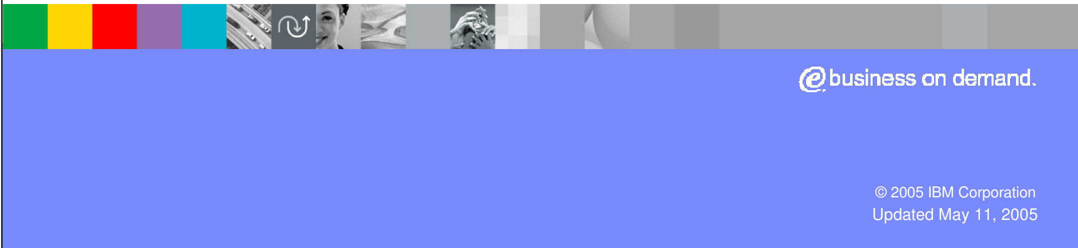
IBM Software Group

IBM® WebSphere® Application Server V6

Application Server Java™ Class Loader Details



© 2005 IBM Corporation

Updated May 11, 2005

This presentation will focus on the details of the Application Server Java Class Loader.

Goals

- Understand the details of the class loaders and different class loading options in WebSphere Application Server V6.
- Other presentations cover examples, Problem Determination, and Dynamic Reload and Preload classes
- Pre-requisite:
  - Class loader - Overview

© 2005 IBM Corporation

The goal of this presentation is to help you understand the details of the class loaders and different class loading options in WebSphere Application Server V6. An understanding of the Class loader from the overview presentation will be helpful in understanding these details.

Agenda

- Class loader hierarchy - recap
- Class loader details and options
- Shared Libraries
- Loading Native Libraries
- Advanced Class loader topics
- Preloading of classes
- Summary and References

© 2005 IBM Corporation

The agenda for this presentation is to cover:

Class loader hierarchy

Class loader details and options

Shared libraries

Native libraries

Advanced class loader topics

Preloading of classes

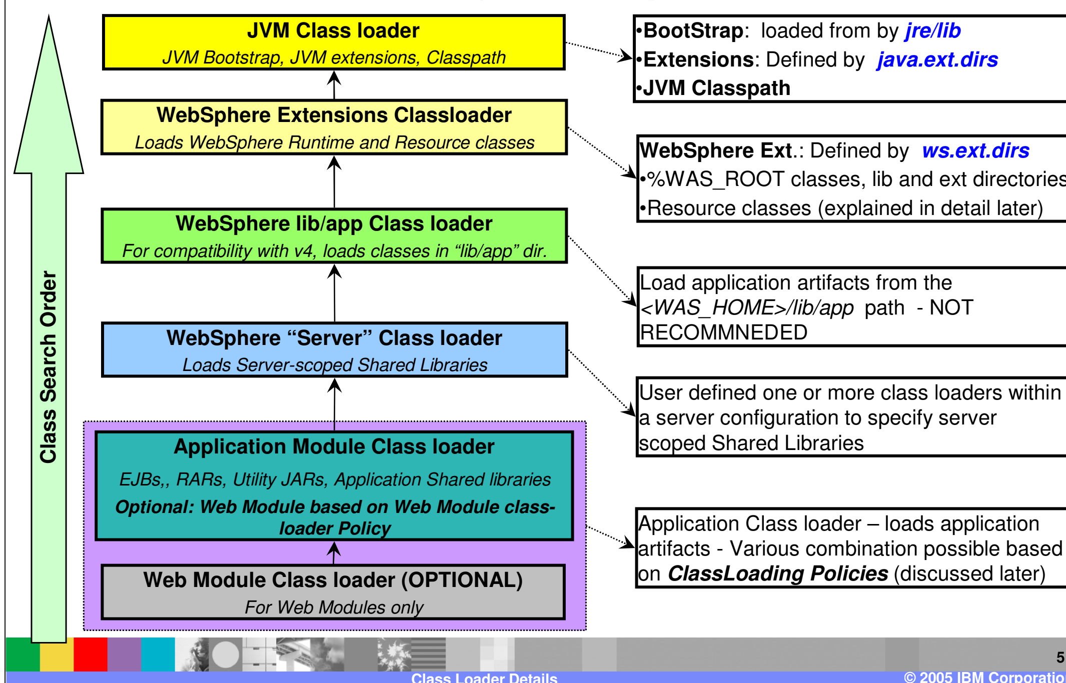
Summary and references

Section

Class Loader Hierarchy

© 2005 IBM Corporation

This section will cover class loader hierarchy.



