1 什么是监听器

监听器就是一个实现特定接口的普通java程序，这个程序专门用于监听另一个java对象的方法调用或属性改变，当被监听对象发生上述事件后，监听器某个方法将立即被执行。

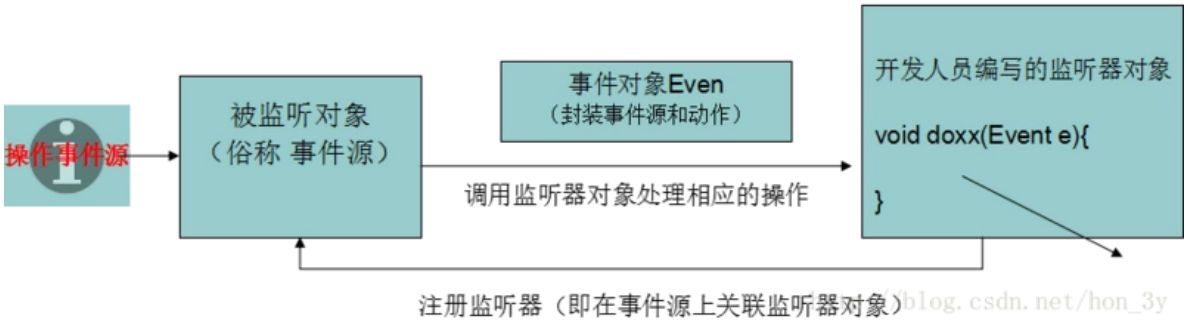
监听器可以用来检测网站的在线人数，统计网站的访问量等等！

2 监听器组件

监听器涉及三个组件：**事件源，事件对象，事件监听器**

当事件源发生某个动作的时候，它会调用事件监听器的方法，并在调用事件监听器方法的时候把事件对象传递进去。

我们在监听器中就可以通过事件对象获取得到事件源，从而对事件源进行操作！



3 模拟监听器

既然上面已经说了监听器的概念了，监听器涉及三个组件：事件源，事件对象，事件监听器。

我们就写一个对象，被监听器监听

3.1 监听器

监听器定义为接口，监听的方法需要事件对象传递进来，从而在监听器上通过事件对象获取得到事件源，对事件源进行修改！

```
/**
 * 事件监听器：监听Person事件源的eat和sleep方法
 */
interface PersonListener{
    void doEat(Event event);
    void doSleep(Event event);
}
```

3.2 事件源

事件源是一个Person类，它有eat和sleep()方法。

事件源需要注册监听器(即在事件源上关联监听器对象)

如果触发了eat或sleep()方法的时候，会调用监听器的方法，并将事件对象传递进去

```
/**
 * 事件源Person：事件源要提供方法注册监听器(即在事件源上关联监听器对象)
 */
class Person {
    //在成员变量定义一个监听器对象
    private PersonListener personListener ;
    //在事件源中定义两个方法
    public void Eat() {
        //当事件源调用了Eat方法时，应该触发监听器的方法，调用监听器的方法并把事件对象传递进去
        personListener.doEat(new Event(this));
    }
    public void sleep() {
        //当事件源调用了Eat方法时，应该触发监听器的方法，调用监听器的方法并把事件对象传递进去
        personListener.doSleep(new Event(this));
    }
    //注册监听器，该类没有监听器对象啊，那么就传递进来吧。
    public void registerLister(PersonListener personListener) {
        this.personListener = personListener;
    }
}
```

3.3 事件对象

事件对象封装了事件源。

监听器可以从事件对象上获取得到事件源的对象(信息)

```
/**
 * 事件对象Even
 * 事件对象封装了事件源
 * 在监听器上能够通过事件对象获取得到事件源
 */
class Event{
    private Person person;
    public Event() {
    }
    public Event(Person person) {
        this.person = person;
    }
    public Person getResource() {
        return person;
    }
}
```

3.4 测试

```
public static void main(String[] args) {
    Person person = new Person();
    //注册监听器()
    person.registerLister(new PersonListener() {
        @Override
        public void doEat(Event event) {
            Person person1 = event.getResource();
            System.out.println(person1 + "正在吃饭呢！");
        }
```

```
        @Override
        public void doSleep(Event event) {
            Person person1 = event.getResource();
            System.out.println(person1 + "正在睡觉呢！");
        }
    });
    //当调用eat方法时，触发事件，将事件对象传递给监听器，最后监听器获得事件源，对事件源进行操作
    person.Eat();
}
```

