课堂笔记

今天内容:

Listener监听器

Filter 过滤器(重点)

Filter案例

1.自动登陆

2.全站编码过滤

一、Listener监听器

Javaweb开发中的监听器，是用于监听web常见对象

HttpServletRequest　　HttpSession　　ServletContext

监听它们的创建与销毁　　属性变化　　session绑定javaBean

1、监听机制

事件　就是一个事情

事件源　产生这个事情的源头

监听器　用于监听指定的事件的对象

注册监听 要想让监听器可以监听到事件产生，必须对其进行注册。

2、Javaweb开发中常见监听器

2.1、监听域对象的创建与销毁

监听ServletContext创建与销毁　　ServletContextListener

监听HttpSession创建与销毁　　HttpSessionListener

监听HttpServletRequest创建与销毁　ServletRequestListener

2.2、监听域对象的属性变化

监听ServletContext属性变化　　ServletContextAttributeListener

监听HttpSession属性变化　　HttpSessionAttributeListener

监听HttpServletRequest属性变化　　ServletRequestAttributeListener

2.3、监听session绑定javaBean

它是用于监听javaBean对象是否绑定到了session域　　HttpSessionBindingListener

它是用于监听javaBean对象的活化与钝化　　HttpSessionActivationListener

3、监听器的快速入门

关于创建一个监听器的步骤

1. 创建一个类，实现指定的监听器接口
2. 重写接口中的方法
3. 在web.xml文件中对监听器进行注册。

3.1、关于域对象创建与销毁的演示

1. ServletContext对象的创建与销毁

这个对象是在服务器启动时创建的，在服务器关闭时销毁的。

```
public class MyServletContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("servletContext对象创建了");
    }

    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("servletContext对象销毁了");
    }

}
```

2. HttpSession对象的创建与销毁

HttpSession session=request.getSession();

Session销毁

1. 默认超时　　30分钟
2. 关闭服务器
3. invalidate()方法
4. setMaxInactiveInterval(int interval) 可以设置超时时间

问题:直接访问一个jsp页面时，是否会创建session?

会创建，因为我们默认情况下是可以在jsp页面中直接使用session内置对象的。

```java
public class MyHttpSessionListener implements HttpSessionListener {

    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("创建session对象");
    }

    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("销毁了session对象");
    }

}
```

3. HttpServletRequest创建与销毁

Request对象是发送请求服务器就会创建它，当响应产生时，request对象就会销毁。

```java
public class MyRequestListener implements ServletRequestListener {

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

    public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("request对象创建了");
    }

}
```

3.2、演示了Request域对象中属性变化

```java
public class MyRequestAttributeListener implements
        ServletRequestAttributeListener {

    public void attributeAdded(ServletRequestAttributeEvent srae) {
        System.out.println("添加了属性");
    }

    public void attributeRemoved(ServletRequestAttributeEvent srae) {
        System.out.println("删除了属性");
    }

    public void attributeReplaced(ServletRequestAttributeEvent srae) {
        System.out.println("替换了属性");
        System.out.println("替换了属性:"+srae.getName());
    }

}
```

在java的监听机制中，它的监听器中的方法都是有参数的，参数就是事件对象，而我们可以通过事件对象直接获取事件源。

3.3、演示session绑定javaBean

1. javaBean对象自动感知被绑定到**session**中.

HttpSessionBindingListener　这个接口是由javaBean实现的，并且不需要在web.xml文件中注册.

```
public class User implements Serializable, HttpSessionBindingListener {
    public void valueBound(HttpSessionBindingEvent event) {
        System.out.println("向session中绑定了一个user对象");
    }

    public void valueUnbound(HttpSessionBindingEvent event) {
        System.out.println("从session中将user对象移除了");
    }
```

```
<%
    User user = new User();
    user.setId(1);
    user.setUsername("tom");
    user.setPassword("123");
    user.setRole("admin");

    //绑定到session
    session.setAttribute("user", user);

    //解除绑定
    session.removeAttribute("user");
%>
```

2. javabean对象可以活化或钝化到**session**中。

HttpSessionActivationListener如果javaBean实现了这个接口，那么当我们正常关闭服务器时，session中的javaBean对象就会被钝化到我们指定的文件中。

当下一次在启动服务器，因为我们已经将对象写入到文件中，这时就会自动将javaBean对象活化到session中。

```
public class User implements Serializable, HttpSessionBindingListener,
        HttpSessionActivationListener {

    public void sessionWillPassivate(HttpSessionEvent se) {
        System.out.println("钝化");
    }

    public void sessionDidActivate(HttpSessionEvent se) {
        System.out.println("活化");
    }
```

我们还需要个context.xml文件来配置钝化时存储的文件

在meta-inf目录下创建一个context.xml文件

<Context>

<Manager className="org.apache.catalina.session.PersistentManager" maxIdleSwap="1">
<Store className="org.apache.catalina.session.FileStore" directory="it315"/>

</Manager>

</Context>

案例-定时销毁session

1. 怎样可以将每一个创建的session全都保存起来?

