学学课堂

课程介绍：

通过前面知识点的学习，我们对于请求的处理已经可以说比较灵活了，但是还不够。我们再介绍两个重要的对象 ServletContext 对象和 ServletConfig 对象

ServletContext 对象：

问题：

1. request:在一次请求里面共享数据，作用域在一个请求中
2. session：在同一个用户下共享数据，作用域在同一个用户下,一个用户下只有一个 session 对象
3. 在不同用户下共享数据怎么办呢？ServletContext 一个工程下只有一个对象

解决：

ServletContext (tomcat 容器的上下文对象,对象在一个 tomcat 中有且只有一个,作用范围?)

设置与获取 ServletContext 对象

```
package com.servlet;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
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 = "ServletA",urlPatterns = "/ServletA")
public class ServletA extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //ServletContext 的创建通过以下三种方式(既是创建 ServletContext,又是获取,如果当前工程
下没有就创建,有就获取,一个工程下只有一个 ServletContext 对象)
        ServletContext sc1 = this.getServletContext();
        ServletContext sc2 = req.getSession().getServletContext();
        ServletContext sc3 = this.getServletConfig().getServletContext();
        System.out.println(sc1.hashCode());
        System.out.println(sc2.hashCode());
        System.out.println(sc3.hashCode());
        //从输出结果可以看出,输出的是相同的哈希码,说明是同一个对象
    }
}
```

总结：

1.创建 ServletContext 对象

2.存储用户共享信息，方式：sc.setAttribute(String name,Object ob)
3.获取用户共享信息：a、创建 ServletContext 对象;
4.获得方式 b、sc.getAttribute(String name);
5.删除共享数据：sc.removeAttribute(String name);

```
package com.servlet;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
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 = "ServletA",urlPatterns = "/ServletA")
public class ServletA extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        ServletContext sc = this.getServletContext();
        sc.setAttribute("a","abc");
        sc.setAttribute("b","bbb");
    }
}
```

```
package com.servlet;

import javax.servlet.ServletContext;
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 = "ServletB",urlPatterns = "/ServletB")
public class ServletB extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        ServletContext sc = this.getServletContext();
        String a = (String)sc.getAttribute("a");
        String b = (String)sc.getAttribute("b");
        resp.getWriter().write(a);
        resp.getWriter().write("b:"+b);
    }
}
```

```
package com.servlet;

import javax.servlet.ServletContext;
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 = "ServletC",urlPatterns = "/ServletC")
public class ServletC extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        ServletContext sc = this.getServletContext();
        sc.removeAttribute("b");
    }
}
```

读取配置文件的配置信息

```
package com.servlet;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
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 = "ServletA",urlPatterns = "/ServletA")
public class ServletA extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        ServletContext sc = this.getServletContext();
        sc.setAttribute("a","abc");
        sc.setAttribute("b","bbb");
        String flag = sc.getInitParameter("flag");
        if(flag.equals("true")){
            System.out.println("定期清理缓存等等....");
        }
        String city = sc.getInitParameter("city");
        System.out.println(city);
```

```
    }
}
```

web.xml 中的配置信息:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!--工程的全局配置信息-->
    <context-param>
        <param-name>flag</param-name>
        <param-value>false</param-value>
    </context-param>
    <context-param>
        <param-name>city</param-name>
        <param-value>changsan</param-value>
    </context-param>
</web-app>
```

作用：

解决了不同用户的数据共享问题

原理：
ServletContext 对象由服务器进行创建，一个项目只有一个对象。不管在项目的任意位置
进行获取得到的都是同一个对象，那么不同用户发起的请求获取到的也就是同一个对象了，
该对象由访问此项目工程(作用域在整个项目工程中)的所有用户共同拥有。

特点：
ServletContext 由服务器进行创建,在服务器启动的时候创建,目的:提供一个对象,用于开发者访问 tomcat 容器的内部数据

用户共享

一个项目只有一个

生命周期：

服务器启动到服务器关闭

作用域：

项目（工程）内

使用：
获取 ServletContext 对象
使用作用域进行共享数据流转
获取 web.xml 中的全局配置

获取已经部署在 项目中 web 下资源流对象及资源的绝对路径

