하나씩 쉽게 따라 해보는 IBM WebSphere Application Server(WAS) v7 – 20

이정운 (juwlee@kr.ibm.com)

하나씩 쉽게 따라 해보는 IBM WAS v7 드디어 그 스무번째 이야기를 시작합니다. 스무번째, 뭔가 딱 떨어지는 느낌이 좋고, 여기까지 힘겹지만 꾸준히 달려왔다는게 좋고, IBM WAS 에 대해서 정말많이 설명드렸어 기분이 좋네요. 스무번째 이야기를 뭐로 할 것인가에 대해서 이리 뒹굴고, 저리 뒹굴며 고민하다가 전체를 아우를 수 있으며 조금은 실용적인 IBM WAS 기본 튜닝으로 잡았습니다.

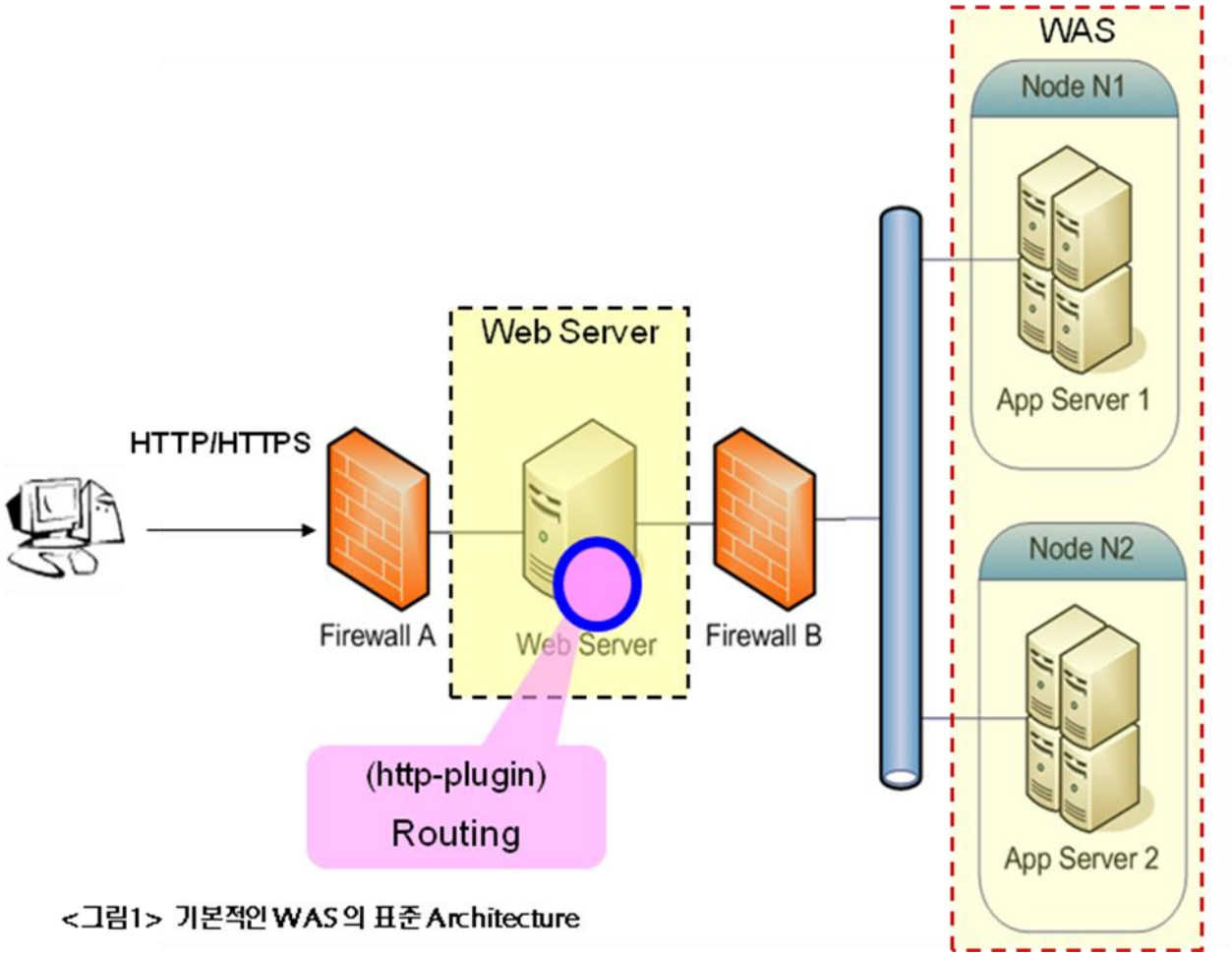
튜닝이라는 것은 WAS 위에서 돌아가는 어플리케이션, 즉 서비스가 좋은 성능을 내고 잘 수행되도록 여러가지 옵션을 조정하는 작업을 의미합니다. IBM WAS v7 Information Center 를 잘 뒤져보신 분들은 이미 아시겠지만 기본적인 튜닝에 대한 가이드가 사이트에 나와 있으며, 저희 www.websphere.pe.kr 사이트에서도 제가 올린 기본 튜닝 가이드를 확인 할 수 있습니다. 이를 참고하여 좀 더 설명하는 기분으로 이번 강좌를 진행하도록 하겠습니다. 사실 튜닝은 좀 더 경험적이고, 실제적인 작업이기 때문에 여기서 튜닝을 다 이해하신다고 생각하는 것 보다는, 이런 것을 가지고 기본 튜닝을 해야 하는구나라는 감을 얻는다고 생각하시고 이후에 실제 업무나 환경에서 직접 몸으로 겪으면서 체득하시기 바라겠습니다.

