하나씩 쉽게 따라 해보는 IBM WebSphere Application Server(WAS) v7 – 22

이정운 (juwlee@kr.ibm.com)

하나씩 쉽게 따라 해보는 IBM WAS v7 스물 두 번째 이야기를 시작합니다. 스물 두 번째는 이전 장까지 진행된 2부를 넘어 진정한 WAS 전문가로 가기위한 실질적인 3부를 진행하도록 하겠습니다. 오래 기다리신 만큼 3부 부터는 조금 더 심도 있고 깊이 있는 내용을 다루기 위해 노력하도록 하겠습니다. (그만큼 약간 어려울 수 있다는 점은 양지 부탁드리겠습니다.^^&;)

이번에 하나씩 쉽게 따라 해보는 IBM WAS v7 에서 진행하게 될 이야기는 클래스 로더(Class Loader) 입니다. 클래스 로더가 무엇인지는 알고 계시나요? (조금 고급 개념이긴 하지만 Java를 해보신 분은 너무 많이 들어서 이미 잘 아시고 있을 것이라고 믿습니다.^^;&) 클래스 로더는 말 그대로 클래스를 Java Virtual Machine(JVM) 에 로딩하는 로더입니다. 너무 간단히 설명드려서 조금 어려우신 가요? 아시겠지만 클래스라는 것은 결국 하나의 소스입니다. 컴퓨터에 존재한다고 해서 아무것도 할 수 있는 것이 없죠. 이 클래스라는 것이 실제로 컴퓨터에서 구동되기 위해서는 로딩이라는 단계를 거쳐서 메모리에 올라가고 실제 수행이 되어야 합니다. 이때, JVM 에서 이 역할을 하는 것이 바로 클래스 로더입니다. (사실 이 클래스 로더 역시 기본적으로 java.lang.ClassLoader 라는 클래스의 인스턴스입니다. 따라서 필요하면 원하시는데로 수정하여 커스텀 클래스 로더를 만들 수도 있습니다. – 단, 상당히 고난이도의 작업이므로 굳이 권장하지는 않습니다.)

실질적으로 클래스를 JVM 에 로딩하는 역할을 하기 때문에 JVM 에서 가장 중요한 메커니즘의 하니이며 가장 민감한 메커니즘입니다. 즉, 반드시 클래스 로더의 지정된 규칙을 잘 이해해야 합니다. 그리고 이를 잘 이해할 수 있어야 WAS 환경을 실질적으로 이해할 수 있으며 그에 적합하게 어플리케이션 아키텍처를 수립할 수 있습니다. 따라서 본 강좌에서는 이제부터 하나씩 IBM WAS 의 클래스 로더의 구조와 특징을 살펴보도록 하겠습니다.

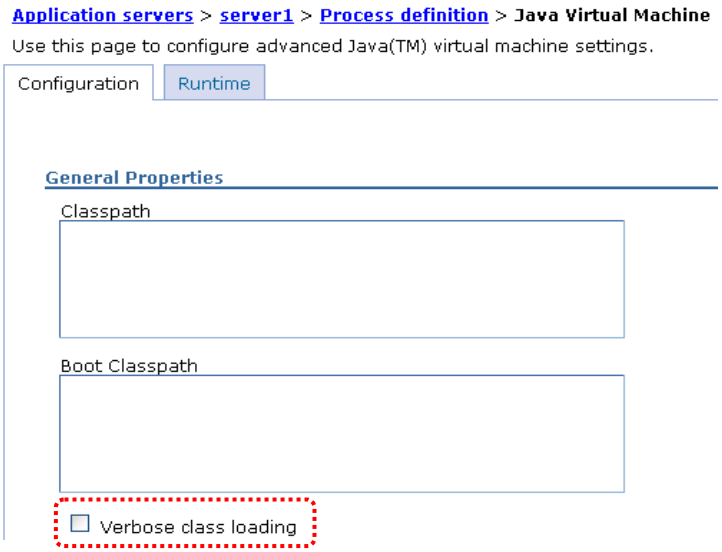
Part 1. 클래스 로더의 규칙

클래스 로더를 잘 이해하기 위해서는 클래스 로더 메커니즘이 가지고 있는 규칙을 잘 이해해야 합니다. 그 규칙을 잘 이해해야 제대로 클래스 로더를 활용할 수 있는 방안을 생각할 수 있습니다.

규칙 1. 클래스는 클래스 로더를 통해서 단 한번만 올라갑니다.

