day28-ServletContext&Response

学习目标

1.能够使用Response对象操作HTTP响应内容

2.能够处理响应乱码

3.能够完成文件下载案例

4.能够使用servletcontext域对象

5.能够说出servlet生命周期方法执行流程

Servlet进阶

一,Servlet的生命周期【重点】

1 ,生命周期概述

一个对象从创建到销毁的过程

2 ,Servlet生命周期方法

servlet从创建到销毁的过程

```
人的生命周期:          Servlet的生命周期:
出生                                初始化              init()
婴儿                                    服务(干活)    service()
童年                                    销毁             destory()
少年
青年
中年
老年
去世
```

出生：（初始化）用户第一次访问时执行。

活着：（服务）应用活着。每次访问都会执行。

死亡：（销毁）应用卸载。

serrvlet生命周期方法:

init(ServletConfig config)

service(ServletRequest req, ServletResponse res)

destroy()

3,Servlet生命周期描述【面试】

当客户端第一次请求的时候，会执行init方法,创建出来

客户端任何一次请求会执行service方法，来处理请求

当servlet从服务器移除或者服务器正常关闭会执行destroy方法,销毁

servlet是单例多线程的，服务器会针对每次请求获得一个线程来处理这个请求.

单例: 只有一个实例, init()调用一次, 只创建一次

多线程: 服务器会针对每次请求获得一个线程来处理这个请求

servlet里面尽量不要用全局的变量, 可能会导致线程不安全.

4.ServletConfig

Servlet的配置对象, 可以使用用ServletConfig来获得Servlet的初始化参数, 在SpringMVC里面会遇到

- 先在配置文件里面配置初始化参数

```
<servlet>
    <servlet-name>ServletDemo01</servlet-name>
    <servlet-class>com.itheima.a_servlet.ServletDemo01</servlet-class>
    <init-param>
        <param-name>akey</param-name>
        <param-value>aaa</param-value>
    </init-param>

</servlet>
<servlet-mapping>
    <servlet-name>ServletDemo01</servlet-name>
    <url-pattern>/demo01</url-pattern>
</servlet-mapping>
```

- 可以通过akey获得aaa

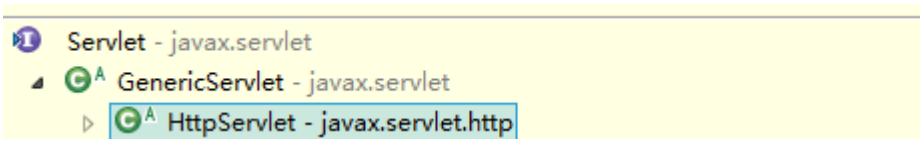
```
public void init(ServletConfig servletConfig) throws ServletException {
    System.out.println("init()....5555555555555"+servletConfig.getInitParameter( s: "akey"));

}
```

5. 配置启动项

- 默认情况下, Servlet是第一次请求时候创建. 能不能当服务器启动的时候创建

```
<servlet>
      <servlet-name>LifeServlet</servlet-name>
      <servlet-class>com.itheima.b_servlet.LifeServlet</servlet-class>
      <!--初始化参数-->
      <init-param>
          <param-name>akey</param-name>
          <param-value>aaa</param-value>
      </init-param>
      <!--配置启动项-->
      <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>LifeServlet</servlet-name>
      <url-pattern>/life</url-pattern>
  </servlet-mapping>
```

二,Servlet体系结构



- Servlet接口

  前面我们已经学会创建一个类实现sevlet接口的方式开发Servlet程序，实现Servlet接口的时候，我们必须实现接口的所有方法。但是，在servlet中，真正执行程序逻辑的是service，对于servlet的初始化和销毁，由服务器调用执行，开发者本身不需要关心。因此，有没有一种更加简洁的方式来开发servlet程序呢？

我们先来查阅API回顾Servlet接口：

由上图可知在servlet接口规范下，官方推荐使用继承的方式，继承GenericServlet 或者HttpServlet来实现接口，那么我们接下来再去查看一下这两个类的API：