1. **하드웨어 및 소프트웨어 요구사항 검토**
2. **최신 갱신팩, 픽스팩 및 권장되는 임시 픽스 설치**
3. **하드웨어 구성 및 설정 확인**
   때때로 일시적인 오류로 인해 이더넷 어댑터 속도가 느려질 수 있습니다. 시스템 메모리가 적정하고 메모리 DIMM(Dual Inline Memory Module)의 수 및 해당 위치가 최적값인지 확인. 일부 시스템에서는 다른 DIMM 구성보다 우수한 성능을 허용하는 메모리 DIMM 구성이 있습니다. 사용되는 하드웨어가 사용할 수 있는 하드웨어인지 여부를 확인.
4. **응용프로그램 설계 검토**
   응용프로그램 설계로 인한 많은 성능 문제점을 추적할 수 있습니다. 설계를 검토하여 이로 인해 성능 문제점이 발생하는지 여부를 판별.
5. **운영 체제 조정**
   운영 체제 구성은 성능에 있어 중요한 역할을 합니다. 많은 경우, 일부 TCP/IP 매개변수를 사용자 응용프로그램에 맞게 조정해야 합니다.
6. **최소 및 최대 JVM(Java Virtual Machine) 힙 크기 설정**
   많은 응용프로그램의 경우 힙 크기를 늘려야 최적의 성능을 나타낼 수 있습니다.
7. **유형 4(또는 순수 Java) JDBC 드라이버 사용**
   유형 4 JDBC 드라이버는 다중 행 Fetch 영역에서 빠르게 수행합니다.
8. **WebSphere Application Server JDBC 데이터 소스 및 연관 연결 풀 조정**
   JDBC 데이터 소스 구성은 성능에 중대한 영향을 줍니다. 예를 들어, 연결 풀 크기 및 준비된 명령문 캐시 크기는 처리되는 동시 요청 수와 응용프로그램 설계에 따라 조정되어야 합니다.
9. **참조에 의한 전달 옵션 사용 가능**
   매개변수를 스택에 복사하지 않으려면 참조에 의한 전달 옵션을 이용할 수 있는 응용프로그램을 사용.
10. **트랜잭션 로그가 고속 디스크에 지정되어 있는지 확인**
    일부 응용프로그램은 WebSphere Application Server 트랜잭션 로그에 대한 고속 쓰기를 생성합니다. 고속 디스크 또는 디스크 배열에서 트랜잭션 로그를 찾으면 응답 시간을 줄일 수 있습니다.
11. **조정 관련 컴포넌트**
    많은 경우에 있어 일부 다른 컴포넌트(예제: 데이터베이스)를 조정해야 전체 구성의 처리량이 증가합니다.
12. **필요하지 않은 기능 사용 안 함**
    예를 들어 응용프로그램에서 웹 서비스 주소 지정(WS-Addressing) 지원 기능을 사용하지 않을 때는 이 기능을 사용 안 함으로 설정하여 성능을 향상시킬 수 있습니다.

참조 : 주요 성능 튜닝 작업 절차 요약(IBM WAS v7 Information Center)

Part 1. 튜닝 대상 선정 및 준비

튜닝 전에 필요한 작업은 당연히 튜닝 대상 선정 및 준비 작업입니다. 이번 강좌에서는 기본 튜닝만 다룰 것이기 때문에 하단의 그림과 같은 기본적인 WAS 의 표준 아키텍처를 기준으로 WAS 를 기본 튜닝 대상으로 삼도록 하겠습니다. 사실적으로는 웹서버 튜닝이나 OS 튜닝, 어플리케이션 튜닝등 여러가지 다양한 튜닝이 더 필요한 것이 사실이지만, 그런 자세한 사항들은 이전에 말씀드렸던 IBM WAS Information Center 의 가이드를 참조 하시기 바라며 가장 중요한 WAS 튜닝부분만 다루도록 하겠습니다.



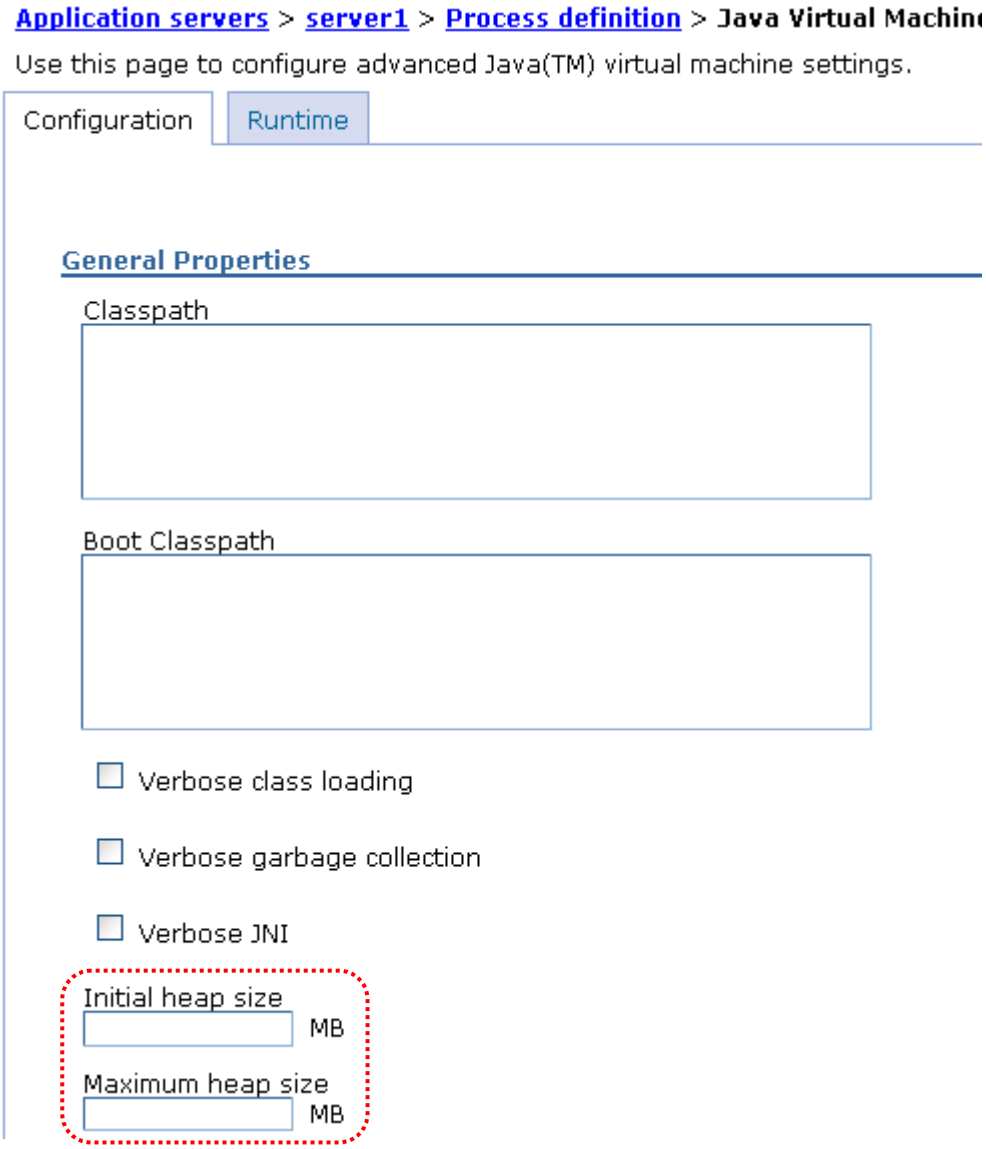
<그림1> 기본적인 WAS 의 표준 Architecture

WAS 는 다시 실제 어플리케이션을 구동시키는 JVM (Java Virtual Machine) 과 요청을 받는 Thread 즉, System 대기열, 어플리케이션으로 대상이 나뉘게 됩니다만 어플리케이션을 다루지 않는다는 것은 이미 말씀드렸기 때문에 JVM 과 System 대기열 부분의 튜닝 포인트에 대해서 집중하여 설명하도록 하겠습니다.

