【拯救者】Ep_前言(必看)

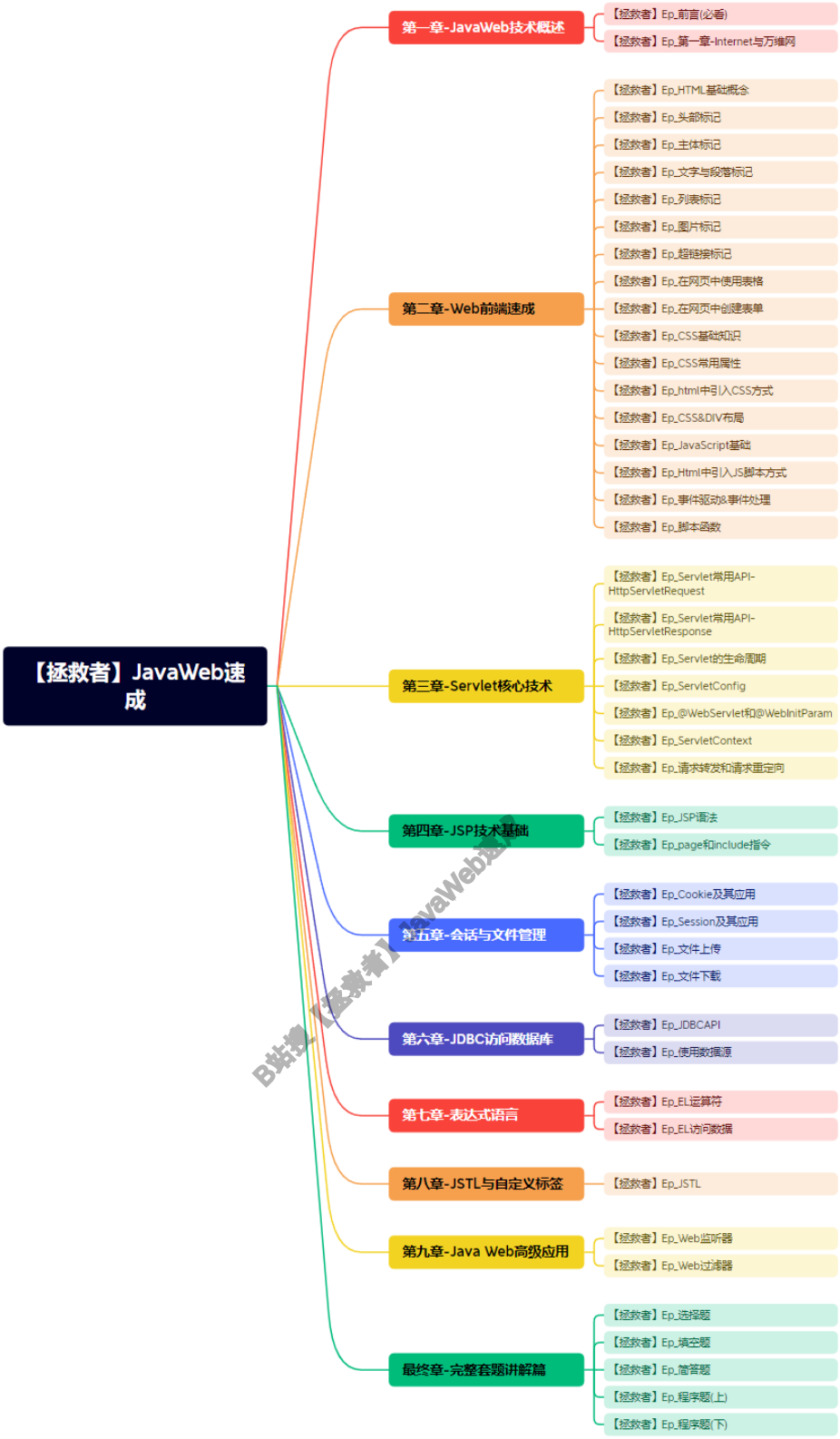
前言:

这里讲的是【 期末速成 速成 速成】版本，按课本章节来， 抽取重点，翻译为大话！！！ 适用于所有JavaWeb课本哈 适用于所有JavaWeb课本 适用于所有JavaWeb课本

学习本门课程，**无需课本， 下载下这个文档**，最后**会解题即可**！**遇到不会的**，**翻看文档**如何去破局！

好评:

课程包括:



因为**不同学校书本不一样**, 但是基本**讲或者考的内容一样**,所以**目录大家不用介意**,把我**讲的跟一遍**在**看你们考的就会了**

学习步骤:

每一章，都会【**先讲会考的基础**】， 坚持**学完全部章**即可 **拿捏期末**,

同时在**课程最后**提供一个**新**的【**整套题的讲解**】，进行**巩固拔高**

个人简介:

现阿里巴巴技术专家,【拯救者】系列课程广受好评,讲课风趣幽默，被誉为BGM之王（慢慢看就知道了）

文档下载&答疑裙：

请看最后一集哈，**文档+整套期末题讲解**都**给大家提供**的哦 **宝们**，**下载**完记得打开**【大纲模式】**（这样就会跟我一样有目录了）

不会打开大纲模式?

教程: https://bbs.csdn.net/topics/609294605

开始大家**不是很熟悉**， **别着急**

当你**二刷时**， 只会**感叹**:

【回首看，已如古人】~

让我们一起**出征【JavaWeb速成】吧** ，都多余了哦~~~

ps:目录为我为了讲课顺序自己制作,

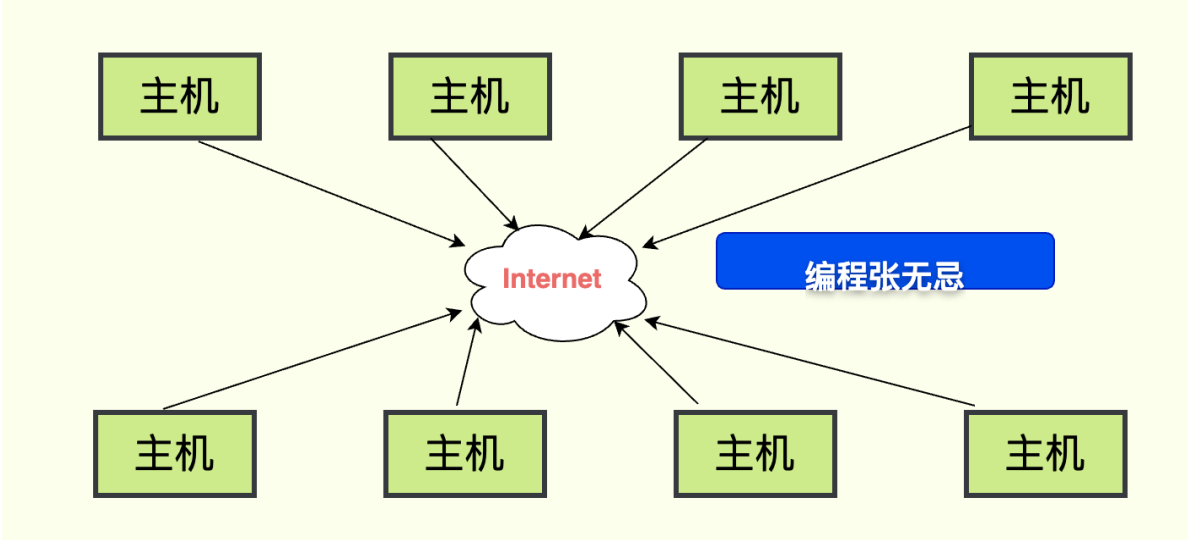
> 别说你的目录跟我书的不一样，我就不学了， 考的内容是一样的哦！

第一章-JavaWeb技术概述

【拯救者】Ep_第一章-Internet与万维网

主机和IP地址

主机:



连接到Internet的所有计算机都叫主机

IP地址

```
目前常用的IP地址用4个字节32位二进制数表示，如某计算机的IP地址
可表示为10101100 00010000 1111110 0000001。
  将它们分为4组,每组8位一个字节,由小数点分开，且将每个字节的二进制用十进制数表示，
  如上述地址可表示为172.16. 254.1
```

分为IPv4与IPv6两个版本。IPv6 采用128位地址长度

域名和DNS

不管用哪种方法表示IP地址,这些数字都很难记住,为了方便人们记忆, **使用域名来表示主机。**

域名

是由一串用**点分隔的名字组成的** 名称, **最末的域称为顶级域,其他的域称为子域**

例如, **tsinghua. edu.cn**是一个域名, 其中最后的.cn是域名的第一层,.edu是第二层, tsinghua是真正的域名

其中第一层的.cn是地理顶级域名。

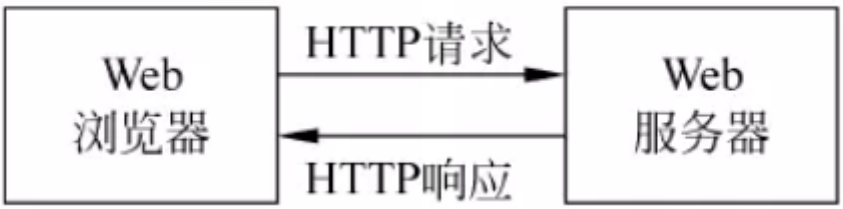
DNS

由于IP地址是Internet 内部使用的地址,因此当Internet主机间进行通信时必须采用IP地址进行寻址,所以**当使用域名时必须把域名转换成IP地址。**

HTTP URL URI

HTTP

HTTP(Hypertext Transfer Protocol)称为**超文本传输协议**，它是 **Web使用的协议。**该协议详细**规定了Web客户端与服务器之间如何通信。**