- GenericServlet 类

javax.servlet

Class GenericServlet

java.lang.Object
  └ **javax.servlet.GenericServlet**

All Implemented Interfaces:
Serializable, Servlet, ServletConfig

Direct Known Subclasses:
HttpServlet

---

public abstract class **GenericServlet**
extends Object
implements Servlet, ServletConfig, Serializable

Implements: Servlet, ServletConfig, java.io.Serializable
Extended by: HttpServlet

定义一般的、与协议无关的 servlet。要编写用于 Web 上的 HTTP servlet，请改为扩展 javax.servlet.http.HttpServlet。

GenericServlet 实现 Servlet 和 ServletConfig 接口。servlet 可以直接扩展 GenericServlet，尽管扩展特定于协议的子类（比如 HttpServlet）更为常见。

GenericServlet 使编写 servlet 变得更容易。它提供生命周期方法 init 和 destroy 的简单版本，以及 ServletConfig 接口中的方法的简单版本。GenericServlet 还实现 log 方法，在 ServletContext 接口中对此进行了声明。

要编写一般的 servlet，只需重写抽象 service 方法即可。

阅读上图API可知，GenericServlet 是一个类，它简化了servlet的开发，已经提供好了一些servlet接口所需的方法，我们开发者只需要重写service方法即可

我们来使用GenericServlet 创建servlet：

1. 创建一个类
2. 继承GenericServlet
3. 重写service方法

```
package cn.itcast.web;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet(name = "GenericDemoServlet",urlPatterns = "/generic")
public class GenericDemoServlet extends GenericServlet {
    @Override
    public void service(ServletRequest servletRequest, ServletResponse servletResponse) throws
ServletException, IOException {
        System.out.println("GenericDemoServlet执行.......");
    }
}
```

虽然，GenericServlet已经简化了servlet开发，但是我们平时开发程序需要按照一种互联网传输数据的协议来开发程序——http协议，因此，sun公司又专门提供了HttpServlet，来适配这种协议下的开发。

- HttpServlet

阅读上图的API可知，继承HttpServlet，我们需要重写doGet、doPost等方法中一个即可，根据Http不同的请求，我们需要实现相应的方法。

我们来使用HttpServlet创建servlet：

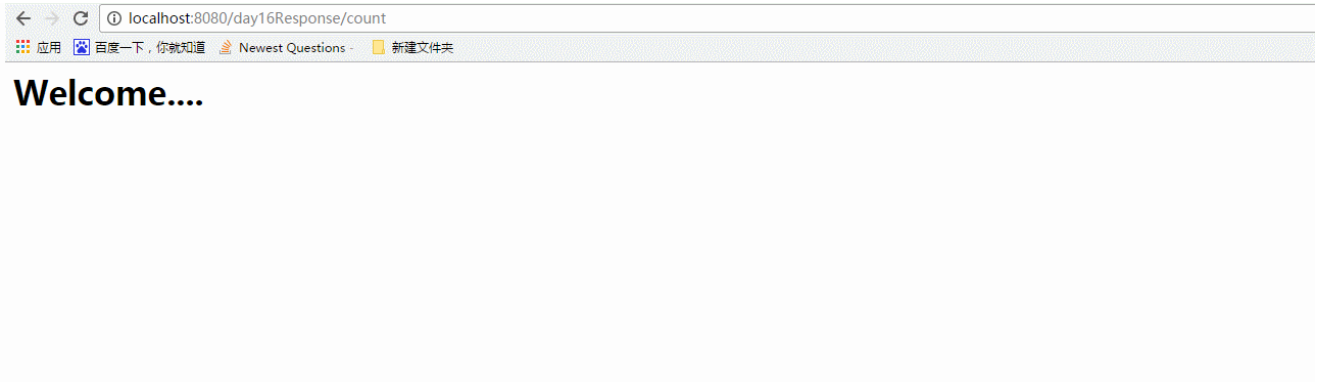
1. 创建一个类
2. 继承HttpServlet
3. 重写doGet方法

```
package cn.itcast.web;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "HttpDemoServlet",urlPatterns = "/http")
public class HttpDemoServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("HttpDemoServlet执行.......");
    }
}
```

通过以上两个API阅读，同学们注意一个细节HttpServlet是GenericServlet的子类，它增强了GenericServlet一些功能，因此，在后期使用的时候，我们都是选择继承HttpServlet来开发servlet程序。

案例一：统计一下网站被访问的总次数.