좀 더 이해하기 쉽게 말씀을 드리자면 한번 JVM 에 로딩된 클래스는 다시 로딩하지 않는다는 것입니다. 즉, 딱 한번만 로딩되서 해당 클래스가 올라간다는 의미입니다. Java 어플리케이션을 작성하면 하나의 클래스를 여러 번 사용하게 되어 있습니다. 그리고 많이들 보시는 것처럼 new 연산자를 이용해서 실제 인스턴스를 만들고 사용을 하게 되죠. (내부적으로 더 들어가면 해당 클래스가 호출되면 생성자를 통해서 실제적으로 클래스가 만들어지는 것인데, 이런 부분은 너무 복잡하므로 패스^^;&) 이때 해당 클래스가 클래스 로더를 통해서 JVM 에 올라가게 됩니다. 클래스가 먼저 올라간 다음에 그 클래스를 이용해서 인스턴스를 만들게 됩니다. 이렇게 한번 올라간 클래스는 보존 되며 다음에 new 연산자를 다시 호출되도 클래스를 다시 올리지 않고 이미 올라간 클래스를 활용하여 인스턴스를 만들게 됩니다. 결국 2번 올라가지 않는다는 것 입니다.

팁1 : 클래스 로딩이 되는 것을 실제로 확인하시고 싶으신 분들은 관리콘솔에서 Servers > 해당 서버 > Process definition > Java Virtual Machine 에서 하단의 옵션을 체크하면 클래스 로딩을 로그(native_stderr.log) 에서 확인할 수 있습니다 (-verbose:class 옵션으로도 확인이 가능 합니다.)



native_stderr.log

```
class load: java/lang/Object
class load: java/lang/J9VMInternals
class load: java/io/Serializable
class load: java/lang/reflect/GenericDeclaration
class load: java/lang/reflect/Type
class load: java/lang/reflect/AnnotatedElement
class load: java/lang/Class
class load: java/lang/Cloneable
class load: java/lang/Comparable
class load: java/lang/CharSequence
```

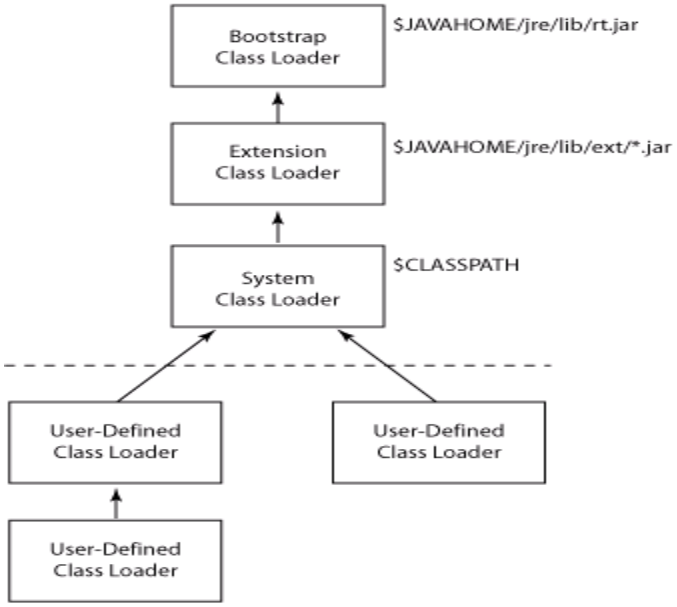
규칙 2. 클래스는 클래스 이름으로 구분되는 것이 아니라 식별자(fully qualified class name)로 구분됩니다.

규칙 1 에서 클래스는 클래스 로더에 의해서 단 한번만 로딩된다고 하였습니다. 그럼 TestClass 라는 것이 로딩되면 TestClass 는 그 뒤로 다시는 로딩되지 않을까요? 그건 아닙니다. 그 이유는, 기본적으로 클래스를 클래스 이름으로 구분하는 것이 아니라 식별자로 구분하기 때문입니다.(식별자라는 것은 해당 클래스가 유일하게 구분될 수 있는 구분자의 모음을 의미합니다.) Java 어플리케이션을 작성하실 때 소스의 맨 처음에 많이 사용하는 것이 package 입니다. 해당 클래스들을 패키지로 묶기 위해서 사용되는데 이 패키지가 식별자의 한 구분으로 사용됩니다. 즉, 패키지이름.클래스이름 이 클래스를 구분하기 위한 식별자로 사용됩니다. TestClass 가 이미 로드된 뒤라도 패키지 이름이 다르면 TestClass 는 로드될 수 있습니다. 규칙 1의 팁에서도 확인할 수 있지만 로드된 클래스로 java/lang/Object 와 같은 형식으로 볼 수 있는 것이 이런 이유입니다.

여기에 한 가지 더 숨은 사실이 있는데 클래스 로더는 하나가 아닌 여러 개가 될 수 있으며(보통 계층적 구조를 가집니다. 이 부분에 대해서는 뒤에서 보충 설명하도록 하겠습니다.), 그렇기 때문에 실질적인 구분자는 패키지이름.클래스이름 이 아니라 클래스로더이름.패키지이름.클래스이름의 형식이 됩니다.

규칙 3. 클래스 로더는 보통 계층적 구조를 가지고 있으며 마마보이(?) 입니다.

규칙 3은 조금 재밌게 이름을 붙였는데, 클래스 로더는 보통 계층적 구조를 가지고 있으며 마마보이입니다. 하단의 Java 의 클래스 로더 위임구조 표와 같이 클래스 로더는 하나가 아니라 여러 개가 존재하며 해당 클래스 로더는 계층 구조로 이루어져 있습니다.