Class loaders are organized in a hierarchy. This means that a child class loader can delegate class finding and loading to its parent, should it fail to load a class.

This flow was discussed in detail in the overview presentation.

Most of this presentation will discuss the details of each of these class loader hierarchies, along with the class loader options and policies.

IBM

Section

Class Loader Details

© 2005 IBM Corporation

This section will discuss the details of the class loader.

JVM Class loader

JVM Class loader
JVM Bootstrap, JVM extensions, Classpath

Not Recommended to add to the JVM classpath – there are better ways to add classes

Class loader	Directories/Files loaded	Delegation
JVM Bootstrap Libraries	***<JAVA_HOME>/jre/lib***	Not applicable
JVM Extension libraries	▪***JAVA_HOME>/jre/ext/lib*** ▪Any directory specified by the **java.ext.dirs** system property	PARENT_FIRST
JVM System Class loader	▪**CLASSPATH** environment variable. ▪Command-line options "**–classpath**" or "**–cp**" ▪If a JAR file on the class path has a manifest with the Class-Path attribute, JAR files specified by the Class-Path attribute will also be searched	PARENT_FIRST

Life cycle of classes: Exists for the duration of the server

- Additional Classpath and Bootclasspath can be configured using administrative console or wsadmin
  - Using Administrative Console: Server ➔ Application Server ➔ <Server> ➔ Java and Process Management ➔ Process Definition ➔ Java Virtual Machine

© 2005 IBM Corporation

At the root of the class loader hierarchy is the JVM class loader. This class loader loads the JVM Boot strap libraries, JVM extension libraries and the JVM System class loader in that order. In this class loader, there is no choice of delegation or search mode.

These classes get unloaded when the Application server is stopped.

Although it is possible to add your own classes to the JVM class path for classes that are used across multiple applications, it is not recommended that you do so. There are better solutions, such as using Shared libraries, which will be discussed later in this presentation.

IBM Software Group

IBM

WebSphere Extension Class loader

WebSphere Extensions Class loader
Loads WebSphere Runtime and Resource provider classes

Class loaders	Directories/Files loaded	Delegation
Java tools directory	<JAVA_HOME>\lib	Not applicable
Runtime Patches (RP) directory	<WAS_HOME>\classes ▪Intended for temporary fixes that are applied to the application server runtime ▪These patches override identical classes and resources that may appear in the RL and RE directories	PARENT_FIRST
Runtime Library (RL) directory	<WAS_HOME>\lib Contains the core application server runtime files	PARENT_FIRST
Runtime Extensions (RE) directory	<WAS_HOME>\lib\ext Contains extensions to the core application server runtime – Not used currently	PARENT_FIRST
Runtime appended class paths	Resource classes (like JDBC) and native paths and Standalone Resource adapters, specified in the server configuration	PARENT_FIRST

Life cycle of classes: Exists for the duration of the server

8

Class Loader Details

© 2005 IBM Corporation

Next in the class loader hierarchy is the WebSphere Extension class loader, whose primary responsibility is to load the WebSphere classes. Within the WebSphere extension class loader, there is a hierarchy of class loaders that are used, as shown in the table. In addition to this, the runtime appends certain class paths, including resource classes, native paths, and Custom Service classes.

WebSphere lib/app Class loader

WebSphere lib/app Class loader
For compatibility with v4, loads classes in "lib/app" dir.