▼ **响应标头**　　查看解析结果

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Methods: POST, GET
Access-Control-Allow-Origin: https://www.baidu.com
Content-Length: 130
Content-Type: application/json
Date: Sun, 29 Jan 2023 01:00:04 GMT
Tracecode: 57299540042980996642623464458012909
```

请求网址: https://ug.baidu.com/mcp/pc/pcsearch

请求方法: POST

状态代码: 200 OK

远程地址: 39.156.66.219:443

引荐来源网址政策: unsafe-url

▾请求标头

```
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: keep-alive
Content-Length: 56
Content-Type: application/json
Cookie: PSTM=1665830602; BIDUPSID=1E1836B24646FBD7DF967D04F67
vwdWWhfos]=mk3SLVN4HKm; H_PS_PSSID=36557_38051_36920_37990_3
```

网页演示:

URL

Web服务器上的**资源是通过URL标识的。**URL(Uniform Resource Locator)称为**统一资源定位器**,指Internet上位于某个主机上的资源。资源包括HTML文件、图像文件和程序等。例如,下面是一些合法的URL。
http : / / www. baidu.com/index.html

http://localhost:8080/helloweb/hello.jsp
URL通常由4部分组成:**协议名称**、所在主机的 **域名或IP地址**、**可选的(端口号和资源的名称)。**

URI

> URI(Uniform Resource Identifier) 是统一资源标志符，可以唯一标识一个资源。
> URL(Uniform Resource Location) 是统一资源定位符，可以提供该资源的路径。即 URL 除了可以用来标识一个资源，而且还指明了如何 locate 这个资源。
> URI的作用像身份证号一样，URL的作用更像家庭住址一样。URL是一种具体的URI，它不仅唯一标识资源，而且还提供了定位该资源的信息。

常见的 web服务器

Tomcat: 由Apache组织提供的一种web服务器，提供对jsp和Servlet的支持。这是一个轻量级的Javaweb服务器，也是当前应用最广的javaweb服务器。

软件安装

百度搜 idea tomcat,请按教程安装. 也可以看我视频讲解

ps:目录为我为了讲课顺序自己制作,

> 别说你的目录跟我书的不一样，我就不学了， 考的内容是一样的哦!

第二章-Web前端速成

请看讲解视频,这个是引用的之前的课程视频哈,考试不是重点

第三章-Servlet核心技术

【拯救者】Ep_Servlet常用API-HttpServletRequest

啥是Servlet

Servlet是**运行在Web服务器**或**应用服务器上的java程序**，它是一个中间层，负责连接来自web浏览器或其他HTTP客户程序和 **[HTTP服务器]** 上应用程序

Servlet执行下面的**任务**：

```
1）读取客户发送的显示数据。
2）读取由浏览器发送的隐式请求数据。如：http请求头
3） 向客户端发送显示数据。servlet和jsp最重要的任务就是将结果包在文本（html）、二进制（图片）等格式的文件中。
4）发送隐式的HTTP响应数据。如：http响应头
```

一、HttpServlet

HttpServlet是我们写Servlet代码创建类时需要**继承**的父类。

HttpServlet核心方法

方法名称	调用时机
init	HttpServlet被实例化时调用一次
destroy	HttpServlet对象被销毁前调用一次
service	每收到一个HTTP请求时调用一次
doGet	收到GET请求时调用(由service方法调用)
doPost	收到POST请求时调用(由service方法调用)
doPut/doDelete/doOptions……	收到其他对应请求时调用(由service方法调用)

二、HttpServletRequest

Tomcat会把收到的HTTP**请求按照HTTP协议的格式解析成一个HttpRequest对象。**

1、HttpServletRequest核心方法

方法	描述
String getProtocol()	返回请求协议的名称和版本
String getMethod()	返回请求的HTTP方法名称
String getURI()	返回URL中端口之后的部分
String getContextPath()	返回请求的ContextPath
String getQueryString()	返回请求中的查询字符串(query string)
Enumeration getParameterNames()	返回query string中所有key的名称， 返回值类型是一个String对象的枚举
String getParameter(String name)	返回query string中指定的key所对应的一个value， 如果key不存在则返回null
String[] getParameterValues(String name)	返回query string中指定的key所对应的所有value， 如果key不存在则返回null
Enumeration getHeaderNames()	返回HTTP请求头中所有的key， 返回值类型是一个String对象的枚举
String getHeader(String name)	返回请求头中指定的key所对应的value
String getCharacterEncoding()	返回请求主体中使用的字符编码名称
String getContentType()	返回请求主体中使用的数据类型， 如果不知道类型则返回null
int getContentLength()	以字节为单位返回请求主体的长度， 如果长度未知则返回-1
InputStream getInputStream()	返回一个InputStream对象，用于读取请求的body内容

2、代码示例

```
  <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
```

打印请求信息

```
public class RequestDemo1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
        //指定响应的数据格式以及使用的字符集
        resp.setContentType("text/html;charset=utf8");
        StringBuilder sb = new StringBuilder();
        //获取请求协议的名称和版本号
        sb.append(req.getProtocol());
        sb.append("<br>");//html的换行
        //获取请求的方法名称
        sb.append(req.getMethod());
        sb.append("<br>");
        //获取请求的URI和ContextPath
        sb.append(req.getRequestURI());
        sb.append("<br>");
        sb.append(req.getContextPath());
        sb.append("<br>");
        //获取query string
        sb.append(req.getQueryString());
        sb.append("<br>");
        //获取请求头中的键值对
        Enumeration<String> headerNames = req.getHeaderNames();
        while (headerNames.hasMoreElements()){
            //获取请求头中的key
            String name = headerNames.nextElement();
            //获取请求头中的value
            String value = req.getHeader(name);
            sb.append(name+": "+value);
            sb.append("<br>");
        }
        resp.getWriter().write(sb.toString());
    }
}
```


获取GET请求中的query string

```
public class GetParameterServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //假设请求中有两个参数：id和name
        String id = req.getParameter("id");
        String name = req.getParameter("name");
        System.out.println(id);
        System.out.println(name);
        resp.getWriter().write(id + ", " + name);
    }
}
```

要在服务器中**指定响应的数据格式和字符集**，否则浏览器可能乱码

```
 resp.setContentType("text/html;charset=utf8");
```

获取POST请求中的query string(form表单形式)

```
public class GetParameterServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //假设请求中有两个参数：id和name
        String id = req.getParameter("id");
        String name = req.getParameter("name");
         req.setCharacterEncoding("utf8");
        System.out.println(id);
        System.out.println(name);
        resp.setContentType("text/html;charset=utf8");
        resp.getWriter().write(id + ", " + name);
    }
}
```

通过form表单构造一个POST请求：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="/getParameter" method="post">
        <input type="text" name="id">
        <input type="text" name="name">
        <input type="submit" value="提交">
    </form>
</body>
</html>
```


【拯救者】Ep_Servlet常用API-HttpServletResponse

Servlet中的**doXXX方法会根据请求计算响应，并把响应的数据设置到HttpResponse对象中**，然后Tomcat就会把这个HttpResponse对象通过Socket写回给浏览器。

1、HttpServletResponse核心方法

方法	描述
void setStatus(int sc)	设置响应的状态码
void setHeader(String name,String value)	设置一个指定名称和值的响应头，如果name已存在，则覆盖旧的值
void addHeader(String name,String value)	设置一个指定名称和值的响应头，如果name已存在，不会覆盖旧的值，添加新的键值对
void setContentType(String type)	设置被发送的客户端的响应的数据格式
void setCharacterEncoding(String charset)	设置被发送到客户端的响应的字符编码
void sendRedirect(String location)	使用指定的重定向位置URL发送临时重定向响应到客户端
PrintWriter getWriter()	用于往body中写入文本格式数据
OutPutStream getOutPutStream()	用于往body中写入二进制格式数据

2、代码示例

自动刷新

让浏览器每3s自动刷新一次，并显示时间戳：

```
public class AutoRefreshServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //通过设置HTTP响应报头中的refresh字段，可以控制浏览器自动刷新的时机
        resp.setHeader("refresh","3");
        //返回当前的时间戳
        resp.getWriter().write(System.currentTimeMillis()+"");
    }
}
```

重定向

实现一个程序，返回一个重定向HTTP响应，自动跳转到CSDN主页：

```
@WebServlet("/redirect")
public class RedirectServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

            resp.sendRedirect("Location","https://www.csdn.net");
    }
}
```

【拯救者】Ep_Servlet的生命周期

Servlet的声明周期指的是**Servlet从创建到销毁的过程**：

(1) 当一个请求从HTTP服务器转发给Servlet容器时，容器会检查对应的Servley是否创建，如果没有创建就实例化该Servlet，并调用init()方法，inint()方法只调用一次，后续的请求都从第二步开始；

(2) Servlet每收到一个请求，就会调用一次service()方法，根据请求类型调用对应的方法，doGet、doPost等；

(3) Servlet销毁前调用一次destroy()方法进行清理操作，该方法只调用一次，随后JVM回收资源。

```
<web-app>
    <servlet>
        <servlet-name>Demo2</servlet-name>
        <servlet-class>Demo2</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Demo2</servlet-name>
        <url-pattern>/demo2</url-pattern>
    </servlet-mapping>
</web-app>

public class Demo2 extends HttpServlet {

  private String message;

  public void init() throws ServletException
  {
      // 执行必需的初始化
      message = "Hello ZwJ";
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
  }

  public void destroy()
  {
       System.out.println("我被销毁了");
  }
}
```