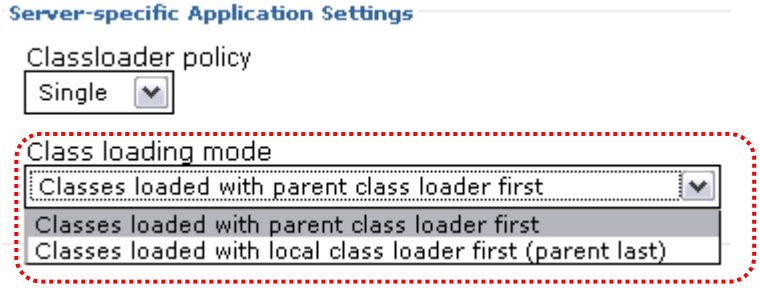


참조: Java 의 클래스 로더 위임 구조
http://www.ibm.com/developerworks/kr/library/j-dclp1/index.html

계층을 가지고 있으면서 본인의 클래스 로더에 연결된 위치의 클래스들을 로딩하는 역할을 합니다.

보다 더 중요한 것은 기본적으로 마마보이라는 표현 입니다. 해당 표현은 부모 위임(parent delegation) 을 의미하는 말입니다. 부모 위임을 쉽게 설명 드리자면 클래스 로더가 클래스를 로딩해야 할 경우 반드시 부모에게 먼저 요청을 위임하는 것을 말 합니다. 예를 들어 java/lang/TestClass 라는 클래스를 로딩해야 할 경우에 본인의 경로에 해당 클래스가 있고 로딩할 수 있다고 하여도 우선 계층적 구조에서의 부모 클래스 로더에게 로딩 해달라고 부탁합니다. 부모 클래스 로더는 해당 클래스의 부모(할아버지?) 클래스에게 다시 요청을 하죠. 이 작업을 반복하다가 더 이상 부모가 없거나, 부모 클래스 로더에서 해당 클래스를 로딩할 수 없다면 그제서야 클래스 로더가 실질적으로 요청된 클래스를 로딩합니다. 위에 첨부한 Java 클래스 로더 위임구조표를 이용해서 설명을 한다면 $CLASSPATH 위치에 해당 클래스가 있어서 System 클래스 로더가 로딩 할 수 있다고 하여도 $JAVA_HOME/jre/lib/rt.jar 에 해당 클래스가 있다면 해당 클래스는 bootstrap 클래스 로더에 의해서 로딩 됩니다. (당연히, 이 후부터는 $JAVA_HOME/jre/lib/rt.jar 에 있는 해당 클래스가 사용되므로 $CLASSPATH 에 위치한 클래스는 아무리 바꾸어 봤자 변경이 안되게 됩니다. 가끔식 이런 삽질(?)을 하시는 분들이 계시더군요.^^;&)

팁2 : 클래스 로더는 항상 부모위임 이긴 하지만 요즘의 WAS 에서는 옵션을 통해서 모드를 변경할 수도 있습니다. 마찬가지로 IBM WAS 에서도 해당 기능을 제공하며 관리콘솔의 Server 를 선택하면 하단처럼 바로 확인할 수 있습니다. 옵션을 "부모 나중(parent last)" 을 선택하시게 되면 부모 클래스 로더에 동일 클래스가 있더라도 그냥 자기 클래스 로더에서 오버라이드 하여 로딩하고 해당 클래스를 사용하는 방식입니다.



규칙 4. 클래스 로더는 자기 부모 밖에 직접 볼 수 없습니다.

규칙 4는 "클래스 로더는 자기 부모 밖에 직접 볼 수 없습니다" 입니다. 해당 클래스 로더에서 부모 클래스 로더에 있는 클래스들은 언제든 메모리 참조가 가능하지만 자식 또는 같은 위치의 클래스 로더에 있는 클래스들은 메모리 참조를 할 수 없습니다. 즉, 직접적으로 클래스들을 볼 수 없다는 의미입니다. 직접 봐야 할 클래스가 EJB 클래스 인 경우를 예로 들자면, 부모 클래스 로더의 EJB 클래스는 직접 볼 수 있기 때문에 메모리 참조를 통한 local call 이 가능하지만, 같은 위치나 자식 클래스 로더의 EJB 클래스의 경우에는 직접 볼 수 없기 때문에 메모리 참조가 불가능하고 remote call 을 해야 합니다. 당연히 메모리 참조에 비해 remote call 은 성능이 매우 떨어집니다. (remote call 인 경우에는 객체의 복사본을 네트워크를 통해서 전달하는 방식입니다. 그러므로 해당 객체는 직렬화를 하여야 하고, 직렬화를 할 경우에는 마샬링과 언마샬링과 같은 부수작업이 수반되므로 성능이 나쁠수 밖에 없습니다.) 클래스 로더의 구조를 잘 이해하고 해당 구조를 잘 활용할 수 있게 Application 아키텍처를 만들어야 하는 이유가 여기 있습니다.