- Used for backward compatibility for v4 that was used to load application artifacts that are common across all applications
- Load classes from <WAS_HOME>/lib/app directory
- Delegation: PARENT_LAST
- Life cycle of classes: Exists for the duration of the server

Not Recommended - There are better solutions than using this class loader

Better Option: Use Shared Library to load application artifacts that provides application level isolation, if needed

© 2005 IBM Corporation

Use of the WebSphere lib/app class loader is not recommended. It is supported in V6 only for backward compatibility. It was used in v4 as a means to place classes that are shared by all the applications running in the server. For V5 and V6, use of Shared libraries is a better option.

WebSphere “Server” Class loaders

- Used to create server-scoped shared libraries
  - Shared libraries that need to be available for all the applications running on that application server
  - You do not have to associate the shared library per application
- User can define one or more “server” class loaders within the server configuration
  - Order in which these are multiple “Server” class loaders are as they appear in the Administrative Console
  - Each Server class loader is linked as the immediate child of the “Server” class loader previously created server class loader
- Delegation Mode: PARENT_FIRST (default) or PARENT_LAST – Configurable
- Life cycle of classes: Exists for the duration of the server
- Configuration: Using the Administrative Console or wsadmin

© 2005 IBM Corporation

The WebSphere **Server** class loader is used when creating server scoped shared libraries. The concept of a shared library is explained later in the presentation. Briefly, shared libraries is a mechanism to specify class libraries and native libraries that are needed by one or more applications in a server.

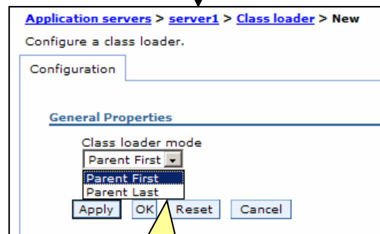
Once the shared library is defined, you can then associate one or more applications with the Shared library. If you would like to associate the shared libraries with all the applications in a server, that can be done by defining the shared library under the **Server** class loader.

Multiple Server class loaders can be defined, forming a hierarchy of server class loaders, as it appears in the Administrative console. You can set the delegation mode for each of the Server class loaders to be parent first or parent last.

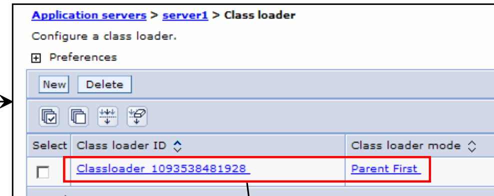
Creating WebSphere "Server" Class loader