```
package com.servlet;

import javax.servlet.ServletContext;
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.File;
import java.io.FileInputStream;
import java.io.IOException;

@WebServlet(name = "ServletD",urlPatterns = "/ServletD")
public class ServletD extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        File f = new File("files/a.txt");
        System.out.println("绝对路径:"+f.getAbsolutePath());
        /*
        取出来的路径与真实路径不一致:D:\Program Files
(x86)\apache-tomcat-9.0.37\bin\files\a.txt
        真实路径 E:\workspace\FourWeb\out\artifacts\FourWeb_war_exploded\files\a.txt
         */
        ServletContext sc = this.getServletContext();
        String path = sc.getRealPath("/");//获取当前工程的根路径

System.out.println("path:"+path);//E:\workspace\FourWeb\out\artifacts\FourWeb_war_expl
oded\
        String filePath = sc.getRealPath("/files/a.txt");
        System.out.println("filePath:"+filePath);
        FileInputStream in = new FileInputStream(filePath);
        System.out.println((char)in.read());
        System.out.println((char)in.read());
        System.out.println((char)in.read());

    }
```

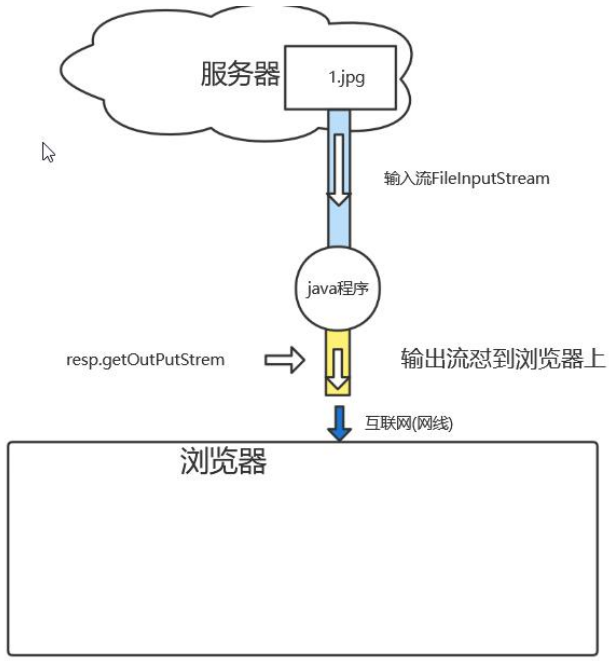
```
}
```

基于服务器文件的上传和下载

从服务器下载文件

Content-Disposition 是 MIME 协议的扩展，MIME 协议指示 MIME 用户代理如何显示附加的文件。当 Internet Explorer 接收到头时，它会激活文件下载对话框。

设置头文件："Content-Disposition", "attachment;filename="+filename




```
package com.servletcontext;

import javax.servlet.ServletContext;
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.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.OutputStream;

@WebServlet("/ServletContextE")
public class ServletContextE extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        ServletContext sc = this.getServletContext();
        //获取到项目的根目录
```

```
        String realPath = sc.getRealPath("/");
        //把服务器的根路径封装成文件对象
        File parentPath = new File(realPath);
        //在指定的父路径(parentPath)下创建子目录 images public File(File parent, String child)
        File imagePath = new File(parentPath,"images");
        //创建目录: /images
        if(!imagePath.exists()) {
            //创建目录
            boolean a = imagePath.mkdir();
            System.out.println(a);
        }
        //下载文件
        //首先要获取文件所在的绝对路径
        String realPath1 = sc.getRealPath("/images/1.jpg");

