Tomcat 5.5 Proposal:
Adding More Flexibility to Config Listener and Deployer

Peter Roßbach & Frank Wegmann
pr@objektpark.de
Date 29.06.2004

Based on 5.0.25/27 CVS Head

Currently, you can automatically deploy, delete or redeploy web applications automatically using the classes HostConfig and StandardHostDeployer. But this is tied to the single webapps directory. However, customer scenarios would benefit from flexible mechanisms that allow deployment of applications from different directories or even a central HTTP repository. The necessity of a revised deployment implementation is further stressed by the new undocumented FarmWar Deployer facilitating an automatic cluster deployment. Here, with the introduction of a StandardHostDeployer variant it gets clear that it makes much sense to use different deployment sources at a single host in a uniform manner

To satisfy these requirements changes are necessary in the following areas:

- Flexible ConfigListener configuration
- Configurable StandardHostDeployer
- More than one deployer or config Listener
- Faster war META-INF/context auto deployment after changing a lot of war files
- JMX Notification at StandardContext und StandardWrapper

The following sections each analyze the current state and then a proposal is made how to improve the code (change requests/enhancements) in order to match the goals sketched above.

1) Config Classes of Container Elements

1.1) Status

HostConfig
    Autodeploy feature works for one webapps dir and one xml context descriptor dir
EngineConfig
    Start/Stop info
ContextConfig
    Context is configured as follows:
        Import DefaultContext
        Interpret the web.xml descriptor
        Auth Config
        Configure SessionManager

Default web.xml Handling

a) The Config Listener can only be changed in server.xml:

```
<Engine engineConfigClass="org.apache.catalina.startup.EngineConfig"
…>
<Host hostConfigClass="org.apache.catalina.startup.HostConfig"
      configClass="org.apache.catalina.startup.ContextConfig">
<Context configClass="org.apache.catalina.startup.ContextConfig" …>
```

See also the implementations for EngineRuleSet, HostRuleSet and LifecycleListenerRule.

b)-d) Creating new container elements at runtime:

b) Create new host at runtime:

```
MBeanFactory.createStandardHost()         (L. 943)
 // add HostConfig for active reloading
        HostConfig hostConfig = new HostConfig();
        host.addLifecycleListener(hostConfig);
```

c) Create new context at runtime:

```
MBeanFactory createStandardContext()      (L.826)
       ContextConfig contextConfig = new ContextConfig();
             context.addLifecycleListener(contextConfig);
```

d) Create new Service/Engine at runtime:
There is no correct setup for an EngineConfig. Currently the EngineConfig implementation only prints out start/stop!

e) Only StandardHost has a default for Context configClass, but the StandardHostDeployer uses the ContextRuleSet to create the ConfigListener.

Deployment without context.xml or context definition at server.xml:
ContextRuleset. 117

```
if (!isDefaultContext()) {
        digester.addRule(prefix + "Context",
                 new CopyParentClassLoaderRule());
        // here is the override
        digester.addRule(prefix + "Context",
                new LifecycleListenerRule
("org.apache.catalina.startup.ContextConfig",
                                "configClass"));
        digester.addRule(prefix + "Context", new
SetDocBaseRule());
        digester.addSetNext(prefix + "Context",
                            "addChild",
                            "org.apache.catalina.Container");
```

Deployment with context.xml or context definition at server.xml:
Second Config Listeners at StandardHostDeployer L.276, L 374
Case: No context.xml exists

```
if (context instanceof Lifecycle) {
          clazz = Class.forName(host.getConfigClass());
          LifecycleListener listener =
```

```
                    (LifecycleListener) clazz.newInstance();
                ((Lifecycle) context).addLifecycleListener(listener);
            }
```

The behaviour is not implemented in the method StandardHostDeployer.install(URL config, URL war) L . 424. – This case use the context.xml.

1.2) Change request:

All container elements in server.xml have a configListenerClass:

```
        <Engine configListenerClass="org.apache.catalina.startup.EngineConfig"
…>
```

The default value is empty so we don't need it! And we would also have:

```
        <Host configListenerClass="org.apache.catalina.startup.HostConfig" …>
        <Context
configListenerClass="org.apache.catalina.startup.ContextConfig" …>
```

The default values are set in the class implementation themselves and the Listener is added either to the init() method or to the installation method of the StandardHostDeployer:

```
        if(configListenerClass != null) {
              Is LifeCycleListener
              Create instance
              Add to Listeners
              Add getter and setter for this attribute
        }
```

Using the configListenerClass would free us from the special MbeanFactory code and we could also drop the digester code in Engine/Host/ContextRuleSet!
Adding the configListenerClass to the DefaultContext as well allows us to default the context config for Engine and Host. We can further remove the special handling of Host.configClass and use the DefaultContext instead.

BTW: Do we really need the method StandardHost.map() (L. 628)?