Servers ➔ Application Server ➔ <Server>

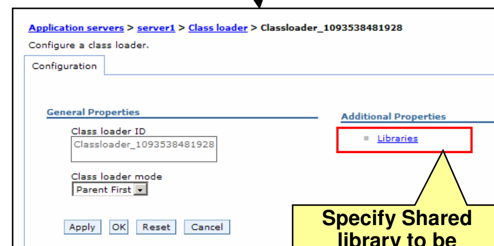
Define New "Server" Class loader



PARENT_FIRST or PARENT_LAST



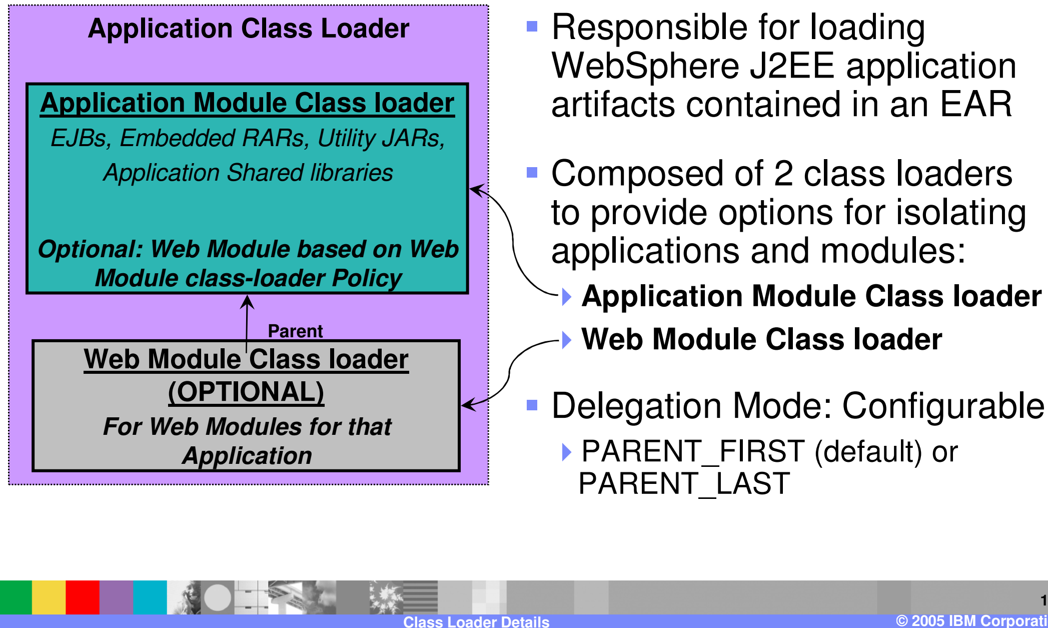
Edit "Server" Class loader



Specify Shared library to be loaded by this classloader

© 2005 IBM Corporation

The Administrative console panels shown here describe how to create and configure the WebSphere **Server** class loaders.

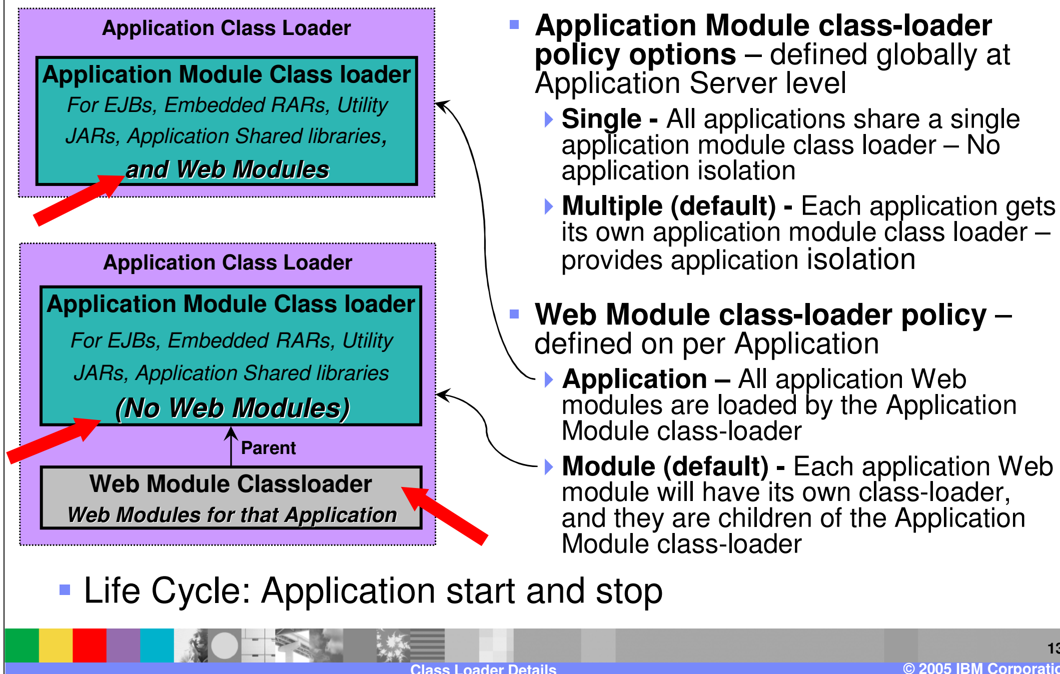


The Application class loader, which is responsible for loading the J2EE applications, have two sub class loaders.

The first one is the Application Module class loader, responsible for loading all application modules, with the potential exception of the Web Modules. The second one is the Web Module class loader that is lower in the class loader hierarchy and is responsible for loading the Web modules.

The Web Module class loader is optional and is created for an application only in certain configurations, as described in the next few pages.

In both these class loaders, the search or delegation mode can be configured to be parent first or parent last.



