie就会丢失！（cookie保存浏览器内存）

1. ：表示删除同名的cookie
2. cookie可以有多个，但是浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。

## 实例：

### 创建Cookie对象，保存会话数据

//如果要发送中文，必须先使用URLEncoder进行加密

String n = URLEncoder.encode("张三", "utf-8");

Cookie c = new Cookie("name",n);

Cookie c2 = new Cookie("email","jacky@qq.com");

#### 1．设置有效路径。默认情况下，设置当前项目的根目录下

什么是有效路径？ 如果把该cookie设置到某个有效路径下，然后当浏览器访问这个有效路径的时候，才会携带这个cookie数据给服务器。（**如果接收cookie的是day10,有效路径该为day11后，访问day10就不会发送此条cookie了**）

c.setPath("/day11");

#### 2．设置有效时间

c.setMaxAge(20); //20秒之后cookie丢失！！！

c.setMaxAge(-1); //浏览器关闭cookie丢失！！！（默认情况下）

//删除名字为name的cookie

Cookie c = new Cookie("name","abc");

c.setMaxAge(0);

response.addCookie(c);

### 发送Cookie对象

#### 1．把cookie发送给浏览器.通过响应头携带cookie数据给浏览器

//response.setHeader("set-cookie", "name=jacky");

/\*\*

\*简化版本

\*/

**response.addCookie(c);**

response.addCookie(c2);

#### 2．浏览器在下次访问时携带cookie数据，通过请求头发送给服务器（cookie）

//服务器获取浏览器发送的cookie数据

//String name = request.getHeader("cookie");

/\*\*

\* 简化版本

\*/

Cookie[] cookies = request.getCookies();

**//取出数据**

if(cookies!=null){

for (Cookie cookie : cookies) {

//cookie的名称

String name = cookie.getName();

//对加密之后的name进行解密

String value = cookie.getValue();

value = URLDecoder.decode(value, "utf-8");

System.out.println(name+"="+value);

}

}else{

System.out.println("没有cookie信息");

}

## 案例-用户上次访问时间

public class LastTimeServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

//制作了当前时间字符串

SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd号 hh时mm分ss秒");

String curDate = sdf.format(new Date());

//获取cookie

Cookie[] cookies = request.getCookies();

String lastTime = null;

if(cookies!=null){

//存在cookie，还有继续找是否name为lastTime的cookie

for (Cookie cookie : cookies) {

if(cookie.getName().equals("lastTime")){

lastTime = cookie.getValue();

//找到了，第n次访问

//lastTime保存了上次访问的时间

//对日期字符串进行解密

lastTime = URLDecoder.decode(lastTime, "utf-8");

//显示内容给浏览器

response.getWriter().write("欢迎再次光临本网站，您上次访问的时间为："+lastTime+"<br/>当前的时间为："+curDate);

//更新cookie上次访问的时间

curDate = URLEncoder.encode(curDate, "utf-8");

cookie.setValue(curDate);

response.addCookie(cookie);

break;

}

}

}

//第一次访问

if(cookies==null || lastTime==null){

//2)显示到浏览器

response.getWriter().write("欢迎首次光临本网站，当前时间为："+curDate);

//3)把当前时间保存到cookie中

//对中文格式的日期字符串进行加密

curDate = URLEncoder.encode(curDate, "utf-8");

Cookie c = new Cookie("lastTime",curDate);