事件源：拥有事件

监听器：监听事件源所拥有的事件（带事件对象参数的）

事件对象：事件对象封装了事件源对象

- 事件源要与监听器有关系，就得注册监听器【提供方法得到监听器对象】
- 触发事件源的事件，实际会提交给监听器对象处理，并且把事件对象传递过去给监听器。

4 Servlet监听器

在Servlet规范中定义了多种类型的监听器，它们用于监听的事件源分别 ServletContext, HttpSession和ServletRequest这三个域对象。

和其它事件监听器略有不同的是，servlet监听器的注册不是直接注册在事件源上，而是由WEB容器负责注册，开发人员只需在web.xml文件中使用<listener>标签配置好监听器。

4.1 监听对象的创建和销毁

HttpSessionListener、ServletContextListener、ServletRequestListener分别监控着Session、Context、Request对象的创建和销毁。

- HttpSessionListener(可以用来收集在线者信息)
- ServletContextListener(可以获取web.xml里面的参数配置)
- ServletRequestListener

```
public class Listener1 implements ServletContextListener,
        HttpSessionListener, ServletRequestListener {
    // Public constructor is required by servlet spec
    public Listener1() {
    }
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("容器创建了");
    }
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("容器销毁了");
    }
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session创建了");
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session销毁了");
    }
    @Override
    public void requestDestroyed(ServletRequestEvent servletRequestEvent) {
```

```
    }
    @Override
    public void requestInitialized(ServletRequestEvent servletRequestEvent) {
    }
}
```

监听器监听到ServletContext的初始化了，Session的创建和ServletContext的销毁。(服务器停掉，不代表Session就被销毁了。Session的创建是在内存中的，所以没看到Session被销毁了)

4.2 监听对象属性变化

ServletContextAttributeListener、HttpSessionAttributeListener、ServletRequestAttributeListener分别监听着Context、Session、Request对象属性的变化。

这三个接口中都定义了三个方法来处理被监听对象中的属性的增加，删除和替换的事件，同一个事件在这三个接口中对应的方法名称完全相同，只是接受的参数类型不同。

- attributeAdded()
- attributeRemoved()
- attributeReplac()

```
public class Listener1 implements ServletContextAttributeListener {
    @Override
    public void attributeAdded(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Context对象增加了属性");
    }
    @Override
    public void attributeRemoved(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Context对象删除了属性");
    }
    @Override
    public void attributeReplaced(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Context对象替换了属性");
    }
}
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    ServletContext context = this.getServletContext();
    context.setAttribute("aa", "123");
    context.setAttribute("aa", "234");
    context.removeAttribute("aa");
}
```

4.3 监听Session内的对象

除了上面的6种Listener，还有两种Linstener监听Session内的对象，分别是HttpSessionBindingListener和HttpSessionActivationListener，实现这两个接口并不需要在web.xml文件中注册

- 实现HttpSessionBindingListener接口，JavaBean 对象可以感知自己被绑定到 Session 中和从 Session 中删除的事件【和HttpSessionAttributeListener的作用是差不多的】

- 实现HttpSessionActivationListener接口，JavaBean 对象可以感知自己被活化和钝化的事件（当服务器关闭时，会将Session的内容保存在硬盘上【钝化】，当服务器开启时，会将Session的内容在硬盘式重新加载【活化】）。

想要测试出Session的硬化和钝化，需要修改Tomcat的配置的。在META-INF下的context.xml文件中添加下面的代码：

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

监听器和事件源：

```
/*
* 由于涉及到了将内存的Session钝化到硬盘和用硬盘活化到内存中，所以需要实现Serializable接口
*
* 该监听器是不需要在web.xml文件中配置的。但监听器要在事件源上实现接口
* 也就是说，直接用一个类实现HttpSessionBindingListener和HttpSessionActivationListener
接口是监听不到Session内对象的变化的。
* 因为它们是感知自己在Session中的变化！
* */
public class User implements
HttpSessionBindingListener,HttpSessionActivationListener,Serializable {
    private String username ;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    @Override
    public void sessionWillPassivate(HttpSessionEvent httpSessionEvent) {
        HttpSession httpSession = httpSessionEvent.getSession();
        System.out.println("钝化了");
    }
    @Override
    public void sessionDidActivate(HttpSessionEvent httpSessionEvent) {
        HttpSession httpSession = httpSessionEvent.getSession();
        System.out.println("活化了");
    }
    @Override
    public void valueBound(HttpSessionBindingEvent httpSessionBindingEvent) {
        System.out.println("绑定了对象");
    }
    @Override
    public void valueUnbound(HttpSessionBindingEvent httpSessionBindingEvent) {
        System.out.println("解除了对象");
    }
}
```