This page describes the options of the Application Class loader, known as the class loader isolation policies.

There are 2 options:

The first one defines whether there will be a single Application module class loader for the entire Application Server that will serve all the applications or a separate Application Module class loader for each application within the server. These are defined at the Application Server level, as **Single** or **Multiple**.

As discussed in the previous page, a Web module of the application can have its own separate class loader at a lower hierarchy than the application Module class loader. This option is defined for each application. The first option is **Application**, indicating that for that application, there should not a separate Web module class loader, and that the Web modules be loaded by the application module class loader. This example is shown in the picture at the top left side of the page.

The second option is **Module**, meaning for that application, the Web modules are loaded by their own separate Web module class loader, and that the application module class loader will not load the web modules for that application. This is the default value. This example is shown in the picture at the bottom left side of the page.

These options help in situations where you need isolation for your own packaged Jar files, like Xerces, and require a different delegation mode.

IBM Software Group

IBM

Class loader Policy UI

Application Module Class Loader Policy

Web Module Class Loader Policy
Need to go down the application settings

Application servers > server1
An application server is a server which pro
Runtime | Configuration
General Properties
Name
server1
Run in development mode
Parallel start
Server-specific Application Settings
Classloader policy
Multiple
Class loading mode
Parent first

Multiple or Single

Delegation
PARENT_FIRST
or
PARENT_LAST

Enterprise Applications > WebSphereBank
Enterprise Applications
Configuration | Local Topology
General Properties
Name
WebSphereBank
Binary Management
Application binaries
C:\WSAD_RATIONAL\
Use metadata from binaries
Enable distribution
Validation
warn
Class Loading and File Update Detection
Class loader mode
Parent First
WAR class loader policy
Module
Module
Application
reloading

Delegation
PARENT_FIRST
or
PARENT_LAST

Module
or
Applications

14

Class Loader Details

© 2005 IBM Corporation

The Administrative console can be used to set the Application class loader policies and delegation, as shown on this page.

The left panel shows the Single or Multiple Application module class loader policy. It also shows whether the search for the application module class loader will be parent first or parent last. Parent first means search the parent class loader in the hierarchy before searching locally.

The right panel shows the setting of the Web module class loader. This is configured separately for each application. It determines where the Web modules of the application will be loaded. If the policy is **Module,** it will be loaded by a new Web module class loader. If the policy is **Application**, it will be loaded by the Application module class loader. If the Web module class loader must be created, the delegation can be specified to be parent first or parent last.

IBM

Section

Shared Libraries

© 2005 IBM Corporation

This section will discuss shared libraries.

Shared Libraries - Overview

- Shared libraries provide a way to use common java or native code and share them across one or more J2EE applications running within the server
  - Example: Dependency (“utility”) JARs, and native libraries
- Shared libraries are defined by the administrator, and then associated with one or more applications
  - These are called “application-associated” shared library, compared to “server-associated” shared library loaded by the “Server” class loader
- Shared libraries can be:
  - Directory that contains collection of JAR files or native libraries
  - Specific JAR file
  - Native libraries
- Dependent Native libraries required by native library specified in Shared library will not loaded by the Application class loader – Should specify them in the JVM Native library path

© 2005 IBM Corporation

In WebSphere Application Server v4, if you had JAR files that needed to be shared by more than one application, you would place them in the %WAS_ROOT%/lib/app directory. The drawback is that the JARs in the %WAS_ROOT%/lib/app directory are exposed to all the applications.

Shared Libraries is a better mechanism, where only applications that need the JARs use them. Other applications are not affected by the shared libraries.

The process consists of defining the Shared libraries, naming them and specifying the file or directories that contain Java code and native code.

After that, these defined shared libraries can be associated with one or more applications running within the server.

The difference between creating Shared libraries this way is that these can be specified for use by just the applications that need them. In comparison, libraries defined when creating “Server” class loaders are available to all the applications. There is no granularity provided by the server class loader, whereas the shared libraries defined here provide a level of granularity in terms of association with one or more applications as needed.