1.3) Enhancements

These enhancements realize a flexible deployment configuration per host.
Therefore we need to create a multi-valued configuration element <config> for the container elements Engine/Host/Context in server.xml which enables the user to configure a Host from more than one webapps or descriptor directory.
While the attribute unpackWars can be set to true for one of these directories, it may be false for others. So it is the decision of the application whether its war files should be unpacked and not that of the host.
The Farm War Deployer is the first extension, but I think more cases exist: For example, a deployer could activate auto-reloading for one application, while

applications in another monitored directory are deployed statically. In a cluster scenario, the Farm War Deployer can distribute the war files to all cluster nodes.

Internally, we can use the Listener/Lifecycle code to implement this Config. Auto-deployment should be possible from a list of URLs so that the user can deploy from a repository of web server applications and/or use a single deployment place for a farm of servers.

Example:

```
<Host >
    <Config className="org.apache.catalina.deploy.DirectoryAutoDeployer"
            deployOnStartup="true"
            appBase="webapps"
            unpackWars="true"/>
    <Config className="org.apache.catalina.deploy.DirectoryAutoDeployer"
            deployOnStartup="true"
            appBase="adminapps"
            unpackWars="false"
            autoDeploy="false"/>
    <Config className="org.apache.catalina.deploy.XMLContextAutoDeployer"
            deployOnStartUp=true"
            directory="context" /> <!—default conf/<engine>/<host> -->
…
</Host>
```

This will lead to a lot of code changes, possibly also by breaking existing interfaces but now the time has come to tackle this issue!

Last but not least: These configuration changes should certainly be accessible in the admin web app as well.

2) HostDeployer

2.1) Status

Currently we can't change the default StandardHostDeployer in a clean way. We would have to override the StandardHost class, but this leads to some problems with the current runtime configuration via JMX. The alternative would be to set with StandardHost.setDeployer() your own Deployer at a special Listener class (start())

--- StandardHost L.1019

```
static String
STANDARD_HOST_DEPLOYER="org.apache.catalina.core.StandardHostDeployer
";

    public Deployer getDeployer() {
        if( deployer!= null )
            return deployer;
        log.info( "Create Host deployer for direct deployment ( non-
jmx ) ");
        try {
            Class c=Class.forName( STANDARD_HOST_DEPLOYER );
            deployer=(Deployer)c.newInstance();
            Method m=c.getMethod("setHost", new Class[] {Host.class}
);
```

```
            m.invoke( deployer,  new Object[] { this } );
        } catch( Throwable t ) {
            log.error( "Error creating deployer ", t);
        }
        return deployer;
    }

    public void setDeployer(Deployer d) {
        this.deployer=d;
    }
```

For clustering scenarios there is another deployer class FarmWarDeployer (in org/apache/catalina/cluster/deploy) that handles cluster applications as follows:

- You have your own directory
- Autodeployment takes place on all nodes in your cluster
- New cluster nodes will get all applications
  (But please check your net for new illegal nodes ☺)

2.2) Change request:

Don't hardcode the parameter STANDARD_HOST_DEPLOYER. Instead use:

```
<host deployerClass="org.apache.catalina.core.StandardHostDeployer">
```

And add getter/setter methods to StandardHost.

2.3) Enhancements

These enhancements let the user define the deployment for a host or cluster in a uniform and transparent manner and also allow customization at deployment time.

Therefore we need to create a multi-valued configuration element <Deployer> for the elements Host and Cluster in server.xml. Any deployment action can be customized by adding a call to a hook (before or after deployment) to easily extend the StandardHostDeployer.

Example:
A very interesting use case is adding security code

```
<Deployer className"org.apache.catalina.core.StandardHostDeployer">
      <Hook action="installRemote"
className="org.xxx.SecurityDeployerHook"/>
</Deployer>
```

Actions are defined by the action attribute. Possible values are:

installRemote	install a war (config and War URL)
installLocal	install a directory or war locally
remove	undeploy existing apps

The FarmWARDeployer needs refactoring to realize this enhancement. Further necessary actions are:
- Fix Security issues (only deploy at auth nodes)
- Integrate a multi-deployer configuration at Host Level.

3) HostConfig

3.1) Status

HostConfig implements the auto host deployment feature:

Call every 10 sec from the backgroundThread System (Engine backgroundProcessorDelay=10)

Validate with an InfoLifecycleListener

```
     [exec] 20:47:35,538 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
     [exec] 20:47:45,572 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
     [exec] 20:47:55,616 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
     [exec] 20:48:05,871 INFO  [StandardHostDeployer] Processing
Context configuration file URL
file:D:\tomcat\project\tomcatexamples\tomcat\examples\tomcat5\advance
d\jmxnotify\webdev-server\conf\Catalina\localhost\myapps.xml
     [exec] 20:48:06,512 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
     [exec] 20:48:16,577 INFO  [StandardHostDeployer] Processing
Context configuration file URL
file:D:\tomcat\project\tomcatexamples\tomcat\examples\tomcat5\advance
d\jmxnotify\webdev-server\conf\Catalina\localhost\myapps1.xml
     [exec] 20:48:17,047 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
     [exec] 20:48:27,092 INFO  [InfoLifecycleListener]
StandardEngine[Catalina].StandardHost[localhost]: check
```

