Servlet

# Servlet概念

### servlet是什么

#### servlet是sun公司提供的用于开发动态web资源的技术

#### 按照servlet的规范开发的servlet可以放在servlet容器(Tomcat)中运行

### 开发servlet步骤

#### 写一个类，实现servlet接口，并实现其中的方法

#### 在web.xml中为servlet配置对外访问路径

获取通过注解配置对外访问路径

# 开发Servlet

### Servlet继承结构

#### Servlet接口

##### Servlet接口提供一个Servlet的基本功能

##### void init(ServletConfig var1)

初始化servlet

##### void destroy();

销毁servlet

##### void service(ServletRequest var1, ServletResponse var2)

处理请求的核心方法

#### 抽象类GenericServlet implements Servlet

##### 实现了Servlet接口，并实现了大部分方法，但Service方法没有实现，这个方法是处理请求的核心方法

#### 抽象类HttpServlet extends GenericServlet，

##### 实现了service方法，在servlice方法中根据不同的请求方式调用不同的doXxx方法

##### (例如：doPost，doGet),因此我们开发的时候继承了HttpServlet，并重写了doGet和doPost方法

#### 自定义Servlet extends HttpServlet

##### 根据业务需要实现doXxx()等方法

### 开发步骤

#### 创建web项目

#### 写一个类，实现servlet接口，实现其方法

##### 实现doGet和doPost

#### 配置对外访问路径

##### 方式一：web.xml

|  |
| --- |
|  |

###### 执行流程：浏览器访问地址时根据/weixin匹配，

###### 然后找到mapping中的servlet-name

###### 匹配servlet中的name

###### 找到servlet-name对应的包

##### 方式二：注解



##### **注意：**

###### 二种方式共存时，对同一个Servlet注册URL不能相同，否则服务器无法启动

# Servlet生命周期

### 加载(启动tomcat时)和实例化Servlet

#### Servlet 容器负责加载和实例化 Servlet。加载和实例化可以发生在容器启动时，或者延迟初始化直到容器决定有请求需要处理时(可配置加载时实例化,默认有请求时才实例化)

### 初始化(init)

#### 实例化(new Servlet)后，调用init方法进行初始化

### 服务（service）

#### init方法 此后该实例一直贮存在服务器内存中，为后续请求提供服务

### 销毁(destroy)

#### **销毁之前调用destroy()方法**

#### 当服务器关闭时，servlet容器销毁时，servlet实例会随之销毁，销毁之前调用destroy

### 卸载

# HttpServletRequest接口

代表请求的对象

### 继承结构

#### 接口ServletRequest

#### 接口HttpServletRequest extends ServletRequest

在父类基础上增加了一些与HTTP有关的方法

### 常用方法

#### request.getContextPath() String

获取当前web应用的名称，如果省略web应用名称返回空白

#### request.getHeader("HOST") String

获取请求头信息，根据请求头key，可以不区分大小写

#### 获取请求参数

##### request.getParameter("username");

根据key获取value 返回 数组的第一个值

##### request.getParameterValues("username");

返回数组，因为一个前端的表单name或者get参数里的name可以重复可以对应多个value

##### request.getParameterMap(); Map<String,String[]>

返回key-value的map，因为一个前端的表单name可能对应多个value (String[])

##### 注意中文乱码问题

浏览器提交的编码是utf-8的，服务器(tomcat)默认使用的编码为iso8859-1(这个编码不支持中文)

###### 解决POST提交乱码：

如果为post提交，请求参数通过请求实体提交，可以设置请求实体内容的编码格式解 决

request.setCharacterEncoding("utf-8");

###### 解决GET提交乱码

如果为get提交，需要手动编解码解决获取二进制(解码)

byte[] bytes= "username".getBytes("iso8859-1");

然后根据某个编码(utf-8)再编码，该方法对post也有效

String name=new String(bytes,"utf-8");

#### 请求转发

##### 特点

###### 一次请求一次响应

###### 地址栏不会发生变化，也就是对浏览器没有任何影响，操作只是在服务器端完成

###### 请求转发只能在同一tomcat服务器内的同一个web应用(一个tomcat对应多个应用)进行转发

###### 可以多级转发A -> B -> C…

##### 方法

###### 获取转发器

request.getRequestDispatcher("/weixin")

###### 转发request和response

.forward(request,response);