The advantage of using application scoped shared library is the capability of using different versions of common application artifacts by different applications. A unique shared library can be defined for each version of a particular artifact, such as a utility JAR

If a native library loaded by shared library requires a second native library, then it must not be specified in the shared library path, but rather in the JVM native library path and will be loaded by the JVM class loader

Defining Shared Library

- Environment -> Shared Library
- Can define at Cell, Node or Server scopes
- Each shared library will have associated Java, Native libraries, or both, defined
- Adding new Shared library definition requires a server restart to be used by applications

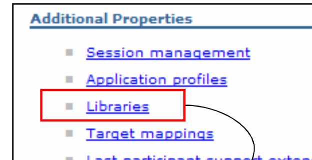
© 2005 IBM Corporation

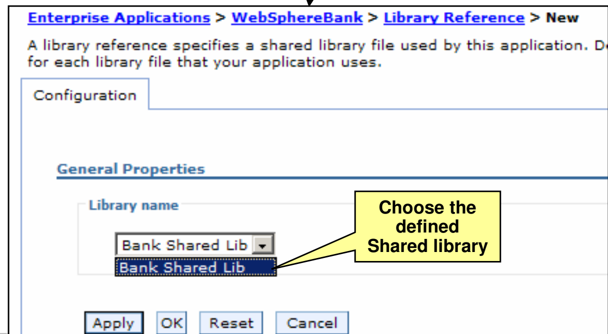
Shared libraries can be defined using the Administrative console panel shown here.

After defining a new shared library and associating it with an application, the Application Server must be restarted.

Applications Using Defined Shared Library

- Select Application -> Additional Properties -> Libraries
- You can add one or more defined shared libraries that will be used by the application code





© 2005 IBM Corporation

Once the shared libraries are defined, they can then be associated with the application. This slide shows the Administrative console panels used to make the association.

Section

Class loading of different files

© 2005 IBM Corporation

This section will cover class loading of different files.

Class Loading of Non-application Files

Artifact / Classes	Class loader	Comments
JVM Classes ▪<JAVA_HOME>/jre/lib ▪<JAVA_HOME>/jre/ext/lib ▪Classpath	**JVM**	
WebSphere Runtime classes ▪<JAVA_HOME>\lib ▪<WAS_HOME>\classes ▪<WAS_HOME>\lib ▪<WAS_HOME>\lib\ext	**WebSphere Extension**	
Custom Service class ▪It is a pluggable extension of the server runtime that allows users to plug in their own classes ▪Implements CustomService interface	**WebSphere Extension**	▪Users should not place Custom Service implementation in JVM classpath – it needs reference to <WAS_HOME>/lib files ▪Custom Service classes will persist in the JVM cache for duration of the Application Server
Resource files ▪Like JDBC, JMS, Resources, etc	**WebSphere Extension**	▪Not possible to have multiple copies of the same Resource class ▪Classes are loaded on demand

© 2005 IBM Corporation

The table shows the loading of the non-application classes like the JVM classes, WebSphere runtime classes, Resources, and Custom Service class.

Custom Service is a pluggable extension of the server runtime that allows you to plug in your own classes

Custom Service classes are loaded by the WebSphere Extensions class loader, which is the parent of every Application class loader. Therefore, Custom Services are visible to all applications hosted by an Application Server. Custom Service classes cannot have any dependencies on application classes

IBM Software Group

IBM

Class Loading of Application Files

Artifact / Classes	Class loader	Comments
Standalone Resource Adapters	**WebSphere Extension**	
Custom User Registry implementation Java class	**Should be placed in the WebSphere Extension class loader path**	WebSphere runtime requires this class – hence should not be placed in a location that is loaded by class loaders below the Extension class loader hierarchy
Server scoped Shared libraries	**WebSphere Server**	
Application associated Shared Libraries	**Application Module**	
EJB modules	**Application Module**	
Embedded Resource Adapters	**Application Module**	
Dependent JARs (Utility JARs)	**Application Module**	Dependency paths specified in the manifest *Class-Path* attribute of the EJB JARs and Web Modules. **Important: Classpath attributes are relative to the EAR directory path**
Web Module Files: ▪WEB-INF/classes ▪WEB-INF/lib	**Application Module** (If Web Module class loader policy = APPLICATION)	
	Web Module (If Web Module class loader policy = MODULE)	

21

Class Loader Details

© 2005 IBM Corporation

This table shows the loading of the application related classes.