总结

web启动 **执行init()方法且只执行一次，每次请求都会执行service方法，项目结束执行destroy方法。**

【拯救者】Ep_ServletConfig

通俗一点说，ServletConfig就代表当前Servlet在web.xml中的配置信息。

当servlet引擎创建一个servlet实例对象后，调用该实例对象的**init(ServletConfig config)方法将Servlet对象传递给Servlet。**

```
<servlet>
    <servlet-name>Demo4</servlet-name>
    <servlet-class>com.zx.Demo4 </servlet-class>
    <init-param>
        <param-name>age</param-name>
        <param-value>99</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Demo4</servlet-name>
    <url-pattern>/demo4</url-pattern>
</servlet-mapping>
```



```
@Override
public void init(ServletConfig servletConfig) throws ServletException {
    //获取一下我们在web.xml中配置的参数
    String age = servletConfig.getInitParameter("age");
    System.out.println(age);
}
```

【拯救者】Ep_@WebServlet和@WebInitParam

在以前的servlet中我们初始化一些**参数都是配置在web.xml**中的，自从**servlet3.0之后给我们提供了注解@WebServlet和@WebInitParam**，@WebServlet是用来配置servlet的属性的，@WebInitParam是用来配置一些初始化属性的。

使用如下：

```
    <servlet>
        <servlet-name>Demo5</servlet-name>
        <servlet-class>com.zx.Demo5 </servlet-class>
        <init-param>
            <param-name>name</param-name>
```

```
            <param-value>bcZwj</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>Demo5</servlet-name>
        <url-pattern>/demo5</url-pattern>
    </servlet-mapping>
```

```
@WebServlet(
    name = "Demo5", urlPatterns = {"/demo5"}
    ,initParams = {
    @WebInitParam(name= "name", value="bcZwj"),
    }
)
public class Demo5 extends HttpServlet{
    @Override
public void init(ServletConfig servletConfig) throws ServletException {
    this.config=servletConfig;
}

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h2> AnnotationDemo</h2>");
        String pValue= config.getInitParameter("name");
        out.println("value: "+pValue);
        out.close();
    }
}
```


【拯救者】Ep_ServletContext

每一个web工程都只有一个ServletContext对象。无论在那哪个servlet里面，或缺到的这个类的对象都是同一个。

1.如何得到对象

```
ServletContext context = this.getServletContext();//获取对象
```

2.方法

```
添加属性：setAttribute(String name, Object obj);
得到值：getAttribute(String name)，//这个方法返回Object
删除属性：removeAttribute(String name)
获取服务器真实文件路径：context.getRealPath("路径+文件名");
    servletContext.getRealPath("/img/a.png");
```

3. 生命周期

ServletContext中的属性的生命周期从创建开始，到服务器关闭结束。

共享数据

HelloServlet1.Java

```
public class HellowServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext context = this.getServletContext();

        String username = "zwj";
        context.setAttribute("username",username);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

HelloServlet2.Java

```
public class HellowServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        String name=context.getAttribute("username");
        System.out.println(name);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

获取初始化参数

```
public class GetParameter extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        String url = context.getInitParameter("url");
        resp.getWriter().println(url);
    }
```

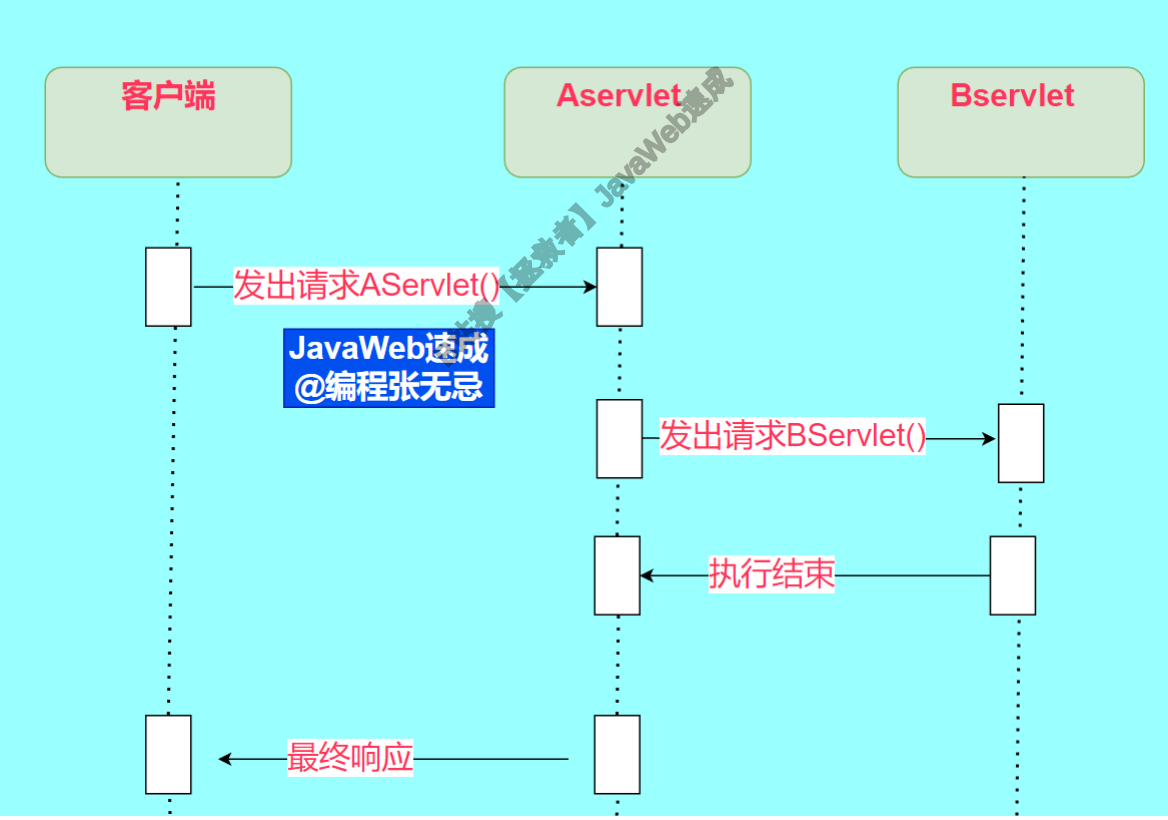
```
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

```
<context-param>
    <param-name>url</param-name>
    <param-value>jdbc:mysql://localhost:3306/digital</param-value>
  </context-param>
```

【拯救者】Ep_请求转发和请求重定向

请求转发

请求转发使用**RequestDispatcher接口中的forward()方**法来实现，该方法可以**把请求转发给另外一个资源**，并让该资源对此请求进行响应。 转发后**浏览器地址栏内容不变。**

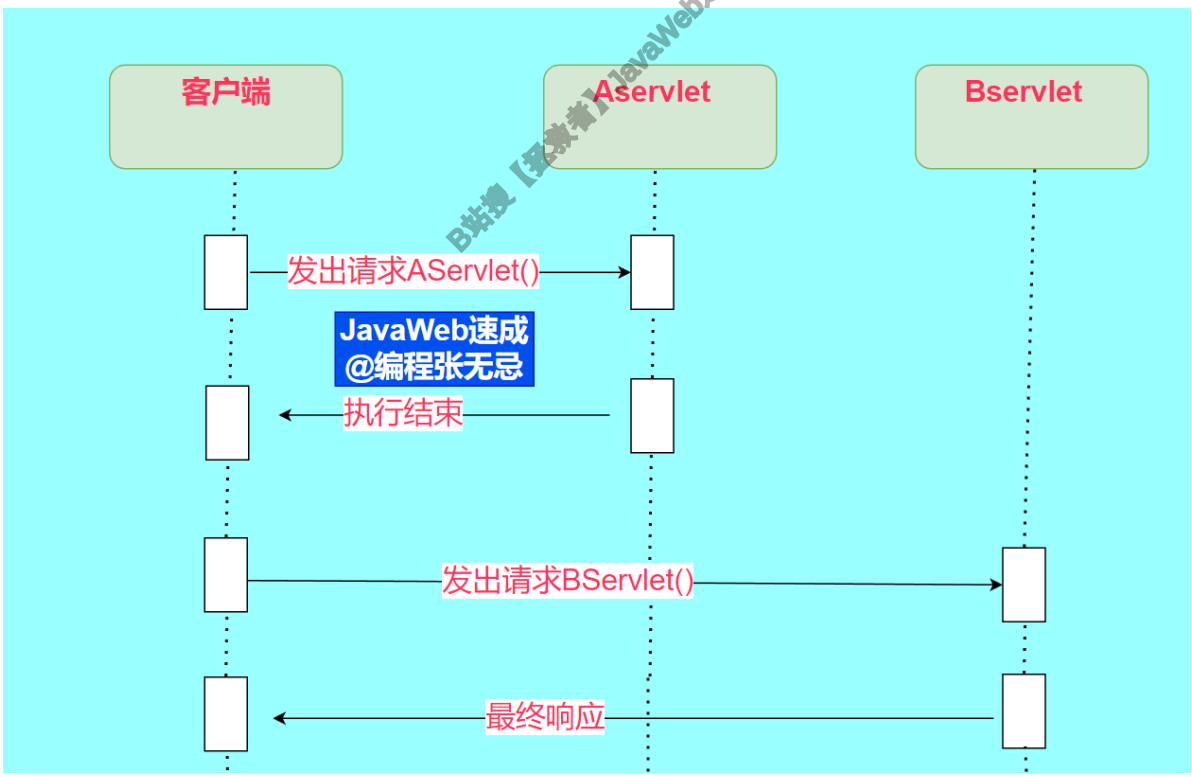