测试代码：

```
User user = new User();
request.getSession().setAttribute("aaa", user);
request.getSession().removeAttribute("aaa");
```

5 统计网站在线人数

我们在网站中一般使用Session来标识某用户是否登陆了，如果登陆了，就在Session域中保存相对应的属性。如果没有登陆，那么Session的属性就应该为空。

现在，我们想要统计的是网站的在线人数。我们应该这样做：我们监听是否有新的Session创建了，如果新创建了Sesssion，那么在线人数就应该+1。这个在线人数是整个站点的，所以应该有Context对象保存。

- 监听Session是否被创建了
- 如果Session被创建了，那么在Context的域对象的值就应该+1
- 如果Session从内存中移除了，那么在Context的域对象的值就应该-1.

监听器代码：

```
public class CountOnline implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent se) {
        //获取得到Context对象，使用Context域对象保存用户在线的个数
        ServletContext context = se.getSession().getServletContext();
        //直接判断Context对象是否存在这个域，如果存在就人数+1,如果不存在就将属性设置到
Context域中
        Integer num = (Integer) context.getAttribute("num");
        if (num == null) {
            context.setAttribute("num", 1);
        } else {
            num++;
            context.setAttribute("num", num);
        }
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        ServletContext context = se.getSession().getServletContext();
        Integer num = (Integer) se.getSession().getAttribute("num");
        if (num == null) {
            context.setAttribute("num", 1);
        } else {
            num--;
            context.setAttribute("num", num);
        }
    }
}
```

我们每使用一个浏览器访问服务器，都会新创建一个Session。那么网站的在线人数就会+1。

使用同一个页面刷新，还是使用的是那个Sesssion，所以网站的在线人数是不会变的。

6 自定义Session扫描器

我们都知道Session是保存在内存中的，如果Session过多，服务器的压力就会非常大。

但是Session的默认失效时间是30分钟(30分钟没人用才会失效)，这造成Seesion可能会过多（没人用也存在内存中，这不是明显浪费吗？）

当然啦，我们可以在web.xml文件中配置Session的生命周期。但是呢，这是由服务器来做的，我嫌它的时间不够准确。（有时候我配置了3分钟，它用4分钟才帮我移除掉Session）

所以，我决定自己用程序手工移除那些长时间没人用的Session。

要想移除长时间没人用的Session，肯定要先拿到全部的Session啦。所以我们使用一个容器来装载站点所有的Session。

只要Sesssion一创建了，就把Session添加到容器里边。毫无疑问的，我们需要监听Session了。

接着，我们要做的就是隔一段时间就去扫描一下全部Session，如果有Session长时间没使用了，我们就把它从内存中移除。隔一段时间去做某事，这肯定是定时器的任务呀。

定时器应该在服务器一启动的时候，就应该被创建了。因此还需要监听Context。

最后，我们还要考虑到并发的问题，如果有人同时访问站点，那么监听Session创建的方法就会被并发访问了。定时器扫描容器的时候，可能是获取不到所有的Session的，这需要我们做同步。

大致的思路：

- 监听Session和Context的创建
- 使用一个容器来装载Session
- 定时去扫描Session，如果它长时间没有使用到了，就把该Session从内存中移除。
- 并发访问的问题

监听器代码：

```
public class Listener1 implements ServletContextListener,
HttpSessionListener {
    //服务器一启动，就应该创建容器。我们使用的是LinkList(涉及到增删)。容器也应该是线程安全的。
    List<HttpSession> list = Collections.synchronizedList(new
LinkedList<HttpSession>());
    //定义一把锁（Session添加到容器和扫描容器这两个操作应该同步起来）
    private Object lock = 1;
    public void contextInitialized(ServletContextEvent sce) {
        Timer timer = new Timer();
        //执行我想要的任务，0秒延时，每10秒执行一次
        timer.schedule(new MyTask(list, lock), 0, 10 * 1000);
    }
    public void sessionCreated(HttpSessionEvent se) {
        //只要Session一创建了，就应该添加到容器中
        synchronized (lock) {
            list.add(se.getSession());
        }
        System.out.println("Session被创建啦");
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session被销毁啦。");
    }
    public void contextDestroyed(ServletContextEvent sce) {
    }
}
```

