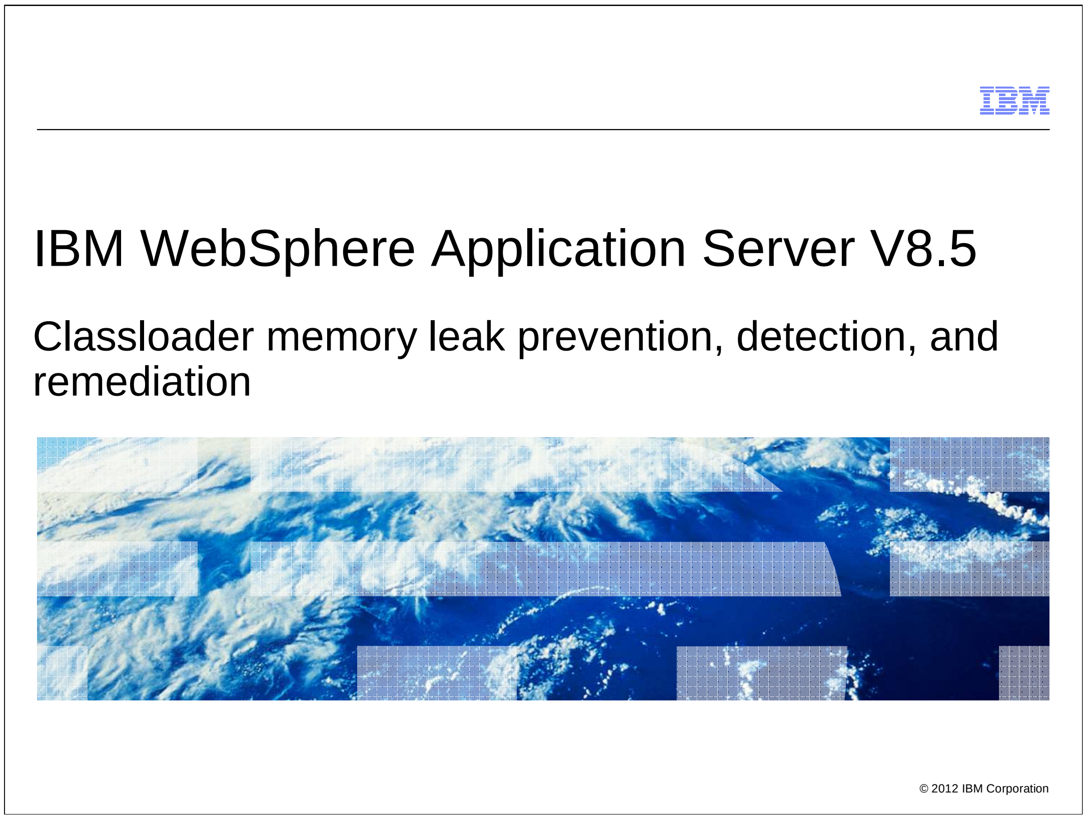


IBM WebSphere Application Server V8.5

Classloader memory leak prevention, detection, and remediation

© 2012 IBM Corporation

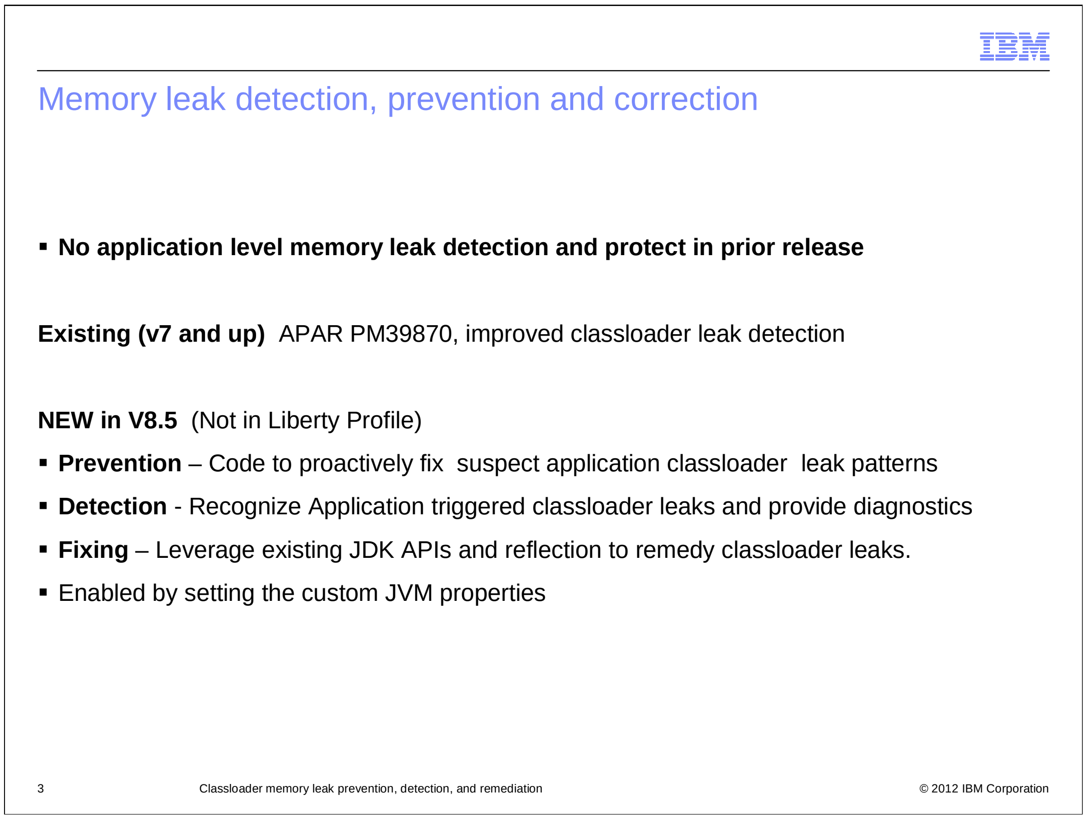
This presentation describes support for Memory leak detection, prevention and remediation included in IBM WebSphere® Application Server V8.5

Section

Overview

2 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

WebSphere Application Server Version 8.5 provides a top down pattern-based memory leak detection, prevention, and action by watching for suspect patterns in application code at run time. WebSphere Application Server provides some means of protection against memory leaks when stopping or redeploying applications. If leak detection, prevention and action are enabled, WebSphere Application Server monitors application and module activity. It performs diagnostic actions to detect and fix leaks when an application or an individual module stops. This feature helps increase application up time with frequent application redeployments without cycling the server. This feature only applies to the full WebSphere Application Server profile and NOT to the Liberty profile.



