Servlet

Servlet是什么?

是一个实现了Servlet接口，运行在服务器端的Java程序。功能：处理请求、返回响应。

Servlet实现类继承关系：

HttpServlet（抽象类，实现了service()，并且提供了受保护的service()方法）-> GenericServlet（抽象类，实现了Servlet接口除service()之外的其它抽象方法） -> Servlet (接口，5个抽象方法)

- void init(ServletConfig config)
- ServletConfig getServletConfig();
- void service(ServletRequest req, ServletResponse res)
- String getServletInfo();
- void destroy();

Servlet流程

1、设置请求编码格式

2、设置响应格式

3、获取请求信息

4、处理请求信息

5、响应处理结果

web.xml文件Servlet虚拟路径的配置

1) 精确匹配 /s1

2) /目录/* 如: /login/，匹配虚拟路径/login/index

3）*.do　注意前面没有/

Request与Response对象

Request对象

作用：封装了当前请求的所有的请求信息

Request对象是由服务器创建的，作为实参传递给处理请求的servlet的service()

作用：

1) 获取请求头数据

```
//获取请求方式
System.out.println(request.getMethod());
System.out.println(request.getRequestURI());
System.out.println(request.getRequestURL());
//获取协议
System.out.println( request.getScheme());
```

2) 获取请求行数据

```
 System.out.println(request.getHeader("Host"));
 Enumeration<String> en=request.getHeaderNames();
        while (en.hasMoreElements()) {
            String str = (String) en.nextElement();
        System.out.print(str+":");
            System.out.println(request.getHeader(str));
        }
```

3) 获取用户数据

```
request.getParameter("u_name");//返回指定的用户数据
 request.getParameterNames();//返回所有用户请求数据的枚举集合
 request.getParameterValues("hobby");返回同键不同值的请求数据（多选）返回的数据
     //如果要获取的数据不存在，不会报错，返回null
```

Response对象

作用： 用来响应数据到客户端的对象

使用:

1）设置响应头

```
response.setHeader("servletdemo", "hello,servlet");
```

2）设置响应状态

```
    response.sendError(500);//自定义响应状态码
```

3）设置响应实体

```
    response.getWriter().write("hello world");//响应具体的内容返回客户端
```

4）设置响应编码

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

service() 请求处理代码流程:

1）设置响应编码格式

2）获取请求数据

3）处理请求数据：数据库操作(MVC思想)

4）响应处理结果

跳转的两种方式

1）请求转发

一次请求、一次响应；参数（Attribute）可在servlet之间共享；地址栏没有变化

```
Servlet1.java
    request.setAttribute("name", "lihuanzhen");
//      请求转发，参数共享的，一次请求，地址不变
request.getRequestDispatcher("s3").forward(request, response);
//或者
//      RequestDispatcher dispatcher=request.getRequestDispatcher("s3");
//      dispatcher.forward(request, response);


Servlet2.java
String u_name=(String)request.getAttribute("name");
```

2）请求重定向：

两次请求，两次响应。地址变化，参数（Attribute）不共享，地址栏有变化，速度比较慢

```
response.sendRedirect("s3");
```

Servlet生命周期

1、加载并实例化Servlet（服务器完成，调用了不含参数的构造方法

2、初始化阶段（服务器完成，调用init(ServletConfig config)，只执行一次）

3、处理请求、返回响应（服务器调用service()，可以执行多次，一次请求，创建一个线程，执行一次service()）

4、销毁阶段（服务器调用destroy()（，只执行一次）

```
<servlet>
        <servlet-name>life</servlet-name>
        <servlet-class>com.edu.lhz.LifeServlet</servlet-class>
        <load-on-startup>0</load-on-startup>
    </servlet>
```

0 在服务器启动时进行实例化

注意：数值越小，优先级越高。值相同，由服务器自己决定加载顺序

如果没有这个设置，那就在第一次访问时，实例化servlet

ServletConfig与ServletContext(4种数据共享的方式)

1、用户提交的参数（参数来自超链接、表单的name属性）

```
<a href="s2?username=tom">请点击 </a>
<form action="login" method=get>
    <input type="text" name="uname" />
     <input type="text" name="pwd" />
    <input type="submit" value="登录"/>

</form>
```

request.getParameter("username")

2、应用不同的Servlet之间（一个请求，请求转发）

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

request.getAttribute("name")

3、Servlet初始化参数的获取（定义web.xml或者当前Servlet的注解中），应用与同一个servlet对象

首先获得：ServletConfig config

config.getInitParameter("today")

或者在service()

getServletConfig().getInitParameter("today");

```
    configdemo
    com.edu.lhz.ServletConfigDemo

        today
        20200330


        search
        baidu

    0
```