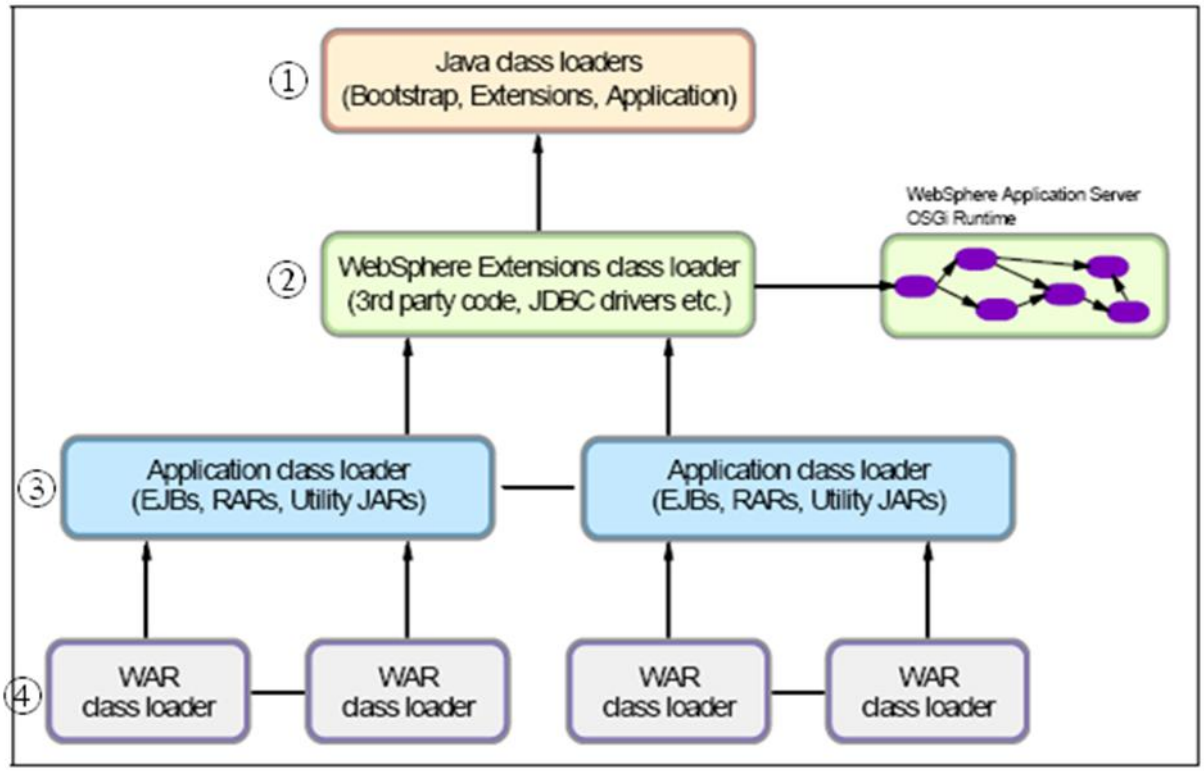
규칙 5. 클래스는 실제 사용될 경우에만 그 시점에 클래스 로더를 통해서 올라갑니다.

클래스 로더를 통해서 올라가는 클래스는 어플리케이션이 시작된다고 해서 모두 다 올라가는 것이 아니라 실제 사용되는 것들만 사용 시점에 올라가게 됩니다. 바로 실제 해당 클래스가 사용될 때, 그 때 해당 클래스를 올리게 되어 있습니다. (공식적으로는 Lazy loading 이라고 합니다.) 다시 말하면, 사용되지 않는 클래스가 있다면 해당 클래스들은 클래스 로더를 통해서 JVM 에 올라가지 않고 필요없는 메모리를 소모하지 않습니다. 이렇게 클래스 로더는 해당 클래스가 필요할 때 로딩하기 때문에 어플리케이션 수행중에서 ClassNotFoundException 이나 NoClassDefFoundError, ClassCastException 를 받을 수 있는 것 입니다.

ClassNotFoundException 는 클래스 로더가 로딩하려는 클래스를 찾을 수 없을 경우 발생 NoClassDefFoundError 는 해당 클래스가 상속받은 내재된 클래스를 클래스 로더가 로딩하려고 할 때 찾을 수 없는 경우 발생 ClassCastException 는 객체를 캐스팅(cast)할 때 캐스팅 할 수 없는 클래스로 캐스팅 할 경우에 발생

Part 2. IBM WAS 의 클래스 로더 구조

이전 파트에서 클래스 로더가 무엇인지 어떤한 규칙을 가지고 있는지 잘 숙지하셨나요? 해당 규칙만 잘 숙지하셨다면 이제부터의 단계는 그냥 이해하는 단계입니다. 기본적인 클래스 로더의 규칙을 준수한 채 IBM WAS 에서 제공하는 클래스 로더의 구조를 이해하는 것이죠. 하단의 그림이 IBM WAS 가 가지고 있는 클래스 로더의 구조입니다.