Memory leak detection, prevention and correction

- **No application level memory leak detection and protect in prior release**

Existing (v7 and up) APAR PM39870, improved classloader leak detection

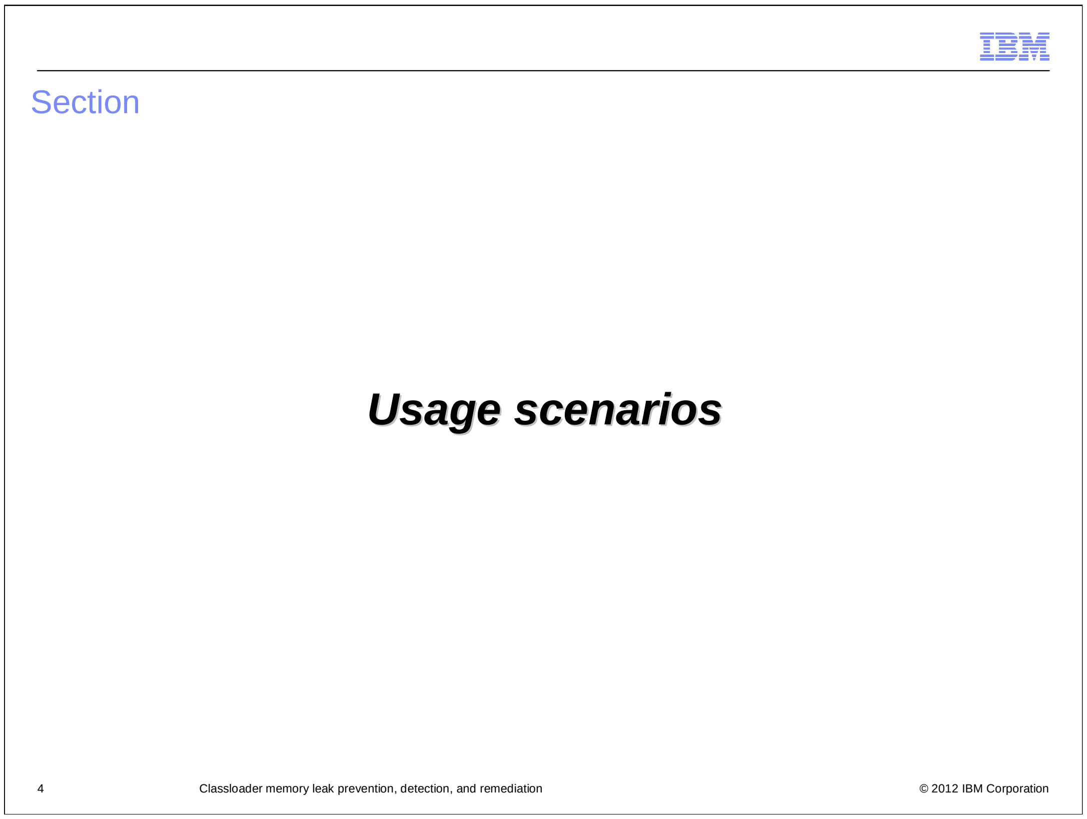
NEW in V8.5 (Not in Liberty Profile)

- **Prevention** – Code to proactively fix suspect application classloader leak patterns
- **Detection** - Recognize Application triggered classloader leaks and provide diagnostics
- **Fixing** – Leverage existing JDK APIs and reflection to remedy classloader leaks.
- Enabled by setting the custom JVM properties

3 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

Before WebSphere Application Server V8.5, customers discovered classloader and threadlocal leaks in the application server environment and in their applications. The application server did not provide application level memory leak detection and protection. Frequent application deployments result in OOM errors.

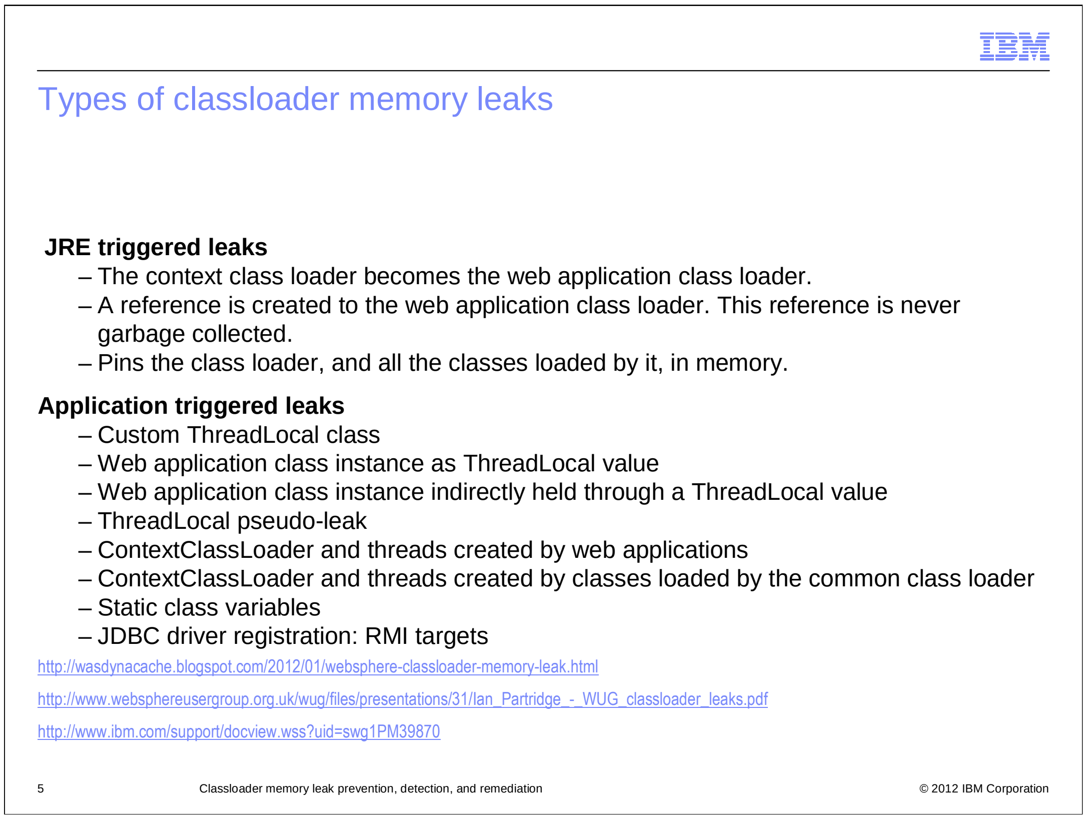
There are three parts to the memory leak feature in WebSphere Application Server V8.5: detection, prevention, and action. Detection issues warnings when a memory leak is detected through a combination of standard API calls and some reflection tricks when a web application is stopped, un-deployed or reloaded. WebSphere Application Server checks for known causes of memory leaks and issues warnings when an application leak is detected. Prevention is on by default and applies only to JRE triggered leaks. JRE triggered leaks are prevented by initializing singletons at server startup when the application server class loader is the context class loader. Action takes proactive action to fix memory leaks. These actions have reasonable defaults and are configured on a case-by-case basis.