//4)把cookie发送给浏览器

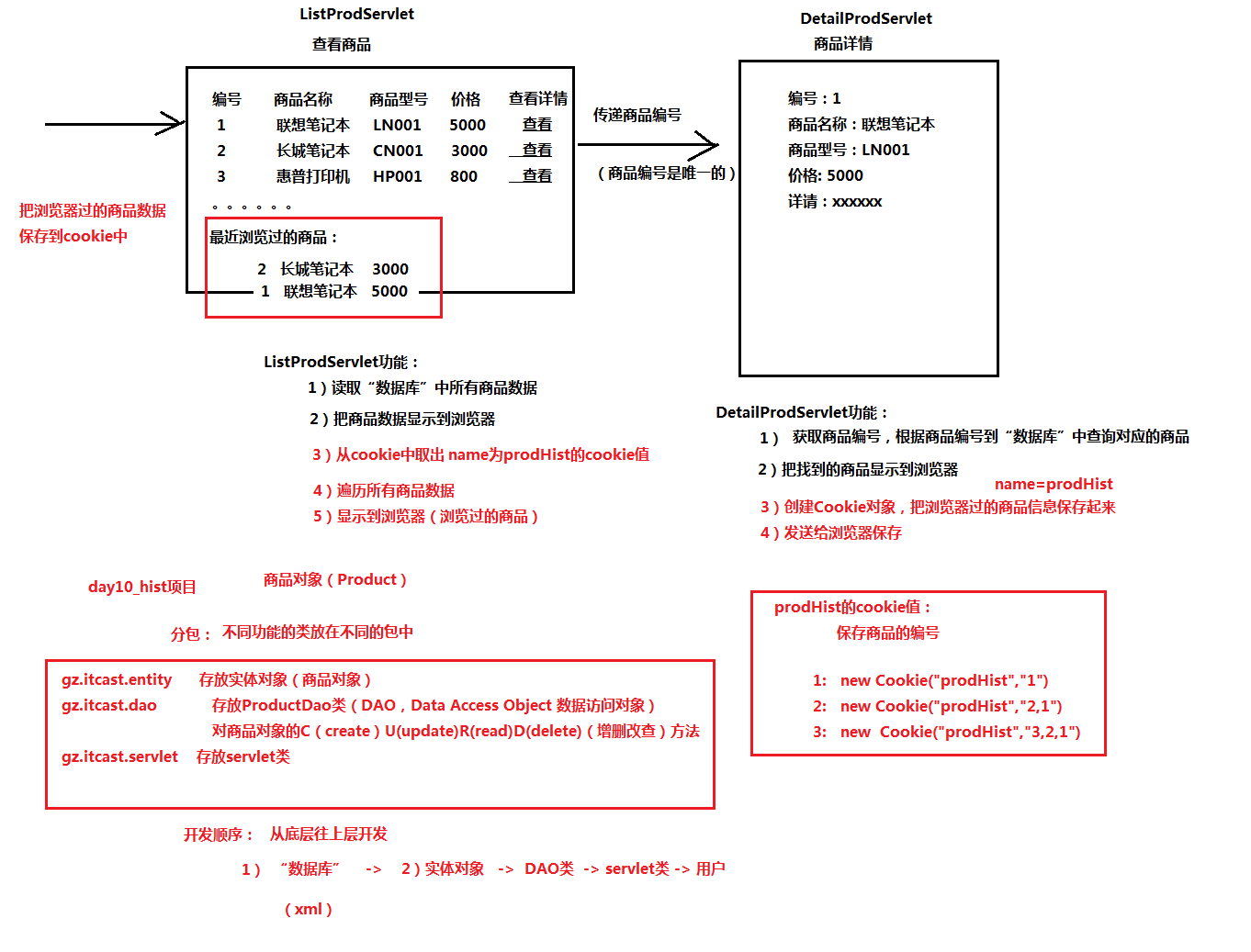
response.addCookie(c);

}

}

}

## 案例-用户浏览过的商品



[D:\教程\Java案例\会话管理（Cookie） - 用户浏览过的商品](file:///D:\教程\Java案例\会话管理（Cookie）%20-%20用户浏览过的商品)

# Session技术

## Session使用步骤

HttpSession对象：

1．创建HttpSession对象，用于保存会话数据

session = request.getSession(true/false); 创建或获取session对象

调用空参数方法和用true为，有及获取，无则创建

用false调用参数方法，有及获取，无则返回null

2．修改HttpSession对象

void setMaxInactiveInterval(int interval) 设置session对象的有效时间

void invalidate() 手动销毁session对象

3．保存会话数据(作为域对象)

session.setAttribute("name",Object) 保存数据

session.getAttribute("name") 获取数据

session.removeAttribute("name") 删除数据

## Session原理

问题： 服务器怎么区分不同的浏览器会话？

**前提：可以从session对象取出数据必须是存放数据的sessinon对象！！！**

1．浏览器1-窗口1(001)：

//1)创建HttpSession对象

HttpSession session = request.getSession(); 给session对象分配001

//2)保存会话数据

session.setAttribute("name", "jacky");

2．浏览器1-窗口2(001)：

//1)得到session对象

HttpSession session = request.getSession(); 搜索001的session对象

//2）获取会话数据

String name = (String)session.getAttribute("name"); 可以得到！！

3．浏览器2（没有标记或不是001）

//1)得到session对象

HttpSession session = request.getSession();

