



# CLASSLOADERS AND YOU

Joel Tosi  
04/07/2011

This technical whitepaper covers classloading in the JBoss Enterprise Application Platform.

## Contents

1. ClassLoader Background.....	2
1.1 What is ClassLoading?.....	2
1.2 Common Issues with ClassLoading.....	2
2. JBoss Classloading Configuration File.....	2
2.1 JBoss ClassLoading Configuration File Format.....	2
2.2 JBoss ClassLoading Configuration File Location.....	4
3. War ClassLoading.....	4
4. Ear Isolation.....	4
5. Debugging.....	5
6. Common Exceptions.....	8
Resources.....	9
Questions/Comments/Issues.....	9



## 1. CLASSLOADER BACKGROUND

### 1.1 What is ClassLoading?

We are familiar with the Java ClassLoader – it is quite simply the part of the JVM that loads our Java classes for execution. Java classes are loaded into the JVM and reside in the permanent generation (PermGen) of the heap. They reside in this space so they are always available to the JVM.

ClassLoading is the act of the class loader loading java class files into memory to be used by the JVM. In addition to the loading of classes, the ClassLoader handles such things as visibility of classes and security permissions around loaded classes.

### 1.2 Common Issues with ClassLoading

Most of the time, we have no reason to worry about class loading. In simple, isolated programs that run independently, there is little risk of issues with class loading. However, as soon as we are developing applications in the same memory space, we increasingly run the risk of class loader issues.

The most common issue is when two applications are attempting to use different versions of a library. For example, say you have some common code for tax calculation and two applications that use that common code. If one application is using version 1 of that tax calculation code and another application is trying to use version 2 – you are fine if those applications are running independently. However, if both applications run in the same memory space, which class should the ClassLoader load? And there is our problem – applications in the same memory space needing different versions of class(es).

The JBoss application server is, at its most simplest, a Java application. Your application that you deploy to JBoss EAP is running in the same memory space as JBoss EAP, and therefore you need to be aware of potential conflicts with class loading between your application and JBoss EAP.

## 2. JBOSS CLASSLOADING CONFIGURATION FILE

How do we go about customizing our classloading behavior? There is one critical file you will want to be aware of. This file is called `jboss-classloading.xml`. Where it is placed depends on what you are trying to apply your classloading policy against. We will cover the various locations for this file as well as the format of this very important file in this section.

### 2.1 JBoss ClassLoading Configuration File Format

The `jboss-classloading.xml` file is relatively straight forward. Here is an example with all possible attributes. An explanation of attributes follows.



```
<?xml version="1.0" encoding="UTF-8"?>
<classloading xmlns="urn:jboss:classloading:1.0"
    name="MySampleApp.war"
    domain="MySampleAppDomain"
    parent-domain="SomeParentDomain"
    top-level-classloader="true"
    parent-first="true"
    export-all="NON_EMPTY"
    import-all="true">
</classloading>
```

As promised, here is what the individual attributes mean in this file:

**Name** – the name of your deployment (war file, ear file, jar file). In this example, we are deploying MySampleApp.war, hence the name in this file.

**Domain** – this is the classloading domain. This value you can be arbitrary, but for simplicity and for troubleshooting, we recommend that this domain be named similar to your application. Be aware that if the domain you choose already exists, then your deployment will join that classloading domain.

**Parent-domain** – Use this to let the classloader know the domain that you want searched when a class is not found in your domain. This attribute is optional and by default it is the 'DefaultDomain' which is that of the container. A good use of this value would be if you have a war file and want it to look at your ear domain for resolution of missing class files.

**Top-level-classloader** – Values here either are true or false. This is used to enable embedded apps, like a war within an ear file, to have their classloader be a true-top level classloader in your domain as opposed to a child classloader with the top-level (usually DefaultDomain) classloader as parent. Set this to true if you want the classloader of your sub-deployment (think war file inside ear) to be a top level class-loader.

**Parent-first** – Values here are either true or false. This controls whether or not the parent classloader is loaded first. Setting this value to false means that this domain will be searched for all class definitions prior to the parent domain which is against j2ee compliance.

**Export-all** - The only value here is 'NON\_EMPTY'. This means that all classes are exposed to other applications.

**Import-all** – Values here are either true or false. Setting this value to true makes all classes exported from other applications visible to this domain / application. You may consider setting this to true when you have a few applications with different versions of libraries / components being used in the same JVM.