Section

Usage scenarios

4 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

Following are usage scenarios for the classloader leak detection feature of WebSphere Application Server V8.5.



Types of classloader memory leaks

JRE triggered leaks
- The context class loader becomes the web application class loader.
- A reference is created to the web application class loader. This reference is never garbage collected.
- Pins the class loader, and all the classes loaded by it, in memory.

Application triggered leaks
- Custom ThreadLocal class
- Web application class instance as ThreadLocal value
- Web application class instance indirectly held through a ThreadLocal value
- ThreadLocal pseudo-leak
- ContextClassLoader and threads created by web applications
- ContextClassLoader and threads created by classes loaded by the common class loader
- Static class variables
- JDBC driver registration: RMI targets

http://wasdynacache.blogspot.com/2012/01/websphere-classloader-memory-leak.html

http://www.websphereusergroup.org.uk/wug/files/presentations/31/Ian_Partridge_-_WUG_classloader_leaks.pdf

http://www.ibm.com/support/docview.wss?uid=swg1PM39870

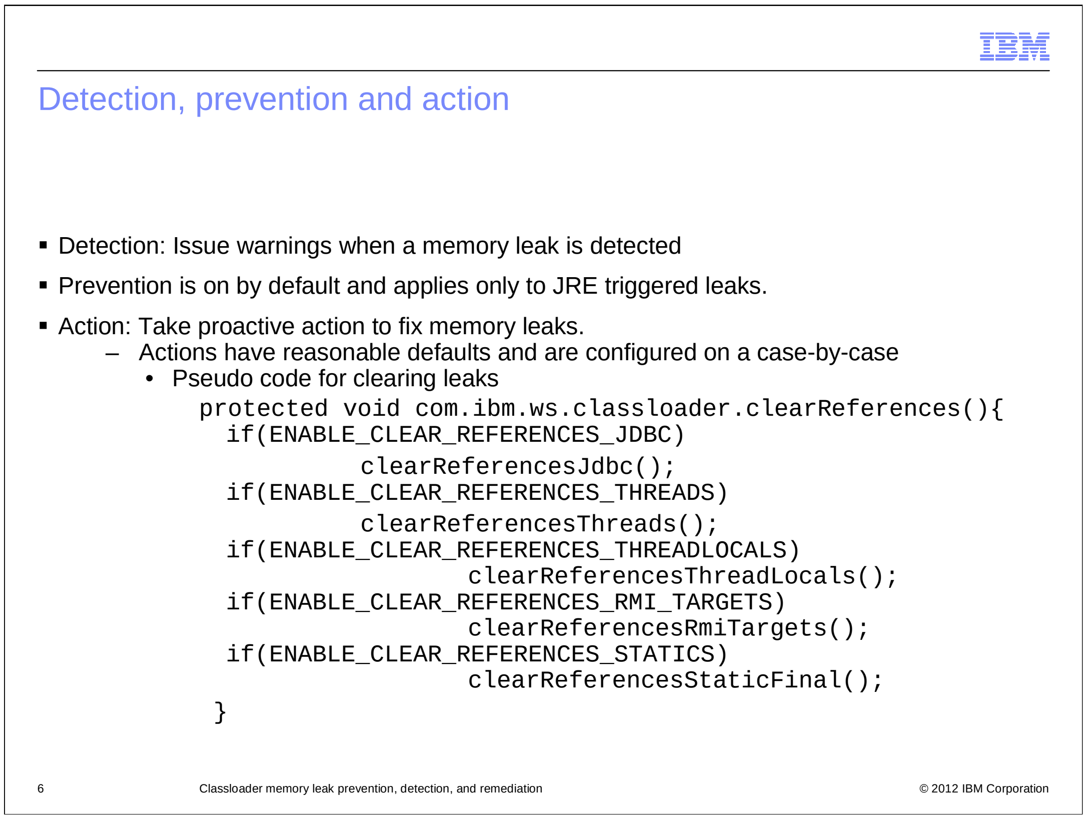
5 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

Many memory leaks manifest themselves as class loader leaks. A Java class is uniquely identified by its name and the class loader that loaded it. Classes with the same name can be loaded multiple times in a single JVM, each in a different class loader. Each web application gets its own class loader and this is what WebSphere Application Server uses for isolating applications.

An object retains a reference to the class it is an instance of. A class retains a reference to the class loader that loaded it. The class loader retains a reference to every class it loaded. Retaining a reference to a single object from a web application causes every class loaded by the web application to be retained in memory. These references often remain after a web application reload. With each reload, more classes are retained in memory which leads to an out of memory error.

Class loader memory leaks are normally caused by the application code or JRE triggered code. JRE triggered leaks are memory leaks that occur when the Java Runtime Environment (JRE) code uses the context class loader to load an application singleton. These singletons can be threads or other objects that are loaded by the JRE using the context class loader. If the web application code triggers the initialization of a singleton or a static initializer, these conditions apply: One, the context class loader becomes the web application class loader and two, a reference is created to the web application class loader. This reference is never garbage collected. As a result the class loader is retained, and all the classes loaded by it, in memory.

Application triggered leaks are categorized as custom ThreadLocal class, web application class instance as threadLocal value, web application class instance indirectly held through a ThreadLocal value, ThreadLocal pseudo-leak, ContextClassLoader and threads created by web applications, ContextClassLoader and threads created by classes loaded by the common class loader, Static class variables, and JDBC driver registration (RMI targets). For more information on application triggered links, see the links on the slide.