4、全局初始化参数的获取（定义web.xml），应用与整个web应用范围

getServletContext().getInitParameter("param1");

```
    param1
    value1
```

```
    getServletContext().setAttribute(name, object);
    getServletContext().getAttribute(name)
```

@WebServlet注解与web.xml

```
//@WebServlet({ "/Servlet5", "/s5" })
//@WebServlet("/s5")
@WebServlet(name ="servletdemo5",
urlPatterns ="/s5",
loadOnStartup = 0,
initParams = {
        @WebInitParam(name="today",value = "20200413"),
        @WebInitParam(name="search",value = "百度"),
}
)
```

```
<servlet>
        <servlet-name>configdemo</servlet-name>
        <servlet-class>com.edu.lhz.ServletConfigDemo</servlet-class>
        <init-param>
            <param-name>today</param-name>
            <param-value>20200330</param-value>
        </init-param>
        <init-param>
            <param-name>search</param-name>
            <param-value>baidu</param-value>
        </init-param>
        <load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>configdemo</servlet-name>
    <url-pattern>/configservlet</url-pattern>
</servlet-mapping>
```

```
    String filePath="d:\\demo\\11.jpg";
        File file =new File(filePath);
        FileInputStream fis=new FileInputStream(file);
        byte[] data=new byte[fis.available()];

        fis.read(data);
        fis.close();

        response.setContentType("image/jpg");
        OutputStream os=response.getOutputStream();
        os.write(data);
        os.flush();
        os.close();
```

会话管理：Cookie的使用

会话：客户端连续不断地与服务器进行请求/响应的一系列交互；多次请求建立关联，称为会话管理。

获取不同请求的数据共享

Cookie技术是浏览器端（客户端）的数据存储技术，解决了不同的请求需要使用相同的请求数据的问题。

1）创建并向客户端发送Cookie

setMaxAge（）设置有效期，存储在用户PC上；如果不设置，有效期为一次会话，浏览器关闭，cookie就消失了

```
    Cookie cookie1=new Cookie("uname", u_name.concat("123"));
            cookie1.setMaxAge(60*60*24);
            response.addCookie(cookie1);

            Cookie cookie2=new Cookie("upwd", u_pwd.concat("456"));
            cookie2.setMaxAge(60*60*24);
            response.addCookie(cookie2);
```

2）从客户端获取Cookie

```
Cookie[] cookies=request.getCookies();
    if(cookies !=null) {
        for(int i=0;i<cookies.length;i++) {
            String s1=cookies[i].getName();
            String s2=cookies[i].getValue();
            out.print(s1+":");
            out.print(s2);
            out.print("</br>");

            Cookie cookie2=new Cookie("upwd", "qa123");
            cookie2.setMaxAge(60*60*24);
            response.addCookie(cookie2);
        }
```

特点：存储在浏览器端；适合少量数据存储，是键值对，不安全的

练习：利用Cookie实现，记录访问的次数的功能，三天免登陆

刷新一次，+1

会话管理：Session的使用

解决问题：

使用Session技术，解决一个用户的不同请求共享数据的问题

原理：

用户使用浏览器第一次向服务器发送请求，服务器在接收到这个请求之后，调用相应的servlet进行处理。在处理过程中会给用户创建一个session的对象，用来存储用户请求的共享数据，会将此session对象的JSESSIONID以Cookie的形式存储在浏览器中（临时存储，浏览器关闭就失效）。

用户发送第二个请求及后续请求，请求信息会附带JSESSIONID，服务器在接收到请求后，调用相应的servlet进行处理，同时根据JSESSIONID返回对应的session对象。

特点：

1）Session技术依赖与Cookie技术的服务器端的数据存储技术，

2）是由服务器创建的，

3）每个用户独立拥有一个session默认存储时间是30分钟

作用：

创建session/获取session对象

HttpSession session=req.getSession();

1、如果请求中有session标识符，也就是jsessionid，返回对应的session对象

2、如果请求中没有session标识符，也就是jsessionid，则创建新的session对象，并且将jsessionid作为cookie数据保存到浏览器内存中

3、session对象失效了，默认是半个小时，也会创建一个新的sessio对象，并且将jsessionid作为cookie数据保存到浏览器内存中

作用域：

一次会话（开始访问，浏览器关闭）

注意：如果在指定时间内，session对象没有被使用，会被销毁。session.setMaxInactiveInterval(5);