## 2.2 JBoss ClassLoading Configuration File Location

Where you place the classloading configuration depends on what you are trying to apply your classloading policy against. The three most common items you will want to control classloading behavior of are war files, ear files, and jar files.

- To control the classloading behavior of a war file, place the `jboss-classloading.xml` file in the `WEB-INF` directory of your war file. For example, if your war file is `myWarFile.war`, your classloading configuration file would be at `myWarFile.war/WEB-INF/jboss-classloading.xml`.
- To control the classloading behavior of an ear file, place the `jboss-classloading.xml` file in the `META-INF` directory of your ear file. For example, if your ear file is `myEarFile.ear`, your classloading configuration file would be at `myEarFile.ear/META-INF/jboss-classloading.xml`.
- To control the classloading behavior of a jar file, place the `jboss-classloading.xml` file in the `META-INF` directory of your jar file. For example, if your jar file is `myJarFile.jar`, your classloading configuration file would be at `myJarFile.jar/META-INF/jboss-classloading.xml`.

## 3. WAR CLASSLOADING

War files are slightly nicer / easier to work with because as part of the Servlet spec, war files are in their own classloader. This gives us some nice benefits such as classes in `WEB-INF/classes` and jars in `WEB-INF/lib` of our war file are not visible to other wars. In essence we are getting some nice war isolation right out of the box. Additionally, war files by default are deployed with a classloading scheme that looks up classes within the war first *before* looking for them in the parent domain. However, ear files by default look to the parent domain first. Be aware of this difference and the problems it can cause, even by simply moving a war file into an ear deployment.

When you have classes that need to be shared, say between an ejb and a war, then you will want to put those classes in either the `lib` directory of the ear (if you want to limit outside application access) or in `jboss/lib` or `jboss/deploy` if you want the classes more globally accessible.

## 4. EAR ISOLATION

Lets look at the following scenario – you have an ear file that you need to deploy. When you deploy it as is, JBoss starts up with errors because the ear file has its own set of libraries and dependencies that conflict with the JBoss runtime. Additionally, the ear file in this example has libraries that you simply don't want exposed to the rest of the environment – lets say a legacy string utility package. What can we do in this situation?

We will want to create a `jboss-classloading.xml` file in our ear file `META-INF` directory. In this file, we will define an new classloader for this ear file that will isolate the ear file from the container.

Our resulting classloading xml file will look as follows:



```
<?xml version="1.0" encoding="UTF-8"?>
  <classloading xmlns="urn:jboss:classloading:1.0"
    domain="IsolatedEARDeploymentDomainWithParent"
    parent-domain="DefaultDomain"
    parent-first="false"
    export-all="NON_EMPTY"
    import-all="true">
  </classloading>
```

*Illustration 1: Sample jboss-classloading.xml that shows ear isolation. This file must be called jboss-classloading.xml and be placed in your deployment META-INF directory*


There are two key points in this example. First is that we are placing this classloading configuration file in the ear deployment META-INF directory. Second we define a new unique domain that this classloader should reside in. Since this domain is different than our parent domain (in this case the container), this ear is properly isolated. Lastly you will notice that we are defining a parent-domain (which is fine because there are services we want from the container) but are defining the classloader so that the parent domain does not load first and take precedence.

**NOTE** – with 'parent-first' set to 'false', the app is no longer in j2ee compliance as the application domain is searched for classes prior to the container.

## 5. DEBUGGING

With all this information now, a natural question to ask is how can you verify what classloaders are being used. The easiest way to achieve this is by looking at the jmx-console under the jboss.classloader group. This is found in the left navigation under 'Object Name Filter.' There you will find a list of all classloaders used.





## JMX Agent View

jtsi.remote (0.0.0.0) - all

**Object Name Filter**

Remove Object Name Filter

- ▶ JMImplementation
- ▶ com.arjuna.ats.properties
- ▶ jboss
- ▶ jboss.admin
- ▶ jboss.alerts
- ▶ jboss.aop
- ▶ jboss.cache
- ▶ jboss.classloader
- ▶ jboss.deployment
- ▶ jboss.ejb
- ▶ jboss.ha
- ▶ jboss.i2ee
- ▶ jboss.jacc
- ▶ jboss.jca
- ▶ jboss.jdrac
- ▶ jboss.jgroups
- ▶ jboss.linx
- ▶ jboss.management.local
- ▶ jboss.messaging
- ▶ jboss.messaging.connectionfactory
- ▶ jboss.messaging.destination
- ▶ jboss.mq
- ▶ jboss.pojo
- ▶ jboss.remoting
- ▶ jboss.ssi