참조: IBM WAS v7.0 Redbook

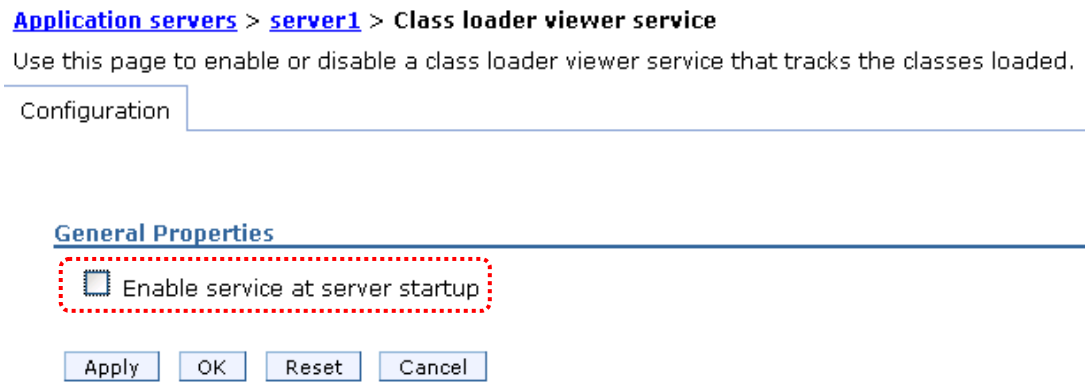
먼저 살펴볼 것은 **Java 클래스 로더** 입니다. 이는 Java 가 기본으로 가지고 있는 클래스 로더로서 java.lang.Object 같은 기본적인 클래스를 로딩하기 위해서 사용됩니다. 특히, WAS 제품에 J2EE API를 제공하기 위한 j2ee.jar 파일이 로드 됩니다. 이 부분은 위에서 어느 정도 설명했으므로 간단하게 넘어가도록 하겠습니다.

WebSphere Extensions 클래스 로더는 런타임에 필요한 WebSphere Application Server 클래스를 로드 합니다. ws.ext.dirs 로 지정된 시스템 속성에서 지정된 경로의 클래스들을 로드 하며 보통 3rd party code 와 JDBC Driver 등이 로드 됩니다.

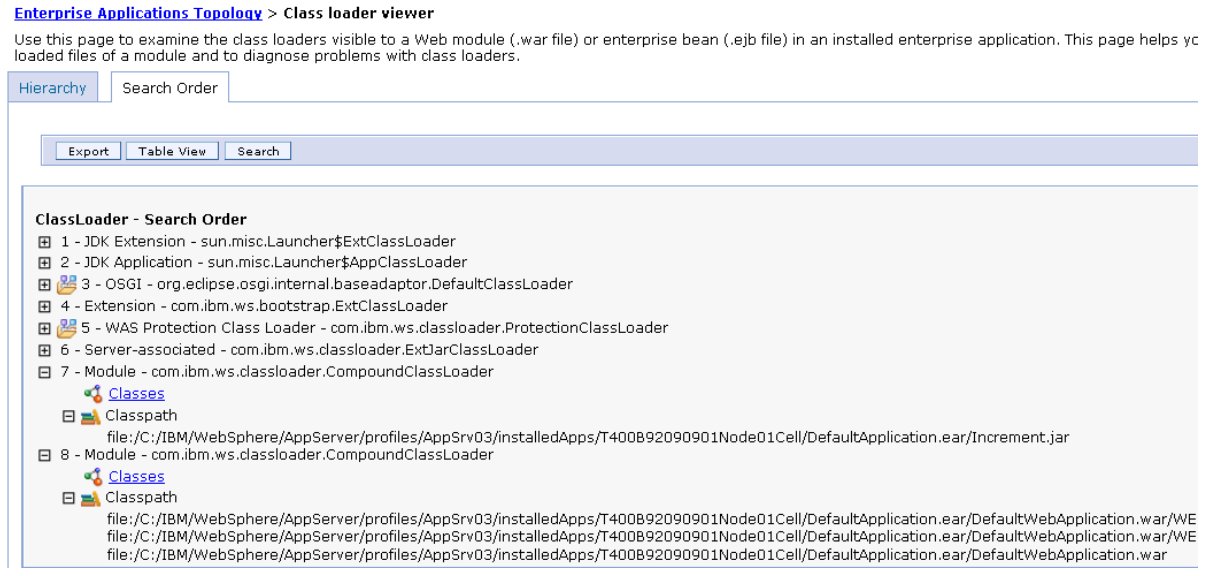
다음으로 가장 중요한 **Application 클래스 로더** 입니다. 웹 모듈, EJB(엔터프라이즈 Bean) 모듈, RAR 파일(자원 어댑터 아카이브) 및 종속성 JAR 파일 같은 Application 의 클래스 나 JAR 파일을 로드하며 공유 라이브러리를 Application 과 연관 시킬 수도 있습니다. 결론적으로 하나의 Application 이 실제로 로드 되어있는 클래스 로더로 보시면 됩니다.

