

-

整理自互联网，仅供学习交流。



2018-10-30

[公司名称]

[公司地址]

目录

**未找到目录项。**

123

**Session与Cookie(自定义Session)**

**课程目标:**

**熟悉Cookie、Session底层实现原理、自定义缓存、自定义Token、表单重复提交解决方案、Servlet之Fileter解决XSS攻击。**

# 会话管理入门

## 生活中会话

我： 小张，你会跳小苹果码？

小张： 会，怎么了？

我： 公司年会上要表演节目，你教教我把

小张：没问题，一顿饭而已。

我： OK。

。。。。。。。。

在这次生活中的会话中产生通话记录（会话数据）

## 软件中的会话

一次会话： 打开浏览器 -> 访问一些服务器内容 -> 关闭浏览器

登录场景：

打开浏览器 -> 浏览到登陆页面 -> 输入用户名和密码 -> 访问到用户主页(显示用户名)

修改密码（输入原密码）

修改收货地址

.......

问题：在此处登录会话过程中产生的数据（用户会话数据）如何保存下来呢？

购物场景：

打开浏览器 -> 浏览商品列表 -> 加入购物车(把商品信息保存下来) -> 关闭浏览器

打开浏览器-> 直接进入购物车 -> 查看到上次加入购物车的商品 -> 下订单 -> 支付

问题： 在购物会话过程中，如何保存商品信息？？

会话管理： 管理浏览器客户端 和 服务器端之间会话过程中产生的会话数据。

域对象： 实现资源之间的数据共享。

request域对象

context域对象

登录场景：

小张： 输入“张三” （保存数据： context.setAttribute("name","张三")） -> 用户主页（显示“张三”）

小李： 输入“李四”(保存数据：context.setAttribute("name","李四")) -> 用户主页（显示“李四”）

问题： context是所有用户公有的资源！！！会覆盖数据。

小张： 输入“张三”（保存数据： request.setAttribute("name","张三")）- > 用户主页（显示“张三”） 问题： 一定要使用转发技术来跳转页面！！！

解决办法： 可以使用session域对象来保存会话数据！！！

## 会话技术

Cookie技术：会话数据保存在浏览器客户端。

Session技术：会话数据保存在服务器端。

# Cooke技术

## 特点

Cookie技术：会话数据保存在浏览器客户端。

## Cookie技术核心

Cookie类：用于存储会话数据

1）构造Coo kie对象

Cookie(java.lang.String name, java.lang.String value)

2）设置cookie

void setPath(java.lang.String uri) ：设置cookie的有效访问路径

void setMaxAge(int expiry) ： 设置cookie的有效时间

如果是负数，浏览器关闭就失效，如果是正数，以秒为单位保存。

cookie.setMaxAge(60 \* 60 \* 24);//cookie保存一天，没有一家公司能做到永久保存。

void setValue(java.lang.String newValue) ：设置cookie的值

3）发送cookie到浏览器端保存

void response.addCookie(Cookie cookie) : 发送cookie

4）服务器接收cookie

Cookie[] request.getCookies() : 接收cookie

## Cookie原理

1）服务器创建cookie对象，把会话数据存储到cookie对象中。

new Cookie("name","value");

2） 服务器发送cookie信息到浏览器

response.addCookie(cookie);

举例： set-cookie: name=eric (隐藏发送了一个set-cookie名称的响应头)

3）浏览器得到服务器发送的cookie，然后保存在浏览器端。

4）浏览器在下次访问服务器时，会带着cookie信息

举例： cookie: name=eric (隐藏带着一个叫cookie名称的请求头)

5）服务器接收到浏览器带来的cookie信息

request.getCookies();

## 2.4 Cookie的细节

1）void setPath(java.lang.String uri) ：设置cookie的有效访问路径。有效路径指的是cookie的有效路径保存在哪里，那么浏览器在有效路径下访问服务器时就会带着cookie信息，否则不带cookie信息。

