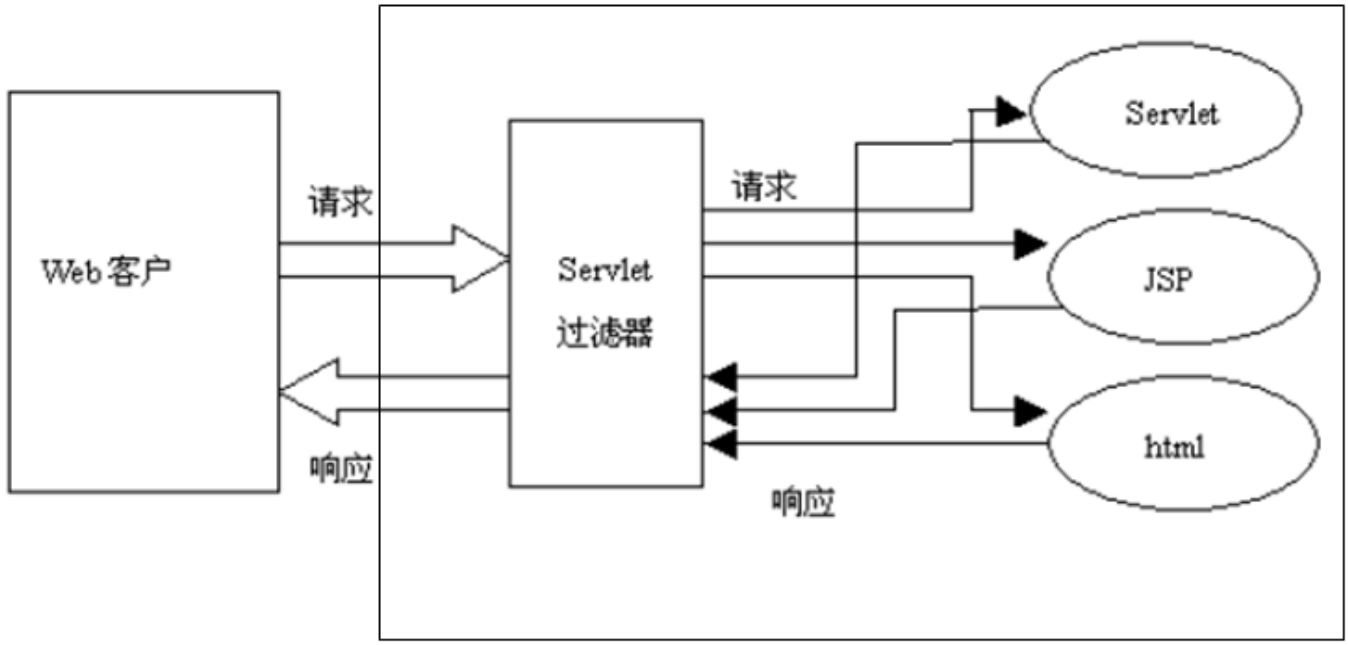
第十四章：Filter&Listener

第1节：Filter

1.1 概述

> 过滤器是一个对象，它对资源（servlet或静态内容）的请求或资源的响应执行过滤任务，或同时对两者执行过滤任务。过滤器在doFilter方法中执行过滤。过滤器是一个服务器端的组件，它可以截取客户端的请求和服务端的响应信息，并对这些信息进行过滤。



1.2 使用

> 方式一：使用web.xml注册过滤器
>
> 1. 创建一个普通类实现Filter接口
> 2. 重写Filter接口中的三个方法
> 3. 在web.xml文件中注册过滤器

> 过滤器java代码：

```
package com.offcn.sevlet;

import javax.servlet.*;
import java.io.IOException;
```

```
public class FirstFilter implements Filter {
    /*初始化*/
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("初始化过滤器");
    }
    /*过滤*/
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        //放行之前
        System.out.println("过滤请器放行之前");
        //放行访问目标资源
        chain.doFilter(request,response);
        //放行之后
        System.out.println("过滤请器放行之后");
    }
    /*销毁*/
    @Override
    public void destroy() {
        System.out.println("过滤器被销毁");
    }
}
```

web.xml注册过滤器：

```
    <!--注册过滤器-->
    <filter>
        <!--过滤器名称-->
        <filter-name>FirstFilter</filter-name>
        <!--过滤器完整路径-->
        <filter-class>com.offcn.sevlet.FirstFilter</filter-class>
    </filter>
    <!--配置过滤路径-->
    <filter-mapping>
        <!--过滤器名称-->
        <filter-name>FirstFilter</filter-name>
        <!--过滤器过滤的路径，如果是"/*"表示所有资源都要被过滤-->
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

方式二：使用注解方式注册过滤器

1. 创建一个普通类实现Filter接口
2. 重写Filter接口中的三个方法
3. 使用注解注册过滤器

案例代码：