마지막으로, **WAR 클래스 로더**는 WEB-INF/classes 및 WEB-INF/lib 의 클래스와 JAR 파일을 로드 합니다. EAR 에서 많이 사용되는 웹 모듈이 보통 로드 된다고 이해하시면 됩니다.

팁3 : 지난 18강에서도 말씀드렸지만 Server > Class loader viewer service 옵션을 하단처럼 설정 하시게되면 Class loader viewer 를 사용하실 수 있습니다

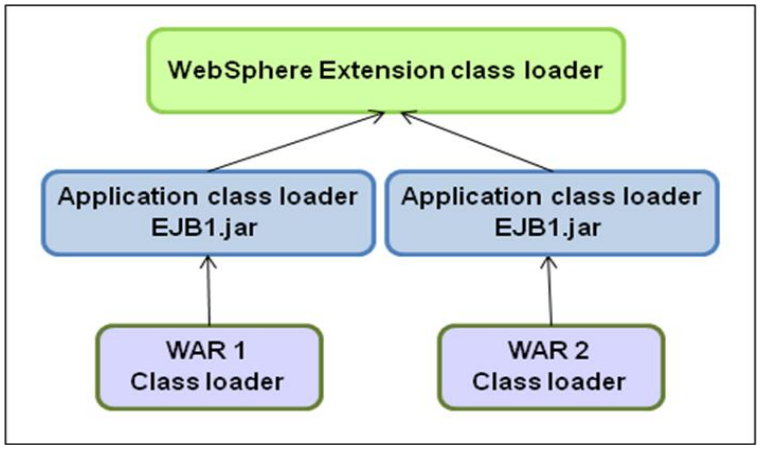


해당 서비스를 시작 시키고 Troubleshooting > Class loader viewer 메뉴를 선택하면 하단처럼 클래스 로더의 구조를 확인할 수 있으며, 또한 하나의 class 가 어느 클래스 로더에 의해서 로드되었는지도 확인할 수 있습니다 (여기서 4번 Extension – com.ibm.ws.bootstrap.ExtClassLoader 는 WebSphere Extensions 클래스 로더를 의미하며, 7번에 존재하는 Module – com.ibm.ws.classloader.CompoundClassLoader 는 Application 클래스 로더는 Application 클래스 로더, 8번에 존재하는 Module – com.ibm.ws.classloader.CompoundClassLoader 는 WAR 클래스 로더를 의미합니다.)



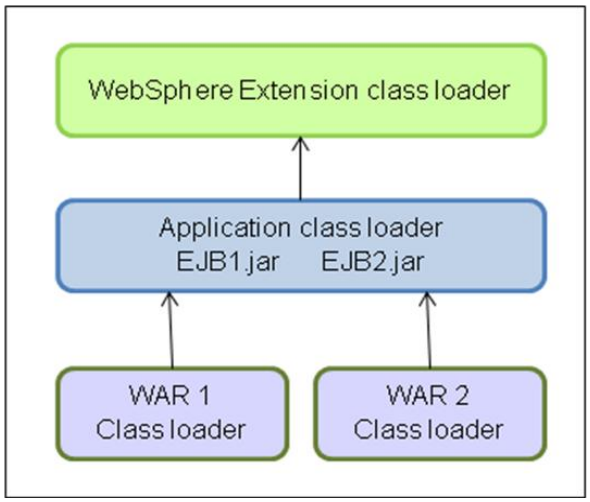
Part 3. IBM WAS 의 클래스 로더 구조 변경

Part 2 에서 설명한 IBM WAS 의 클래스 로더 구조는 기본적인 클래스 로더 구조입니다. 만약 사용자가 원한다면 옵션을 통해서 클래스 로더 구조의 변경이 가능합니다. (그렇다고 전부 다 마음대로 커스터마이즈 가능한 것은 아니고 몇몇 조정이 가능할 뿐입니다. 그리고 가장 기본적인 클래스 로더가 가장 많이 사용되고 추천되는 클래스 로더 구조입니다.)