2）void setMaxAge(int expiry) ： 设置cookie的有效时间。

正整数：表示cookie数据保存浏览器的缓存目录（硬盘中），数值表示保存的时间。

负整数：表示cookie数据保存浏览器的内存中。浏览器关闭cookie就丢失了！！

零：表示删除同名的cookie数据

3）Cookie数据类型只能保存非中文字符串类型的。可以保存多个cookie，但是浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。

## 2.5 案例- 显示用户上次访问的时间

|  |
| --- |
| @WebServlet("/LastAccessTime")  public class LastAccessTime extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  resp.setContentType("text/html;charset=utf-8");// 防止浏览器显示乱码  String lastAccessTime = null;  Cookie[] cookies = req.getCookies();  for (Cookie cookie : cookies) {  String name = cookie.getName();  if (name.equals("lastAccessTime")) {  lastAccessTime = cookie.getValue();  break;  }  }  if (StringUtils.*isEmpty*(lastAccessTime)) {  resp.getWriter().print("您是首次访问!");  } else {  resp.getWriter().print("你上次访问时间:" + lastAccessTime);  }  // 保存访问时间  // 创建cookie 将当前时间作为cookie保存到浏览器  String currenttime = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss").format(new Date());  Cookie cookie = new Cookie("lastAccessTime", currenttime);  cookie.setMaxAge(60 \* 60 \* 24);  // 发送cookie  resp.addCookie(cookie);  }  String addCookie(String key, String value, HttpServletResponse resp) {  return key;  }  } |

# Session技术

## 引入

Cookie的局限：

1）Cookie只能存字符串类型。不能保存对象

2）只能存非中文。

3）1个Cookie的容量不超过4KB。

如果要保存非字符串，超过4kb内容，只能使用session技术！！！

Session特点：

会话数据保存在服务器端。（内存中）

## Session技术核心

HttpSession类：用于保存会话数据

1）创建或得到session对象

HttpSession getSession()

HttpSession getSession(boolean create)

2）设置session对象

void setMaxInactiveInterval(int interval) ： 设置session的有效时间

void invalidate() ： 销毁session对象

java.lang.String getId() ： 得到session编号

3）保存会话数据到session对象

void setAttribute(java.lang.String name, java.lang.Object value) ： 保存数据

java.lang.Object getAttribute(java.lang.String name) ： 获取数据

void removeAttribute(java.lang.String name) ： 清除数据

## Session原理

问题： 服务器能够识别不同的浏览者！！！

现象：

前提： 在哪个session域对象保存数据，就必须从哪个域对象取出！！！！

浏览器1：(给s1分配一个唯一的标记：s001,把s001发送给浏览器)

1）创建session对象，保存会话数据

HttpSession session = request.getSession(); --保存会话数据 s1

浏览器1 的新窗口（带着s001的标记到服务器查询，s001->s1,返回s1）

1）得到session对象的会话数据

HttpSession session = request.getSession(); --可以取出 s1

新的浏览器1：(没有带s001,不能返回s1)

1）得到session对象的会话数据

HttpSession session = request.getSession(); --不可以取出 s2

浏览器2：(没有带s001,不能返回s1)

1）得到session对象的会话数据

HttpSession session = request.getSession(); --不可以取出 s3

代码解读：HttpSession session = request.getSession();

1）第一次访问创建session对象，给session对象分配一个唯一的ID，叫JSESSIONID

new HttpSession();

2）把JSESSIONID作为Cookie的值发送给浏览器保存

Cookie cookie = new Cookie("JSESSIONID", sessionID);

response.addCookie(cookie);

3）第二次访问的时候，浏览器带着JSESSIONID的cookie访问服务器

4）服务器得到JSESSIONID，在服务器的内存中搜索是否存放对应编号的session对象。

if(找到){

return map.get(sessionID);

}

Map<String,HttpSession>]