我们可以做一个HttpSessionListener，当session对象创建时，就将这个session对象装入到一个集合中.

将集合List<HttpSession>保存到ServletContext域中。

2. 怎样可以判断session过期了?

在HttpSession中有一个方法public long getLastAccessedTime()

它可以得到session对象最后使用的时间。

可以使用invalidate方法销毁。

我们上面的操作需要使用任务调度功能.

在java中有一个Timer定时器类

```
public class TimerDemo {

    public static void main(String[] args) {
        Timer t = new Timer(); // 创建了一个定时器

        t.schedule(new TimerTask() {

            @Override
            public void run() {

                System.out.println(new Date().toLocaleString());
            }
        }, 1000, 1000);
    }
}
```

关于三个域对象获取

如果在Servlet中要获取　request，在方法上就有，request.getSession()

getServletContext();

如果我们有request对象了, request.getSession()　request.getSession().getServletCotnext();

程序在使用时，需要考虑并发问题,因为我们在web中，它一定是一个多线程的，那么我们的程序对集合进行了添加，还有移除操作。

二、Filter过滤器(重要)

Javaweb中的过滤器可以拦截所有访问web资源的请求或响应操作。

1、Filter快速入门

1.1、步骤:

1. 创建一个类实现Filter接口
2. 重写接口中方法　doFilter方法是真正过滤的。
3. 在web.xml文件中配置

注意:在Filter的doFilter方法内如果没有执行chain.doFilter(request,response)那么资源是不会被访问到的。

```
public class Demo1Filter implements Filter {

    public void init(FilterConfig filterConfig) throws ServletException {

    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("开始拦截了");

        // 放行
        chain.doFilter(request, response);

        System.out.println("结束拦截");
    }

    public void destroy() {

    }

}
```

```xml
<filter>
    <filter-name>demo1Filter</filter-name>
    <filter-class>cn.itcast.web.filter.Demo1Filter</filter-class>
</filter>

<filter-mapping>
    <filter-name>demo1Filter</filter-name>
    <url-pattern>/demo1</url-pattern>
</filter-mapping>
```

1.2、FilterChain

FilterChain 是 servlet 容器为开发人员提供的对象，它提供了对某一资源的已过滤请求调用链的视图。过滤器使用 FilterChain 调用链中的下一个过滤器，如果调用的过滤器是链中的最后一个过滤器，则调用链末尾的资源。

问题:怎样可以形成一个Filter链？

只要多个Filter对同一个资源进行拦截就可以形成Filter链

问题:怎样确定Filter的执行顺序？

由<filter-mapping>来确定

1.3、Filter生命周期

Servlet生命周期：

实例化 --》 初始化 --》 服务 --》 销毁

1 当服务器启动，会创建Filter对象，并调用init方法，只调用一次.

1 当访问资源时，路径与Filter的拦截路径匹配，会执行Filter中的doFilter方法，这个方法是真正拦截操作的方法.

1 当服务器关闭时，会调用Filter的destroy方法来进行销毁操作.

1.4、FilterConfig

在Filter的init方法上有一个参数，类型就是FilterConfig.

FilterConfig它是Filter的配置对象，它可以完成下列功能

1. 获取Filtr名称
2. 获取Filter初始化参数
3. 获取ServletContext对象。

Method Summary	
String	**getFilterName**() Returns the filter-name of this filter as defined in the deployment descriptor.
String	**getInitParameter**(String name) Returns a String containing the value of the named initialization parameter, o
Enumeration	**getInitParameterNames**() Returns the names of the filter's initialization parameters as an Enumeration c parameters.
ServletContext	**getServletContext**() Returns a reference to the ServletContext in which the caller is executing.

问题:怎样在Filter中获取一个FIlterConfig对象？

```
public class Demo2Filter implements Filter {
    private FilterConfig config;

    public void init(FilterConfig config) throws ServletException {

        this.config = config;
    }
```

1.5、Filter配置

基本配置

<filter>

<filter-name>filter名称</filter-name>

<filter-class>Filter类的包名.类名</filter-class>

</filter>

<filter-mapping>

<filter-name>filter名称</filter-name>

<url-pattern>路径</url-pattern>

</filter-mapping>

关于其它配置

1.<url-pattern>

完全匹配　　以”/demo1”开始，不包含通配符*

目录匹配　　以”/”开始　　以*结束

扩展名匹配　*.xxx　　不能写成/*.xxx

2.<servlet-name>

它是对指定的servlet名称的servlet进行拦截的。

3.<dispatcher>

可以取的值有　REQUEST　FORWARD　ERROR　INCLUDE

它的作用是：当以什么方式去访问web资源时，进行拦截操作.