```
package com.offcn.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;
@WebFilter("*.do")
public class SecondFilter  implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("second初始化过滤器");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        System.out.println("second过滤器放行之前");
        chain.doFilter(request,response);
        System.out.println("second过滤器放行之后");
    }

    @Override
    public void destroy() {
        System.out.println("second销毁过滤器");
    }
}
```

这里我们可以直接通过`@WebFilter("/*")`注解的方式进行过滤器的注册。

说明：过滤器常用的路径配置格式有以下三种

1./*过滤所有的服务器端资源

2.*.do表示过滤所有以.do结尾的服务器端资源

3./hello 只能过滤hello这个路径的服务器端资源

1.3 特点

1. 一个过滤器可以过滤多个servlet或者请求路径

web.xml

```
<!--注册过滤器-->
 <filter>
     <!--过滤器名称-->
```

```
    <filter-name>FirstFilter</filter-name>
    <!--过滤器完整路径-->
    <filter-class>com.offcn.filter.FirstFilter</filter-class>
</filter>
<!--配置第一个过滤路径-->
<filter-mapping>
    <!--过滤器名称-->
    <filter-name>FirstFilter</filter-name>
    <!--过滤器过滤的路径，如果是/*表示所有资源都要被过滤-->
    <url-pattern>/hello</url-pattern>
</filter-mapping>
<!--配置第二个过滤路径-->
<filter-mapping>
    <!--过滤器名称-->
    <filter-name>FirstFilter</filter-name>
    <!--过滤器过滤的路径，如果是/*表示所有资源都要被过滤-->
    <url-pattern>/demo.do</url-pattern>
</filter-mapping>
```

注解：

```
@WebFilter(urlPatterns = {"/hello","/demo.do"})
```

2.过滤器默认情况下只过滤重定向的路径，不过滤转发路径。

```
web.xml方式配置：
<filter-mapping>
<!--过滤器名称-->
<filter-name>ThirdFilter</filter-name>
<!--过滤器过滤的路径，如果是/*表示所有资源都要被过滤-->
<url-pattern>/second</url-pattern>
<!--设置过滤转发路径-->
<dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

注解方式配置：

```
@WebFilter(urlPatterns = {"/one","/second"},dispatcherTypes =
DispatcherType.FORWARD)
```

1.4 使用场景

场景1。字符编码过滤器

```
package com.offcn.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.sound.midi.Soundbank;
```

```
import java.io.IOException;
@WebFilter("/*")
public class CharaterFilter implements Filter {
 @Override
 public void init(FilterConfig filterConfig) throws ServletException {

 }

 @Override
 public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
throws IOException, ServletException {
     //编码处理
     req.setCharacterEncoding("UTF-8");
     resp.setContentType("text/html;charset=UTF-8");
     //放行
     chain.doFilter(req,resp);
 }

 @Override
 public void destroy() {

 }
}
```

场景2：登录控制

```
package com.offcn.filter;

import com.offcn.bean.Admin;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;

@WebFilter("/*")
public class LoginContrlFilter implements Filter {
 @Override
 public void init(FilterConfig filterConfig) throws ServletException {

 }

 @Override
 public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
throws IOException, ServletException {
     //将请求和响应对象进行向下转型
     HttpServletRequest request = (HttpServletRequest) req;
     HttpServletResponse response = (HttpServletResponse) resp;
```

```
        //创建sessoin
        HttpSession session = request.getSession();
        //获取session中的信息
        Admin admin = (Admin) session.getAttribute("admin");
        //判断哪些资源不需要过滤
        //获取请求的资源名称
        String requestURI = request.getRequestURI();
        String sourceName = requestURI.substring(requestURI.lastIndexOf("/") + 1);
        //除了登录和注册相关的资源不需要过滤，其他都需要登录成功后才可以访问
        if
(sourceName.equals("login.jsp")||sourceName.equals("login")||sourceName.equals("re
gist.jsp")||sourceName.equals("regist")){
            //放行
            chain.doFilter(req,resp);
        }else{
            //判断是否为空
            if (admin==null){
                //跳转到登录页面进行先登录
                response.sendRedirect("login.jsp");
            }else{
                //放行
                chain.doFilter(req,resp);
            }
        }
    }

    @Override
    public void destroy() {

    }
}
```