```
@WebServlet("/AServlet")
 public class DispatcherServlet extends HttpServlet {
/**
 * 请求转发，希望在转发的时候，还可以携带原有的数据
 * 原有的数据，就会保留着，不会丢失。
 */

protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

```
        System.out.println("请求AServlet，A请求转发BServlet");
        request.getRequestDispatcher("BServlet").forward(request, response);


    }

}
```

```
@WebServlet("BServlet")
public class Dispatcher2Servlet extends HttpServlet {

    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

                System.out.println("BServlet处理完毕");

    }
```

请求重定向



重定向是A找B帮忙，B做不了，让A去找C帮忙;且可以重定向到其他项目中去。

- **重定向是两次请求，转发是同一次请求。**
- **重定向浏览器地址栏的URL改变；而转发浏览器地址栏的URL不变。**
- **重定向传输的信息会丢失；** `req.setAttribute("user",user);`，这是将数据存储到请求域中，它只能在一次请求中存在，因此在请求转发中可以共享数据，而在重定向中是无法进行数据共享的。`Session对象` **重定向 时，session数据不会丢失**
- **重定向可以跳转到第三方服务器**

如果在登录页面输入的用户名和密码正确就 到欢迎页面（welcome.html），否则重定向到登录页面（login.html)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>WELCOME To Login!!!!</h2>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>登录页面</h2>
<form action="/WEB_TASK/LoginServlet" method="post">
用户名:<input type="text" name="name"><br/>
密　　码:<input type="password" name="password"><br/>
    <input type="submit" name="提交">
</form>
</body>
</html>
```



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

public class LoginServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String name="admin";
        String pw="123456";
        String username=request.getParameter("name");
        String password=request.getParameter("password");
        if( name.equals(username)&&pw.equals(password)) {
            response.sendRedirect("/login/welcome.html");
        }else {
            response.sendRedirect("/login/login.html");
        }
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
```

```
    }
}
```

第四章-JSP技术基础

【拯救者】Ep_JSP语法

JSP页面的基本构成

一个JSP页面

- 在**HTML静态**页面文件中加入和**java相关的动态元素**就构成了一个**JSP页面。**

组成:

```
普通的HTML标记(第二章讲了)
JSP注释
Java脚本元素，包括声明、Java程序片和Java表达式
JSP标记，例如指令标记、动作标记和自定义标记等(不重要,不讲)
```

JSP注释

1. HTML注释**格式：** `<!--HTML-->`
2. JSP注释**格式：** `<%--JSP注释--%>`

jsp脚本元素

java程序片:

放在**<% 内容%>中的**被称为java程序片

JSP成员变量和局部变量:

通常来说：定义在**<%! 内容%>**中的为**成员变量**，定义在**<% 内容%>**中的为**局部变量。**

例如:

```
    <%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>hello.jsp</title>
</head>
<%!
    int n=0;
    int add(int x,int y){
        return x+y;
    }
%>
<body>
<%
    int m=0;
    n++;
    m++;
    int result=add(1,2);
```

```
    out.print("成员2变量n的值为:"+n+"<br>");
    out.print("成员变量m的值为:"+m+"<br>");
    out.print("1+2 = "+result+"<br>");
    out.print("第"+n+"个客户");
%>
</body>
</html>
```

n是成员变量，被所有客户共享，jsp页面 不管执行多少次，上一次对该全局变量执行的**结果都是下一次执行的初始值。**

m是局部变量，为每个客户独享， **调用多少次就会重新初始化多少次，所以局部变量的值始终保持不变。**

java表达式

<%=%>之间用来**放表达式**，中间不能有其他内容

```
<%!
  int visited=1;
%>
<body>
  <p>
    您是第<%=visited%>访问的顾客
    <%
      visited++;
    %>
  </p>

</body>
```


【拯救者】Ep_page和include指令

JSP指令标记

Page指令

page指令称为页面指令，用来**定义JSP页面的全局属性，该配置会作用于整个页面。**

```
  <%@ page attribute="value" %>
```

属性:

属性	描述
contentType	指定当前JSP页面的MIME类型和字符编码
import	导入要使用的Java类
language	定义JSP页面所用的脚本语言，默认是Java
pageEncoding	是值JSP文件自身存储所用的编码

include指令

通过include指令来**包含其他文件。被包含的文件可以是JSP文件、HTML文件**或文本文件。

```
<%@ include file="文件相对 url 地址" %>
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>index.jsp</title>
</head>
<body>
  <%@include file="header.jsp"%>

</body>
</html>
```

其他不重要的不讲~

第五章-会话与文件管理

【拯救者】Ep_Cookie及其应用

Cookie的基本使用

```
创建Cookie对象，并设置数据
    Cookie cookie = new Cookie("key","value");

发送Cookie到客户端：使用response对象
    response.addCookie(cookie);

获取客户端携带的所有Cookie，使用request对象
    Cookie[] cookies = request.getCookies();

使用Cookie对象方法获取数据
    cookie.getName();
    cookie.getValue();

设置Cookie存活时间
    setMaxAge(int seconds)
```

案例:

```
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <!--jsp-->
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
```

```
        <version>2.2</version>
        <scope>provided</scope>
    </dependency>
    <!--jstl-->
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
    </dependency>
```

在Servlet中创建Cookie对象，存入数据，发送给前端

```
@WebServlet("/aServlet")
public class AServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //1. 创建Cookie对象
        Cookie cookie = new Cookie("username","ZWJ");
        //2. 发送Cookie，response
        response.addCookie(cookie);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

启动测试，在浏览器Application中查看Cookie对象中的值

```
http://localhost:8080/xxxxx/aServlet
```

编写一个新Servlet类，名称为BServlet

```
@WebServlet("/bServlet")
public class BServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        //1. 获取Cookie数组
        Cookie[] cookies = request.getCookies();
        //2. 遍历数组
        for (Cookie cookie : cookies) {
            //3. 获取数据
```

```
            String name = cookie.getName();
            if("username".equals(name)){
                String value = cookie.getValue();
                System.out.println(name+":"+value);
                break;
            }
        }

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

启动测试控制台打印

```
http://localhost:8080/xxxx/bServlet
```

7天内免登陆的案例

```
@WebServlet("/aServlet")
public class AServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //发送Cookie
        //1. 创建Cookie对象
        Cookie cookie = new Cookie("username","ZwJ");
        //设置存活时间    ，1周 7天
        cookie.setMaxAge(60*60*24*7);
        //2. 发送Cookie，response
        response.addCookie(cookie);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

把汉字`张无忌`存入到Cookie中，会报错

- **Cookie不能直接存储中文**

解决方案:

对中文进行URL编码，采用URLEncoder.encode()，将编码后的值存入 Cookie中

```
        value = URLEncoder.encode("张无忌", "UTF-8");
        //将编码后的值存入Cookie中
        Cookie cookie = new Cookie("username",value);
```

2.将获取的值在进行**URL解码,采用URLDecoder.decode()，就可以获取**到对应的中文值

```
            value = URLDecoder.decode(value,"UTF-8");
            System.out.println(name+":"+value);
```

【拯救者】Ep_Session及其应用

当浏览器第一次访问服务器一个页面, 由**服务器创建一个Session的ID(JSESSIONID)**，以Cookie的方式**保存在客户端**中，在**Session被创建**之后，就可以**调用Session的相关方法往Session中增加内容**了，而这些**内容只会保存在服务器中**

Session的方法使用

```
获取Session对象,使用的是request对象
    HttpSession session = request.getSession();
存储数据到 session 域中
    void setAttribute(String name, Object o)
根据 key，获取值
    Object getAttribute(String name)
根据 key，删除该键值对
    void removeAttribute(String name)
设置Session失效时间的S数
    void setMaxInactiveInterval(int var1);
```


案例

在一个Servlet中往Session中存入数据，在另一个Servlet中获取Session中的数据

```
@WebServlet("/demo1")
public class SessionDemo1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        //1. 获取Session对象
        HttpSession session = request.getSession();
        //2. 存储数据
        session.setAttribute("username","ZwJ");

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
        this.doGet(request, response);
    }
}
```

```
@WebServlet("/demo2")
public class SessionDemo2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
         //1. 获取Session对象
        HttpSession session = request.getSession();
        //2. 获取数据
        Object username = session.getAttribute("username");
        System.out.println(username);

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

```
1先访问http://localhost:8080/xxxx/demo1,将数据存入Session
2在访问http://localhost:8080/xxxx/demo2,从Session中获取数据
3查看控制台打印
```


【拯救者】Ep_文件上传

服务器端通过程序**接收本地文件内容，并将其保存在服务器端磁盘中。**

文件上传

上传大多数情况是通**过表单的形式提交给服务器**，使用 选择文件 **未选择任何文件** 标签。

需要注意以下两点：

```
1.必须设置name属性，不然浏览器不会发送上传文件的数据。

2.必须将method属性设置为post，ectype属性设置为“multipart/form-data”类型。
```

导入开源组件Commons-FileUpload的两个jar包，

commons-fileupload-1.3.3.jar，commons-io-2.6.jar。

```
<dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.2.1</version>
    </dependency>

    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-io</artifactId>
      <version>1.3.2</version>
    </dependency>
```

思路

1 创建工厂类对象，用于将每一个项目封装成单独的DiskFileItemFactory也就是FileItemFactory的具体实现类

2 创建文件上传的核心类，ServletFileUpload对象，将工厂类对象作为参数进行传递

3 通过核心类的parseRequest方法，传入一个请求对象为参数，获取到FileItem的集合，在真正进行文件的操作的时候，是使用FileItem对象进行操作

4 遍历FileItem集合，判断是普通的表单数据还是上传的文件，
    对于普通的表单数据，我们关注的是表单项的属性值和属性值对应的具体的值，
    对于上传的文件，我们关注的是属性值和文件名，并在获取到信息之后，
        将文件写入到输入，使用FileItem对象的write方法，参数为一个File对象，最好在构建File对象的时候将文件名添加进去