注意：jsessionid存储在cookie的临时空间，浏览器不关闭，不会失效。

只要不关闭浏览器，并且session不失效（没有超时），同一个用户的任何Servlet请求访问的都是同一个session对象

```
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {


//1、设置请求编码方式
    req.setCharacterEncoding("UTF-8");
//2、设置响应编码方式
    resp.setContentType("text/html;charset=UTF-8");
    PrintWriter out=resp.getWriter();

//3、获取请求信息
    String user_name=req.getParameter("username");
//  System.out.println(user_name);
//4、处理请求信息
//  如果是第一次访问，服务器会创建一个session的对象，会创建一个cookie，
//  把session的标识符JSESSIONID保存在cookie中，并且添加到响应
    HttpSession session=req.getSession();
```

```
    String mem_name=(String)session.getAttribute("name");
    if(mem_name == null) {
//      第一次访问
        out.write("你是第一次访问");
        out.write("<h1>"+user_name+"</h1>");
        session.setAttribute("name", user_name);
    }else {
        out.write(mem_name);
    }


//  5秒内，没有被访问，session对象就销毁了；如果访问了，就不会被，会重新计时
//  session.setMaxInactiveInterval(5);
    //  session.setMaxInactiveInterval(60*60*3);

//  session强制失效，适用于退出操作
//  session.invalidate();

//5、响应处理结果
//  resp.sendRedirect("member");


}
```

监听器

1、什么是监听器?

监听器主要监听（ServletContext、Session、Request）对象的创建、销毁和属性的变化，是一个实现特定接口的JAVA类，监听事件源上发生的行为或动作。

三类对象：事件源（被监听对象）、事件对象、监听器对象（分三类、如ServletContextListener、Session、Request）

监听ServletContext

实例：实现了访问量计数器的功能

1) ServletContextListener　用来监听 ServletContext的生命周期，即创建（启动服务器）、销毁（关闭服务器）

contextInitialized(ServletContextEvent sce)

contextDestroyed(ServletContextEvent sce)

ServletContextEvent　可用方法getServletContext（）

```
  public void contextDestroyed(ServletContextEvent sce)  {
        System.out.println("contextDestroyed");
        ServletContext context=sce.getServletContext();
        context.removeAttribute("count");
    }
```

```
    public void contextInitialized(ServletContextEvent sce)  {
        System.out.println("contextInitialized");
        ServletContext context=sce.getServletContext();
        context.setAttribute("count", 0);

    }
```

2） ServletContextAttributeListener

用来监听ServletContext属性的变化 setAttribute（增加/修改/删除属性）、removeAttribute

attributeAdded(ServletContextAttributeEvent scae)

attributeRemoved(ServletContextAttributeEvent scae)

attributeReplaced(ServletContextAttributeEvent scae)

ServletContextAttributeEvent 可以使用的方法 getName() getValue()

```
scae.getName()
scae.getValue()
```

```
 public void attributeAdded(ServletContextAttributeEvent scae)  {


        System.out.println("attributeAdded---"+ scae.getName()+":"
        + (int)scae.getValue());

    }

    public void attributeRemoved(ServletContextAttributeEvent scae)  {
         System.out.println("attributeRemoved-----"+ scae.getName()+":"+
(int)scae.getValue());
    }


    public void attributeReplaced(ServletContextAttributeEvent scae)  {
         System.out.println("attributeReplaced-----"+ scae.getName()+":"+
(int)scae.getValue());

    }
```

测试类

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        ServletContext context=getServletContext();
        int num=(int)context.getAttribute("count");
        num++;
        System.out.println(num);
        out.write("<h1>"+num+"</h1>");
        context.setAttribute("count", num);

    }
```

监听HttpSession

实例：实现了在线人数的统计的功能

1、初始化count值写在哪？答案：ServletContext

2、打开浏览器来一个用户，创建session写在哪？答案：Servlet；count++写在哪？答案：监听器sessionCreated方法

3、关闭浏览器，代表用户不在线，count-- 写在哪？答案：监听器

3）HttpSessionListener

sessionCreated(HttpSessionEvent se)

sessionDestroyed(HttpSessionEvent se)

形参HttpSessionEvent 可用方法 getSession()

4）HttpSessionAttributeListener

attributeAdded(HttpSessionBindingEvent se)

attributeRemoved(HttpSessionBindingEvent se)

attributeReplaced(HttpSessionBindingEvent se)

HttpSessionBindingEvent 可用方法

```
 se.getName()
 se.getValue()
```

```
public void sessionCreated(HttpSessionEvent se) {
    System.out.println("sessionCreated");
        HttpSession session = se.getSession();
//        获取ServletContext的count的值
        ServletContext context = session.getServletContext();
        Integer num = (Integer) context.getAttribute("count");
        if (num == null) {
            context.setAttribute("count", 1);
```

```
        } else {
            num++;
//          更新ServletContext的count的值
            context.setAttribute("count", num);
        }

}
```

```
public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("sessionDestroyed");