第2节：Listener

2.1 概述

> 监听器就是特定的Java接口，主要用于监听某个对象的状态变化的组件。

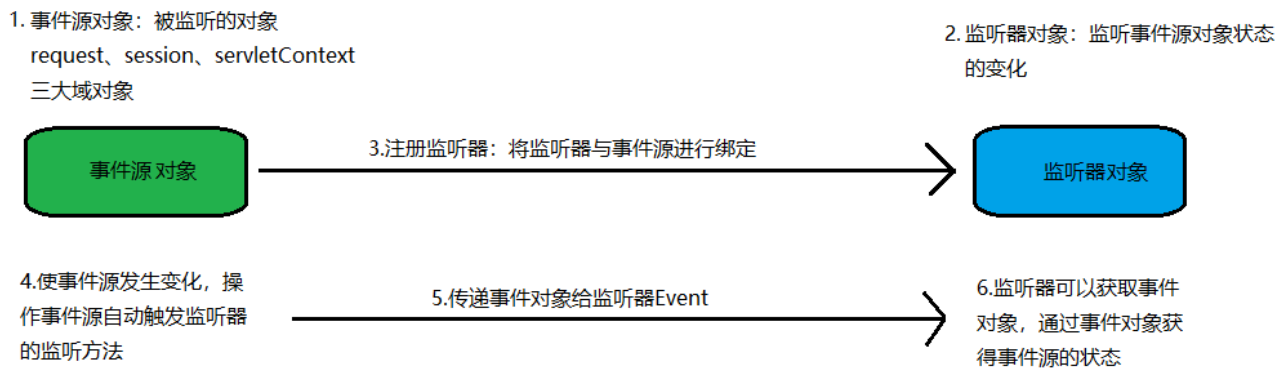
2.2 分类

> 第一维度：按照被监听的对象划分：ServletRequest域 、HttpSession域 、 ServletContext域
> 第二维度：监听的内容分：域对象的创建与销毁的、域对象的属性变化的、绑定到HttpSession域中某个对象状态的

维度	ServletContext域	HttpSession域	ServletRequest域
域对象的创建与销毁	ServletContextListener	HttpSessionListener	ServletRequestListener
域对象的属性变化	ServletContextAttributeListener	HttpSessionAttributeListener	ServletRequestAttributeListener
绑定到HttpSession域中某个对象的状态		HttpSessionBindingListener HttpSessionActivationListener	

2.3 原理

原理图：



1. 事件源：被监听的对象，即：request、session、servletContext三大域对象。
2. 监听器：监听事件源对象，事件源对象状态的变化都会触发监听器。
3. 注册监听器：将监听器与事件源进行绑定，有两种注册方式：web.xml 或 @WebListener注解
4. 事件：域对象发生改变

2.4 使用

1. 定义一个普通类实现监听器接口；
2. 重写监听器接口方法；
3. 注册监听器：配置 web.xml 或 @WebListener注解

2.4.1 ServletContextListener

用来监听ServletContext域对象的创建和销毁。

```
package com.offcn.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class MyServletContextListener implements ServletContextListener {
 //创建方法
 @Override
 public void contextInitialized(ServletContextEvent servletContextEvent) {
     System.out.println("ServletContextListener监听器被创建");
 }
  //销毁方法
```

```
 @Override
 public void contextDestroyed(ServletContextEvent servletContextEvent) {
     System.out.println("ServletContextListener监听器被销毁");
 }
}
```

注册监听器：

```
<listener>
 <listener-class>com.offcn.listener.MyServletContextListener</listener-class>
</listener>
```