Part 2. JVM 튜닝

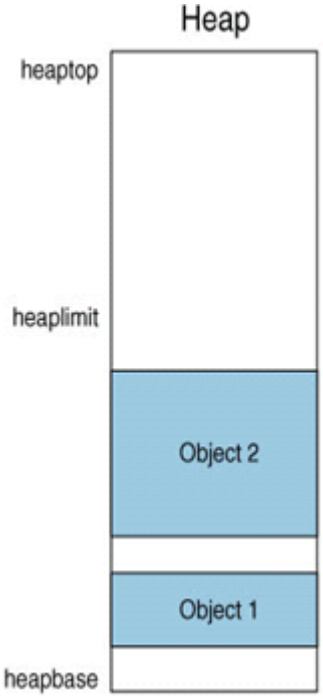
이미 몇 번 강조하였지만 WAS 에서 실질적으로 서비스를 하기 위해서 구동되는 하나의 서버 인스턴스는 결국 하나의 JVM 입니다. 즉, JVM 튜닝이 WAS 튜닝의 제일 핵심이고 말할수 있습니다.. 이를 다시 좀 더 세부적으로 살펴보면 JVM 이 사용할 수 있는 Heap 메모리와 GC(Garbage Collection)에 관련된 튜닝으로 나눌 수 있습니다. JVM 은 자체적으로 Heap 메모리를 만들어서 어플리케이션이 이를 활용하여 구동되기 때문에 어떻게 어플리케이션이 필요한 만큼의 최적의 메모리를 줄 수 있는가가 큰 성능 이슈중의 하나입니다. 튜닝의 가장 중요한 점은 최대한 많이가 아니라 어떻게 필요한 만큼 최적으로 줄 수 있는가 입니다. 또한 한때, Java 의 사생아라고 천대받던 GC 도 JVM 튜닝의 가장 중요한 포인트입니다. GC가 발생하면 순간적이긴 하지만 Thread 의 중지가 발생하므로 이를 최소화 하는 것이 결국 성능을 최대한 발휘하는 것 입니다.

이제 Heap 에 대한 튜닝 포인트를 보다 자세히 살펴보도록 하겠습니다. 관리콘솔에서 Application servers > 해당 서버 > Process definition > Java virtual Machnine 메뉴를 클릭하면 하단과 같은 JVM 설정 화면을 보실 수 있습니다.



JVM 설정 메뉴를 보면 아시겠지만, Heap 에 대한 튜닝 포인트중에서 가장 중요한 것은 Heap 사이즈의 최소값과 최대값을 잘 설정하는 것입니다. Initial heap size 는 JVM 이 initialize 될 때 생성되는 초기 Heap 크기이며 Maximum heap size 는 JVM heap 이 최대한 커질 수 있는 크기입니다. 따라서 초기 Heap 크기는 가급적 어플리케이션이 사용하는 평균치가 되어야 하며 최대값은 실제로 어플리케이션이 사용하는 heap 메모리의 최대값의 20% 정도 여유분이 더 들어가야 합니다. 정말 단순하게 보이는 말이지만 그 적정 값을 찾는 것은 가장 어려운 일중의 하나입니다. 이는 WAS 서버에 어떤 어플리케이션이 돌아가냐에 따라서 전부 틀리기 때문에 실제로 부하테스트를 통한 튜닝이 가장 적합한 방법입니다. 제 경험치에 의한 그나마 추천 값은 초기값이 256 MB 정도이며 최대의 경우 2GB 입니다. ^^&;

참고적으로, 많은 분들이 오해하시는 것 중의 하나가 WAS 가 시작되자 마자 설정되어 있는 Maximum heap size 의 값을 사용하는 것 처럼 생각하시는 경우가 있는데 이는 조금은 틀린 이야기 입니다.



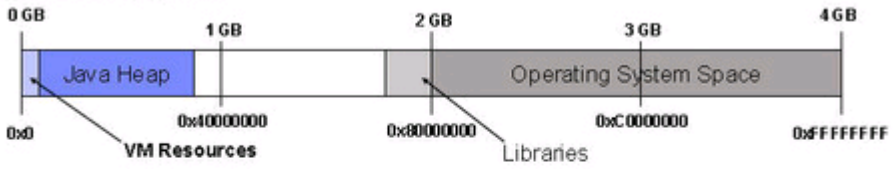
위에 있는 그림이 heap 이 가진 구조를 이해하기 쉽게 그림으로 표현된 것입니다. WAS 가 시작되면 Initial heap size 가 heaplimit 값이 되고, 그만큼의 Memory 를 차지합니다. 그리고 요청이 많아지고 객체가 점점 heap 에 쌓이면서 heaplimit 도 계속 늘어나고 heaptop 까지 찰 수 있습니다. 그때, heaptop 이 나타내는 것이 바로 Maximum heap size 입니다. 즉, 요청이 지속적으로 없다거나, 객체가 heap 에 충분히 쌓이지 않는다면 Maximum heap size 까지의 메모리를 안 쓸수도 있습니다. (대부분 그렇습니다.)

그러면 여러분들이 추가적으로 궁금증이 생기실텐데, 어느 정도의 객체가 메모리에 차야 heaplimit 이 증가할 것인가 입니다. 이때 사용되는 옵션이 -Xminf 과 –Xmaxf 입니다. –Xminf 는 minimum free 를 의미하며 GC 후에 남는 메모리의 영역이 적어도 이정도는 되어야 한다는 의미

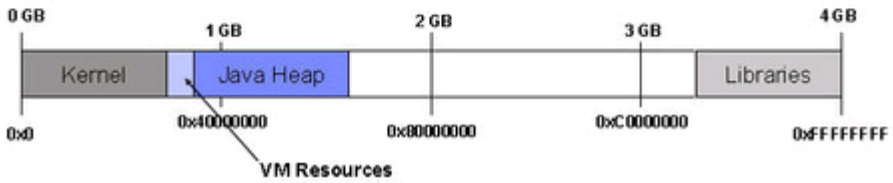
입니다. (기본 값은 0.3 으로 30%를 의미합니다.) 다시 말해서, GC 후에 남은 메모리 영역이 전체의 30%가 되지 않는다면 heaplimit 이 증가합니다. 반대로 –Xmaxf 는 maximum free 를 의미하며 GC 후에 남는 메모리의 영역이 지정된 값 이상이라면 heaplimit 은 줄어듭니다. (기본 값은 0.6 으로 60%를 의미합니다.) 어때 이해가 되시나요? JVM 을 이해하기 위해서는 반드시 이 Heap 을 잘 이해할 수 있어야 합니다.