The Stand-alone resource adapters are loaded by the WebSphere extension class loader, whereas the embedded resource adapter is loaded by the Application Module class loader.

The Server scoped shared library is loaded by the "Server" class loader, and therefore is available to all the applications, whereas the shared libraries associated with the application are loaded by the application module class loader, and available only to the application.

Section

Loading Native Libraries

© 2005 IBM Corporation

This section will cover loading native libraries.

Native Libraries - Overview

- Native libraries are platform specific non-Java code used by Java via JNI (Java Native Interface) – like ‘.dll’ in Windows™, ‘.so’ and ‘.a’ in Unix
- Java applications use System.loadLibrary(*libName*) method to load the native library
  - It is loaded at the time of System.loadLibrary(…) call
- JVM uses the caller’s classloader to load the native library and if that fails, it then uses the JVM System class loader
  - If both fail to load, you will see UnsatisfiedLinkError
- Native libraries are located on the native library paths of the JVM class loader and the WebSphere (Extensions, Server, and Application module) class loaders

© 2005 IBM Corporation

Native libraries are loaded by Java using the System load library when they are needed. Native libraries could be located in the JVM class loader or one of the WebSphere class loaders such as the Extensions, Server or the Application module class loader.

WebSphere Extensions, Server, and Application module class loaders define a local native library path, similar to java.library.path supported by the JVM class loader

Native Libraries – Overview (cont.)

- JVM native library: java.library.path setting - path for different systems
  - Windows: PATH environment variable, current working directory, and system directories
  - AIX®: LIBPATH environment variable
  - Solaris, Linux®: LD_LIBRARY_PATH environment variable
  - HP-UX: SHLIB_PATH environment variable
- JVM native library path can also be specified in the Administrative Console by specifying the appropriate OS string (like LIBPATH, PATH, etc.) and the value
  - Servers ➔ Application Server ➔ <Server> ➔ Java process and Management ➔ Process Definition ➔ Environment Entries

© 2005 IBM Corporation

The JVM native library path settings for different platforms are shown here. In addition, the native library path can also be configured using the Administrative console or wsadmin scripting.

Native Libraries – Errors

- If a different class loader already loaded the nativelibrary, the JVM throws an UnsatisfiedLinkError and indicates the problem
- JVM also throws UnsatisfiedLinkError whenever a dependent native library cannot be resolved
  - Native library specified in Shared library needs to load another dependent Native library and the dependent native lib. is not in the JVM Native lib path
- JVM will load only one instance of a particular native library per class loader
- Good Practice:
  - Call System.loadLibrary() within a static block of exactly one class within an application
  - If possible, load native library from the code that has life cycle of the server like WebSphere “Server” class loader

© 2005 IBM Corporation

When loading a native library, the JVM can throw an exception if the library was already loaded or if the library is not found. Therefore, it is a good practice to load the native library from a static block within the Java code using class loaders that have the life cycle of the server, if you are using WebSphere class loaders to load the native library.

Native libraries loaded by WebSphere Class loader

Artifact	Native Class loader	Appended Native Library Path (NLP)
Resources – JDBC, Generic JMS, etc.	**WebSphere Extension**	NLP for every JDBC provider defined in the server configuration
Standalone Resource Adapters	**WebSphere Extension**	NLP of the Resource Adapter
Embedded Resource Adapters	**Application Module**	NLP of the Resource Adapter
Server scoped Shared libraries	**WebSphere Server**	NLP of Shared Library
Application associated Shared libraries	**Application Module**	NLP of Shared Library

© 2005 IBM Corporation

This table shows the different WebSphere class loaders that are used for the native code in different artifacts.

IBM

Section

Class Loader

Some Advanced Topics

© 2005 IBM Corporation

This section will cover advanced class loader topics.