测试：

当tomcat服务器开启时执行被创建的方法contextInitialized，服务器被关闭时执行被销毁的方法contextDestroyed。

2.4.2 HttpSessionListener

HttpSessionListener监听器：用来监听HttpSession域对象的创建和销毁。

Session何时创建：Servlet中是request.getSession() ，JSP页面中自带Session。

Session何时销毁：非正常关闭服务器，Session过期，session.invalidate()

```
package com.offcn.listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
@WebListener
public class MyHttpSessionListener implements HttpSessionListener {
 //创建方法
 @Override
 public void sessionCreated(HttpSessionEvent httpSessionEvent) {
     System.out.println("HttpSessionListener监听器被创建");
 }
 //销毁方法
 @Override
 public void sessionDestroyed(HttpSessionEvent httpSessionEvent) {
     System.out.println("HttpSessionListener监听器被销毁");
 }
}
```

注册监听器：

在MyHttpSessionListener监听器类的上方添加注解@WebListener

测试：直接访问index.jsp页面，由于session是jsp的内置对象，意味着在访问index.jsp页面时创建了session对象，因此会触发监听器的创建方法。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>测试</title>
</head>
<body>
<h2>测试HttpSessionListener监听器</h2>
</body>
</html>
```

测试session对象的销毁：

方式1：可以在web.xml文件中配置一下session的有效期，让期在1分钟过期，当超过1分钟没有操作服务器页面就会触发销毁方法；

```
<session-config>
<session-timeout>1</session-timeout>
</session-config>
```

方式2：执行session.invalidate()方法强制销毁session

```
<%
session.invalidate();
%>
```

2.4.3 ServletRequestListener

ServletRequestListener监听器：用来监听ServletRequest域对象的创建和销毁。
Request何时创建：请求发起时创建
equest何时销毁：响应结束时销毁

```
package com.offcn.listener;

import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpServletRequest;
@WebListener
public class MyServletRequestListener implements ServletRequestListener {
 @Override
 public void requestInitialized(ServletRequestEvent servletRequestEvent) {
     System.out.println("ServletRequestListener监听器被创建");
     //通过事件参数获取被监听的对象
```

```
        ServletRequest servletRequest = servletRequestEvent.getServletRequest();
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        //可以获取请求对象的相关信息
        System.out.println(request.getMethod());
    }
    @Override
    public void requestDestroyed(ServletRequestEvent servletRequestEvent) {
        System.out.println("ServletRequestListener监听器被销毁");
    }
}
```

注册监听器
在MyServletRequestListener监听器类的上方添加注解@WebListener

测试：创建一个jsp页面request.jsp，在该页面使用request做一次转发操作，先触发创建方法，转发结束后触发销毁方法。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试request监听器</title>
</head>
<body>
    <h2>测试ServletRequestListener监听器</h2>
    <%
        request.getRequestDispatcher("index.jsp").forward(request,response);
    %>
</body>
</html>
```

2.4.4 ServletContextAttributeListener

ServletContextAttributeListener监听器：监听ServletContext中属性的变化。

2.4.5 HttpSessionAttributeListener

HttpSessionAttributeListener监听器： 监听HttpSession中属性的变化。

2.4.6 ServletRequestAttributeListener

ServletRequestAttributeListener 监听器：监听ServletRequest中属性的变化。

以下是此类监听器对域中属性进行不同操作时所触发的方法：

1. attributeAdded监听属性添加 – 当数据范围对象没有该属性，第一次添加时会自动触发调用执行
2. attributeRemoved 监听属性移除 – 从一个数据范围对象删除一个已经存在的属性时会自动触发执行。
3. attributeReplaced监听属性替换 – 当一个数据范围已经存在一个属性，向数据范围添加相同名称属性时自动触发替换方法。

HttpSessionAttributeListener监听器为例，其他两个同理。

1. 创建一个普通类，实现HttpSessionAttributeListener接口，并重写内部的三个方法。

```
package com.offcn.listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;
@WebListener
public class MyHttpSessionAttributeListener implements
HttpSessionAttributeListener {
    //监听添加新的属性信息
    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
        System.out.println("被添加的属性名："+event.getName()+",属性的
值："+event.getSession().getAttribute(event.getName()));
    }

    //监听移除属性信息
    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
        System.out.println("被移除的属性名："+event.getName()+",属性的
值："+event.getSession().getAttribute(event.getName()));
    }
    //监听对已存在书属性的替换
    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
        System.out.println("被替换的属性名："+event.getName()+",属性的
值："+event.getSession().getAttribute(event.getName()));
    }
}
```