**jboss.classloader**

- domain="DefaultDomain",system=33212498
- domain="jboss.console.sar=console-mgr.sar",system=33212498
- domain="org.jboss.classloader-embedded",system=33212498
- domain="seam.jboss.org.loader=seam-bookinq",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/ROOT.war",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/httpaha-invoker.sar/invoker.war",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/jboss.war/jboss-management.war",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/jmx-console.war",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/juddi-service.sar/juddi.war",system=33212498
- domain="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/management/console-mgr.sar/web-console.war",system=33212498
- domain="vfszip:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/farm/jboss-seam-bookinq.ear/jboss-seam-bookinq.war",system=33212498
- id="aop-classloader:0.0.0\$MODULE"
- id="bootstrap-classloader:0.0.0\$MODULE"
- id="deployers-classloader:0.0.0\$MODULE"
- id="jmx-classloader:0.0.0\$MODULE"
- id="profile-classloader:0.0.0\$MODULE"
- id="security-classloader:0.0.0\$MODULE"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/conf"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/conf/bindingservice.beans"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/conf/jboss-service.xml"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/ROOT.war"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/\_jms\_Topic-service.xml"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy\_topic\_MyTopic-service.xml"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/admin-console.war"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/cache-invalidation-service.xml"
- id="vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/depnvr/cluster/denlow-ha/inleton-jboss-heapnc.xml"

Illustration 2: jboss.classloader listing under the jmx-console. This is accessible on your JBoss EAP instance at the /jmx-console app, i.e. http://localhost:8080/jmx-console

Clicking on any of these items gives us more information about those deployments and their classloader. For example, we could choose the one for the jmx-console war file and see the following:

Java Class	domain	[vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/jmx-console.war]		
Description	org.jboss.classloader.spi.ClassLoaderDomain	Management Bean.		
Attribute Name	Access	Type	Description	Attribute Value
Name	R	java.lang.String	MBean Attribute.	vfsfile:/home/jtsi/Downloads/eap5.1temp/jboss-eap-5.1/jboss-as/server/all/deploy/jmx-console.war/
ParentPolicyName	R	java.lang.String	MBean Attribute.	AFTER_BUT_JAVA_BEFORE
ParentDomainName	R	java.lang.String	MBean Attribute.	DefaultDomain
ResourceCacheSize	R	int	MBean Attribute.	0
ClassBlackListSize	R	int	MBean Attribute.	96
ClassCacheSize	R	int	MBean Attribute.	10
ParentDomain	R	javax.management.ObjectName	MBean Attribute.	jboss.classloader:system=33212498,domain="DefaultDomain" <a href="#">View MBean</a>
System	R	javax.management.ObjectName	MBean Attribute.	jboss.classloader:service=ClassLoaderSystem <a href="#">View MBean</a>
ResourceBlackListSize	R	int	MBean Attribute.	482

Operation	Return Type	Description	Parameters
findClassLoaderForClass	javax.management.ObjectName	MBean Operation.	<p>p1 java.lang.String (no description)</p> <p style="text-align: right;"><input type="button" value="Invoke"/></p>

Illustration 3: Drilling down on the classloader information for the jmx-console war file

Of interest here is the fact that we can see the policy being enforced on this class loader as well as there are some exposed methods that we can execute. This is a great place to start debugging by finding out which classloaders are loading your specific classes as well as the policies in place around those classloaders.

**Please Note** - The JMX console *will not* show you every class loaded. It will only show those loaded through JBoss. To see all classes loaded you could add the `-verbose:class` switch to your startup script and all the loaded classes will be output to the console.

```
if [ "x$JAVA_OPTS" = "x" ]; then
    JAVA_OPTS="-Xms1303m -Xmx1303m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Dsun.lang.ClassLoader.allowArraySyntax=true -verbose:class"
fi
## Specify the Security Manager options
#JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -Djava.security.policy=$POLICY"

# Sample JPDA settings for remote socket debugging
#JAVA_OPTS="$JAVA_OPTS -Xrunjdpw:transport=dt_socket,address=8787,server=y,suspend=n"

# Sample JPDA settings for shared memory debugging
#JAVA_OPTS="$JAVA_OPTS -Xrunjdpw:transport=dt_shmem,address=jboss,server=y,suspend=n"
```