一，需求分析



- 在页面中显示您是第x位访问的用户.

二，技术分析

1.servletContext概述

服务器为每一个应用(项目)都创建了一个servletContext。servletContext属于整个应用的，不局限于某个servlet。

servletContext当做全局管理者;

eg: 项目当做咱班, Servlet当做每一位同学,ServletContext当做班主任

2.作用

作为域对象存取数据

获得文件mini类型（文件上传和下载）

获得全局初始化参数

获取web资源路径

3作为域对象存取数据

范围: 在当前应用，使多个Servlet共享数据

- getAttribute(String name) ;向ServletContext对象的map取数据
- setAttribute(String name, Object object) ;从ServletContext对象的map中添加数据
- removeAttribute(String name) 根据name去移除数据

三，思路分析

- 在CountServlet的init方法里面，存储登录次数（0次）
- 在CountServlet的doGet()方法里面，取出值+1，再存储
- 在ShowServlet里面取出值展示

四，代码实现

五，总结

1.servletContext

1.1作为域对象存取值【重点】

- getAttribute(String name) ;向ServletContext对象的map取数据
- setAttribute(String name, Object object) ;从ServletContext对象的map中添加数据
- removeAttribute(String name) ;根据name去移除数据

1.2获得文件mini类型（文件下载）

- getMimeType(String fileName)

1.3获得全局初始化参数【重点】 后面Spring会用到

- String getInitParameter(String name) ; //根据配置文件中的key得到value

在web.xml配置

```
<!--全局的初始化参数,通过ServletContext来获得的,也就意味着任何的Servlet都有可以获得-->
<context-param>
    <param-name>bkey</param-name>
    <param-value>bbb</param-value>
</context-param>
```

通过ServletContext来获得

```
//2.获得全局的初始化参数
String value = getServletContext().getInitParameter( s: "bkey");
System.out.println("value="+value);
```

1.4获取web资源路径【重点】

- String getRealPath(String path);根据资源名称得到资源的绝对路径.
- getResourceAsStream(String path) ;返回制定路径文件的流

注意: filepath:直接从项目的根目录开始写

案例二：完成文件下载

一需求分析

- 创建文件下载的列表的页面,点击列表中的某些链接,下载文件.

文件下载的列表页面

超链接的下载

hello.txt
cs10001.jpg
hello.zip

手动编码方式下载

hello.txt
cs10001.jpg
hello.zip
美女.jpg

二，技术分析

1. HttpServletResponse概述

在Servlet API中，定义了一个HttpServletResponse接口，它继承自ServletResponse接口，专门用来封装HTTP响应消息。由于HTTP响应消息分为响应行、响应消息头、消息体三部分，因此，在HttpServletResponse接口中定义了向客户端发送响应状态码、响应头、响应体的方法。

2.操作响应三部分

2.1操作响应行

```
HTTP/1.1 200 OK
```

void	**setStatus**(int sc) Sets the status code for this response.

常用的状态码：

200：成功

302：重定向

304：访问缓存

404：客户端错误

500：服务器错误

2.2操作响应头

一个key对应多个value

void	**setDateHeader**(String name, long date) Sets a response header with the given name and date-value.
void	**setHeader**(String name, String value) Sets a response header with the given name and value.
void	**setIntHeader**(String name, int value) Sets a response header with the given name and integer value.

一个key对应一个value

void	**addDateHeader**(String name, long date) Adds a response header with the given name and date-value.
void	**addHeader**(String name, String value) Adds a response header with the given name and value.
void	**addIntHeader**(String name, int value) Adds a response header with the given name and integer value.

掌握的方法: setHeader(String key,String value);

常用的响应头

Refresh:定时跳转

Location:重定向

Content-Disposition:设置文件下载时候的头

Content-Type：设置响应内容的MIME类型

2.3响应体

ServletOutputStream	**getOutputStream**() Returns a ServletOutputStream suitable for writing binary data in the response.
PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client.