2. 注册监听器
   在MyHttpSessionAttributeListener监听器类的上方添加注解@WebListener

3. 测试监听器：创建一个jsp页面session.jsp，在该页面中分别对session域做添加，移除，替换的操作，这样就会触发不同的方法执行。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Session监听器</title>
</head>
<body>
    <%
        //添加新属性
        session.setAttribute("age",23);
        //替换属性
        session.setAttribute("age",35);
        //移除属性
        session.removeAttribute("age");
    %>
</body>
</html>
```

2.4.7 HttpSessionBindingListener

实现HttpSessionBindingListener接口的Java对象，可以感知自身被绑定到Session或者从Session中解除绑定。

1. 创建一个学生类Student实现HttpSessionBindingListener接口，并重写内部两个方法。

```
package com.offcn.listener;

import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class Student implements HttpSessionBindingListener {
 public Student() { }
 public Student(int id, String name) {
     this.id = id;
     this.name = name;
 }
 private int id;//学号
 private String name;//姓名

 public int getId() {
     return id;
```

```
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void valueBound(HttpSessionBindingEvent httpSessionBindingEvent) {
        System.out.println(httpSessionBindingEvent.getName()+"对象被绑定到Session中");
    }
    @Override
    public void valueUnbound(HttpSessionBindingEvent httpSessionBindingEvent) {
        System.out.println(httpSessionBindingEvent.getName()+"对象从Session中解除绑定");
    }
}
```

2. 测试：创建一个jsp页面，创建学生对象并分别绑定和移除Session进行测试。

```
<%@ page import="com.offcn.listener.Student" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>HttpSessionBindingListener</title>
</head>
<body>
    <h2>测试HttpSessionBindingListener监听器</h2>
    <%
        Student stu1 = new Student(1,"张三");
        //将学生对象绑定到Session中会触发valueBound方法
        session.setAttribute("stu1",stu1);
        //从session中移除会触发valueUnbound方法
        session.removeAttribute("stu1");
    %>
</body>
</html>
```

2.4.8 HttpSessionActivationListener

> 实现HttpSessionActivationListener接口的Java对象，可以感知从内存被钝化到硬盘，从硬盘活化到内存中。
> 钝化时机：服务器关闭或重启，指定时间内（长时间）不操作服务器。
> 活化时机：服务器再次开启。

> 1. 创建一个老师类Teacher实现HttpSessionActivationListener接口，并重写内部两个方法。

```
package com.offcn.listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionActivationListener;
import javax.servlet.http.HttpSessionEvent;
import java.io.Serializable;

public class Teacher implements HttpSessionActivationListener, Serializable {
 public Teacher() {
 }

 public Teacher(int age, String name) {
     this.age = age;
     this.name = name;
 }

 private int age;//年龄
 private String name;//姓名

 public int getAge() {
     return age;
 }

 public void setAge(int age) {
     this.age = age;
 }

 public String getName() {
     return name;
 }

 public void setName(String name) {
     this.name = name;
 }

 @Override
 public void sessionWillPassivate(HttpSessionEvent httpSessionEvent) {
     System.out.println(httpSessionEvent.getSession().getId()+"已钝化");
 }
```

```
 @Override
 public void sessionDidActivate(HttpSessionEvent httpSessionEvent) {
     System.out.println(httpSessionEvent.getSession().getId()+"已活化");
 }
}
```

2. 配置tomcat根目录下conf文件夹里的context.xml文件，其中directory="e:\test"表示自定义钝化文件的存储路径，maxIdleSwap="1"表示超过1分钟未操作服务器会自动钝化。

```
<Manager className="org.apache.catalina.session.PersistentManager"
saveOnRestart="true" maxIdleSwap="1">
 <Store className="org.apache.catalina.session.FileStore" directory="e:\test"/>
</Manager>
```

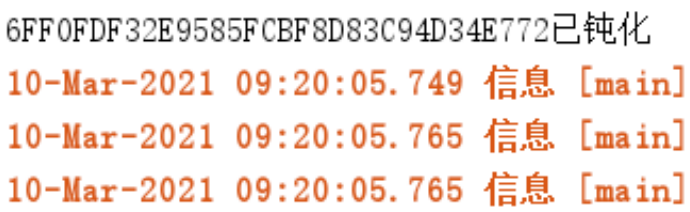
3. 测试：创建一个test01.jsp页面，向session中保存一个老师对象。

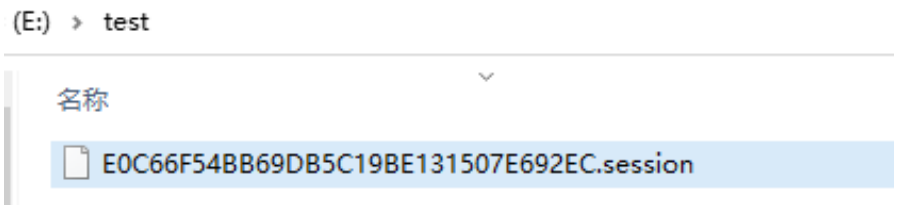
```
<body>
 <h2>测试HttpSessionActivationListener监听器</h2>
 <%
     Teacher teacher = new Teacher(45,"韩红");
     session.setAttribute("teacher",teacher);
 %>
</body>
```

4. 再创建一个test02.jsp页面获取活化后session中的数据是否存在