5 最后抛出异常，关闭流


```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>文件上传</title>
</head>
<body>
<!--表单的enctype属性要设置为multipart/form-data-->
<form action="/demo17" method="post" enctype="multipart/form-data">
    <p>上传文件1：<input type="file" name="file1"></p>
    <p>上传文件2：<input type="file" name="file2"></p>
   <p>
        <input type="submit" value="提交"> | <input type="reset" value="重置">
    </p>
</form>
</body>
</html>
```

创建Servlet

```
@WebServlet("/demo17")
public class DemoServlet17 extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //1. 判断表单是否带上传文件
        if(!ServletFileUpload.isMultipartContent(req))
        {
            return;
        }
        //2. 创建上传文件保存在服务器中的目录，建议在WEB-INF路径下，可以一定程度上保证安全
        String uploadPath = this.getServletContext().getRealPath("/WEB-
INF/upload");
        System.out.println("上传文件根目录："+uploadPath);
        File uploadFile = new File(uploadPath);
        if (!uploadFile.exists())
        {
            uploadFile.mkdir();
        }
        //3. 创建磁盘文件库
        DiskFileItemFactory diskFileItemFactory = new DiskFileItemFactory();
        //4. 获取servlet文件上传对象
        ServletFileUpload fileUpload = new
ServletFileUpload(diskFileItemFactory);

        //4.2 处理乱码问题
        fileUpload.setHeaderEncoding("UTF-8");
        //4.3 设置单个文件大小的最大值
        fileUpload.setFileSizeMax(1024*1024*10);
        //4.4 设置总共能上传的文件大小最大值
        fileUpload.setSizeMax(1024*1024*10);
        //5. 处理上传的文件
        try {
            //5.1 获取表单中的每一个控件
            List<FileItem> fileItems = fileUpload.parseRequest(req);
            for (FileItem fileItem : fileItems) {
                if (!fileItem.isFormField()) {
                    {
                        //5.2 获取文件路径
                        String uploadFileName = fileItem.getName();
                        //5.3 对获取的文件字符串路径进行处理
                        if (uploadFileName.trim().equals("") || uploadFileName
== null)
                            continue;
                        //5.3.1  获取文件名
                        String fileName = uploadFileName.substring(0,
uploadFileName.lastIndexOf("."));

                        //5.4 生成唯一的字符串，标识文件名，保证文件不会因为重名和覆盖
                        UUID uuidName = UUID.randomUUID();
                        //5.5 为上传的文件创建一个唯一命名的文件夹
                        String realUploadPath = uploadPath + "//" + uuidName;
```

```
                        File realUploadFile = new File(realUploadPath);
                        if (!realUploadFile.exists())
                            realUploadFile.mkdir();
                        //5.6 将上传的文件保存到上面存储的唯一文件夹中
                        //5.6.1  获取上传文件的流
                        InputStream inputStream = fileItem.getInputStream();
                        //5.6.2 将文件流写出到指定服务器文件    upload//123-3132-
32322//1.png
                        FileOutputStream fos = new
FileOutputStream(realUploadPath + "//" + uploadFileName);
                        byte[] buffer = new byte[1024 * 1024];
                        int len = 0;
                        while ((len = inputStream.read(buffer)) > 0) {
                            fos.write(buffer, 0, len);
                        }
                        fos.close();
                        inputStream.close();
                        fileItem.delete();//上传成功，清除临时文件
                    }
                }
            }
        }catch (FileUploadException e) {
            e.printStackTrace();
        }
        String msg="文件上传成功";
        System.out.println(msg);
    }
}
```


【拯救者】Ep_文件下载

文件下载

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>文件下载</title>
</head>
<body>
<a href="/demo18?filename=1.png">文件下载</a>
</body>
</html>
```

创建Servlet

```
 */
@WebServlet("/demo18")
public class DemoServlet18 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
        doGet(request,response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        //获取文件名
        String filename = request.getParameter("filename");//1.png
        //文件所在的文件夹
        String folder="WEB-INF/upload/5708fb9c-acc0-43c0-8de6-b1918f353731/";
        //通知浏览器以下载的方式打开
        response.addHeader("Content-Type","application/octet-stream");
        response.addHeader("Content-
Disposition","attachment;filename="+filename);
        //通过文件输入流读取文件
        InputStream in=getServletContext().getResourceAsStream(folder+filename);
        OutputStream out=response.getOutputStream();
        byte[] bytes=new byte[100024];
        int len=0;
        while ((len=in.read(bytes))!=-1){
            out.write(bytes,0,len);
        }
    }
}
```


第六章-JDBC访问数据库

【拯救者】Ep_JDBCAPI

jdbc: **java database connection** （Java数据库连接）

定义了一套Java操作数据库的规范，称之为JDBC

定义了一套JDBC接口，这套接口由数据库厂商去实现，这样，**开发人员只需要学习JDBC接口， 并通过JDBC加载具体的驱动，就可以操作数据库。**

（1）java.sql

类：DriverManager

接口：Connection Statement ResultSet PreparedStatement

```
1. Driver接口：  表示java驱动程序接口。所有的具体的数据库要来实现此接口。
          connect(url, properties):  连接数据库的方法。
          url: 连接数据库的URL
             URL语法：  jdbc协议:数据库子协议://主机:端口/数据库
          user:  数据库的用户名
          password:  数据库用户密码

2. DriverManager类：  驱动管理器类，用于管理所有注册的驱动程序
        registerDriver(driver)  : 注册驱动类对象
        Connection getConnection(url,user,password):  获取连接对象
```

```
3. Connection接口：  表示java程序和数据库的连接对象。

        Statement createStatement() ：  创建Statement对象
        PreparedStatement prepareStatement(String sql)：创建PreparedStatement对象

4. Statement接口：  用于执行静态的sql语句
        int executeUpdate(String sql)  ：  执行静态的更新sql语句
        ResultSet executeQuery(String sql)  ：执行的  查询sql语句

5. PreparedStatement接口：用于执行预编译sql语句
         int executeUpdate() ：  执行预编译的更新sql语句
         ResultSet executeQuery()  ：  执行预编译的查询sql语句

6. ResultSet接口：用于封装查询出来的数据
          boolean next() ：  将光标移动到下一行
          getXX() ：获取列的值
```

使用

```
0 引入依赖

    <!--mysql依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.22</version>
    </dependency>

1 在程序中加载数据库驱动（称为注册驱动）
    DriverManager.registerDriver(Driver driver);
2 建立连接（Connection）
    Connection conn = DriverManager.getConnection(url, user, pass);
3 创建用于向数据库发送SQL的Statement对象，并发送SQL
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery(sql);
4 从代表结果集的ResultSet中取出数据，打印到命令行窗口
5 断开与数据库的连接，并释放相关资源。
        rs.close();
        st.close();
        con.close();
```



```
public class JdbcDemo1 {

    public static void main(String[] args) throws SQLException {

        // 1.注册驱动
         Class.forName("com.mysql.jdbc.Driver");
```

```
        // 2.获取连接对象
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/数据库名字", "root", "root");

        // 3.通过连接对象获取操作sql语句Statement
        Statement st = con.createStatement();

        // 4.操作sql语句
        String sql = "select * from user";
        ResultSet rs = st.executeQuery(sql);
          //可以在演示插入语句

        // 5.遍历结果集
        while(rs.next()){
            int id=rs.getInt("id");
            String username=rs.getString("username");
            String password=rs.getString("password");
            String email=rs.getString("email");

            System.out.println(id+"  "+username+"  "+password+"  "+email);
        }

        //6.释放资源
        rs.close();
        st.close();
        con.close();
    }
}
```


(2) javax.sql

接口 DataSource

(下节课讲)

【拯救者】Ep_使用数据源

javax.sql

接口 DataSource

DataSource方式 ：在内部创建Connection对象的连接池，“池”资源 是可以重复利用的。
当一个Connection对象调用Connection.close()方法之后，它不是真正的关闭,这个对象会被DataSource回收进入连接池。
若此时有别的用户需要建立连接，不是创建新的连接，而是看连接池中是否有空闲的连接，直接使用空闲的连接。

它有两个被覆盖的方法。

获取连接（）
getConnection(字符串用户名，字符串密码)
这两种方法都返回一个Connection对象。

例如 MySQL JDBC 驱动程序包括实现 **MysqlDataSource**

```
public class DataSource {

   static MysqlDataSource mysqlDataSource= new MysqlDataSource();
   static DataSource ds=null;

     public static DataSource getInstance(){
         if(ds==null){
             mysqlDataSource.setPassword("123456");
             mysqlDataSource.setUser("root");
             mysqlDataSource.setURL("jdbc:mysql://localhost:3306/edu?
serverTimezone=UTC&characterEncodeing=utf8");
             return new DataSource();
         }else {
             return ds;
         }

     }

    public MysqlDataSource getMysqlDataSource() {

        return mysqlDataSource;
    }
}
```



```
@WebServlet("/demo20")
public class DemoServlet20 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        Statement statement =null;
        Connection con =null;
        ResultSet resultSet =null;
        try {
            MysqlDataSource mysqlDataSource =
DataSource.getInstance().getMysqlDataSource();
              con = mysqlDataSource.getConnection();
            String sql="select * from student";
             statement = con.createStatement();
              resultSet = statement.executeQuery(sql);
            while(resultSet.next()){
                String name = resultSet.getString("NAME");
                String userphone = resultSet.getString("userphone");
                int age = resultSet.getInt("age");
```

```
                String password = resultSet.getString("password");
                System.out.println(name+" "+userphone+" "+age+" "+password);

            }

        } catch ( Exception e) {
            throw new RuntimeException(e);
        } finally {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }

            try {
                statement.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }

            try {
                con.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```