*Illustration 4: Adding `verbose:class` to `run.conf` - located in the `bin` directory of your JBoss EAP install*

```

[Loaded java.text.MessageFormat$Field from /usr/lib/jvm/java-1.6.0-openjdk-1.6.0
.0/jre/lib/rt.jar]
run.sh: invalid option -- v
[Loaded java.util.ArrayList$Itr from /usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0/jre
/lib/rt.jar]
[Loaded java.util.IdentityHashMap$KeySet from /usr/lib/jvm/java-1.6.0-openjdk-1.
5.0.0/jre/lib/rt.jar]
[Loaded java.util.IdentityHashMap$IdentityHashMapIterator from /usr/lib/jvm/java
-1.6.0-openjdk-1.6.0.0/jre/lib/rt.jar]
[Loaded java.util.IdentityHashMap$KeyIterator from /usr/lib/jvm/java-1.6.0-openj
dk-1.6.0.0/jre/lib/rt.jar]
[Loaded java.io.DeleteOnExitHook from /usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0/jr
e/lib/rt.jar]
[Loaded java.util.LinkedHashSet from /usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0/jre
/lib/rt.jar]
[Loaded java.util.HashMap$KeySet from /usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0/jr
e/lib/rt.jar]
[Loaded java.util.LinkedHashMap$LinkedHashMapIterator from /usr/lib/jvm/java-1.6.0-
openjdk-1.6.0.0/jre/lib/rt.jar]
[Loaded java.util.LinkedHashMap$KeyIterator from /usr/lib/jvm/java-1.6.0-openjdk
-1.6.0.0/jre/lib/rt.jar]

```

*Illustration 5: Sample Output from console when verbose:class is applied*

## 6. COMMON EXCEPTIONS

Here are some common exceptions that you may see in your log files that could be caused by a class loading issue.

### **ClassNotFoundException**

This exception is normally caused by a class not being in the class path. It can also be caused by a class or jar file missing from the application deployment altogether. For example, the jar file might not be listed in META-INF / MANIFEST.MF Classpath and also is not in the lib directory in the ear. A final scenario where this can happen is when ear isolation is done (more information on ear isolation in section 4). If the ear is isolated (using META-INF / jboss-app.xml) then the jar files and classes are only available for the particular ear file they are in. If those jar files or classes are needed outside that ear, you would see this ear. To resolve this issue, the second ear would need to specify the same loader repository name in its jboss-app.xml.

### **ClassCastException**

ClassCastExceptions, when they aren't simply coding issues, usually occur when there are multiple versions of a class available to the classloader. You will see this exception when the developer is trying to package a different version of a jar file or class in their application that has been isolated using jboss-app.xml. You will see this, for example, around xml parsers. With XML parsers, JBoss EAP uses its own version of common



xml parsers and therefore the application that is deployed gets a reference and cast to a class that isn't what it expects, resulting in a ClassCastException.

### **NoSuchMethodException**

This exception usually indicates a class version mismatch between jar files. This means that one class is calling a method on another class that does not exist in the version that is currently loaded. This most frequently occurs when customers package different versions of common libraries or container provided libraries in their application, for example hibernate or jboss-cache. This can also happen if you are migrating between versions of JBoss EAP and do not recompile your application against the updated EAP jars.

### **RESOURCES**

Permanent Generation, Heap Space - [http://blogs.sun.com/jonthecollector/entry/presenting\\_the\\_permanent\\_generation](http://blogs.sun.com/jonthecollector/entry/presenting_the_permanent_generation)

JBoss ClassLoader Glossary - <http://community.jboss.org/wiki/JBossClassLoaderGlossary>

JBoss ClassLoader Documentation - [http://docs.redhat.com/docs/en-US/JBoss\\_Enterprise\\_Application\\_Platform/5/html/JBoss\\_Microcontainer\\_User\\_Guide/sect-JBoss\\_Microcontainer\\_User\\_Guide-The\\_ClassLoading\\_Layer-ClassLoading.html](http://docs.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/5/html/JBoss_Microcontainer_User_Guide/sect-JBoss_Microcontainer_User_Guide-The_ClassLoading_Layer-ClassLoading.html)

Detailed information on JBoss ClassLoading - <http://java.dzone.com/articles/jboss-microcontainer-classloading>

### **Questions/Comments/Issues**

If you have questions or comments about this whitepaper, please enter them in the Red Hat customer portal for this specific whitepaper: <https://access.redhat.com/knowledge/techbriefs> . If you have a technical issue following this whitepaper please open a support case: <https://access.redhat.com/support/cases/new>