Detection, prevention and action

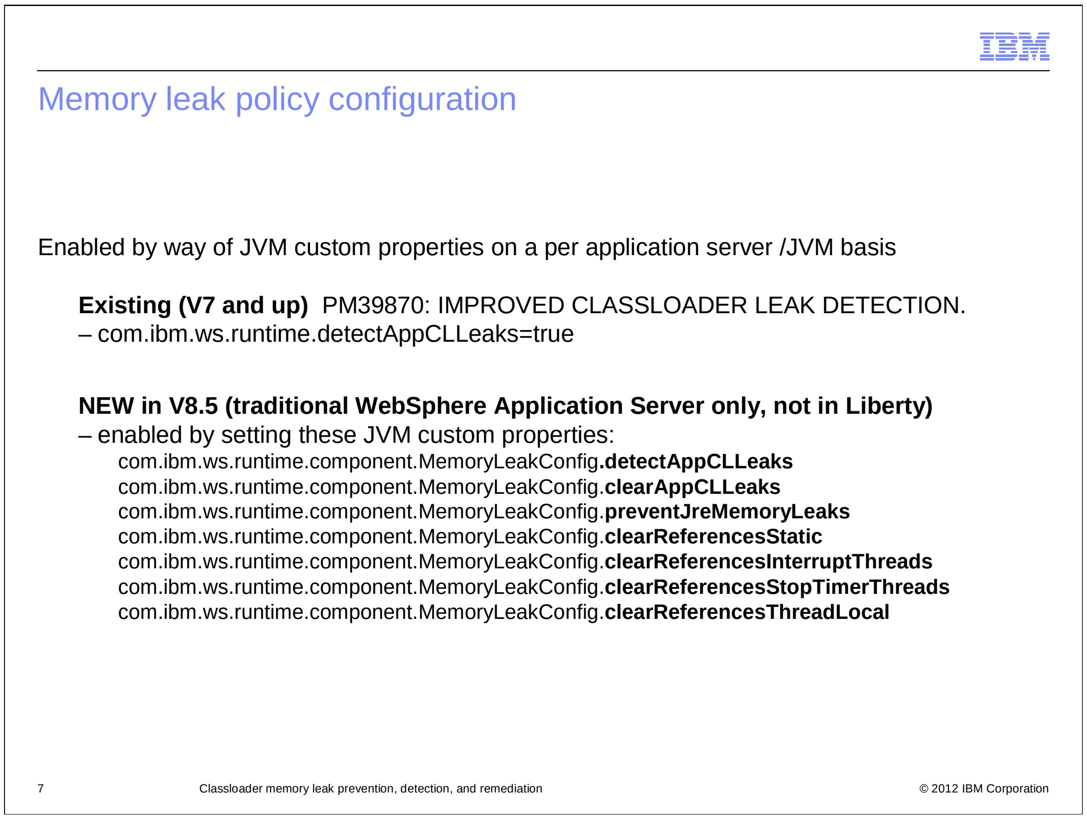
- Detection: Issue warnings when a memory leak is detected
- Prevention is on by default and applies only to JRE triggered leaks.
- Action: Take proactive action to fix memory leaks.
  - Actions have reasonable defaults and are configured on a case-by-case
    - Pseudo code for clearing leaks

```
protected void com.ibm.ws.classloader.clearReferences(){
  if(ENABLE_CLEAR_REFERENCES_JDBC)
           clearReferencesJdbc();
  if(ENABLE_CLEAR_REFERENCES_THREADS)
           clearReferencesThreads();
  if(ENABLE_CLEAR_REFERENCES_THREADLOCALS)
                   clearReferencesThreadLocals();
  if(ENABLE_CLEAR_REFERENCES_RMI_TARGETS)
                   clearReferencesRmiTargets();
  if(ENABLE_CLEAR_REFERENCES_STATICS)
                   clearReferencesStaticFinal();
 }
```

6 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

Through a combination of standard API calls and some reflection tricks, when a web application is stopped, undeployed or reloaded a memory leak is detected.

Prevention is on by default and applies only to JRE triggered leaks. JRE triggered leaks are prevented by initializing singletons at server startup when the application server class loader is the context class loader. Proactive actions to fix memory leaks have reasonable defaults and are configured on a case-by-case basis.



Memory leak policy configuration

Enabled by way of JVM custom properties on a per application server /JVM basis

Existing (V7 and up) PM39870: IMPROVED CLASSLOADER LEAK DETECTION.
– com.ibm.ws.runtime.detectAppCLLeaks=true

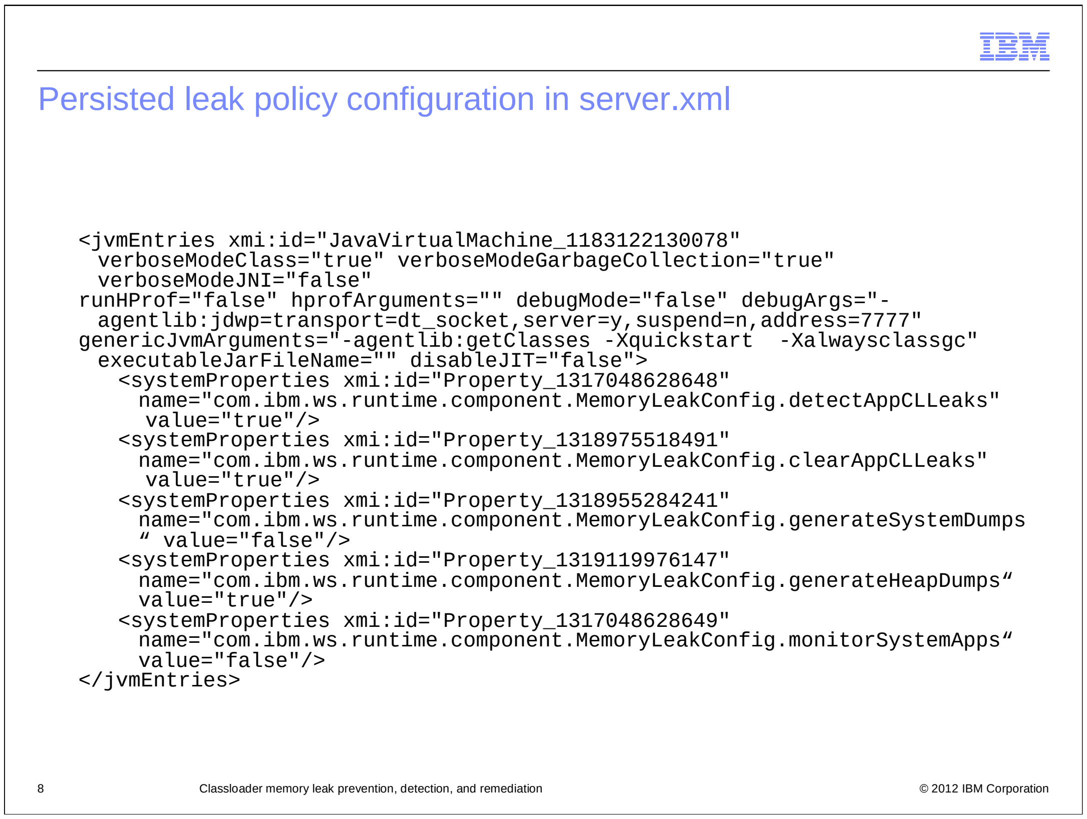
NEW in V8.5 (traditional WebSphere Application Server only, not in Liberty)
– enabled by setting these JVM custom properties:

com.ibm.ws.runtime.component.MemoryLeakConfig**.detectAppCLLeaks**
com.ibm.ws.runtime.component.MemoryLeakConfig.**clearAppCLLeaks**
com.ibm.ws.runtime.component.MemoryLeakConfig.**preventJreMemoryLeaks**
com.ibm.ws.runtime.component.MemoryLeakConfig.**clearReferencesStatic**
com.ibm.ws.runtime.component.MemoryLeakConfig.**clearReferencesInterruptThreads**
com.ibm.ws.runtime.component.MemoryLeakConfig.**clearReferencesStopTimerThreads**
com.ibm.ws.runtime.component.MemoryLeakConfig.**clearReferencesThreadLocal**

7 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

The MemoryLeak service and its mbean are active only in an application server that hosts applications and services requests. This service is not active on a Deployment Manager, node agent, administrative agent, or other server types like WebSphere proxy server.