第七章-表达式语言

【拯救者】Ep_EL运算符

引入依赖

```
    <!--jstl-->
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
    </dependency>
```

EL表达式全称：**Expression Language**

作用

主要是代替 **jsp页面** 中的**表达式脚本**在jsp页面中进行数据的输出

因为**EL表达式在输出数据时，要比jsp的表达式脚本要简洁很多。**

格式

EL表达式的格式： ${表达式}

EL表达式支持如下运算符：

语法：${运算表达式}，EL表达式支持如下运算符：

1、关系运算符

关系运算符	说　　明	范　　例	结　果
= = 或 eq	等于	${ 5 = = 5 } 或 ${ 5 eq 5 }	true
!= 或 ne	不等于	${ 5 != 5 } 或 ${ 5 ne 5 }	false
< 或 lt	小于	${ 3 < 5 }或 ${ 3 lt 5 }	true
> 或 gt	大于	${ 3 > 5 }或 ${ 3 gt 5 }	false
<= 或 le	小于等于	${ 3 <= 5 }或 ${ 3 le 5 }	true
>= 或 ge	大于等于	${ 3 >= 5 }或 ${ 3 ge 5 }	false

2、逻辑运算符：

逻辑运算符	说　明	范　　例	结　果
&& 或 and	交集	${ A && B }或 ${ A and B }	true / false
|| 或 or	并集	${ A || B } 或 ${ A or B }	true / false
! 或 not	非	${ !A } 或 ${ not A }	true / false

3、empty运算符：检查对象是否为null(空)

4、二元表达式：${user!=null?user.name :""}

5、[ ] 和 . 号运算符

1.点运算符(.)

访问JSP页面中某些对象的属性，如 对象、List对象、Array对象等

```
<%
    Person p = new Person();
    p.setName("ZwJ");
    p.setAddress("杭州");
    session.setAttribute("person", p);//存入sessionScope
%>
${person.name}
```

2.方括号运算符([])

```
${person.name}  等价于${person["name"]}。
```

3.算数运算符

```
+加${10+2}  12
-减${10-2}   8
*乘${10*2}   20
/除${10/4}   2.5  “/” 除法运算时，商为小数。
%取模${10%4}  2
```

4.比较运算符

```
==（或eq）等于${10==2}或${10 eq 2}false

!=（或ne）不等于${10!=2}或${10 ne 2}true

<（或lt）小于${10<2}或${10 lt 2}false

>（或gt）大于${10>2}或${10 gt 2}true

<=（或le）小于等于${10<=2}或${10 le 2}false

>= （或ge）大于等于${10 >=2}或${10 ge 2}true
```

5.逻辑运算符

```
&& 逻辑与 ${true&&false}或者{ture and false}false

|| 逻辑或 ${fasle||true}或者{false or true}true

! 逻辑非  ${!true}或${not ture}false
```


6.empty运算符

empty运算符可以判定变量(或表达式)是否为null

```
${empty name}，如果不存在name变量或者值为null，就返回true。
```

7.条件运算符

```
${A?B:C}
${(1==2)?3:4}的结果就为4。
```

【拯救者】Ep_EL访问数据

获取并显示数据: 从**四个域中**通过key找到简单数据并显示出来

四个域的寻找顺序是 **page，request，session，application**

```
<%
    request.setAttribute("key1","张无忌的面试宝典");
 pageContext.setAttribute("u",user1);

%>

EL表达式输出key的值是：${key1}
```

对象的某个属性值并显示出来

```
    Person p = new Person();
    p.setName("ZwJ");
    p.setAddress("杭州");
    session.setAttribute("person", p);//存入sessionScope
%>
${person.name}
```

从List集合对象中获取某个值并显示

```
<%
    List<Person> list = new ArrayList<Person>();
    list.add(new Person("张无忌"));
    list.add(new Person("刘能"));
    list.add(new Person("金秀贤"));
    application.setAttribute("list", list);
%>
${list[1].name }
```


从Map中获取某个值并显示

```
<%
    Map map = new HashMap();
    map.put("面试", "看[张无忌的面试宝典]");
    map.put("复试",  "看[考研复试面试宝典]");
    map.put("期末", "看我的速成");
    request.setAttribute("map", map);
%>
${map['面试']  }
```

第八章-JSTL与自定义标签

【拯救者】Ep_JSTL

JSTL全称为**JSP Standard Tag Library，即JSP标准标签库。**

EL 表达式主要是为了替换 jsp 中的表达式， 而**标签库则是为了替换代码**

JSTL核心标签库

标签名称	功能分类	属性	作用
<标签名:if>	流程控制	核心标签库	用于条件判断
<标签名:choose>	流程控制	核心标签库	用于多条件判断
<标签名:when>	流程控制	核心标签库	用于多条件判断
<标签名:otherwise>	流程控制	核心标签库	用于多条件判断
<标签名:forEach>	迭代遍历	核心标签库	用于循环遍历

使用

引入依赖

```
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-spec</artifactId>
    <version>1.2.1</version>
</dependency>
<dependency>
    <groupId>org.apache.taglibs</groupId>
    <artifactId>taglibs-standard-impl</artifactId>
<version>1.2.1</version>
 </dependency>
```

在 jsp 标签库中使用 taglib 指令引入标签库:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
这里的prefix="c"，c就是标签名称
```

案例

流程控制:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <title>流程控制</title>
</head>
<body>
<%--向域对象中添加成绩数据--%>
${pageContext.setAttribute("score","A")}
<%--对成绩进行判断--%>
<c:if test="${score == 'A'}">
    优秀
</c:if>

<c:choose>
```

```
    <c:when test="${score == 'A'}">优秀</c:when>
    <c:when test="${score == 'D'}">良好</c:when>
    <c:when test="${score == 'C'}">及格</c:when>
    <c:when test="${score == 'D'}">较差</c:when>
    <c:otherwise>成绩类型不符</c:otherwise>
</c:choose>
</body>
</html>
```

循环遍历

```
<%
//    向域对象中添加集合 默认作用域是pageContext，仅当前页面
    ArrayList<String> list = new ArrayList<>();
    list.add("aa");
    list.add("bb");
    list.add("cc");
    list.add("dd");
    pageContext.setAttribute("list",list);
%>
<%--遍历集合--%>
<c:forEach items="${list}" var="str">
    ${str} <br>
</c:forEach>
```


第九章-Java Web高级应用

【拯救者】Ep_Web监听器

javaweb有三大组件,**分别是servlet,listener和filter.**

其中listener就是监听器.

基本概念

是Servlet规范中定义的一种特殊类，它用于监听web应用程序中的**ServletContext, HttpSession和ServletRequest等域对象的创建与销毁事件**，以及监听这些域对象中的属性发生修改的事件。

1.ServletContextListener

监听ServletContext域的创建与销毁

```
创建 ：  服务器启动针对每一个Web应用创建ServletContext,触发contextInitialized ( )方法
```

销毁：服务器关闭前先关闭代表每一个web应用的ServletContext,触发contextDestroyed( )方法。

```
public class MyServletContextListener implements ServletContextListener{

    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
        System.out.println("ServletContext对象销毁了！");
    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
        System.out.println("ServletContext对象创建了！");
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">

    <listener>
        <listener-class>com.zwj.MyServletContextListener</listener-class>
    </listener>
</web-app>
```


注解

```
类上加
@WebListener
```

2.ServletRequestListener

监听ServletRequest域创建与销毁

创建：用户每一次访问都会创建request对象,触发requestInitialized ()方法。

销毁：当前访问结束，request对象就会销毁,触发requestDestroyed 方法。

```
public class MyServletRequestListener implements ServletRequestListener {

    @Override
    public void requestDestroyed(ServletRequestEvent arg0) {
        System.out.println("request对象销毁了");
    }

    @Override
    public void requestInitialized(ServletRequestEvent arg0) {
        System.out.println("request对象创建了");
    }
}
```

web.xml 或者 注解 与1类似

3.ServletContextAttributeListener

监听ServletContext域的属性变化的

当ServletContext对象添加属性时，触发attributeAdded( )方法。

当ServletContext对象修改属性时，触发attributeReplaced( )方法。

当ServletContext对象删除属性时，触发attributeRemoved( )方法。

```
public class MyServletContextAttributeListener implements
ServletContextAttributeListener {

    @Override
    public void attributeAdded(ServletContextAttributeEvent arg0) {
        System.out.println("属性被添加,name:" + arg0.getName() + "value:" +
arg0.getValue());
    }

    @Override
    public void attributeRemoved(ServletContextAttributeEvent arg0) {
        System.out.println("属性被删除,name:" + arg0.getName() + "value:" +
arg0.getValue());
    }

    @Override
    public void attributeReplaced(ServletContextAttributeEvent arg0) {
        //获得是修改前的数据
        System.out.println("属性被修改,name:" + arg0.getName() + "value:" +
arg0.getValue());
    }
}
```



```
@WebServlet(
    name = "ListenerServlet", urlPatterns = {"/listenerServlet"}
public class ListenerServlet extends HttpServlet{

    @Override
```

```
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
         ServletContext context = this.getServletContext();
        //存储一个数据
        context.setAttribute("name", "Zwj");
        //修改一个数据
        context.setAttribute("name", "Zwj2");
        //删除一个数据
        context.removeAttribute("name");

    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doGet(req, resp);
    }
}
```