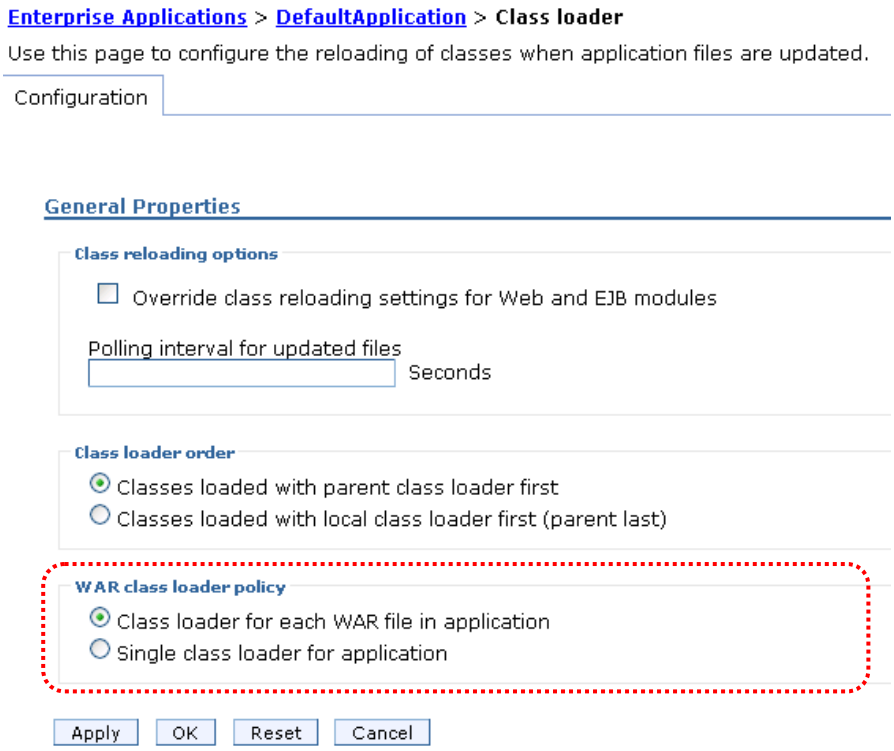


관리콘솔의 Server 옵션에서 보았던 Server-specific Application Settings 옵션을 통해서 Application 클래스 로더를 조정할 수 있습니다.

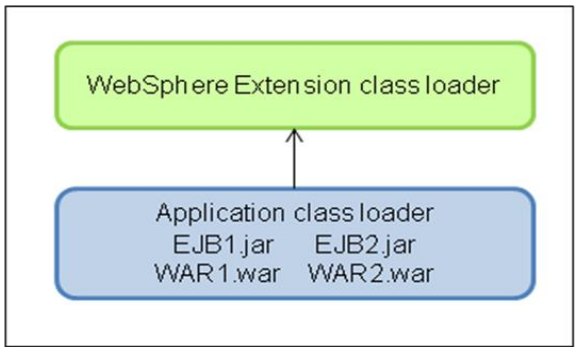
기본인 Multiple 이 아니라 Single 로 설정을 변경하게 되면 Application 단위로 Application 클래스 로더가 생성되는 것이 아니라 WAS 에서 단, 하나의 Application 클래스 로더가 생성되어 모든 Application 관련 클래스와 Jar 파일이 하나의 Application 클래스 로더를 통해서 로드 됩니다



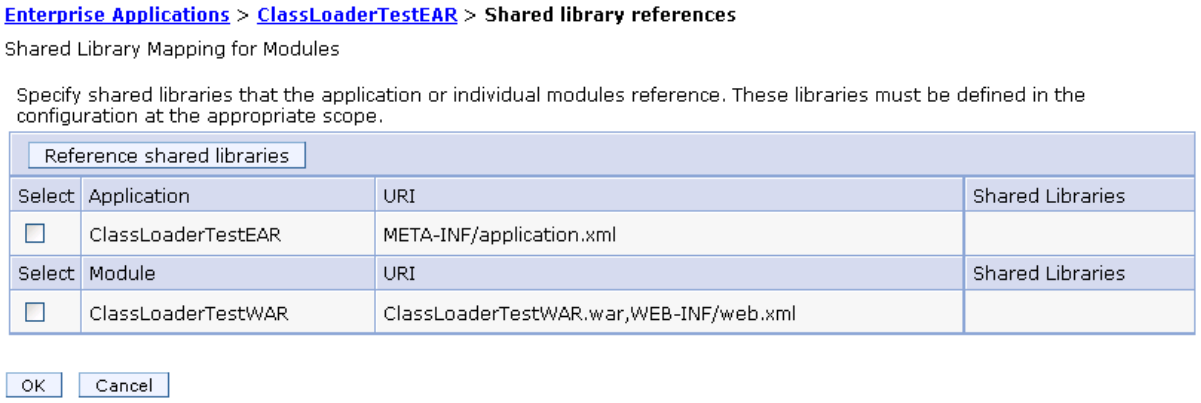
그리고 관리콘솔의 Enterprise application 메뉴의 Class loading and update detection 메뉴를 클릭하여 나타난 세부 설정 메뉴를 통해서 하나의 Application 에 대한 하위의 WAR 클래스 로더 정책을 조정할 수 있습니다



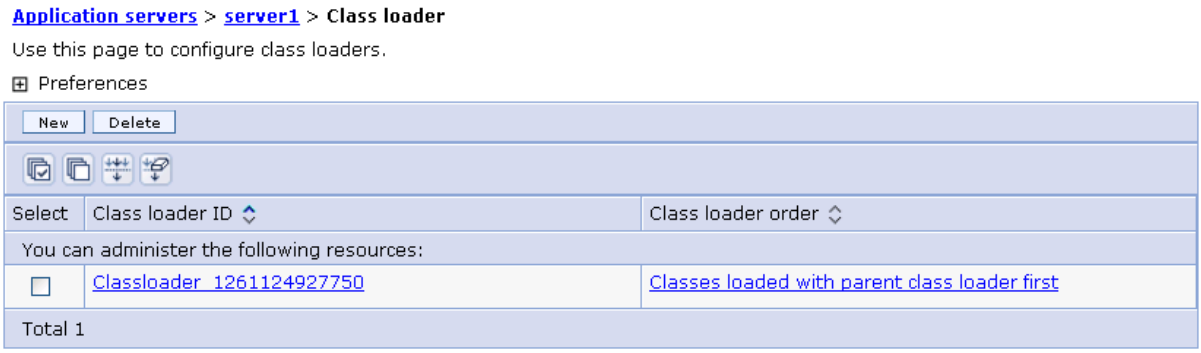
WAR class loader policy 옵션을 Single class loader for application 으로 변경하게 되면 웹 모듈을 WAR 클래스 로더로 분리하여 로딩하는 대신에 웹 모듈도 Application 클래스 로더에 포함하여 로드할 수 있습니다.