Java Heap 사이즈를 지정하면서 주의할 것이 하나 있는데, 64bit 의 경우에는 크게 문제가 없지만 32bit 의 JVM 을 사용할 경우에는 Native heap 도 같이 신경을 쓰셔야 합니다. Native heap 은 보통 native code 들이 직접 들어가 있는 메모리 영역으로 JNI 코드나 class 등이 위치하게 됩니다. 32 bit 의 경우, 이론적으로 최대 활용가능한 메모리 레퍼런스의 최대 크기는 2^32 입니다. 대략 4GB 정도이나, OS 가 반드시 사용해야 하는 부분등을 빼면 2~3GB 정도 됩니다. 이 메모리를 Java heap 만이 쓰는 것이 아니라 Native heap 도 나눠서 쓰게 됩니다. 즉, Java heap 이 커지면 커질수록 Native heap 은 작아질 수 밖에 없는 현실에 도달되는 것이죠. 이런 현상이 심해지면 Java heap 이 많이 남아있음에도 불구하고 heap 이 모자라다고 OOM(OutofMemory) 현상이 발생되는 경우가 나타납니다. 이 경우에는 보통 native heap 이 부족해서 OOM 이 발생하는 것 입니다. 그렇기 때문에 Java heap 을 최대한 확보하는 것보다, 최적의 값을 찾는 것이 중요하다라는 말씀을 드린 것 입니다.

- **On Windows**



- **On AIX**



참조 : 32bit 에서 어플리케이션에 대한 Native heap 가능영역
(http://blog.naver.com/hslee93/110079836668)

Heap 의 사이즈 말고 추가로 Heap 에 대해서 살펴볼 만한 옵션을 간단히 설명드리자면 Allocating Page size 가 있습니다. 이 옵션은 –Xlp64k 옵션으로 설정가능하며 Heap을 64KB page 크기로 할당하도록 합니다. 가상 메모리 페이지를 64KB로 사용하도록 함으로 성능과 처리량을 향상시키기 때문에 64bit 환경의 경우에는 보통 사용하는 것이 권장됩니다.

Part 3. GC 튜닝

지금까지는 간단하게 Java Heap 에 대한 튜닝 옵션에 대해서 살펴봤습니다. 그럼 이제부터는 JVM 튜닝 옵션의 또다른 중요 포인트인 GC 튜닝에 대해서 알아보도록 하겠습니다.

그런데 먼저 짚고 넘어갈 것이 하나 있는데, GC 가 뭔지는 잘 알고 계신가요? 이 강좌를 읽으시는 분들은 Java 를 어느 정도는 다 하실수 있을 거라는 생각이기 때문에 GC 를 잘알고 있다고 생각되지만 간단히만 짚고 넘어가도록 하겠습니다. GC 는 쉽게 얘기해서 청소부 입니다. Java 의 가장 큰 장점은 C 언어와는 다르게 포인터를 이용해서 memory 를 직접 제어하지 않는다는 것입니다. JVM 이 알아서 메모리를 관리하고 제어합니다. 그중에 가장 중요한게 이 GC 입니다. 어플리케이션에서 사용을 위해서 객체를 메모리에 올린 후에 그 객체가 사용이 끝나면 메모리에서 반환하여야 하는데, 이 때 이 작업을 수행하는 것이 GC 입니다. 즉, 필요없는 객체를 쓱쓱 쓸어모아서 버리는 것을 GC가 하는 것 입니다.그런데, 이 GC 가 수행되는 동안에는 모든 Thread 들이 멈추게 되어 있기 때문에, 이 GC를 얼마나 잘 튜닝하느냐가 JVM 성능에 지대한 영향을 미치게 되는 것입니다.

그럼 GC를 좀더 분석해 볼까요? (GC의 경우에는 IBM JDK 를 기준으로 설명하도록 하겠습니다.) GC는 Mark, Sweep, Compaction 의 3단계로 구성됩니다. Mark 를 통하여 사용되지 않는 객체를 찾고, Sweep 에서 free 영역의 풀에 추가 하고, Compact 을 통하여 빈 공간을 메우게 됩니다.(좀더 자세한 설명은 하단의 표를 참조하세요)

✓ GC 의 3 단계

항목	설명	비고
Mark phase	Mark 단계에서는 thread stack 이나 static, interned string, JNI reference 에서 참조하는 모든 object 를 식별한다. 이 행위는 JVM이 참조하는 object 의 root 집합을 생성한다.각 object 는 각각 다른 것들을 하므로 결국, 프로세스의 두 번째 부분에서 그것이 만든 다른 참조들을 위하여 각 object 를 스캔 한다. 이 두 가지 프로세스가 함께 도달 할 수 있는 object의 시작을 정의하고 있는 bit 벡터를 생성한다. (markbits)	
Sweep phase	Sweep 단계에서는 다음 할당을 위해서 회수될 수 있는 heap storage 의 부분을 인식하기 위하여 mark 단계에서 만들어진 mark bit 을 사용한다. 그래서 이러한 부분들은 free 영역의 pool 에 추가된다.	Mark 와 Sweep 단계는 compaction 단계에 비해서 아주 작은 시간을 소비한다.
Compaction phase	Garbage 가 heap 에서 회수될 때 GC 는 그들 사이의 공간을 제거하기 위하여 object 의 결과 집합에 대하여 압축작업을 할지 고려한다. compaction 과정은 오랜 시간이 걸릴 수 있기 때문에 GC 는 절대적으로 그것이 필요할 때만 수행하며, 따라서 compaction은 드문 이벤트이다.	실질적으로 가장 많은 시간을 소비하므로 compaction 단계를 줄이는 것이 tuning 의 대표 기법이다.

이러한 GC 를 수행할 때 어떤 방식으로 수용할 것인가의 알고리즘이 성능에 가장 밀접하게 연관되는데 IBM JDK 의 경우에는 총 4가지 방식의 GC 알고리즘을 제공합니다. 각각 장, 단 점이 있으므로 목적에 맞게 4가지 중에 한가지를 선택하여 사용하시면 됩니다.

✓ GC 정책 (-Xgcpolicy 옵션 이용)

항목	설명	비고
Optthruput 정책	처리량은 높지만 GC 중간시간이 길다. GC 중 compaction 이 필요하면 mark, sweep, compaction 을 하기 위한 모든 어플리케이션 thread 는 중지된다.	기본값이며, 대부분의 어플리케이션은 이 옵션으로도 충분한 성능을 낼 수 있음.
Optavgpause 정책	어플리케이션 실행과 함께 GC의 mark, sweep 단계를 실행시킴으로써 GC 중단 시간을 줄인다. 이런 동시실행은 전반적인 처리량에 작은 성능 영향을 야기할 수 있다.	
Gencon 정책	IBM JVM 을 위하여 세대가 있는 GC를 사용한다. 세대의 개념은 GC 중단시간을 줄임으로써 높은 처리량을 달성하기 위해 사용된다. 이러한 목적을 이루기 위하여 heap 을 new 와 old 부분으로 나눈다. 짧게 살아있는 객체가 new영역에서 빠르게 GC 되는 반면에 오랫동안 살아있는 객체는 old 공간으로 이동한다.	Java 5.0 에서 새로 도입된 옵션으로 성능상에 많은 이점이 있는 경우가 많이 해당값을 사용하는 것을 권장한다.
Subpool 정책	보통 8 개 이상의 프로세스를 사용할 경우에 멀티프로세서 시스템에서 성능을 향상시킬 수 있다. 이 정책은 IBM pSeries 와 zSeries 에서만 사용가능하며 향상된 확장성과 객체 할당을 제공하기 위하여 heap이 subpool 들로 나누어져 있다는 것만 제외하고는 optthruput 정책과 비슷하다.	.