访问http://localhost:8081/listenerServlet.html

在线人数统计

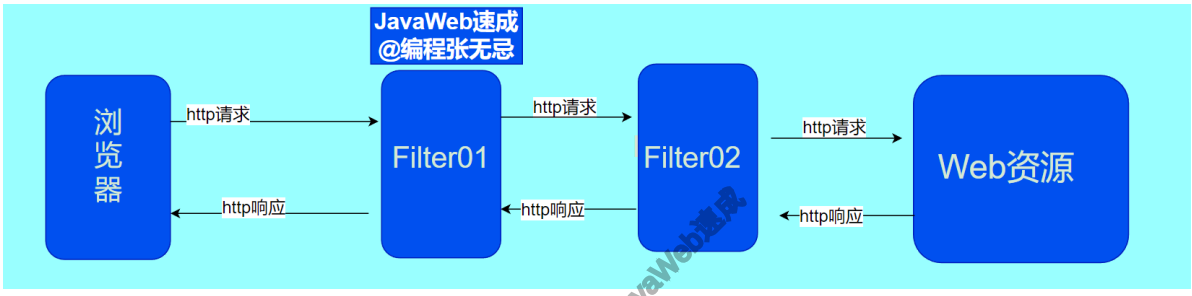
创建session对象的监听器，当session创建时，人数+1，销毁时，人数-1；

```
@WebListener
public class OnlineListener implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent hse) {
        // 从ServletContext对象中获取当前的在线人数
        ServletContext context = hse.getSession().getServletContext();
        Long online = (Long)context.getAttribute("online");

        // 判断online是否为null：说明是第一个用户
        if ( online == null ) {
            //online = 45751245L; // 初始化在线人数
            online = 0L; // 初始化在线人数
        }

        // 人数+1
        online++;
        context.setAttribute("online", online);
    }

    public void sessionDestroyed(HttpSessionEvent hse) {
        ServletContext context = hse.getSession().getServletContext();
        Long online = (Long)context.getAttribute("online");
        context.setAttribute("online", --online);
    }
}
```

- 显示在线人数的online.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Insert title here</title>
    </head>
    <body>
        当前的在线人数是：${online }
    </body>
</html>
```

【拯救者】Ep_Web过滤器



使用过滤器 可以对请求进行拦截，然后做相应的处理，实现许多特殊功能。如登录控制，权限管理，过滤敏感词汇等.

```
    <dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>javax.servlet-api</artifactId>
          <version>3.0.1</version>
       </dependency>
```

Filter有3个阶段，分别是初始化，拦截和过滤，销毁。

初始化阶段：当服务器启动时，我们的服务器(Tomcat)就会读取配置文件，扫描注解，然后来创建我们的Filter。

拦截和过滤阶段：只要请求资源的路径和拦截的路径相同，那么过滤器就会对请求进行过滤，这个阶段在服务器运行过程中会一直循环。

销毁阶段：当服务器(Tomcat)关闭时，服务器创建的Filter也会随之销毁。

`doFilter` 方法中处理完业务逻辑之后，必须添加 `filterChain.doFilter(servletRequest,servletResponse);`**否则请求/响应无法向后传递，一直停留在过滤器中。**

使用

我们创建Filter，只需要继承Filter接口就行。

```
public class MyFilter implements Filter {

     @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        //这个方法就是初始化方法，在Filter创建时调用
        System.out.println("调用了init()方法");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        //这个方法就是过滤和拦截的方法，当请求和拦截匹配时调用
        System.out.println("调用了doFilter()方法");
    }

    @Override
    public void destroy() {
        //这个方法就是销毁方法，在Filter销毁前调用
        System.out.println("调用了destroy()方法");
    }

}
```


这个方法就是我们写过滤代码的地方

xml方式

```
    <filter>
        <filter-name>myFilter</filter-name>
        <filter-class>com.zwj.filter.MyFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>myFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

注解

```
@WebFilter("/*")
```

```
String description() default "";

String displayName() default "";

WebInitParam[] initParams() default {};

String filterName() default "";

String smallIcon() default "";

String largeIcon() default "";

String[] servletNames() default {};

String[] value() default {};

String[] urlPatterns() default {};
```

```
filterName：该filter的名字
initParams：初始化参数
displayName：filter显示名称
servletNames：指定对哪些servlet进行过滤
asyncSupported：是否支持异步模式
urlPatterns：指定拦截路径
value：指定拦截路径
    urlPatterns和value只能配置一个，不能两个都配置，两个都配置就会报错。
```

```
@WebFilter(value={"/*"},filterName = "myFilter")
```


多个过滤器的先后顺序

多个过滤器之间会依次执行，这个依次就是按照过滤器名字的**字典顺序来执行的**，也就是说，在案例中，**Filter1比Filter2的字典顺序靠前，所以就会先执行Filter1，再执行Filter2。**在执行完放行前的语句之后，在**执行放行后的语句的时候，是和之前的执行顺序是相反的**，也就是说，多个过滤器链遵循先进先出，后进后出的原则

```
我们可以看见Filter01先进行过滤，然后交给Filter02，然后访问资源，然后Filter02对响应进行过滤，然后Filter01对响应进行过滤。
```

实现敏感词汇过滤

我们写一个评论页面，如果评论中含有我们定义的敏感词汇，那么我们就进行过滤，使用**来进行代替。

> # 你的昵称:**好

创建一个`input.jsp`

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>elDemo</title>
</head>
<body>
<form action="/filter3"  method="post">
    <input type="text"  name="username" />
    <input type="submit" value="提交"  />
</form>
</body>
</body>
</html>
```

show.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java"
isELIgnored="false"  %>
<html>
<head>
    <title>elDemo</title>
</head>
<body>
<h2>你的昵称:${username}</h2>
</body>
</body>
</html>
```

再创建一个`TestServlet`

```
@WebFilter("/filter3")
public class MyFilter3  implements Filter {
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        servletRequest.setCharacterEncoding("UTF-8");
        String username = servletRequest.getParameter("username");

        username=username.replace("尼玛","**");
        servletRequest.setAttribute("username",username);
```

```
 servletRequest.getRequestDispatcher("/show.jsp").forward(servletRequest,servlet
Response);
        filterChain.doFilter(servletRequest,servletResponse);
    }

    public void destroy() {

    }
}
```

最终章-完整套题讲解篇

【拯救者】Ep_选择题

当访问一个Servlet时，以下Servlet中的哪个方法先被执行？（ D ）

(A) destroy() (B) doGet() (C) service() (D) init()

编写一个Filter，需要（B）

A. 继承Filter 类

B. 实现Filter 接口

C. 继承HttpFilter 类

D. 实现HttpFilter接口

重定向到另一个页面，以下（C）语句是正确的

A. request . sendRedirect(“https ://www . csdn.net”);

B. request . sendRedirect();

C. response . sendRedirect(“https: // www . csdn.net”);

D. response .sendRedirect();

按作用域从大到小排列正确的是 D

A. application page requestresponse

B. session pageContext requestapplication

C. public application session request

D. application session request pageContext

以下哪个是Web服务器（ C ）。

A、JCreator
B、JBuilder
C、Tomcat
D、Eclips

一个Servlet可以被映射的虚拟路径个数是（ D ）。

A、0
B、1
C、2
D、多

在一个JSP文件中，有表达式<%=2+3 %>，它将输出（ B ）。

A、2+3
B、5
C、23
D、输出报错

创建Servlet后可以在以下那个文件中对Servlet进行配置（ A ）。

A、web.xml
B、application.xml
C、config.xml
D、web-config.xml

在配置Servlet是，可以使用一下哪个注解来映射路径（ D ）。

A、@Controller
B、@WebService
C、@WebFilter
D、@WebServlet

在web.xml中使用_A_ 标签配置过滤器

A. 和

B. 和

C. 和

D. 和

下列不是JSP隐式对象的是（ C ）。

A、Request
B、out
C、Context
D、Session

在login.JSP中存在以下代码

```
<form action="login"  method="post">
    <input type="text"  name="yourname"  id="myName"  value= "username">
<form>
```

当表单被提交时，下列选项可以获取到input框的值的是（ B ）。

A、request.getParameter("username")
B、request.getParameter("yourname")
C、request.getParameter("name")
D、request.getParameter("myName")

在JSP页面中，能够完成输出操作的内置对象是 A

A. out

B. response

C. request

D. config

某JSP中有如下代码，显示结果为（B）

```
<%
    Int a = 5;
    request.setAttribute(“a”,”123”);
    session.setAttribute(“a”,”456”);
%>
<c:out value="${a}"/>
```

A. 5

B. 123

C. 456

D. null

【拯救者】Ep_填空题

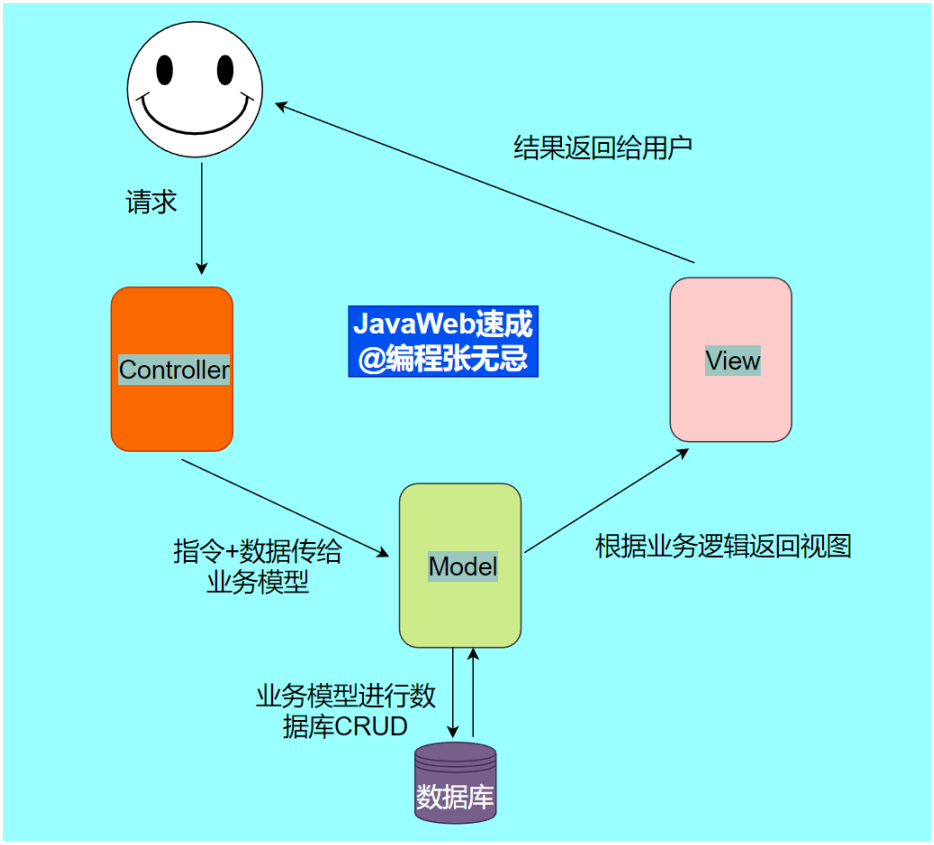
Servlet容器为每一个HttpSession对象分配一个唯一标识符，叫做**sessionID** __。