JVM Class loading – Dependent classes

- Dependent classes must be visible on the local class path of the object's class loader, or one its parent class loaders.
- Violations will cause the JVM to throw ClassNotFoundException and NoClassDefFoundError whenever the dependent class cannot be located

© 2005 IBM Corporation

If a class being loaded needs to load another class, the dependent class must be visible in the current class loader or any of its parent class loaders. The search or delegation mode of the current class loader will decide whether to search parent first or parent last.

If the class is not found in the current or the parent class loaders, an exception will be thrown.

JVM Class loading – Multiple Definitions and Cache

- JVM can maintain multiple definitions of classes having the same fully-qualified name
  - JVM recognizes a class definition as unique using the couplet <class-loader, class-name>, not <class-name>,
- JVM will cache several class definitions having the same name, so long as each definition is loaded by a different class loader
- Can otherwise cause confusing class loading anomalies in multiple environments
- Example: If a class loaded by a child class loader depends on a class that is inadvertently loaded by a parent class loader, and the two classes are binary incompatible (e.g., the dependent class was built with a different JDK or contains an interface change), the JVM will throw a linkage error, such as an InvalidClassChangeError

© 2005 IBM Corporation

The JVM can load multiple definitions of classes by different class loaders. For example, the Xerces parser will be loaded by the WebSphere Extension class loader, and if the application has bundled the Xerces class, the application module class loader could load the local Xerces class. Situations like these can cause anomalies that may have side effects, as shown in this example.

Section

Class Pre-loading

(Introduced in WebSphere V5.1.1)

© 2005 IBM Corporation

This section will discuss class preloading.

Class Pre-loading Overview

- Preloading classes speeds up the startup of the Application Server process
  - Class loader will not have to search for the classes when needed – they will already be available
- How is it done:
  - The 1st time the Application Server process starts up, the name of each class loaded and the name of the JAR file containing the class are written to a preload file
  - Subsequent startups of the process use the preload file to start the process more quickly
  - New classes required during startup of a process are added to the preload file
  - Any classes removed from a process are ignored during subsequent startups

© 2005 IBM Corporation

Pre loading classes improves performance during server startup.

A preload file is created when the server first starts up. The file contains the name of each class loaded and the JAR file containing the class. The preload file is used for subsequent server startups.

New files that might have been loaded during the server startup are added to the preload file. This could occur if a fix pack was applied, which caused addition of new classes.

Class Pre-loading – Administrative Tasks

- Administration Tasks: None required
  - No configuration is needed
  - By default, class preloading is ON
- To disable class preloading
  - Add the Generic JVM property to the Application Server -Dibm.websphere.preload.classes=false
- To Regenerate the preload file
  - Delete the preload file – the next startup of the server will recreate the file
- File name and directory for the preload file:
  - **Directory**: <%WAS_INSTALL%>/logs/preload directory
  - **File name**:
    - Application Server: *cell_name.node_name.server_name*.preload
    - startserver: WsServerLauncher.preload
    - launchClient: launchClient.preload

© 2005 IBM Corporation

By default, class preloading is enabled. The Dibm.websphere.preload.classes JVM property allows you to disable preloading.

If the preload file is deleted, a new one will be generated. Absence of a preload file does not affect the running of the Server.

The preload files are saved in the logs directory. There are separate preload files for Application server startup, start server and launch client.

Section

Summary and References

© 2005 IBM Corporation

This section will provide a summary of the concepts discussed in this presentation.

Summary

- Class-loaders are part of the Java virtual machine (JVM) and are responsible for finding and loading class files:
  - JVM Classloaders
  - WebSphere Extensions Classloader
  - WebSphere Server Classloader
  - Application Module Classloader
  - Web Module Classloader
- Few policy options to control your class loader scheme to suit your application needs
- Follow-on presentations on class loader will cover Examples, problem determination and Best Practice, etc.

© 2005 IBM Corporation

In summary, this presentation focused on class loaders, which are a part of the JVM and are responsible for finding and loading class files. It also gave a few policy options to control your class loader scheme to suit your application needs.

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.