```
<body>
<h2>获取session的数据</h2>
 ${sessionScope.teacher.name}<br>
 ${sessionScope.teacher.age}<br>
</body>
```

首先运行test01.jsp页面，向session中存入老师信息，然后关闭服务器或通过设置maxIdleSwap="1"等待1分钟不操作服务器， 都会触发钝化方法执行，同时在指定的路径e:\test下会看到一个.session文件，将session中的数据钝化到该文件中。





然后重新启动服务器，再次访问test01.jsp，此时会触发活化方法将.session文件中的数据读取到内存，这里再访问test02.jsp，会读取到数据。

特别注意：
实现这两个接口的类不需要有 web.xml 文件或注解中进行注册监听器，都是由Session自主完成的。

2.5 使用场景

统计在线人数：
思路分析：通过ServletContextListener监听，当Web应用上下文启动时，在ServletContext中添加一个List集合，用来准备存放在线的用户名；然后，可以通过HttpSessionAttributeListener监听器，当用户登录成功，把用户名设置到Session中时，同时将用户名存放到ServletContext中的List列表中；最后通过HttpSessionListener监听，当用户注销会话时，将用户名从应用上下文范围中的List列表中删除。

实现步骤：
1.创建监听器，分别实现ServletContextListener、HttpSessionAttributeListener、HttpSessionListener接口。

```
package com.offcn.listener;

import javax.jws.Oneway;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
```

```
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
import java.util.LinkedList;
@WebListener
public class OnlineListener implements ServletContextListener,
HttpSessionAttributeListener, HttpSessionListener {
//创建一个全局作用域对象用于后面保存在线人数
private ServletContext application = null;
//启动web应用时会自动触发调用
@Override
public void contextInitialized(ServletContextEvent servletContextEvent) {
//初始化一个application对象
application=servletContextEvent.getServletContext();
//定义一个集合保存到application中
application.setAttribute("online",new LinkedList<String>());
}
//向session域中添加属性时触发调用
@Override
public void attributeAdded(HttpSessionBindingEvent httpSessionBindingEvent) {
//从application中把list集合取出
LinkedList<String> online = (
LinkedList<String>)application.getAttribute("online");
//判断向session中存储的属性名是否是username,如果是则取出值保存到集合中
if ("username".equals(httpSessionBindingEvent.getName())){
   String username = (String) httpSessionBindingEvent.getValue();
   //添加到集合中
   online.add(username);
}
System.out.println(online);
//将更新后的集合在保存回application中
application.setAttribute("online", online);
}
//修改session的属性值触发调用，内部操作同上
@Override
public void attributeReplaced(HttpSessionBindingEvent httpSessionBindingEvent) {
//从application中把list集合取出
LinkedList<String> online = (
LinkedList<String>)application.getAttribute("online");
//判断向session中存储的属性名是否是username,如果是则取出值保存到集合中
if ("username".equals(httpSessionBindingEvent.getName())){
   String username = (String)
httpSessionBindingEvent.getSession().getAttribute("username");
   //添加到集合中
   online.add(username);
}
System.out.println(online);
```

```
//将更新后的集合在保存回application中
application.setAttribute("online", online);
}

//当销毁session时触发调用
@Override
public void sessionDestroyed(HttpSessionEvent httpSessionEvent) {
//从application中把list集合取出
LinkedList<String> online = (
LinkedList<String>)application.getAttribute("online");
//取出当前用户名
String username = (String) httpSessionEvent.getSession().getAttribute("username");
//将当前用户名从列表中移除
online.remove(username);
//将更新后的集合在保存回application中
application.setAttribute("online", online);
}