表单标记中的 **action 属性**用于指定处理表单数据程序url的地址。
form的method属性如果不指定,默认为 **GET 请求。**
JSP本质上是一个 **servlet 。**
getAttribute (“”) 返回一个 **Object** 类型。
page指令中的import 属性可以在页面中出现 **多** 次。
里面的内容必须以 **/** 开头。

jsp 主要内置对象有：__application__、exception_____、_pageContext_____、__request_____、__session__、_____response_____out、config、page。

Servlet 的生命周期分三个时期：__装载 Servlet_____、__创建一个 Servlet 实例_____、__销毁_____。

javaweb在mvc设计模式下，**Javabean***是模型*，***__Jsp****是视图*，**servlet**_是控制器。



【拯救者】Ep_简答题

解释一下什么是Servlet?

Servlet 就 JavaWeb 三大组件之一。三大组件分别是：Servlet 程序、Filter 过滤器、Listener 监听器。
Servlet 是运行在服务器上的一个 java 程序，它可以接收客户端发送过来的请求，并响应数据给客户端。

讲述请求重定向和请求转发的区别

一个web资源收到客户端请求后,通知服务器去调用另外一个web资源进行处理,称之为请求转发.

整个请求过程中，请求和返回的数据是共享的；

整个请求的过程中 URL 地址是不变的

一个web资源收到客户端请求后,通知浏览器去访问另外一个web资源,称之为请求重定向.

请求重定向客户端发送两次完全不同的请求，所以两次请求中的数据是不同的。

URL 地址发生了改变。

jsp和servlet的声明周期和工作原理?

jsp运行原理

发送请求到服务器->服务器识别出这是将jsp页面的请求->交给jsp引擎->磁盘载入jsp->转成servlet->处理完成后传给servlet引擎->产生html格式的输出返回相应

jsp声明周期

编译(当浏览器请求JSP页面时，JSP引擎会首先去检查是否需要编译这个文件。如果这个文件没有被编译过，或者在上次编译后被更改过，则编译这个JSP文件。)
初始化(容器载入JSP文件后，它会在为请求提供任何服务前调用jspInit()方法)
执行(调用_jspService())
销毁

servlet运行原理

发出请求->产生req和res对象->找到servlet创建线程->调用service(),根据请求类型执行doGet或者其他->执行完成后返回->线程销毁

servlet声明周期

服务器加载servlet->创建servlet->调用init->调用service->service处理后返回客户端->等待下一个请求或者服务器卸载->卸载调用destroy方法

Session与Cookie区别和联系

Cookie 一般用来保存用户信息
Session 的主要作用就是服务端记录用户的状态。
1存在的位置：
cookie 存在于客户端，临时文件夹中； session存在于服务器的内存中，一个session域对象为一个用户浏览器服务

2安全性
cookie是以明文的方式存放在客户端的，安全性低
session存放于服务器的内存中，所以安全性好

3生命周期
cookie的生命周期是 累计 的，从创建时，就开始计时，直到cookie生命周期结束；
session的生命周期是 间隔 的，从创建时，开始计时如在20分钟内，没有访问session，那么session生命周期被销毁。
但是，如果在20分钟内（如在第19分钟时）访问过session，那么，将重新计算session的生命周期。
关机会造成session生命周期的结束，但是对cookie没有影响

4保存的数据类型
Cookie只能保存ASCII，Session可以保存任意类型。

什么情况下调用doGet()和doPost(),这两种方法与url流有什么关系?

Form表单的method的方法如果是post的话,那么表单提交的时候就会调用doPost().

其他的访问方式如链接访问、表单里method值为get、地址栏直接提交的都默认是调用doGet().

这两种方法有本质的区别,get只有一个流,参数附加在url后,大小个数有严格的限制且只能是字符串.post参数是通过另外的流传递的,不通过url,所以可以很大,也可指传递二进制数据.

Jsp有哪些内置对象?

page、pageContext、request、response、session、application、out、config、exception

page指的是JSP被翻译成Servlet的对象的引用

pageContext对象可以用来获得其他8个内置对象，还可以作为Jsp的域范围对象使用。pageContext中存的值的作用范围就在当前页面

request代表的是请求对象，可以用于获取客户机的信息，也可以作为作用域对象使用，使用request保存的数据在一次请求范围内有效

session代表的是一次会话，可以用于保存用户的私有信息，也可以作为域对象使用，使用session保存的数据在一次会话范围内有效、

application代表的是整个应用范围，使用这个对象保存的数据在整个web应用中都有效

response是响应对象，代表的是从服务器向浏览器响应数据

out是JSPWriter用于向页面输出内容的对象

config值得是ServletConfig用于JSP翻译成Servlet后获取servlet的配置的对象

exception只要在页面中设置isErrorPage="true"，即可使用,是Throwable的引用,用来获得页面的错误信息

【拯救者】Ep_程序题(上)

1.网站登录页,如果账号密码正确,跳转到欢迎界面, 否则重定向到登陆失败页面

代码实现

login.html

```
<html>
<head>
    <title>登录页面</title>
</head>
<body>
<div align="center">
    <h2>欢迎登录</h2>
    <form name="login">
        用户名：<input type="text" name="username" placeholder="请输入用户名"><br/>
        密 码：<input type="password" name="password" placeholder="请输入密码">
<br/>
        <button type="submit" value="登录"/>
    </form>
</div>
```

```
</body>
</html>
```

welcome.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>首页</title>
</head>
<body>
欢迎你<%=request.getAttribute("username")%>
</body>
</html>
```

loginError.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
    <head>
        <title>用户登录错误</title>
    </head>
    <body>
        <h3>用户登录错误界面</h3>
    </body>
</html>
```


ServletLogin

```
@WebServlet(name = "ServletLogin",urlPatterns = "/login")
public class ServletLogin extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //用户列表
        String user="zhangwuji";
        String pass="666";
        //获取客户端传来的参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        if( user.equals(username)&&pass.equals(password)){
             request.setAttribute("username",username);

request.getRequestDispatcher("/welcome.jsp").forward(request,response);


        }else{
                    response.sendRedirect("/loginError.jsp");
```

```
        }
    }
}
```

2.拓展,从数据库中读取账号 密码

代码实现

```
CREATE TABLE USER(
        id INT PRIMARY KEY AUTO_INCREMENT,
        username VARCHAR(32) UNIQUE NOT NULL,
        password VARCHAR(32) NOT NULL
);
```

```
@WebServlet(urlPatterns = "/Login")
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //获取表单页传过来的参数信息
        String username1 = request.getParameter("username");
        String password1 = request.getParameter("password");

          // 1.注册驱动
         Class.forName("com.mysql.jdbc.Driver");
        // 2.获取连接对象
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/数据库名字", "root", "root");

        // 3.通过连接对象获取操作sql语句Statement
        Statement st = con.createStatement();
        // 4.操作sql语句
        String sql = "select * from user";
        ResultSet rs = st.executeQuery(sql);
        while (true){
            try {
                if (!rs.next()) break;
                String username = rs.getString("username");
                String password = rs.getString("password");
                if(username.equals(username1)&&password.equals(password1)){
                     request.setAttribute("username",username);

 request.getRequestDispatcher("/welcome.jsp").forward(request,response);
                    return;
                }
            } catch (SQLException e) {
                e.printStackTrace();
```

```
            }
        }

        //6.释放资源
        rs.close();
        st.close();
        con.close();

        response.sendRedirect("/loginError.jsp");
    }
```

【拯救者】Ep_程序题(下)

3.编写一个 统计当前在线人数案例

代码实现

创建HttpSessionListener进行初始化

```
@WebListener
public class OnlineCountListener  implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent httpSessionEvent) {
        ServletContext servletContext =
httpSessionEvent.getSession().getServletContext();
        Long count =(Long) servletContext.getAttribute("count");
        if(count==null){
            count=0L;
        }
        count++;

        servletContext.setAttribute("count",count);
    }

    public void sessionDestroyed(HttpSessionEvent httpSessionEvent) {
        ServletContext servletContext =
httpSessionEvent.getSession().getServletContext();
        Long count = (Long) servletContext.getAttribute("count");
        count--;
        servletContext.setAttribute("count",count);
    }
}
```

jsp页面 index.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>在线人数统计案例</title>
</head>
<body>
<h1>在线人数：${count}</h1>
</body>
</html>
```