System.out.println("realPath1:"+realPath1);//E:\workspace\MyWebTest\out\artifacts\MyWebTest_war_exploded\
images\1.jpg
        //网络传输,搞一根管子怼到服务器上
        FileInputStream fileInputStream = new FileInputStream(realPath1);
        //先获取要下载的文件的名字
        //先把文件的路径封装从 File 对象
        File file = new File(realPath1);
        //取文件名
        String fileName = file.getName();
        //使用附件的形式下载
        //设置响应头
        resp.setHeader("Content-Disposition", "attachment;filename="+fileName);
        //把服务器中的文件数据响应给浏览器
        OutputStream out = resp.getOutputStream();//相当于输出流怼到客户端的浏览器上
        //fileInputStream 与 out 进行数据传递
        byte[] b = new byte[1024];
        int len;
        while((len= fileInputStream.read(b)) !=-1){
            out.write(b,0,len);
        }
        out.close();
        fileInputStream.close();
    }
}
```

ServletConfig 对象（了解）

问题：

使用 ServletContext 对象可以获取 web.xml 中的全局配置文件，

在 web.xml 中每个 Servlet 也可以进行单独的配置，那么该怎么获取配置信息呢？

解决：

使用 ServletConfig 对象

作用：

ServletConfig 对象是 Servlet 的专属配置对象，每个 Servlet 都单独拥有一个 ServletConfig 对象，用来获取 web.xml 中的局部配置信息。

使用：

获取 ServletConfig 对象

获取 web.xml 中 servlet 的配置信息

```
<servlet>
    <servlet-name>ServletE</servlet-name>
    <servlet-class>com.servlet.ServletE</servlet-class>
    <!--每个 servlet 自己的配置信息-->
    <init-param>
        <param-name>city</param-name>
        <param-value>beijing</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>ServletE</servlet-name>
    <url-pattern>/ServletE</url-pattern>
</servlet-mapping>
```

```
package com.servlet;

import javax.servlet.ServletConfig;
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 = "ServletE",urlPatterns = "/ServletE")
public class ServletE extends HttpServlet {
```

```
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        ServletConfig sc = this.getServletConfig();//域对象,类似于ServletContext()
        String city = sc.getInitParameter("city");
        System.out.println(city);
    }
}
```

针对每个 servlet 可以设置配置信息,通过 servletconfig 获取

Web.xml 文件使用总结（了解）：

每个 web 项目下都具备 web.xml 文件。通过前面的学习，我们对 web.xml 文件有了一定了解，但是还不够，本节课就 web.xml 文件进行总结。同时对 Tomcat 下的 server.xml 进行介绍。

tomcat 下还有一个 Web.xml 文件,它的作用：
存储项目相关的全局配置信息，保护 Servlet。解耦一些数据对程序的依赖。

使用位置：
每个 Web 项目中
Tomcat 服务器中(在服务器目录 conf 目录中)

区别：

Web 项目下的 web.xml 文件为局部配置，针对本项目的位置。
Tomcat 下的 web.xml 文件为全局配置，配置公共信息，针对部署到 tomcat 中的所有项目有效。
Web 项目和 Tomcat 下都配置相同的信息：
就近原则,取 web 项目下 web.xml 文件中的配置信息
区别：

内容(核心组件)：
全局上下文配置(全局配置参数)<context-param></context-param>
Servlet 配置
过滤器配置
监听器配置

加载顺序(了解)：
Web 容器会按 ServletContext -> context-param -> listener ->filter -> servlet 这个顺序加载组件，这些元素可配置在 web.xml 文件中的任意位置。

Web 项目下的 web.xml 和 Tomcat 下的 web.xml 加载时机：都是在服务器启动时。

在 idea 中启动服务器默认会访问 index 开头的 jsp 或 html 文件是因为
在 tomcat/cof/web.xml 下含有这个配置

```
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
```

server.xml 文件(了解):

问题:

浏览器发起请求后，服务器根据请求在 webapps 目下调用对应的 Servlet 进行请求处理。那么为什么是 webapps 目录难道不能是其他的目录吗?