//以下方法不需要实现
@Override
public void contextDestroyed(ServletContextEvent servletContextEvent) {

}
@Override
public void attributeRemoved(HttpSessionBindingEvent httpSessionBindingEvent) {

}
@Override
public void sessionCreated(HttpSessionEvent httpSessionEvent) {

}
}
```

2.创建用户登录的Servlet

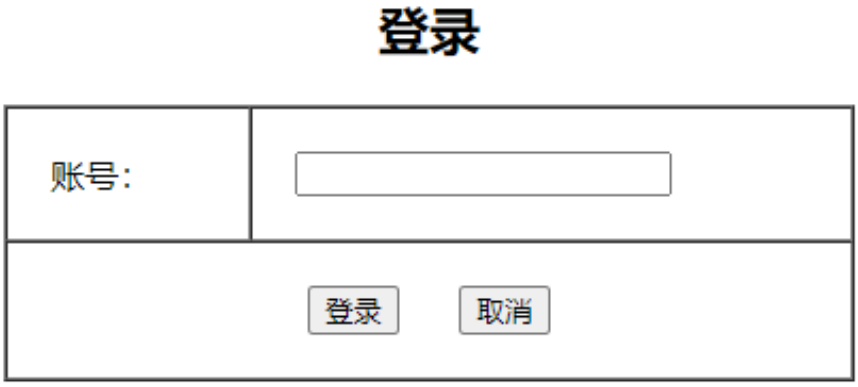
```
package com.offcn.servlet;

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;
import java.util.LinkedList;
import java.util.List;

/*登录的servlet*/
```

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
//获取用户名和密码
String username = req.getParameter("username");
//用户名不为空则认为登录成功
if (username!=null && !username.equals("")){
   //向session中存用户名,此时会触发attributeAdded方法向集合中存值
   req.getSession().setAttribute("username",username);
}
//从全局域中取在线人数列表做显示
ServletContext application = req.getServletContext();
List<String> online = (LinkedList<String>)application.getAttribute("online");
req.setAttribute("online",online);
req.getRequestDispatcher("online.jsp").forward(req,resp);
}
}
```

3. 创建用户注销的Servlet

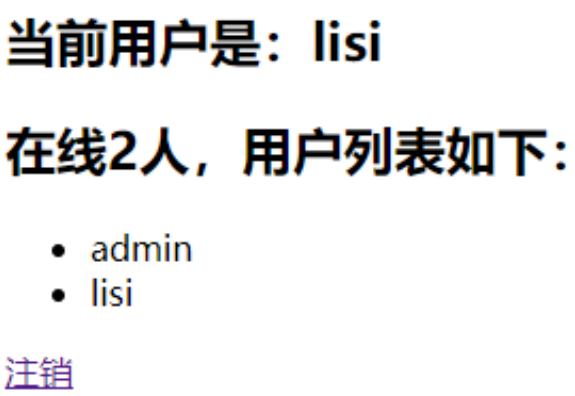
```
package com.offcn.servlet;

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;

/*登出的servlet*/
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
  req.getSession().invalidate();
  resp.sendRedirect("online.jsp");
}
}
```

4.创建登录页面login.jsp




```
<%--
Created by IntelliJ IDEA.
User: dell
Date: 2021/3/8
Time: 8:59
To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
<title>登录</title>
</head>
<body>
<center>
<form action="login" method="post">
<table width="400px" cellspacing="0" cellpadding="20" border="1">
    <caption><h2>登录</h2></caption>
    <tr>
        <td>账号: </td>
        <td><input type="text" name="username"></td>
    </tr>
    <tr align="center">
        <td colspan="2">
            <input type="submit" name="regist" value="登
录">     
            <input type="reset" name="ret" value="取消">
        </td>
    </tr>
</table>
</form>
</center>
</body>
</html>
```

5.创建显示在线人数页面online.jsp




```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>显示在线人数</title>
</head>
<body>
<h2>当前用户是：${sessionScope.username}</h2>
<h2>在线${onlie.size()}人，用户列表如下：</h2>
<ul>
  <c:forEach items="${online}" var="username">
      <li>${username}</li>
  </c:forEach>
</ul>
<a href="logout">注销</a>
</body>
</html>
```