1.REQUEST 当是从浏览器直接访问资源，或是重定向到某个资源时进行拦截方式配置的 它也是默认值

2.FORWARD 它描述的是请求转发的拦截方式配置

3.ERROR 如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

4.INCLUDE 如果目标资源是通过RequestDispatcher的include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用

三、自动登陆

1、创建库与表

```
CREATE DATABASE day17
    USE day17
CREATE TABLE USER(
id INT PRIMARY KEY AUTO_INCREMENT,
username VARCHAR(100),
PASSWORD VARCHAR(100)
)
    INSERT INTO USER VALUES(NULL,"tom","123");
```

2、自动登陆功能实现:

1.当用户登陆成功后，判断是否勾选了自动登陆，如果勾选了，就将用户名与密码持久化存储到cookie中

2.做一个Filter，对需要自动登陆的资源进行拦截

问题

1.如果用户想要登陆操作，还需要自动登陆吗?

2.如果用户已经登陆了，还需要自动登陆吗?

四、MD5加密

在mysql中可以对数据进行md5加密

Md5(字段)

UPDATE USER SET PASSWORD=MD5(PASSWORD);

在java中也提供了md5加密

```
/**
   * 使用md5的算法进行加密
 */
public static String md5(String plainText) {
byte[] secretBytes = null;
try {
secretBytes = MessageDigest.getInstance("md5").digest(
plainText.getBytes());
} catch (NoSuchAlgorithmException e) {
throw new RuntimeException("没有md5这个算法！");
}
String md5code = new BigInteger(1, secretBytes).toString(16);
for (int i = 0; i < 32 - md5code.length(); i++) {
md5code = "0" + md5code;
}
return md5code;
```

五、全局的编码过滤器

分析:

我们之前做的操作，只能对post请求是ok

```
    FilterChain chain) throws IOException, ServletException
// 1.将request,response强制转换成Http协议下使用request,与response
HttpServletRequest request = (HttpServletRequest) req;
HttpServletResponse response = (HttpServletResponse) resp;

// 2.操作--将编码问题解决
request.setCharacterEncoding("utf-8");
response.setContentType("text/html;charset=utf-8");
// 3.放行
chain.doFilter(request, response);
```

怎样可以做成一个通用的，可以处理post,get所有的请求的?

在java中怎样可以对一个方法进行功能增强?

1. 继承
2. 装饰设计模式

1. 创建一个类让它与被装饰类实现同一个接口或继承同一个父类
2. 在装饰类中持有一个被装饰类的引用
3. 重写要增强的方法

问题:我们获取请求参数有以下方法

1. getParameter
2. getPrameterValues
3. getParameterMap

这三个方法都可以获取请求参数。

分析后，我们知道getParameter与getParameterValue方法可以依赖于getParamterMap方法来实现。

```
// 这个就是我们对request进行装饰的类
class MyRequest extends HttpServletRequestWrapper {

private HttpServletRequest request;// 是用于接收外部传递的原始的request

public MyRequest(HttpServletRequest request) {
super(request); // 是因为父类没有无参数构造
this.request = request;
```

```
}
// @Override
// public String getParameter(String name) {
// // 1.得到原来的getParameter方法的值
// String value = request.getParameter(name); // 乱码
//
// try {
// return new String(value.getBytes("iso8859-1"), "utf-8");
// } catch (UnsupportedEncodingException e) {
// e.printStackTrace();
// }
// return null;
// }


@Override
public String getParameter(String name) {

if (name != null) {
String[] st = (String[]) getParameterMap().get(name);
if (st != null && st.length > 0) {
return st[0];
}
}
return null;
}

@Override
public String[] getParameterValues(String name) {
if (name != null) {

return (String[]) getParameterMap().get(name);
}
return null;
}
```

```
private boolean flag = true;

@Override
public Map getParameterMap() {
// 1.得到原始的map集合
Map<String, String[]> map = request.getParameterMap();// 乱码

if (flag) {
// 2.将map集合中的String[]得到，解决每一个元素的乱码问题.
for (String key : map.keySet()) {

String[] st = map.get(key); // 得到每一个数组

for (int i = 0; i < st.length; i++) {

try {
st[i] = new String(st[i].getBytes("iso8859-1"), "utf-8");
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
}

}
flag = false;
}
return map;

}


}
```