任务代码：

```
/*
* 在任务中应该扫描容器，容器在监听器上，只能传递进来了。
* 要想得到在监听器上的锁，也只能是传递进来
```

```
* */
class MyTask extends TimerTask {
    private List<HttpSession> sessions;
    private Object lock;
    public MyTask(List<HttpSession> sessions, Object lock) {
        this.sessions = sessions;
        this.lock = lock;
    }
    @Override
    public void run() {
        synchronized (lock) {
            //遍历容器
            for (HttpSession session : sessions) {
                //只要15秒没人使用，我就移除它啦
                if (System.currentTimeMillis() - session.getLastAccessedTime() >
(1000 * 15)) {
                    session.invalidate();
                    sessions.remove(session);
                }
            }
        }
    }
}
```

15秒如果Session没有活跃，那么就被删除！

- 使用集合来装载我们所有的Session
- 使用定时器来扫描session的声明周期【由于定时器没有session，我们传进去就好了】
- 关于并发访问的问题，我们在扫描和检测session添加的时候，同步起来就好了【当然，定时器的锁也是要外面传递进来的】

7 踢人小案列

列出所有的在线用户，后台管理者拥有踢人的权利，点击踢人的超链接，该用户就被注销了。

首先，怎么能列出所有的在线用户呢？一般我们在线用户都是用Session来标记的，所有的在线用户就应该用一个容器来装载所有的Session。

我们监听Session的是否有属性添加(监听Session的属性有添加、修改、删除三个方法。如果监听到Session添加了，那么这个肯定是个在线用户！)。

装载Session的容器应该是在Context里边的【属于全站点】，并且容器应该使用Map集合【待会还要通过用户的名字来把用户踢了】

思路：

- 写监听器，监听是否有属性添加在Session里边了。
- 写简单的登陆页面。
- 列出所有的在线用户
- 实现踢人功能(也就是摧毁Session)

监听器代码：

```
public class KickPerson implements HttpSessionAttributeListener {
    // Public constructor is required by servlet spec
    public KickPerson() {
    }
```

```
    public void attributeAdded(HttpSessionBindingEvent sbe) {
        //得到context对象，看看context对象是否有容器装载Session
        ServletContext context = sbe.getSession().getServletContext();
        //如果没有，就创建一个呗
        Map map = (Map) context.getAttribute("map");
        if (map == null) {
            map = new HashMap();
            context.setAttribute("map", map);
        }
        //得到Session属性的值
        Object o = sbe.getValue();
        //判断属性的内容是否是User对象
        if (o instanceof User) {
            User user = (User) o;
            map.put(user.getUsername(), sbe.getSession());
        }
    }
    public void attributeRemoved(HttpSessionBindingEvent sbe) {
      /* This method is called when an attribute
         is removed from a session.
      */
    }
    public void attributeReplaced(HttpSessionBindingEvent sbe) {
      /* This method is invoked when an attibute
         is replaced in a session.
      */
    }
}
```

处理登陆Servlet：

```
//得到传递过来的数据
String username = request.getParameter("username");
User user = new User();
user.setUsername(username);
//标记该用户登陆了！
request.getSession().setAttribute("user", user);
//提供界面，告诉用户登陆是否成功
request.setAttribute("message", "恭喜你，登陆成功了！");
request.getRequestDispatcher("/message.jsp").forward(request, response);
```

处理踢人的Servlet：

```
String username = request.getParameter("username");
//得到装载所有的Session的容器
Map map = (Map) this.getServletContext().getAttribute("map");
//通过名字得到Session
HttpSession httpSession = (HttpSession) map.get(username);
httpSession.invalidate();
map.remove(username);
//摧毁完Session后，返回列出在线用户页面
request.getRequestDispatcher("/listUser.jsp").forward(request, response);
```

监听Seesion的创建和监听Session属性的变化有啥区别？

- Session的创建只代表着浏览器给服务器发送了请求。会话建立

- Session属性的变化就不一样了，登记的是具体用户是否做了某事(登陆、购买了某商品)