이해를 위하여 조금 더 부가적인 설명을 보태자면, Optthruput 정책의 경우는 Parallel Mark Sweep Collector 를 사용하여 GC 가 이루어 지며(가능한 compaction 단계를 피함) 처리량을 최적화하기 위하여 사용됩니다. 이전 IBM JDK 와 동일하게 단일 Java heap으로 구성됩니다.

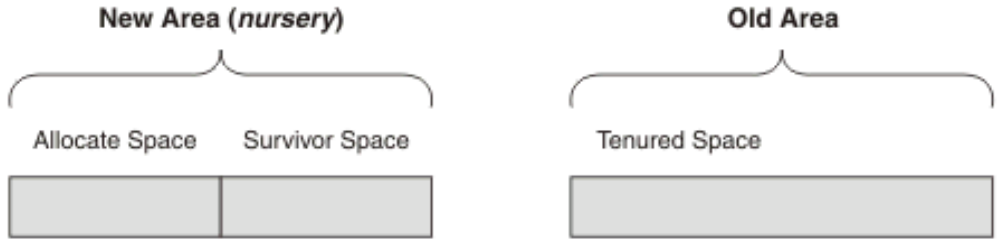
Optavgpause 정책은 평균적인 GC 중단을 가급적 줄이기 위한 방안으로 사용되며 Concurrent Marking 과 Concurrent Sweeping 을 사용합니다. 다만, GC 를 너무 크게 줄이게 되면 부가적으로 처리량에 약간의 오버헤드를 주게 되므로 빠른 사용자 응답시간 기준을 가진 시스템에 이상적인 정책입니다.

Subpool 정책은 대량의 프로세서를 가진 시스템에서 오브젝트 할당을 가속화하기 위하여 사용되며 적어도 8 개 이상의 프로세스를 사용할 경우에 의미가 있는 GC 정책입니다.

마지막으로, 요즘들어(IBM JDK 5 이상) 가장 많이 사용하는 GC 정책인 Gencon 정책 입니다. 위의 표에 설명이 자세히 나와있지만 기존 IBM JDK 의 메모리 구조와 다르게(그 전에는 전체가 한 영역) Sun 의 Hotspot 처럼 메모리 구조를 New 영역(nursery)과 Old 영역(tenured)으로 나누어서 GC 에 의한 중단 시간을 줄이고 처리량을 높일 수 있습니다. 뿐만 아니라, IBM WAS 7 에서 사용하고 있는 IBM JDK 6 에서는 더 완벽한 Gencon 알고리즘이 적용되어 더 좋은 성능을 기대할 수 있습니다. (Gencon 은 내부적으로 Generational Concurrent GC 알고리즘을 사용합니다.)

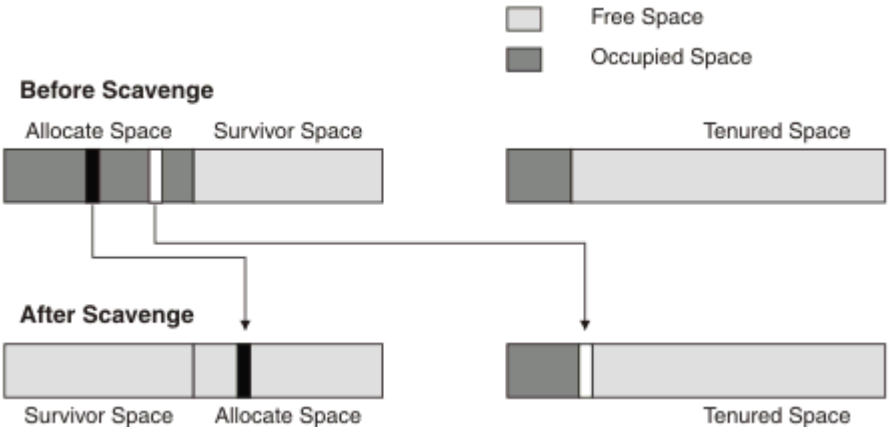
Gencon 이 요즘들어 많이 사용되는 방식중의 하나이기 때문에 이해를 돕고자 좀 더 자세한 설명

을 하도록 하겠습니다. New 영역은 다시 하단에서 보는 것처럼 allocate 영영과 survivor 영역으로 나누어 지며, Old 영역은 tenured 영역이라고 지칭됩니다.



참조 : IBM JDK 6 information Center

실제로 객체가 들어오면 해당 객체는 allocate 영역에 할당되고 해당 영역이 다 차게되면, GC 프로세스는 필요없는 객체를 청소하기 위해 scavenge 를 작동시킵니다. 이 scavenge 가 작동되는 동안, 살아있는 객체는 Survivor 영역에 복사되거나 tenured age 를 넘긴 경우(일정 시간 이상 살아있는 오래된 객체) Tenured 영역으로 복사됩니다. 이러한 모든 복사 작업이 끝나면 new 영역은 allocate 와 survivor 영역을 맞바꾸어서 사용하게되며 survivor 영역은 clear 됩니다.(이때 allocate 에 있는 객체들은 tenured age 가 +1 됩니다.) 이해가 잘 안되시는 분들은 하단의 그림을 참조하시면 보다 효율적으로 이해를 하실 수 있을 것 입니다.



참조 : IBM JDK 6 information Center

이 때, 이 scavenge 작업이 Sun 에서 이야기하는 Minor GC 와 같은 역할을 하고(객체만 다른 위치로 복사) Global GC (상위에서 설명한 실제적인 Mart, Sweep, Compaction 수행) 가 Major GC 와 같은 역할을 합니다.
따라서 GC 를 잘 튜닝하기 위해서는 New 영역과 Old 영역의 비중을 어떻게 조절할 것인가도 중요한 요소 중의 하나입니다. Gencon 정책을 사용했을 때 이 두 영역(New 와 Old영역)은 –Xmns, -Xmnx, -Xmos, -Xmox 옵션으로 사이즈 조절이 가능합니다. –Xmns 는 New 영역의 초기 크기를, -Xmnx 는 new 영역의 최대크기를 지정하며 반대로 –Xmos 는 Old 영역의 초기 크기를, -Xmox 는 Old 영역의 최대 크기를 지정할 수 있습니다.

여기서 재미있는 것이 하나있는데 바로 오래된 객체들이 존재하는 Tenured 영역의 GC 입니다.