        HttpSession session = se.getSession();
//          获取ServletContext的count的值
        ServletContext context = session.getServletContext();
        int num = (int) context.getAttribute("count");
        num--;
//          更新ServletContext的count的值
        context.setAttribute("count", num);
    }
```

测试类Servlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
    //如果第一次方法，服务器会创建session对象，就会执行HttpSessionListener监听器中的
sessionCreated
        HttpSession session=request.getSession();
        session.setAttribute("myvalue", "lhz");

        //******************


        PrintWriter out=response.getWriter();

        out.write("<h1>"+session.getId()+"</h1>");

        ServletContext context=getServletContext();
        int num=(int)context.getAttribute("count");
        out.write("<h1>当前在线人数："+num+"</h1>");



    }
```

监听ServletRequest

5）ServletRequestListener

requestInitialized(ServletRequestEvent sre)

requestDestroyed(ServletRequestEvent sre)

形参ServletRequestEvent 可用方法

```
sre.getServletRequest()
sre.getServletContext()
```

6） ServletRequestAttributeListener

attributeRemoved(ServletRequestAttributeEvent srae)

attributeAdded(ServletRequestAttributeEvent srae)

attributeReplaced(ServletRequestAttributeEvent srae)

ServletRequestAttributeEvent 可用方法

```
srae.getName()
srae.getValue()
srae.getServletContext()
srae.getServletRequest()
```

有哪些方法可以获取ServletContext

ServletRequestEvent； Session对象获取，HttpSessionEvent；ServletContextEvent；Servlet中getServletContext()

使用监听器的步骤

1）创建一个监听器的类（实现一个接口）

2）注册监听器

方法1：@WebListener

方法2：webxml

com.listeners.ServletContextListenerImpl

3） 编写测试类（Servlet）

过滤器

1、过滤器的作用及生命周期

用于拦截用户的请求或者服务器返回给客户端的响应重新包装。

作用3个方面：

1）拦截用户请求，对用户请求进行预处理，再传递给目标资源

2）如果是过滤器链，把请求传递给下一个过滤器

3) 拦截对客户端的响应，对于响应进行业务逻辑处理，再返回给浏览器

生命周期：

1）（Servlet容器）服务器在启动时，会实例化过滤器的对象，并且调用init()方法；

2）如果有发往服务器的请求，服务器会调用

doFilter(request,response,chain){

//对于请求的业务逻辑代码1

chain.doFilter(request,response)

//对于响应的业务逻辑代码2，例如弹幕的效果

}

3）服务器关闭，会执行destroy()

2、过滤器的使用

2）创建过滤器：实现Filter接口的类

3）配置过滤器

方法1：使用注解@WebFilter("/Filter1")

方法2：web.xml配置

```
<filter>
        <filter-name>myfilter1</filter-name>
        <filter-class>com.filters.Filter1</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>myfilter1</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

/* 表示拦截该站点所有的请求，例如编码方式的设置

/s1 表示拦截指定的url请求，针对某个servlet的请求进行拦截

*.do 表示以".do"结尾的请求，一般用来进行模块拦截处理。例如等于登录的验证

3、多个过滤器的顺序是怎么样的?

加载、实例化、init()：在web.xml文件 倒序

拦截请求并对请求进行预处理，执行doFilter()：正序

拦截响应的顺序，执行doFilter()：倒序

销毁destroy()：在web.xml文件 倒序

最基本是作用：

1）设置统一的编码方式

```
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        System.out.println("Filter1-----doFilter");
        request.setCharacterEncoding(filterConfig.getInitParameter("encoding"));
        response.setContentType("text/html;charset=UTF-8");

        chain.doFilter(request, response);

        System.out.println("Filter1-----doFilter---after");

    }
```

2）Session的管理，登录信息放在session中

登录页面 login.jsp

```
<form action="login.do" method="post">
用户名：<input type="text" name="username">
<input type="submit" value="登录">

</form>
```

servlet类，对登录信息进行判断

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        // TODO Auto-generated method stub
//      doGet(request, response);

        String user_name=request.getParameter("username");
        if(user_name.equals("admin")) {
        HttpSession session=request.getSession();
        session.setAttribute("uname", user_name);
            request.getRequestDispatcher("welcome.jsp").forward(request,
response);
```

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

    }
```

过滤器，对session信息的判断

```
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        // TODO Auto-generated method stub
        System.out.println("Filter2-----doFilter");
        HttpSession session=((HttpServletRequest)request).getSession();
        if(session.getAttribute("uname")==null) {
            ((HttpServletResponse)response).sendRedirect("login.jsp");
        }else {
            chain.doFilter(request, response);
        }

        System.out.println("Filter2-----doFilter-----after");



    }
```

登录成功 welcome.jsp

```
<body>
欢迎你！！
<%=session.getAttribute("uname")%>
</body>
```