The leak detection option is disabled by default. You can use Java virtual machine (JVM) custom properties to adjust the leak policy values such as, enable and disable leak detection, action, and prevention. These custom properties are only applicable to a stand-alone server or managed application server and not to a node agent, administrative agent, job manager, or deployment manager. When the application or the server is shutting down, WebSphere Application Server determines the classloaders that have references to associated loaded classes and objects. If a classloader leak is detected a heapdump or systemdump is taken. All persistent configurations of this service are completed using JVM custom properties. At runtime, use the MemoryLeakConfig and MemoryLeakAdmin mbeans for configuration and administration. The configuration changes are not persisted until JVM custom properties are configured.

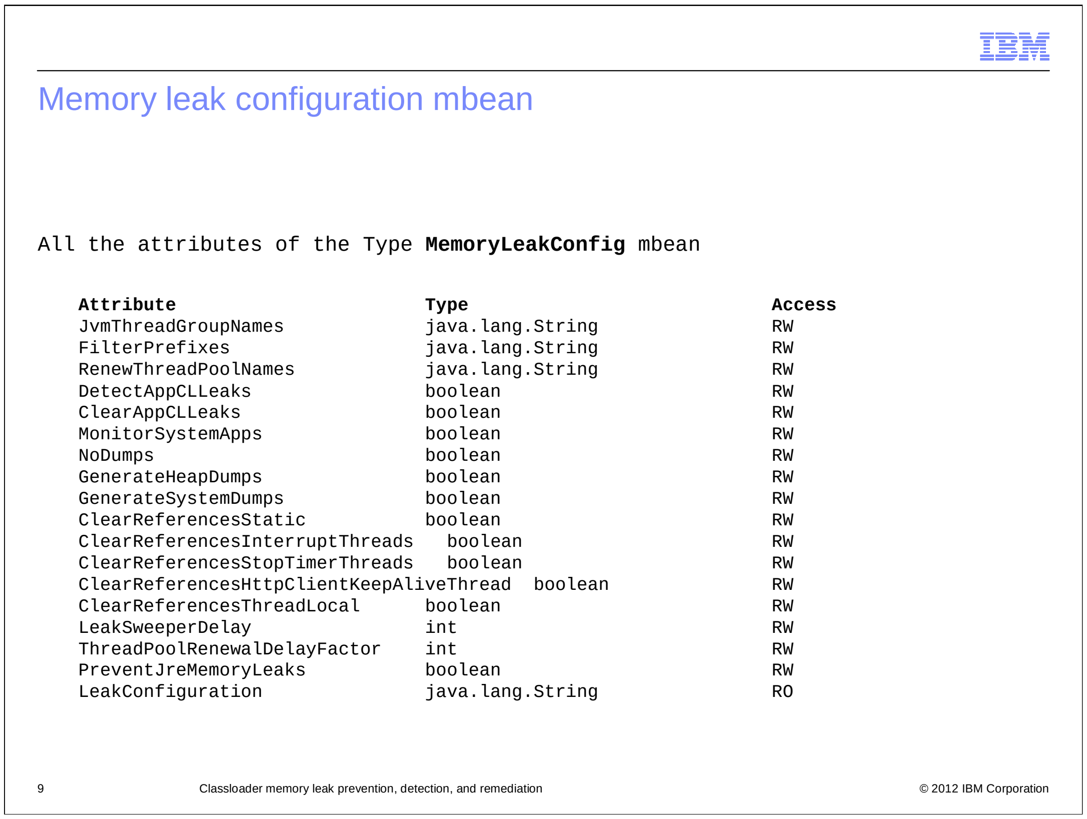


Persisted leak policy configuration in server.xml

```
<jvmEntries xmi:id="JavaVirtualMachine_1183122130078"
  verboseModeClass="true" verboseModeGarbageCollection="true"
  verboseModeJNI="false"
runHProf="false" hprofArguments="" debugMode="false" debugArgs="-
  agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777"
genericJvmArguments="-agentlib:getClasses -Xquickstart  -Xalwaysclassgc"
  executableJarFileName="" disableJIT="false">
    <systemProperties xmi:id="Property_1317048628648"
      name="com.ibm.ws.runtime.component.MemoryLeakConfig.detectAppCLLeaks"
       value="true"/>
    <systemProperties xmi:id="Property_1318975518491"
      name="com.ibm.ws.runtime.component.MemoryLeakConfig.clearAppCLLeaks"
       value="true"/>
    <systemProperties xmi:id="Property_1318955284241"
      name="com.ibm.ws.runtime.component.MemoryLeakConfig.generateSystemDumps
      " value="false"/>
    <systemProperties xmi:id="Property_1319119976147"
      name="com.ibm.ws.runtime.component.MemoryLeakConfig.generateHeapDumps"
      value="true"/>
    <systemProperties xmi:id="Property_1317048628649"
      name="com.ibm.ws.runtime.component.MemoryLeakConfig.monitorSystemApps"
      value="false"/>
</jvmEntries>
```

8 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

These JVM custom properties are persisted in the WebSphere Application Server configuration model in the server.xml file. The example here is of a persisted leak policy configuration in the server_home/config/cells/nodes/servers/server.xml file of an unmanaged server:



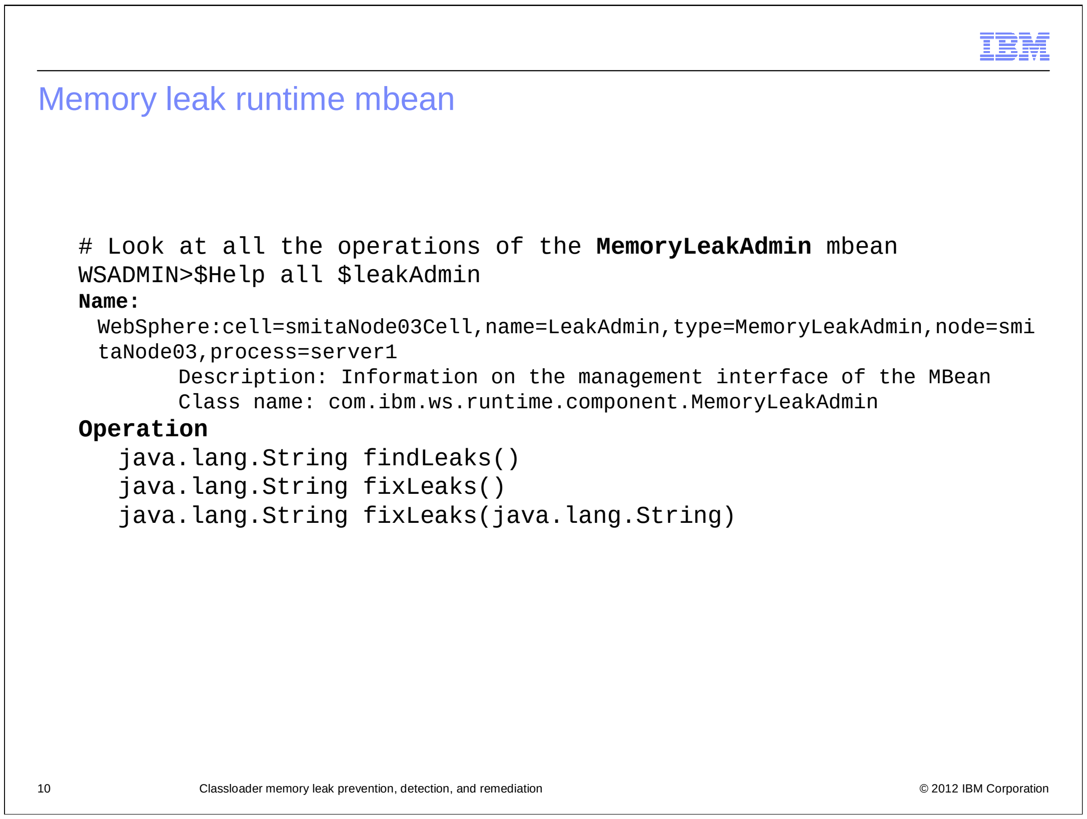
Memory leak configuration mbean

All the attributes of the Type **MemoryLeakConfig** mbean

Attribute	**Type**	**Access**
JvmThreadGroupNames	java.lang.String	RW
FilterPrefixes	java.lang.String	RW
RenewThreadPoolNames	java.lang.String	RW
DetectAppCLLeaks	boolean	RW
ClearAppCLLeaks	boolean	RW
MonitorSystemApps	boolean	RW
NoDumps	boolean	RW
GenerateHeapDumps	boolean	RW
GenerateSystemDumps	boolean	RW
ClearReferencesStatic	boolean	RW
ClearReferencesInterruptThreads	boolean	RW
ClearReferencesStopTimerThreads	boolean	RW
ClearReferencesHttpClientKeepAliveThread	boolean	RW
ClearReferencesThreadLocal	boolean	RW
LeakSweeperDelay	int	RW
ThreadPoolRenewalDelayFactor	int	RW
PreventJreMemoryLeaks	boolean	RW
LeakConfiguration	java.lang.String	RO

9 Classloader memory leak prevention, detection, and remediation

At runtime the memory leak detection, prevention and policy configuration can be changed using the MemoryLeakConfig mbean. Administration of the memory leak policy can be carried out using the MemoryLeakAdmin mbean. The leak policy affects how the application server responds to a classloader memory leak when an application or server is stopped. You can adjust the memory leak policy settings by using the WSADMIN scripting interface. These changes take effect immediately. However, they are not persisted to the server configuration and are lost when the server is restarted.

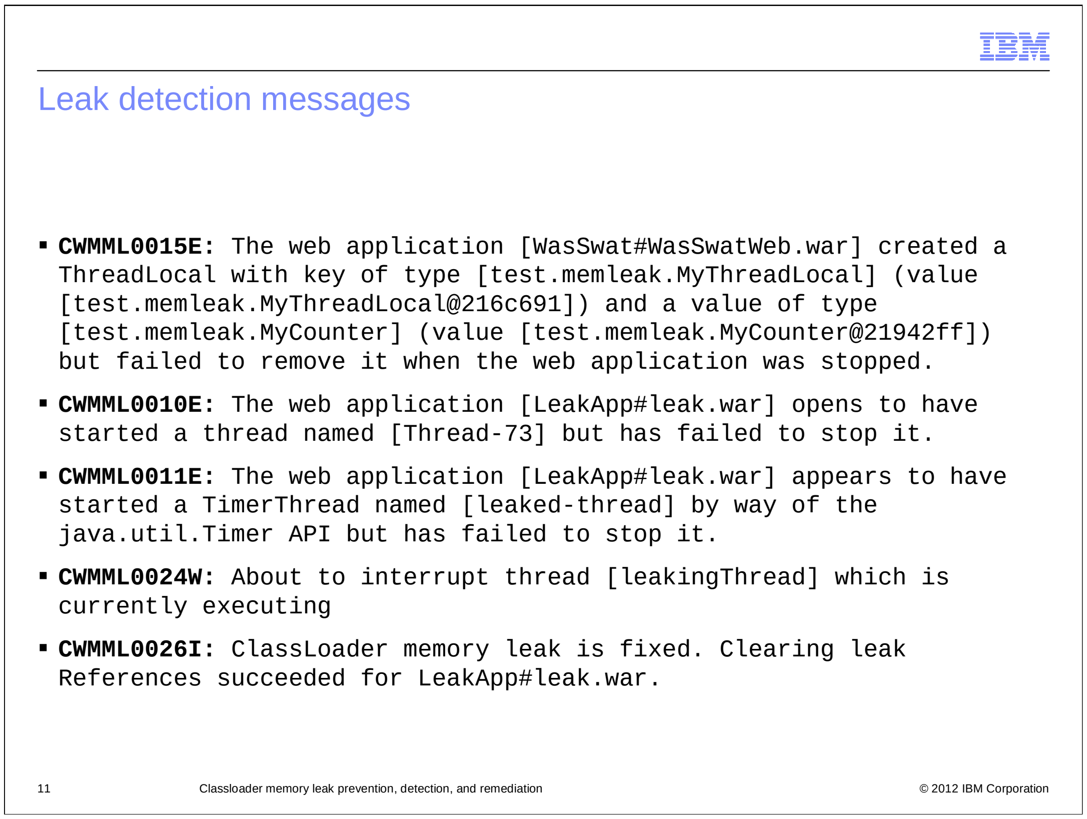


Memory leak runtime mbean

```
# Look at all the operations of the MemoryLeakAdmin mbean
WSADMIN>$Help all $leakAdmin
Name:
  WebSphere:cell=smitaNode03Cell,name=LeakAdmin,type=MemoryLeakAdmin,node=smi
  taNode03,process=server1
        Description: Information on the management interface of the MBean
        Class name: com.ibm.ws.runtime.component.MemoryLeakAdmin
Operation
   java.lang.String findLeaks()
   java.lang.String fixLeaks()
   java.lang.String fixLeaks(java.lang.String)
```

10 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

The MemoryLeakAdmin Mbean is available to you that provides operations to find and fix leaks at runtime. The MemoryLeakAdmin mbean also provides multiple operations to find and fix leaks for applications that have been stopped.