IBM JDK 에서는 Tenured 영역의 GC 를 위해서 별도로 optavgpause 정책과 동일한 concurrent GC 알고리즘을 사용합니다. 그래서 gencon 이 사용하는 GC 알고리즘의 이름이 Generational Concurrent GC 알고리즘 입니다. (New 영역에 대한 GC 는 Generational 알고리즘을, Old 영역에 대한 GC 는 Concurrent GC 알고리즘을 사용한다는 의미입니다.)

이렇듯, 해당 어플리케이션에 적합한 GC 알고리즘을 선택하는 것도 튜닝의 중요한 포인트인데, 이 경우에는 JVM의 옵션에서 확인할 수 있는 GC 모니터링 기능을 작동하여 GC 로그를 직접 모니터링하면서 최적의 옵션을 찾는 것이 더 도움이 됩니다.

☐ Verbose garbage collection

일반적인 적정치는 GC 시간 : 2초 이하, GC 주기 : 10초 이상 이며 GC 사이클이 정상 범위를 넘어갈 경우에는 Heap 사이즈와 GC 세부 옵션들을 통해서 튜닝하여야 합니다.
다음으로 약간은 도움이 될 만한 GC 관련 기타 옵션을 소개하자면 Class GC 옵션이 있습니다. Class GC 여부에 대한 옵션으로 남아 있는 클래스가 살아있지 않는다는 것이 판명될 때마다 JVM 은 클래스를 메모리에서 해제하며 이러한 작용은 성능에 영향을 미치게 됩니다. -Xnoclassgc 옵션으로 설정 가능하며 같은 클래스를 여러 번 로딩하고 해제하는 오버헤드를 제거하기 위하여 사용됩니다.

그리고, 만약 사용하는 객체들의 사이즈가 대부분 큰 편이고 라고 생각된다면 `-Xloa` 옵션을 사용해 보는 것도 성능에 도움이 될 수 있습니다. `64k` 이상의 큰 객체들만을 담는 `LOA(large object area)`을 활성화 하여 이 영역 안에다 대용량 객체를 몰아 놓는 방식입니다.

마지막으로 –Xdisableexplicitgc 옵션이 있습니다. 이는 용어 그대로 명시적인 GC 를 작동하지 못하게 하는 옵션입니다. 여기서 명시적인 GC란 System.gc() 명령에 의한 사용자 GC를 의미하며, GC의 종류와 무관하게 항상 Full GC를 발생하므로 특별한 경우가 아니라면 일반적으로 System.gc() 는 가급적 권장하지 않으며 Java 의 자동화된 GC 알고리즘에 맡기는 것을 추천합니다.

여기서 번외로 JVM 튜닝에 대한 다른 이야기를 하자면, 많은 고객분들이 질문하기도 하고 관리자로서 가장 많이 하는 질문 중의 하나가 CPU 당 몇 개의 WAS 인스턴스, 즉 JVM 인스턴스를 돌리는 것이 최적인가라는 질문입니다. 그런데, 이 질문 만큼 답변하기 어려운 질문도 없습니다. 그건 어플리케이션에 따라서, CPU 칩에 따라서 너무도 달라지는 내용이기 때문입니다. 특정 어플리케이션은 CPU 를 많이 쓸수도 있고, 특정 어플리케이션은 메모리를 많이 사용할 수 있고 특성이 다 다릅니다. 이런 상황에서 일률적으로 CPU 당 몇 개의 WAS 인스턴스를 추천합니다 라는 이야기는 사실 거짓말이 될 수 밖에 없습니다.

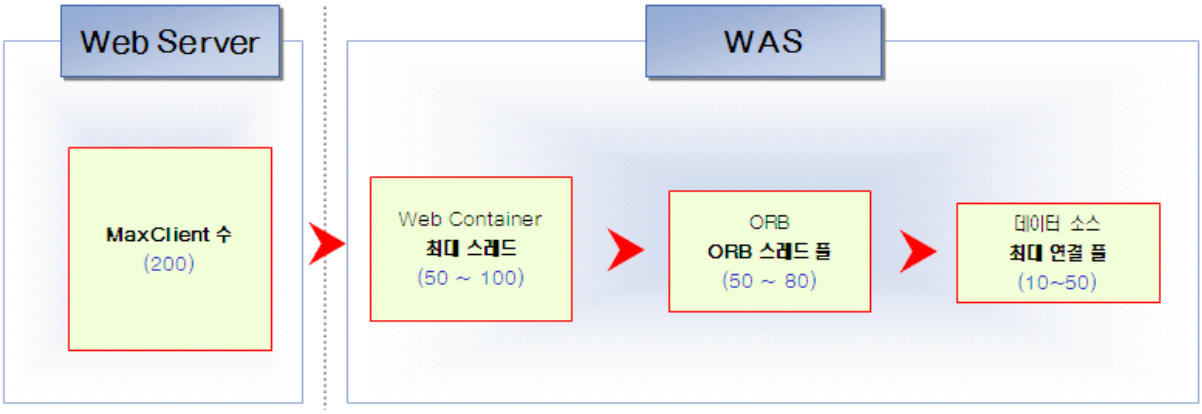
더군다나, IBM 의 Power 칩만 예로 들자면, Power 5 칩과 이번에 출시된 Power 7 칩과의 성능 차

이는 어마어마 합니다. 거기다가 일반적인 Intel 칩과 비교하자면 그건 절대 1:1로 비교가 불가능 합니다. 이런 상황에서 단순하게 CPU 당 몇 개의 WAS 인스턴스가 좋나요라는 질문은 결국 우문이 될 수 밖에 없습니다. 그래도 이런 상황이 오게된다면 제 경험으로 추천드리는 현답은 성능 테스트를 통해서 계산을 해야합니다란 답변입니다. 실제로 많이 사용하는 방법도 어플리케이션 제작 후에 실제 성능 테스트에서 WAS 인스턴스를 2대, 4개 등등으로 변경하여 부하 테스트를 직접해보고 해당 환경에서 가장 좋은 성능을 내는 WAS 인스턴스 개수를 찾는 것 입니다. 참고하시기 바라겠습니다.^^&;

지금까지 IBM WAS 의 JVM 에 대한 튜닝에 대해 많은 말씀을 드렸는데, 어짜피 3사가 동일하게 JVM 을 사용하는 것이므로 인터넷에 돌아다니는 다양한 자료를 참조하는 것도 튜닝을 배우기 위한 좋은 방법중의 하나입니다. IBM 의 JDK 나 SUN(이제는 Oracle 이라고 해야하나요^^&) 의 Hotspot 의 차이점이 있기는 하지만 기본 뼈대는 같으므로 서로의 자료를 참조하는 것도 훌륭한 공부법 중의 하나라는 점 잊지 마시길 바라겠습니다.