<"s001", s1>

<"s001,"s2>

5）如果找到对应编号的session对象，直接返回该对象

6）如果找不到对应编号的session对象，创建新的session对象，继续走1的流程

结论：通过JSESSION的cookie值在服务器找session对象！！！！！

## Sesson细节

1）java.lang.String getId() ： 得到session编号

2）两个getSession方法：

getSession(true) / getSession() : 创建或得到session对象。没有匹配的session编号，自动创建新的session对象。

getSession(false): 得到session对象。没有匹配的session编号，返回null

3）void setMaxInactiveInterval(int interval) ： 设置session的有效时间

session对象销毁时间：

3.1 默认情况30分服务器自动回收

3.2 修改session回收时间

3.3 全局修改session有效时间

|  |
| --- |
| <!-- 修改session全局有效时间:分钟 -->  <session-config>  <session-timeout>1</session-timeout>  </session-config> |

3.4.手动销毁session对象

void invalidate() ： 销毁session对象

4）如何避免浏览器的JSESSIONID的cookie随着浏览器关闭而丢失的问题

|  |
| --- |
| /\*\*  \* 手动发送一个硬盘保存的cookie给浏览器  \*/  Cookie c = **new** Cookie("JSESSIONID",session.getId());  c.setMaxAge(60\*60);  response.addCookie(c); |

总结：

1）会话管理： 浏览器和服务器会话过程中的产生的会话数据的管理。

2）Cookie技术：

new Cookie("name","value")

response.addCookie(coookie)

request.getCookies()

3）Session技术

request.getSession();

setAttrbute("name","会话数据");

getAttribute("会话数据")

# 自定义缓存

## 定义缓存实体类

|  |
| --- |
| package com.itmayiedu;  public class Cache {  public Cache(String key, Object value, Long timeOut) {  super();  this.key = key;  this.value = value;  this.timeOut = timeOut;  }  public Cache() {  }  /\*\*  \* key  \*/  private String key;  /\*\*  \* 缓存数据  \*/  private Object value;  /\*\*  \* 超时时间  \*/  private Long timeOut;  public String getKey() {  return key;  }  public void setKey(String key) {  this.key = key;  }  public Object getValue() {  return value;  }  public void setValue(Object value) {  this.value = value;  }  public Long getTimeOut() {  return timeOut;  }  public void setTimeOut(Long timeOut) {  this.timeOut = timeOut;  }  } |

## 4.2 定义缓存类

|  |
| --- |
| /\*\*  \*  \* @classDesc: 功能描述:(缓存map)  \* @author: 余胜军  \* @createTime: 2017年9月1日 下午3:19:24  \* @version: v1.0  \* @copyright:上海每特教育科技有限公司  \*/  public class CacheManager {  private Map<String, Cache> cacheMap = new HashMap<>();  /\*\*  \*  \* @methodDesc: 功能描述:(往缓存存值)  \* @author: 余胜军  \* @param: @param  \* key  \* @param: @param  \* oj  \* @createTime:2017年9月1日 下午3:20:58  \* @returnType:@param key  \* @returnType:@param oj void  \* @copyright:上海每特教育科技有限公司  \*/  public void put(String key, Object oj) {  put(key, oj, null);  }  /\*\*  \*  \* @methodDesc: 功能描述:(往缓存存值)  \* @author: 余胜军  \* @param: @param  \* key  \* @param: @param  \* oj  \* @createTime:2017年9月1日 下午3:20:58  \* @returnType:@param key  \* @returnType:@param oj void  \* @copyright:上海每特教育科技有限公司  \*/  public synchronized void put(String key, Object oj, Long timeOut) {  if (oj == null) {  return;  }  Cache cache = new Cache();  cache.setKey(key);  if (timeOut != null)  cache.setTimeOut(timeOut + System.*currentTimeMillis*());  cache.setValue(oj);  cacheMap.put(key, cache);  }  /\*\*  \*  \* @methodDesc: 功能描述:(删除)  \* @author: 余胜军  \* @param: @param  \* key  \* @createTime:2017年9月1日 下午3:26:54  \* @returnType:@param key void  \* @copyright:上海每特教育科技有限公司  \*/  public synchronized void deleteCache(String key) {  cacheMap.remove(key);  }  /\*\*  \*  \* @methodDesc: 功能描述:(获取缓存中数据)  \* @author: 余胜军  \* @param: @param  \* key  \* @param: @return  \* @createTime:2017年9月1日 下午3:24:03  \* @returnType:@param key  \* @returnType:@return Cache  \* @copyright:上海每特教育科技有限公司  \*/  public synchronized Object get(String key) {  Cache cache = cacheMap.get(key);  Object oj = null;  if (cache != null) {  oj = cache.getValue();  }  return oj;  }  /\*\*  \*  \* @methodDesc: 功能描述:(检查数据是否在有效期内)  \* @author: 余胜军  \* @param:  \* @createTime:2017年9月1日 下午3:25:21  \* @returnType: void  \* @copyright:上海每特教育科技有限公司  \*/  public synchronized void checkValidityData() {  for (String key : cacheMap.keySet()) {  Cache cache = cacheMap.get(key);  Long timeOut = cache.getTimeOut();  if (timeOut == null) {  return;  }  long currentTime = System.*currentTimeMillis*();  long endTime = timeOut;  long result = (currentTime - endTime);  if (result > 0) {  System.*out*.println("清除:"+key);  cacheMap.remove(key);  }  }  }  public static void main(String[] args) throws InterruptedException {  CacheManager cacheManager = new CacheManager();  // cacheManager.put("lisi", 0);  cacheManager.put("zhangsan", "jj", 5000l);  ScheduledExecutorService scheduledThreadPool = Executors.*newScheduledThreadPool*(5);  scheduledThreadPool.schedule(new Runnable() {  public void run() {  cacheManager.checkValidityData();  }  }, 5000, TimeUnit.*MILLISECONDS*);  Thread.*sleep*(5000);  System.*out*.println(cacheManager.get("zhangsan"));  }  } |