//2．获取会话数据

String name = (String)session.getAttribute("name"); 不可以得到！

4．新的浏览器1（没有标记或不是001）

//1)得到session对象

HttpSession session = request.getSession();

//2．获取会话数据

String name = (String)session.getAttribute("name"); 不可以得到！！

1．服务器创建Session对象，服务器会给这个session对象分配一个唯一的标记JSESSIONID

2．把JSESSIONID作为Cookie发送给浏览器

3．浏览器得到JSESSIONID保存下来，在下次访问时携带这个JSESSIONID去访问服务器

4．服务器得到JSESSIONID，在服务器内存中搜索是否存在指定JSSESSINOID的session对象

5．如果找到，则返回这个session对象

6．如果找不到，可能直接返回null，或者再创建新的session对象。

HttpSession session = request.getSession();

**结论： 通过JSESSIONID在服务器中查询对应的session对象。**

## Session细节

1．setMaxInactiveInterval（秒数）: 设置session对象的有效时间

问题：session在什么销毁？

注意：不是浏览器关闭，session对象就销毁！！！

默认情况： 等待30分钟空闲时间，session对象才会销毁。

|  |
| --- |
| <!-- 设置全局的session对象的过期时间 （分钟）-->  <session-config>  <session-timeout>1</session-timeout>  </session-config> |

2．可以让JSESSIONID不会随着浏览器关闭而丢失！！！！

|  |
| --- |
| /\*\*  \* 设置JSESSIONID的时间，不会随着浏览器关闭而丢失！  \*/  Cookie c = **new** Cookie("JSESSIONID",session.getId());  c.setMaxAge(1\*30\*24\*60\*60);//1个月  response.addCookie(c); |

3．直接手动销毁sessino对象

invalidate()

4．创建或得到session对象

request.getSession() / request.getSession(true)

创建或得到session对象，查询session对象，如果没有sessino对象，则创建新的session对象

request.getSession(false)

得到session对象。 查询session对象，如果session对象，直接返回null

## 实例：

### 把数据保存到HttpSession对象中

**//1．创建HttpSession对象**

HttpSession session = request.getSession(true);

//设置session的有效时间

session.setMaxInactiveInterval(20);

//设置JSESSIONID的时间，不会随着浏览器关闭而丢失！

Cookie c = new Cookie("JSESSIONID",session.getId());

c.setMaxAge(1\*30\*24\*60\*60); //1个月

response.addCookie(c);

**//2．保存会话数据**

session.setAttribute("name", "jacky");

### 从HttpSession对象中获取数据

**//1．得到session对象**

HttpSession session = request.getSession(false);

if(session==null){

System.out.println("没有session对象");

}else{

**//2．获取会话数据**

String name = (String)session.getAttribute("name");

System.out.println("name="+name);

}

### 手动销毁session对象

**//1．得到session对象**

HttpSession session = request.getSession();

**//2．销毁**

session.invalidate();

System.out.println("销毁成功");

## 案例-用户登录

[D:\教程\Java案例\会话管理（Session） - 用户登录](file:///D:\教程\Java案例\会话管理（Session）%20-%20用户登录)

# 会话管理总结：

1．会话管理： 浏览器与服务器之间会话过程中产生的会话数据

2．Cookie技术：会话数据保存在浏览器

核心的API：

a）创建Cookie对象

Cookie c = new Cookie("name","value")

b）发送给浏览器

respone.addCookie(c);

c）浏览器发送cookie给服务器，服务器接收cookie

Cookie[] cookies = request.getCookies()

3．Session技术： 会话数据保存在服务器（内存中）

核心API：

a）创建或得到HttpSession对象

HttpSession session = request.getSession(true) 创建或得到

HttpSession session = request.getSession(false) 得到

b）使用HttpSession保存和获取，删除会话数据

session.setAttribure("name",Object);

session.getAttribute("name");

session.removeAttrbute("name");

# servlet总结：

第一天：

tomcat服务器：

1．tomcat服务器如何使用‘

2．web应用如何写

3．web应用如何部署？

第二天：

http协议：

http请求：

HttpServletRequest对象： 获取请求数据

http响应：

HttpServletResponse对象： 设置响应数据

第三天：

servlet编程：

1．servlet生命周期

2．ServletConfig对象： 获取初始户参数

3．ServletContext对象： 5大作用

第四天：

会话管理

Cookie对象

HttpSession对象