Part 4. System 대기열 튜닝

이번 파트에서는 System 대기열 튜닝에 대해서 말씀드리도록 하겠습니다. System 대기열이란 쉽게 이해하도록 설명하자면 요청이라는 물이 흐르는 파이프라인에서의 그 흐름에 관련된 사항입니다. 클라이언트가 요청을 보내면 먼저 웹서버에서 받고, 이를 다시 WAS 로 보내면 웹 컨테이너가 받습니다. 그 후에 EJB 요청인 경우 이를 다시 ORB 객체로 보내고 데이터 소스를 통해서 데이터베이스 처리 작업을 하게 됩니다. 이를 잘 살펴보면 마치 하나의 흐름처럼 보이지 않나요? 이 흐름은 연속적으로 연결된 파이프라인 같아서 이 수치들의 조정, 즉 파이프가 넓거나, 좁은것에 따라서 실제로 안에서 흘러가는 요청의 흐름의 빠르기(성능)에 많은 영향을 미칠 수 있습니다. 결국, System 대기열 튜닝의 가장 중점 사항은 흐름의 중단, 즉 병목현상이 없이 요청이 처리될때까지 흐름을 가장 빠르게 지속적으로 연결할 수 있게 해주는 것이며, 추가적으로 가급적 하드웨어가 가지고 있는 리소스 자원을 충분히 활용할 수 있도록 파이프라인의 두께를 넓게 하는 것 입니다. 일반적으로 이런 흐름에서 각각의 파이프의 지점들은 웹 서버의 MaxClient 옵션과 WAS 의 Web Container 최대 스레드 설정, EJB 에 대한 ORB 스레드 풀, 데이터소스 최대 연결 풀이 될 수 있으며 하단의 그림처럼 MaxClient > Web Container 최대 스레드 > ORB 스레드 풀 > 데이터소스 최대 연결 풀 의 형태를 갖추도록 하는 것이 좋습니다.



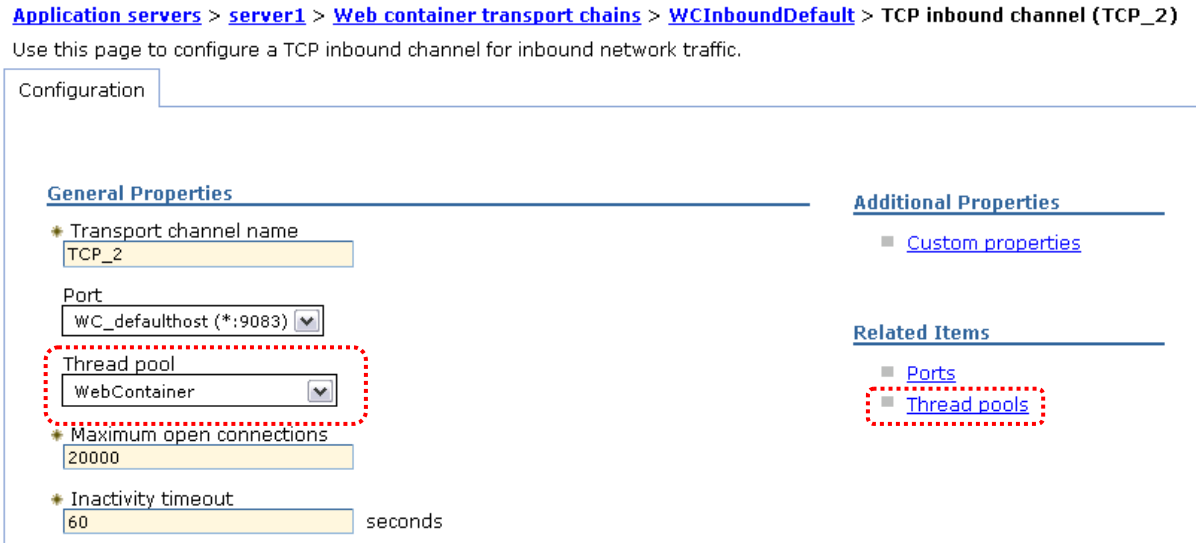
마치 삼각형을 오른쪽으로 90도정도 회전한듯한 파이프라인의 형태가 알려진 가장 이상적인 튜닝입니다. 그것은 보통 위에 첨부한 그림처럼 오른쪽으로 갈수록 파이프의 라인이 더 좁아지는 것인데 그것은 오른쪽으로 갈수록 요청에 대한 실제 처리해야할 작업이 많고 시간이 더 오래 걸리기 때문입니다. 예를 들어 맨 왼쪽의 웹 서버에서 요청을 한 개 처리하는데 1 초가 걸린다면 WAS 의 웹컨테이너에서 요청을 처리하는데 5초 정도가 걸릴 수 있습니다. 이를 성능의 관점에서 최적화 하려면 웹 서버에서는 동시에 5개의 요청을 처리해야 하며 WAS의 웹 컨테이너에서는 그대로 한 개한 요청을 처리하는 것이 가장 이상적인 그림이 됩니다. 바로 순간의 중단없이 지속적으로 흐를 수 있는 상태가 되는 것입니다.

이제 위의 그림에 나와있는 각 파이프의 지점, 옵션들에 대해서 좀 더 자세히 설명하도록 하겠습니다. 먼저 MaxClients 옵션은 WAS가 아니라 웹 서버에 있는 옵션이며, 웹 서버 설정 파일인 httpd.conf 파일에서 해당 내용을 확인할 수 있습니다.(단, 웹 서버가 Windows 가 아닌 다른 OS 에서 설치되었을 때의 기준입니다. Windows 는 MaxClients 옵션을 지원하지 않습니다.)

```
ServerLimit           1
ThreadLimit           1024
StartServers          1
MaxClients            1024
MinSpareThreads       1
MaxSpareThreads       1024
ThreadsPerChild       1024
MaxRequestsPerChild   0
```

웹 서버에서 동시에 처리할 수 있는 최대 HTTP 클라이언트 요청 수를 의미하므로 실제 요청량을 기준으로 값을 설정해야 합니다. 실제 요청보다 MaxClients 수가 작으면 웹 서버에서 처리해서 WAS 로 넘기지 못하고 그대로 요청을 튕겨 내버리는 상황이 발생됩니다. 그럼 당연히 성능만이 아니라 서비스 운영에도 문제가 발생하겠죠.

WAS 로 넘어오게 되면 가장 먼저 웹 서버의 요청을 앞에서 받는 것이 WAS 에 내장되어 있는 웹 컨테이너 입니다. 웹 컨테이너에서 HTTP 요청을 받으면 하나의 스레드를 통해서 들어온 HTTP 요청을 처리합니다. 따라서 여기서 성능에 민감한 옵션은 실제로 요청을 처리하는 스레드에 관련된 옵션이며, 이는 다시 말하면 해당 스레드 풀의 최대 스레드 수 입니다. 관리콘솔에서 Application servers > 해당 서버 > Web_container_transport_chains > WCInboundDefault > TCP invound channel 메뉴를 클릭하면 하단처럼 스레드 풀에 매핑된 별명(alias)을 확인할 수 있습니다.



해당 스레드 풀에 할당된 정확한 수치를 알고자 한다면 Thread pools 옵션을 클릭하면 하단의 그림처럼 해당 풀의 최소, 최대 사이즈를 확인할 수 있습니다.