###### 注意

“/weixin”为servlet的urlPatterns值，或者是页面(例/index.jsp)即 对外访问路径

“/ ”可以省略

转发后之前response.getWriter().write的内容是在缓冲区中，会清空，即转发之前不会输 出

#### 作为域对象使用

##### 域对象

如果一个对象具有一个可以被看见的范围，利用该对象的map可以在整个范围内共享数据,那么这个对象就是域对象。

##### 方法

###### 请求转发前设置，将姓名放到域中

request.setAttribute("name","shu");

###### 目的Servlet接收或jsp页面

request.getAttribute("name")

###### **从域中删除** request.removeAttribute("name");

###### 只是请求转发，针对请求的数据，所以response没有这二个方法

### 生命周期及作用范围

#### 生命周期：请求开始时创建，请求结束时销毁

#### 作用范围：整个请求链, 即包括请求转发的下一个servlet

# HttpServletResponse接口

### 概述

#### 代表响应的对象

### 继承结构

#### 接口ServletResponse

#### 接口HttpServletResponse extends ServletResponse

在父类基础上增加了一些与HTTP有关的方法

### 功能

#### 设置响应状态码

response.setStatus(666);

#### 设置响应头

response.setHeader("set-cookie","pord=phone");

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

#### 设置响应实体内容

##### 方法

//字符流 需要设置请求头编码，不然可能乱码  
response.getWriter().write("你好Writer");  
//字节流 需要设置编码，不然可能乱码  
response.getOutputStream().write("你好OutPutStream".getBytes(“utf-8”));

##### 注意

字节流和字符流不能同时使用,网页可能会出现错误码500

##### 乱码问题

由于服务器发送的编码和浏览器的解码不一致，字符流服务器(tomcat)默认使用的是iso8859-1，浏览器读取数据时默认使用的是平台(系统)码，即GBK。

###### 字符流*：*字符流不能设置编码，只能通过设置请求头*。*

可以指定浏览器读取数据时使用utf-8读取，同时使用utf-8发送数据。

告诉浏览器发送的数据格式和编码，设置了这个头，服务器发送数据时也会默认此编码 发送。  
 response.setHeader("Content-Type","text/html;charset=utf-8");

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

###### 字节流：字节流可以设置发送数据的编码

在字符流设置header基础上多一步设置” 字符串编码 ”, 和发送编码无关

response.getOutputStream().write("你好OutPutStream".getBytes(“utf-8”));



#### 请求重定向

##### 方法

response.sendRedirect(request.getContextPath()+"/weixin2");

##### 特点：

###### 二次请求，二次响应

###### 浏览器地址栏会发生变化

###### 在浏览器端完成

###### 可以跳转到当前web应用外的网页(即所有网页)

###### 重定向不能通过request对象共享数据，第一次请求响应request已销毁

#### 定时刷新

##### 方法

response.setHeader("refresh","5;url="+request.getContextPath()+"/weixin2");

延时5秒请求重定向

response.setHeader("refresh","5;url=http://www.baidu.com");

跳转到外部，需加http协议，否则会认为是当前web应用的目录,会自动加上当前网页的目录，可以设置0秒

##### 特点

只是延迟的重定向，因为是通过设置请求头的方式，没有违反一次请求一次响应原则，所以重定向之前可以通过字节流或字符流输出内容

# ServletContext接口

### 概述

#### 代表整个Web应用的对象，即一个web应用只有一个ServletContext

### 生命周期

#### 启动

##### 服务器启动时，web应用加载时创建ServletContext，这个对象唯一代表web应用

#### 销毁

##### 直到服务器关闭，web应用销毁时，随着其销毁

### 获取ServletContext对象

#### 在当前web应用的任意Servlet中获取

##### this.getServletContext()

### ServletContext功能

#### 获取WEB应用初始化参数

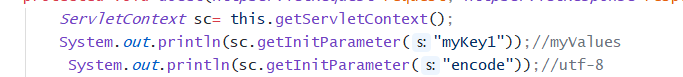
##### 配置初始化参数

###### 在web.xml中配置初始化参数



##### Servlet中获取参数

###### 如果希望在整个WEB应用中配置一些参数，可以通过ServletContext对象获取这些参数



#### 作为域对象

##### 特点：

###### 生命周期：和web应用的生命一样长

Web应用启动时创建，关闭时销毁

###### 作用范围：整个web应用

###### 功能：在整个web应用内共享数据

##### API:

和其他域的方法一样的

**setAttribute**(“key”,”value”)

**getAttribute**(“key”)

**removeAttribute**(“key”)

等…

#### 获取WEB资源文件路径

##### String str=sc.getRealPath("1.propertiies")

// E:\kaifa\JAVA\project\_idea\EasyMall\out\artifacts\EasyMall\_war\_exploded\1.propertiie