3.文件下载

3.1什么是文件下载

将服务器上已经存在的文件,输出到客户端浏览器.

说白了就是把服务器端的文件拷贝一份到客户端, 文件的拷贝---> 流(输入流和输出流)的拷贝

3.2文件下载的方式

- 第一种:超链接方式（不推荐）

  链接的方式：直接将服务器上的文件的路径写到href属性中.如果浏览器不支持该格式文件(压缩文件),那么就会提示进行下载,如果浏览器支持这个格式(eg: png, jpg....)的文件,那么直接打开,不再下载了

- 第二种:手动编码方式（推荐）

  手动编写代码实现下载.无论浏览器是否识别该格式的文件,都会下载.

3.3手动编码方式要求

设置两个头和一个流

设置的两个头:

Content-Dispostion:浏览器识别该格式文件,提示浏览器下载.

Content-Type:文件类型.(MIME的类型)

设置一个流：

获得要下载的文件的输入流.

三，思路分析

1.超链接方式

- 创建资源下载页面。
- 设置超链接，href值设置资源的路径

2.编码方式

- 创建资源下载页面
- 设置超链接，把文件名提交到downloadServlet中，设置两头一流，进行下载

四，代码实现

```
String fileName = request.getParameter("fileName");
System.out.println("fileName=" + fileName);

// 设置两头一流(文件下载类型和指示客户端下载文件)
// 告诉浏览器要下载
response.setHeader("Content-Disposition", "attachment;filename=" + fileName);

// 告诉浏览器的文件类型
String type = getServletContext().getMimeType(fileName);
// response.setContentType(type);
response.setHeader("Content-Type", type);

// 设置文件的输入流
String realPath = getServletContext().getRealPath("/download/" + fileName);
InputStream is = new FileInputStream(realPath);

OutputStream os = response.getOutputStream();

byte[] b = new byte[1024];
int len = 0;

while ((len = is.read(b)) != -1) {
        os.write(b, 0, len);
}
```

五，总结

1.下载中文的文件

中文文件在不同的浏览器中编码方式不同：

火狐是Base64编码,

其它浏览器(谷歌 360...)是URL编码, 字符集写成utf-8

```
if(agent.contains("Firefox")){
                        // 火狐浏览器
                        filename = base64EncodeFileName(filename);
                }else{
                        // IE，其他浏览器
                        filename = URLEncoder.encode(filename, "UTF-8");
                }

public static String base64EncodeFileName(String fileName) {
                BASE64Encoder base64Encoder = new BASE64Encoder();
                try {
                        return "=?UTF-8?B?"
                                        + new String(base64Encoder.encode(fileName
                                                        .getBytes("UTF-8"))) + "?=";
                } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                        throw new RuntimeException(e);
                }
        }
```

2 Response其他操作

2.1定时刷新

```
response.setHeader("refresh","秒数;url=跳转的路径"); //几秒之后跳转到指定的路径上
```

2.2重定向【重点】

void	**sendRedirect**(String location) Sends a temporary redirect response to the client using the specified redirect location URL.

- 重定向是两次请求
- 重定向地址栏路径改变
- 重定向路径写绝对路径(可以是远程的[服务器外部的], 也可以写当前项目里面的)

2.3 向页面输出内容:

ServletOutputStream	**getOutputStream**() Returns a ServletOutputStream suitable for writing binary data in the response.
PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client.

页面输出只能使用其中的一个流实现,两个流是互斥的.

- 解决字符流输出中文乱码问题

```
        //设置缓冲器的编码
        response.setCharacterEncoding("utf-8");
        //告知浏览器用什么编码解析
        response.setHeader("Content-Type", "text/html;charset=utf-8");
        //或者,是前面两句的封装
        response.setContentType("text/html;charset=utf-8");
        response.getWriter().println("你好！")
```

- 使用字节输出流输出中文乱码问题

```
//设置浏览器打开方式
response.setHeader("Content-type", "text/html;charset=utf-8");
//得到字节输出流
ServletOutputStream outputStream = response.getOutputStream();

outputStream.write("你好".getBytes("utf-8"));// 使用平台的默认字符(utf-8)集将此 String 编码为 byte 序列
```