Leak detection messages

- **CWMML0015E:** The web application [WasSwat#WasSwatWeb.war] created a ThreadLocal with key of type [test.memleak.MyThreadLocal] (value [test.memleak.MyThreadLocal@216c691]) and a value of type [test.memleak.MyCounter] (value [test.memleak.MyCounter@21942ff]) but failed to remove it when the web application was stopped.
- **CWMML0010E:** The web application [LeakApp#leak.war] opens to have started a thread named [Thread-73] but has failed to stop it.
- **CWMML0011E:** The web application [LeakApp#leak.war] appears to have started a TimerThread named [leaked-thread] by way of the java.util.Timer API but has failed to stop it.
- **CWMML0024W:** About to interrupt thread [leakingThread] which is currently executing
- **CWMML0026I:** ClassLoader memory leak is fixed. Clearing leak References succeeded for LeakApp#leak.war.

11 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

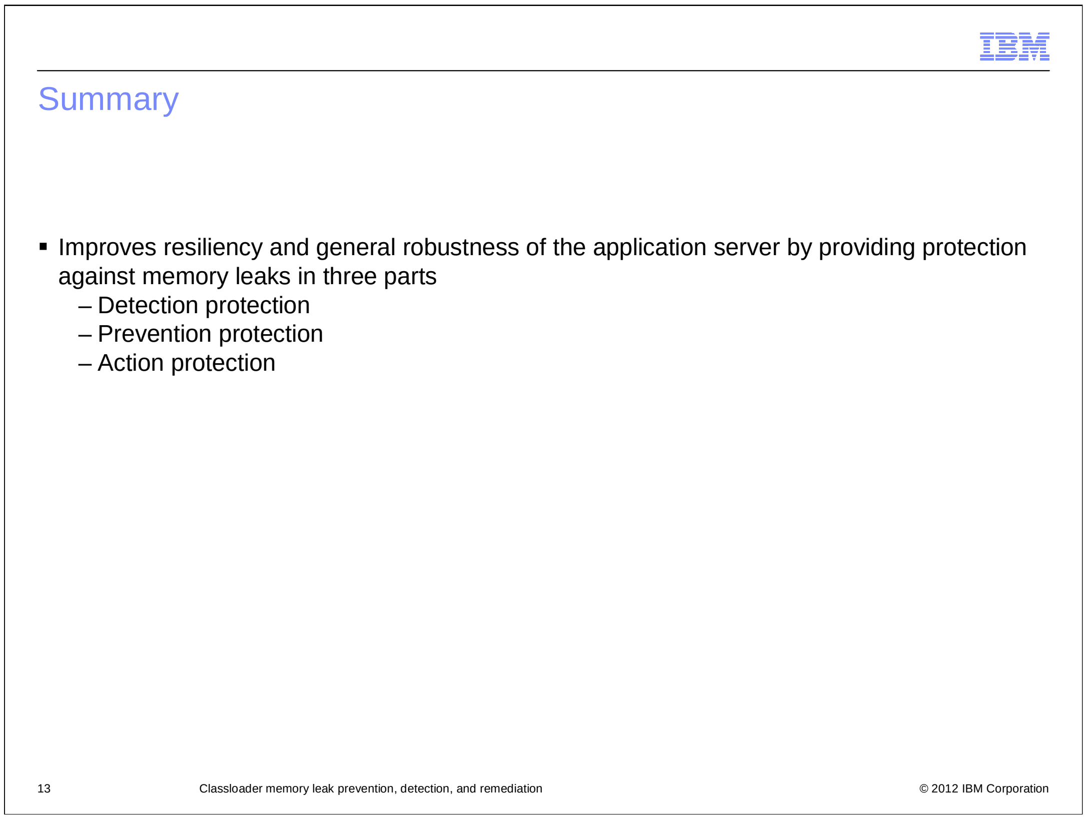
These are samples of the different memory leak warning messages that output when a memory leak is detected. The messages describe the type of memory leak and its root cause. These messages provide guidance on how to fix leaks in the source code.

Section

Summary

12 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

To summarize,

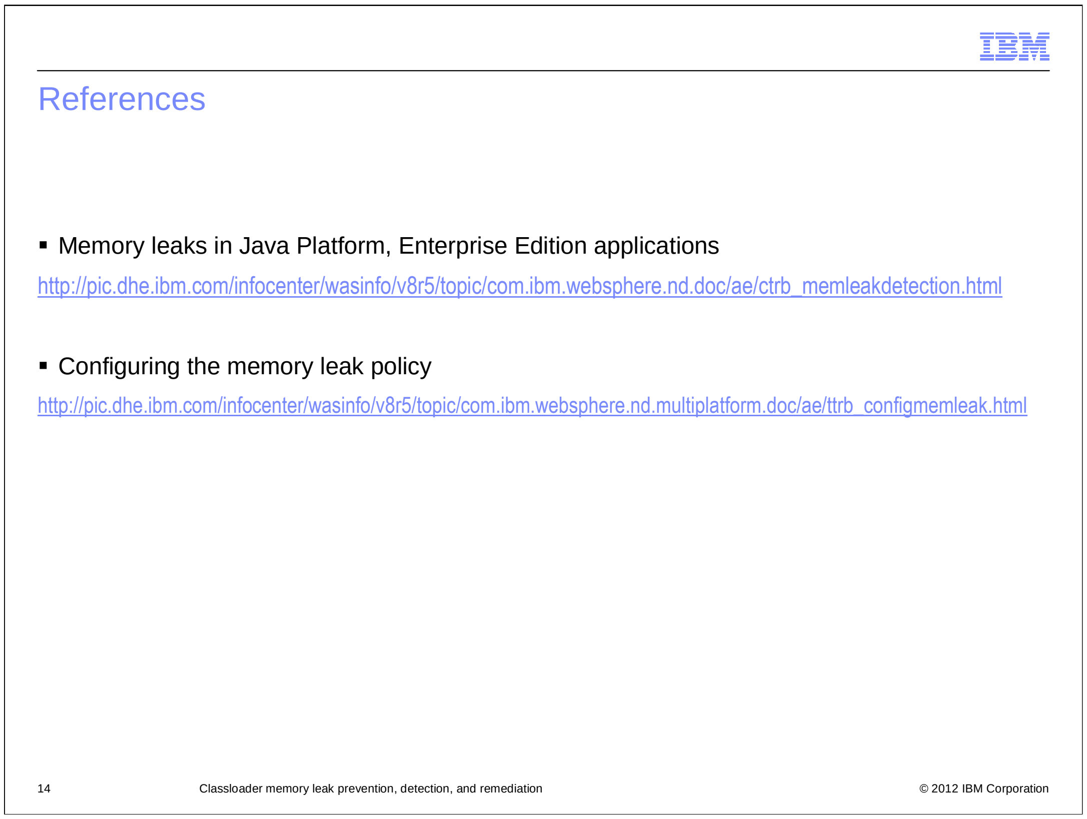


Summary

- Improves resiliency and general robustness of the application server by providing protection against memory leaks in three parts
  - Detection protection
  - Prevention protection
  - Action protection

13 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

The classloader leak detection feature of WebSphere Application Server V8.5 provides protection against memory leaks in three parts: detection, prevention, and action. Detection issues warnings when a memory leak is detected. Prevention protects from JRE triggered leaks by initializing singletons at server startup when the application server class loader is the context class loader. Action takes proactive action to fix memory leaks by way of reasonably configured defaults on a case-by-case basis.