# 五、自定义Token

## 5.1 什么是token

token其实就是一个令牌,具有随机性,类似于sessionId。

在对接一些第三方平台的时候,为了能够保证数据安全性,通常会使用一些令牌进行交互

例如: https://mp.weixin.qq.com/wiki?t=resource/res\_main&id=mp1421140183

## 5.2 如何自定义token

token生成规则,只要保证token生成一个不重复的唯一字符串即可。

使用jdk自带的uuid生成规则。

## 5.3 什么是UUID

UUID含义是通用唯一识别码 (Universally Unique Identifier)，这是一个软件建构的标准，也是被开源软件基金会 (Open Software Foundation, OSF)   
的组织应用在分布式计算环境 (Distributed Computing Environment, DCE) 领域的一部分。

     UUID 的目的，是让分布式系统中的所有元素，都能有唯一的辨识资讯，而不需要透过中央控制端来做辨识资讯的指定。如此一来，每个人都可以建立不与其它人冲突的 UUID。  
在这样的情况下，就不需考虑数据库建立时的名称重复问题。目前最广泛应用的 UUID，即是微软的 Microsoft's Globally Unique Identifiers (GUIDs)，而其他重要的应用，  
则有 Linux ext2/ext3 档案系统、LUKS 加密分割区、GNOME、KDE、Mac OS X 等等

## UUID组成

UUID保证对在同一时空中的所有机器都是唯一的。通常平台会提供生成的API。按照开放软件基金会(OSF)制定的标准计算，用到了以太网卡地址、纳秒级时间、芯片ID码和许多可能的数字  
UUID由以下几部分的组合：  
（1）当前日期和时间，UUID的第一个部分与时间有关，如果你在生成一个UUID之后，过几秒又生成一个UUID，则第一个部分不同，其余相同。  
（2）时钟序列。  
（3）全局唯一的IEEE机器识别号，如果有网卡，从网卡MAC地址获得，没有网卡以其他方式获得。  
UUID的唯一缺陷在于生成的结果串会比较长。关于UUID这个标准使用最普遍的是微软的GUID(Globals Unique Identifiers)。在ColdFusion中可以用CreateUUID()函数很简单地生成UUID，  
其格式为：xxxxxxxx-xxxx- xxxx-xxxxxxxxxxxxxxxx(8-4-4-16)，其中每个 x 是 0-9 或 a-f 范围内的一个十六进制的数字。而标准的UUID格式为：xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12);