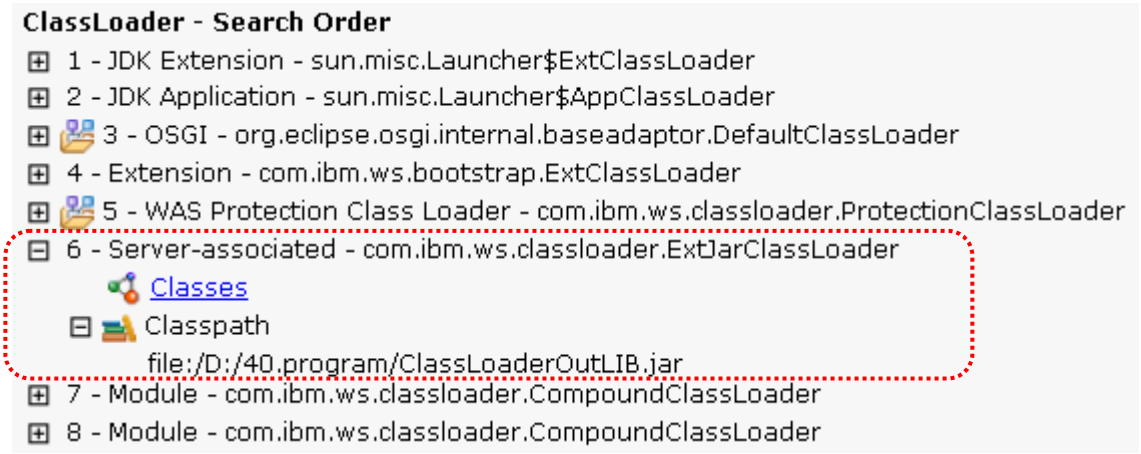


팁4 : Java 의 공유 라이브러리(Shard library)의 경우에는 이와 같은 Application 클래스 로더의 구조의 각 부분에 원하는 대로 설정할 수 있습니다.

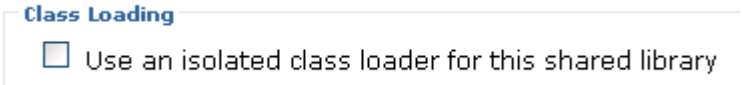


뿐만 아니라, 관리콘솔의 Server > Java > Class loader 메뉴를 통해서 Application 클래스 로더 상위이며, WebSphere Extension 클래스 로더 하위로 공유 라이브러리를 위치하여 해당 WAS 의 모든 Application 이 공유 가능하도록 할 수 있습니다





또한, 공유 라이브러리의 isolate 옵션을 이용하면 공유 라이브러리의 인스턴스를 해당하는 모든 Application 이 공유 가능하게 하여 리로드하지 않고 마치 Server 의 클래스 로드처럼 동일한 생명주기를 가지게 할 수도 있습니다.



여기까지 잘 이해가 되시나요? 지금까지 기본적인 클래스 로더의 규칙과 IBM WAS 의 클래스 로더에 대해서 배우는 시간을 가져봤습니다. 많은 경우 클래스 로더의 구조를 변경하는 경우는 드물지만, 때에 따라서는 Application 을 잘 관리하는데 효율적인 방법이 될 수 있습니다. 그렇기 때문에 IBM WAS 의 클래스 로더 구조를 잘 이해하시고 Application 아키텍처링을 하시는게 많은 도움이 될 수 있습니다. 또한, 클래스 로더의 구조를 잘 이해하시면 여러가지 어플리케이션 클래스 관련 문제가 발생되었을 때, 머리속에서 클래스 로더의 구조가 그려지고 즉각적인 대처가 가능합니다. 그러니 반드시 숙지하시기 바라겠습니다. 그럼 이번 강좌도 여기서 마무리 하도록 하겠습니다. 이만~~~~~휘리릭... ^^&;

참고 1) IBM WebSphere Application Server v7.0 InfoCenter
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.multiplatform.doc/info/welcome_nd.html

참고 2) IBM WebSphere Application Server v7.0 InfoCenter
- Class loaders in WebSphere Application Server
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/crun_classload.html?resultof=%22%63%6c%61%73%73%22%20%22%6c%6f%61%64%65%72%22%20