设置session失效时间

在一般系统登录后，都会设置一个当前session失效的时间，以确保在用户没有使用系统一定时间后，自动退出登录，销毁session。

1.具体设置很简单，方法有三种：

（1）在主页面或者公共页面中加入：

session.setMaxInactiveInterval(900);   
\*单位是秒，设置为-1将永不关闭。

（2）在项目的web.xml中设置

<session-config>

<session-timeout>30</session-timeout>

</session-config>

\*这里的15也就是15分钟失效.

（3）直接在应用服务器中设置，如果是tomcat，可以在tomcat目录下conf/web.xml中 找到<session-config>元素，tomcat默认设置是30分钟，只要修改这个值就可以了。   
 <session-config>

<session-timeout>30</session-timeout>

</session-config>

（4）在server.xml中设置超时时间

<Context path="/livsorder" docBase="/home/httpd/html/livsorder"

　　defaultSessionTimeOut="3600" isWARExpanded="true"

　　isWARValidated="false" isInvokerEnabled="true"

　　isWorkDirPersistent="false"/>

需要注意的是如果上述三个地方如果都设置了，有个优先级的问题，从高到低：   
(1)---(2)---(3)----(4)

session的过期时间计算是从当前session的最后一次请求开始的。

**2.在一般系统中，也可能需要在session失效后做一些操作:**  
（1）控制用户数，当session失效后，系统的用户数减少一个等，控制用户数在一定范围内，确保系统的性能。   
（2）控制一个用户多次登录，当session有效时，如果相同用户登录，就提示已经登录了，当session失效后，就可以不用提示，直接登录了   
  
那么如何在session失效后，进行一系列的操作呢？   
这里就需要用到监听器了，即当session因为各种原因失效后，监听器就可以监听到，然后执行监听器中定义好的程序，就可以了。   
监听器类为：HttpSessionListener类，有sessionCreated和sessionDestroyed两个方法   
  
自己可以继承这个类，然后分别实现。   
sessionCreated指在session创建时执行的方法   
sessionDestroyed指在session失效时执行的方法   
给一个简单的例子：   
public class SessionListener implements HttpSessionListener{   
  
public void sessionCreated(HttpSessionEvent event) {   
HttpSession ses = event.getSession();   
String id=ses.getId()+ses.getCreationTime();   
SummerConstant.UserMap.put(id, Boolean.TRUE); //添加用户   
}   
  
public void sessionDestroyed(HttpSessionEvent event) {   
HttpSession ses = event.getSession();   
String id=ses.getId()+ses.getCreationTime();   
synchronized (this) {   
SummerConstant.USERNUM--; //用户数减一   
SummerConstant.UserMap.remove(id); //从用户组中移除掉，用户组为一个map   
}   
}   
}   
  
然后只需要把这个监听器在web.xml中声明就可以了   
例如：   
<listener>   
<listener-class>   
com.summer.kernel.tools.SessionListener   
</listener-class>   
</listener> 

 补充：

做一个过滤器，实现Filter接口，对指定路径下的请求进行session的失效验证，如失效则跳转到登录页面：

public class RequestFilter implements Filter {

public void doFilter(ServletRequest req, ServletResponse res,  
   FilterChain chain) throws IOException, ServletException {

         //在这里判断session是否已失效，如已失效则重定向到登录页面。

        /\*

        User user = (User)request.getSession().getAttribute("user");

        if(user == null) {

            response.sendRedirect(request.getContextPath()+"/login.jsp");  
            return;

        }

        \*/

   }

   public void init(FilterConfig config) throws ServletException {  
   }  
   
   public void destroy() {  
   }

}

在web.xml里配置过滤器：

  <filter>  
    <filter-name>requestFilter</filter-name>  
    <filter-class>com.xxx.RequestFilter</filter-class>  
  </filter>  
  <filter-mapping>  
    <filter-name>requestFilter</filter-name>  
    <url-pattern>/\*</url-pattern>  
  </filter-mapping>

至于你说的直接跳到登录页面，由于页面响应到达客户端浏览器后就已经与服务端断开连接并呈静态，必须要等到用户再次操作向服务端发起请求时才能执行。