## UUID代码

|  |
| --- |
| UUID.randomUUID().toString() |

# 表单重复提交解决方案(防止Http重复提交。)

## 场景模拟

创建一个from.jsp页面

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  </head>    <body>  <form action="${pageContext.request.contextPath}/DoFormServlet" method="post">  用户名：<input type="text" name="userName">  <input type="submit" value="提交" id="submit">  </form>  </body>  </html> |

DoFormServlet 代码

|  |
| --- |
| import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  @WebServlet("/DoFormServlet")  public class DoFormServlet extends HttpServlet {  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.setCharacterEncoding("UTF-8");  String userName = req.getParameter("userName");  try {  Thread.*sleep*(300);  } catch (Exception e) {  // TODO: handle exception  }  System.*out*.println("往数据库插入数据...."+userName);  resp.getWriter().write("success");  }  } |

### 网络延时

在平时开发中，如果网速比较慢的情况下，用户提交表单后，发现服务器半天都没有响应，那么用户可能会以为是自己没有提交表单，就会再点击提交按钮重复提交表单，我们在开发中必须防止表单重复提交。

### 重新刷新

表单提交后用户点击【刷新】按钮导致表单重复提交

### 点击浏览器的【后退】按钮回退到表单页面后进行再次提交

用户提交表单后，点击浏览器的【后退】按钮回退到表单页面后进行再次提交

## 解决方案

### 使用javascript 解决

既然存在上述所说的表单重复提交问题，那么我们就要想办法解决，比较常用的方法是采用JavaScript来防止表单重复提交，具体做法如下：

修改form.jsp页面，添加如下的JavaScript代码来防止表单重复提交

代码:

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  <script type="text/javascript">  var isFlag = false; //表单是否已经提交标识，默认为false  function submitFlag() {  if (isFlag == false) {  isFlag = true;  return true;  } else {  return false;  }  }  </script>  </head>  <body>  <form action="${pageContext.request.contextPath}/DoFormServlet"  method="post" onsubmit="return submitFlag()">  用户名：<input type="text" name="userName"> <input type="submit"  value="提交" id="submit">  </form>  </body>  </html> |

除了用这种方式之外，经常见的另一种方式就是表单提交之后，将提交按钮设置为不可用，让用户没有机会点击第二次提交按钮，代码如下：

|  |
| --- |
| function dosubmit(){  //获取表单提交按钮  var btnSubmit = document.getElementById("submit");  //将表单提交按钮设置为不可用，这样就可以避免用户再次点击提交按钮  btnSubmit.disabled= "disabled";  //返回true让表单可以正常提交  return true;  } |

### 使用后端提交解决

对于【场景二】和【场景三】导致表单重复提交的问题，既然客户端无法解决，那么就在服务器端解决，在服务器端解决就需要用到session了。

具体的做法：**在服务器端生成一个唯一的随机标识号，专业术语称为Token(令牌)，同时在当前用户的Session域中保存这个Token。然后将Token发送到客户端的Form表单中，在Form表单中使用隐藏域来存储这个Token，表单提交的时候连同这个Token一起提交到服务器端，然后在服务器端判断客户端提交上来的Token与服务器端生成的Token是否一致，如果不一致，那就是重复提交了，此时服务器端就可以不处理重复提交的表单。如果相同则处理表单提交，处理完后清除当前用户的Session域中存储的标识号。**

在下列情况下，服务器程序将拒绝处理用户提交的表单请求：

1. **存储Session域中的Token(令牌)与表单提交的Token(令牌)不同。**
2. **当前用户的Session中不存在Token(令牌)。**
3. **用户提交的表单数据中没有Token(令牌)。**

转发代码:

|  |
| --- |
| @WebServlet("/ForwardServlet")  public class ForwardServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.getSession().setAttribute("sesionToken", TokenUtils.*getToken*());  req.getRequestDispatcher("form.jsp").forward(req, resp);  }  } |

转发页面:

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  </head>  <body>  <form action="${pageContext.request.contextPath}/DoFormServlet"  method="post" onsubmit="return dosubmit()">  <input type="hidden" name="token" value="${sesionToken}"> 用户名：<input type="text"  name="userName"> <input type="submit" value="提交" id="submit">  </form>  </body>  </html> |