References

- Memory leaks in Java Platform, Enterprise Edition applications

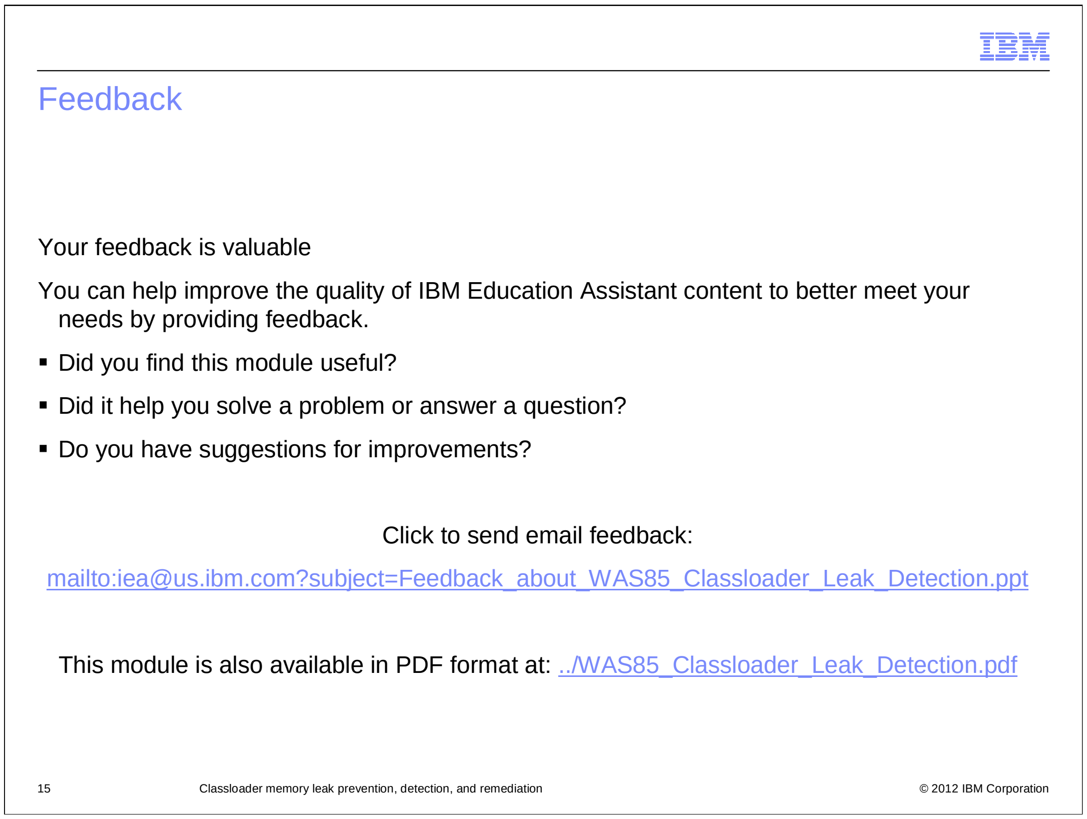
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/ctrb_memleakdetection.html

- Configuring the memory leak policy

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_configmemleak.html

14 Classloader memory leak prevention, detection, and remediation © 2012 IBM Corporation

See these references for additional information.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

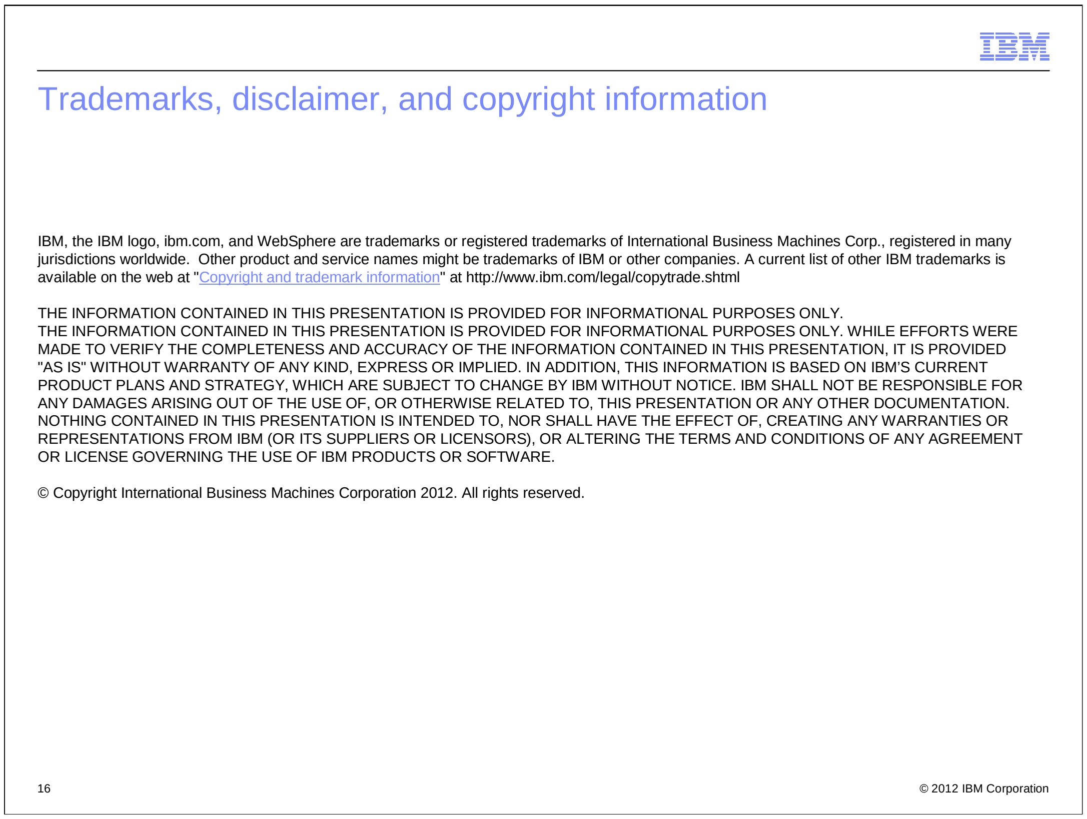
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WAS85_Classloader_Leak_Detection.ppt

This module is also available in PDF format at: ../WAS85_Classloader_Leak_Detection.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information