##### 返回当前项目下的路径,返回值会自动在参数前面补全web项目全路径

##### 即参数为当前项目的相对路径，返回绝对路径

##### 

# SESSION及Servlet中的HttpSession接口

### SESSION原理

#### 4.1.session原理

将会话中产生的数据保存在服务器端

计算机生成了可选文字:
服务器
浏览器
浏览器2
ServletA
电脑
ServletB
手机
电脑
手机
电脑
sesslon
手机
sesslon
电脑

浏览器向服务器发送请求，服务器接收请求参数，然后在服务器中检查是否有为当前浏览器服务的session，如果有则直接拿来使用，如果没有则创建一个session，并将数据保存到session中

当浏览器再次访问服务器时，服务器可以找到为当前浏览器服务的session，从中取出数据

通过这种方式也可以保存会话中产生的数据

每个session对应一个id，此id通过cookie（临时）保存在浏览器中，访问服务器时携带cookie，服务器根据这个id判断是哪个session

### Session域

#### 生命周期

##### 创建

第一次调用request.getSession()方法时创建session对象

*HttpSession* session = request.getSession();

如果服务器中已经有session，会直接拿来使用，如果没有才创建

##### 销毁

###### 超时死亡：

默认30分钟不使用会超时销毁，及时关闭浏览器未超时也不会销毁，Cookie关闭浏览器时会销毁(如果没有设置超时时间)

可以通过web.xml配置超时时间，单位为分钟

<session-config>

<session-timeout>1</session-timeout>

</session-config>

###### 主动杀死：

可以调用session.invalidate()方法，立刻杀死session  
session.invalidate();

###### 意外身亡：

当服务器意外关闭，session也会销毁

如果服务器正常关闭，session会被钝化，当服务器启动时再活化

#### Session数据的钝化与活化：

由于session中保存大量访问网站相关的重要信息，因此过多的session数据就会服务器性能的下降，占用过多的内存。因此类似数据库对象的持久化，web容器也会把不常使用的session数据持久化到本地文件或者数据中。这些都是有web容器自己完成，不需要用户设定。

不用的session数据序列化到本地文件中的过程，就是钝化；

当再次访问需要到该session的内容时，就会读取本地文件，再次放入内存中，这个过程就是活化。

#### 作用范围

整个会话

#### 作用

整个会话范围内共享数据

#### 使用

request.getSession(true)：若存在会话则返回该会话，否则新建一个会话。

request.getSession(false)：若存在会话则返回该会话，否则返回NULL

request.getSession()：默认为true

# 请求转发和请求重定向、定时刷新的区别

### 请求转发

在服务器端完成，所以在request对象中

|  |
| --- |
| 这里写图片描述 |

### 请求重定向

在浏览器端完成，所以在response对象中(响应给浏览器)

|  |
| --- |
| 这里写图片描述 |

### 定时刷新

只是延迟的重定向，因为时通过设置请求头的方式，没用违反一次请求一次响应原则，所以重定向之前 可以输出内容

|  |
| --- |
|  |

在同一个web应用内部跳转，需要通过request对象携带数据到目的地 -- 请求转发

在同一个web应用内部跳转，地址栏不能发生变化 -- 请求转发

需要跳转到其他web应用 -- 重定向或者定时刷新，根据是否需要指定多长时间跳转决定

如果在同一个web应用跳转，没有其他需求，可以选用任意方式跳转

# 四大作用域总结

### ServletContext

#### 生命周期：

Web应用启动时创建，web应用销毁时销毁

#### 作用范围：

整个web应用

#### 功能：

整个web应用共享数据

### Session

#### 生命周期：

第一次调用request.getSession()方法时创建

默认30分钟超时销毁、或者调用invalidate()方法主动销毁、或服务器宕机意外身亡

#### 作用范围：

整个会话

#### 功能：

整个会话范围共享数据

### Request

#### 生命周期：

一次请求开始创建，请求结束销毁

#### 作用范围：

整个请求链

#### 功能：

整个请求链范围内共享数据

### pageContext

#### 生命周期：

访问jsp页面时创建，访问jsp页面结束时销毁

#### 作用范围：

当前jsp页面

#### 功能：

整个jsp页面共享数据