Deploy context.xml from host conf/<enginename>/<host> directory
Deploy all wars from host.appBase (webapps) directory
Deploy all webapp dirs from host.appBase (webapps) directory

For every war file that includes a META-INF/context.xml, the following actions are performed:

Detect the descriptor
Extract the descriptor to conf/<enginename>/<host>/
Then deploy again all descriptors
Stop deploy cycle (Hups)

The relevant code in HostConfig:

```
File xml = new File
            (configBase, files[i].substring
                     (0, files[i].lastIndexOf(".")) + ".xml");
            if (!xml.exists()) {
                try {
                    jar = new JarFile(dir);
                    entry = jar.getJarEntry("META-INF/context.xml");
                    if (entry != null) {
                        istream = jar.getInputStream(entry);
                        ostream = new BufferedOutputStream
                                  (new FileOutputStream(xml), 1024);
                        byte buffer[] = new byte[1024];
                        while (true) {
```

```
                            int n = istream.read(buffer);
                            if (n < 0) {
                                 break;
                            }
                            ostream.write(buffer, 0, n);
                        }
                        ostream.flush();
                        ostream.close();
                        ostream = null;
                        istream.close();
                        istream = null;
                        entry = null;
                        jar.close();
                        jar = null;
                        deployDescriptors(configBase(),
configBase.list());
                        return;
                    }
                } catch (Exception e) {
                    // Ignore and continue
                    if (ostream != null) {
                        try {
                            ostream.close();
```

See the return call after the call to deployDescriptors()!
Why stop the deployment cycle and wait 10 seconds for the next call?
Why do we not install only the single context.xml instead to loop over the complete conf/<enginename>/<hostname> directory?

The StandardContext has an attribute unpackWAR, but only the SetDocBaseRule class handles this attribute. StandardHostDeployer doesn't:

See StandardHostDeployer method install(URL,URL) (L. 458):

```
 // Expand war file if host wants wars unpacked
        if (isWAR && host.isUnpackWARs()) {
            docBase = ExpandWar.expand(host, war);
        }
```

Currently we can't use unpackWAR=false when deploying with META-INF/context.xml.

3.2) Change request:

Don't return after deployDescriptors and the first WAR deployment will be faster!
Don't extract war at StandardHostDeployer and wait for Context SetDocBaseRule handling

For War without a META-INF/context.xml, generate a default context.xml
Also check the deployment of the ManagerServlet.

But also remove the ExpandWars and generate a default conext.xml instead.
And check for special case ROOT app.

3.3) Enhancements

Introduce directory deployment also for META-INF/context.xml
Check META-INF/context.xml for missing or older conf/<enginename>/<host>/app.xml.
Refactor the ManagerServlet code:
- Extract some methods
- Delete duplicated code
- Simpler interfaces – extract Beans
- Split the servlet into useful components:
  - Manager servlet
    - List Apps
    - Reload
    - Start/stop
    - Deploy/undeploy
  - All other information
    - Sessions
    - Resources
    - …

Add Config and Deployer elements to Host.

4) JMX Notification at StandardContext und StandardWrapper

4.1 Status

StandardContext and StandardWrapper has a broadcastsupport object, but nobody can register a NotificationListener.

Central JMX-based monitoring for all applications is a nice feature.

4.2 Change Request

We must add the NotificationEmitter Interface and the implementation
- Done and tested.
- One problem still exists: it is not possible to directly register NotificationListener at MX4j MBeanServer. However, a direct registration at a started Context works.

```
try {
    ObjectName managercontext = new ObjectName(

     "Catalina:j2eeType=WebModule,name=//localhost/,J2EEAp
plication=none,J2EEServer=none");
     log.info(managercontext);
// work:
     StandardContext context =(StandardContext)
           mserver.getAttribute(managercontext,
"managedResource");
     context.addNotificationListener(this, null, null);

// not work:
```



```
//         mserver.addNotificationListener(
//                         managercontext,this,null,null);
      log.info("Notification to ROOT added");
} catch (Exception e) {
                log.error(e);
}
```

StandardHost.removeChild()

The destroy method of the child is not called.

Problem:

StandardHostDeployer.remove(String) or StandardHostDeployer.remove(String,Boolean) destroy() not called at removed Context.

Result:

The JMX NotifcationBroadcast support does not emit a deleted j2ee.object event.

Question: How we can register a Notification Listener before init() of Context or Wrapper is called?

Currently there is no chance to get the notification “j2ee.state.starting” for StandardHostDeployer installed applications or servlets.

Reason:

The Context JMX registration takes place after the call of the init method call.

JMX Registration happens only after the complete start.

Regards
Peter Roßbach (pr@objektpark.de)
Frank Wegmann