后端Java代码:

|  |
| --- |
| /\*\*  \* @description: 功能描述：()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午4:16:16  \*/  package com.ysy.servlet;  import java.io.IOException;  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 javax.websocket.Session;  /\*\*  \* @description: TODO (表单servlet)  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午4:16:16  \*/  @WebServlet("/DoFormServlet")  public class DoFormServlet extends HttpServlet {  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.setCharacterEncoding("UTF-8");  if (!isSubmit(req)) {  System.out.println("你已经提交了数据，或者tokenu错误");  resp.getWriter().write("你已经提交了数据，或者tokenu错误");  }  String userName = req.getParameter("userName");  try {  Thread.sleep(300);  } catch (Exception e) {  // TODO: handle exception  }  System.out.println("往数据库插入数据...." + userName);  resp.getWriter().write("保存成功");  req.getSession().removeAttribute("sessionToken");  }  public Boolean isSubmit(HttpServletRequest request) {  String parameter = request.getParameter("parameterToken");  Session sessionToken = (Session) request.getSession().getAttribute("sessionToken");  // 判断是否提交  if (sessionToken == null) {  return false;  }  // 判断是否是伪造token  if (!(parameter.equals(sessionToken))) {  return false;  }  return true;  }  } |

|  |
| --- |
| /\*\*  \* @description: 功能描述：()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午5:14:12  \*/  package com.ysy.servlet;  import java.io.IOException;  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 javax.servlet.http.HttpSession;  /\*\*  \* @description: 功能描述 ()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午5:14:12  \*/  @WebServlet("/local")  public class LocalFromServlet extends HttpServlet {  /\*\*  \* (非 Javadoc) Title: doGet Description: 跳转到from.jsp  \*  \* @param req  \* @param resp  \* @throws ServletException  \* @throws IOException  \* @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,  \* javax.servlet.http.HttpServletResponse)  \*/  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  // 生成Token  String token = TokenUtils.getToken();  HttpSession session = req.getSession();  session.setAttribute("sessionToken", token);  req.getRequestDispatcher("index.jsp").forward(req, resp);  }  } |

# Servlet之Fileter

## 什么是Fileter？

Filter 也称之为过滤器，它是 Servlet 技术中最实用的技术，Web 开发人员通过 Filter 技术，对 web 服务器管理的所有 web 资源：例如 Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现 URL 级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

它主要用于对用户请求进行预处理，也可以对 HttpServletResponse 进行后处理。使用 Filter 的完整流程：Filter 对用户请求进行预处理，接着将请求交给 Servlet 进行处理并生成响应，最后 Filter 再对服务器响应进行后处理。

## 什么是Fileter使用

|  |
| --- |
| import java.io.IOException;  import java.util.Map;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 使用Filter 打印参数  \* @author Administrator  \*  \*/  public class FilterDemo implements Filter {  public FilterDemo(){  System.*out*.println("FilterDemo 构造函数被执行...");  }  /\*\*  \* 销毁  \*/  public void destroy() {  System.*out*.println("destroy");  }  public void doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse,  FilterChain paramFilterChain) throws IOException, ServletException {  System.*out*.println("doFilter");  HttpServletRequest request = (HttpServletRequest) paramServletRequest;  HttpServletResponse response = (HttpServletResponse) paramServletResponse;  // 请求地址  String requestURI = request.getRequestURI();  System.*out*.println("requestURI:"+requestURI);  // 参数  Map<String, String[]> parameterMap = request.getParameterMap();  for (String key : parameterMap.keySet()) {  String[] arr=parameterMap.get(key);  }  }  /\*\*  \* 初始化  \*/  public void init(FilterConfig paramFilterConfig) throws ServletException {  System.*out*.println("init");  }  } |

web.xml配置

|  |
| --- |
| <filter>  <filter-name>FilterDemo</filter-name>  <filter-class>com.itmayiedu.servlet.FilterDemo</filter-class>  </filter>  <filter-mapping>  <filter-name>FilterDemo</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## 使用Fileter防止XSS攻击

### 什么是XSS攻击？

XSS攻击使用Javascript脚本注入进行攻击

例如在表单中注入: <script>location.href='http://www.itmayiedu.com'</script>

实例:

演示:

代码: fromToXss.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="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>  <form action="XssDemo" method="post">  <input type="text" name="userName"> <input type="submit">  </form>  </body>  </html> |

代码: XssDemo

|  |
| --- |
| **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.annotation.WebServlet;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  @WebServlet("/XssDemo")  **public** **class** XssDemo **extends** HttpServlet {  @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  String userName = req.getParameter("userName");  req.setAttribute("userName", userName);  req.getRequestDispatcher("showUserName.jsp").forward(req, resp);  }    } |

代码: showUserName.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="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>userName:${userName}  </body>  </html> |

### 解決方案

使用Fileter过滤器过滤器注入标签

FilterDemo

|  |
| --- |
| import java.io.IOException;  import java.util.Map;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 使用Filter 打印参数  \*  \* @author Administrator  \*  \*/  public class FilterDemo implements Filter {  public FilterDemo() {  System.*out*.println("FilterDemo 构造函数被执行...");  }  /\*\*  \* 销毁  \*/  public void destroy() {  System.*out*.println("destroy");  }  public void doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse,  FilterChain paramFilterChain) throws IOException, ServletException {  System.*out*.println("doFilter");  HttpServletRequest request = (HttpServletRequest) paramServletRequest;  XssAndSqlHttpServletRequestWrapper xssRequestWrapper = new XssAndSqlHttpServletRequestWrapper(request);  // HttpServletResponse response = (HttpServletResponse)  // paramServletResponse;  // // 请求地址  // String requestURI = request.getRequestURI();  // System.out.println("requestURI:" + requestURI);  // // 参数  // Map<String, String[]> parameterMap = request.getParameterMap();  // for (String key : parameterMap.keySet()) {  // String[] arr = parameterMap.get(key);  // System.out.print("key:");  // for (String string : arr) {  // System.out.println(string);  // }  // }  paramFilterChain.doFilter(xssRequestWrapper, paramServletResponse);  }  /\*\*  \* 初始化  \*/  public void init(FilterConfig paramFilterConfig) throws ServletException {  System.*out*.println("init");  }  } |

**XssAndSqlHttpServletRequestWrapper**

|  |
| --- |
| **import javax.servlet.http.HttpServletRequest;**  **import javax.servlet.http.HttpServletRequestWrapper;**  **import org.apache.commons.lang3.StringEscapeUtils;**  **import org.apache.commons.lang3.StringUtils;**  **/\*\***  **\* 防止XSS攻击**  **\*/**  **public class XssAndSqlHttpServletRequestWrapper extends HttpServletRequestWrapper {**  **HttpServletRequest request;**  **public XssAndSqlHttpServletRequestWrapper(HttpServletRequest request) {**  **super(request);**  **this.request = request;**  **}**  **@Override**  **public String getParameter(String name) {**  **String value = request.getParameter(name);**  **System.*out*.println("name:" + name + "," + value);**  **if (!StringUtils.*isEmpty*(value)) {**  **// 转换Html**  **value = StringEscapeUtils.*escapeHtml4*(value);**  **}**  **return value;**  **}**  **}** |

|  |
| --- |
| **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletRequestWrapper;  **import** org.apache.commons.lang3.StringEscapeUtils;  **import** org.apache.commons.lang3.StringUtils;  /\*\*  \* 防止XSS攻击  \*/  **public** **class** XssAndSqlHttpServletRequestWrapper **extends** HttpServletRequestWrapper {  HttpServletRequest request;  **public** XssAndSqlHttpServletRequestWrapper(HttpServletRequest request) {  **super**(request);  **this**.request = request;  }  @Override  **public** String getParameter(String name) {  String value = request.getParameter(name);  System.***out***.println("name:" + name + "," + value);  **if** (!StringUtils.*isEmpty*(value)) {  // 转换Html  value = StringEscapeUtils.*escapeHtml4*(value);  }  **return** value;  }  } |

作业:使用Cookie实现自动登录效果。

本节课用到maven坐标

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.4</version>  </dependency> |

作业题： 完成自动登录。