Application servers > testServer01 > Web container transport chains > WCInboundDefault > TCP inbound channel (TCP_2) > **Thread pools**

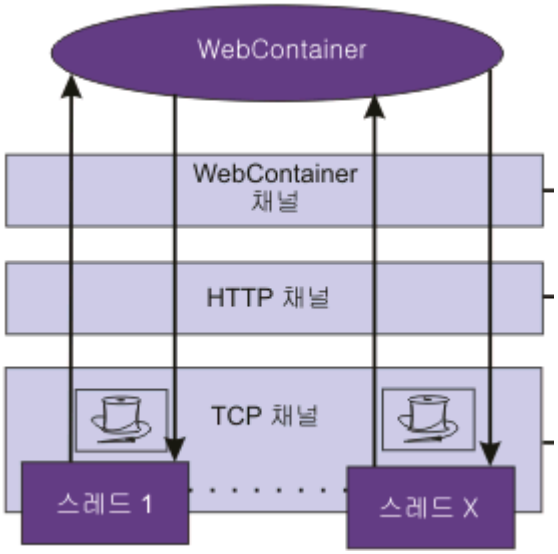
Use this page to specify a thread pool for the server to use. A thread pool enables server components to reuse threads instead of creating new threads at run time. Creating new threads is typically a time and resource intensive operation.

Preferences

New Delete

Select	Name	Description	Minimum Size	Maximum Size
You can administer the following resources:				
☐	Default		20	20
☐	ORB.thread.pool		10	50
☐	SIBFAPInboundThreadPool	Service integration bus FAP inbound channel thread pool	4	50
☐	SIBFAPThreadPool	Service integration bus FAP outbound channel thread pool	4	50
☐	SIBJMSRAThreadPool	Service Integration Bus JMS Resource Adapter thread pool	35	41
☐	TCPChannel.DCS		5	20
☐	WMQCommonServices	WebSphere MQ common services thread pool	1	40
☐	WMQJCAResourceAdapter	wmqJcaRaThreadPoolDescription	5	25
☐	WebContainer		50	50
☐	server.startup	This pool is used by WebSphere during server startup.	1	3
Total 10				

그런데 여기서 하나 더 짚고 넘어갈 것이 있는데, 우선적으로는 웹 컨테이너에서 요청을 처리하는 스레드의 최대치가 요청을 처리하는데 중요한 요소이긴 하지만 튜닝할 때 꼭 살펴볼 것 중의 하나는 그 앞단에 실질적으로 HTTP 요청이 들어오는 통로인 웹 컨테이너 채널과 HTTP 전송 채널, TCP 전송 채널입니다. 그러나 이 부분은 이미 18강에서 많은 부분 설명했기 때문에 해당 강좌를 참조하시기 바라겠습니다.



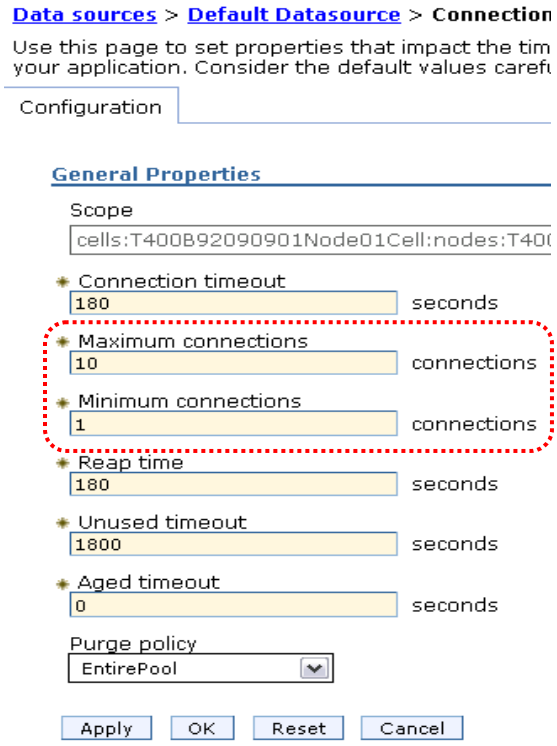
참조: IBM WAS v7 Information Center

다음으로 튜닝해야 할 것은 EJB 요청일 경우에 해당 객체를 처리하기 위한 ORB 객체의 스레드 풀 옵션 조정입니다. 관리콘솔에서 Application servers > 해당 서버 > Container Service > ORB Service 메뉴를 클릭하면 메뉴의 하단에서 확인할 수 있습니다.

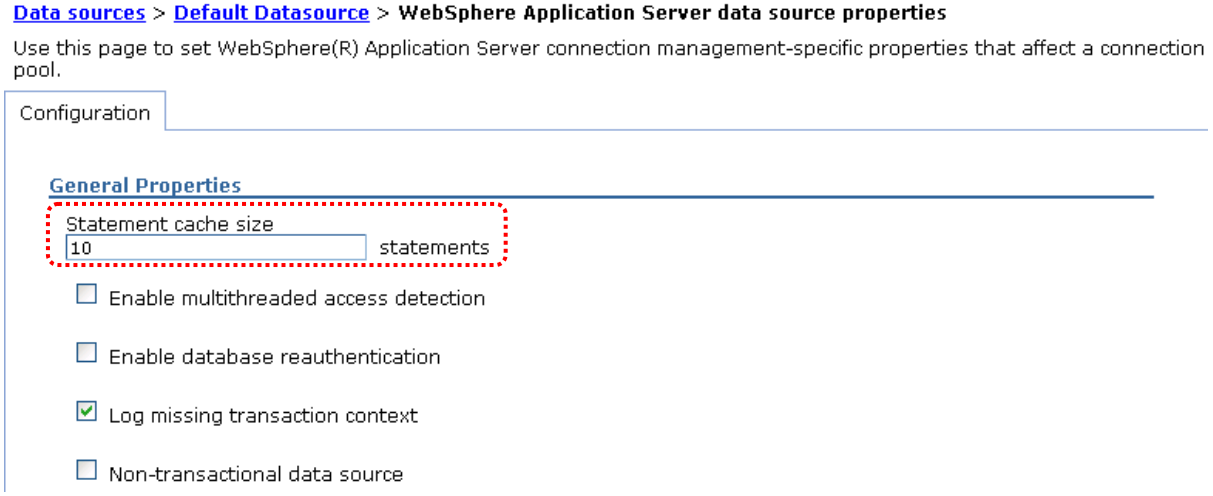
WAS 에서 EJB 클라이언트(servlet or 다른 EJB)가 EJB 메소드 요청을 처리하기 위해 풀에 넣을 수 있는 스레드 수를 조정하는 것으로, EJB 클라이언트(servlet, 다른 EJB)가 서로 다른 JVM에서 실행될 때에만 ORB 객체를 사용하고 스레드 풀에 들어가게 됩니다. EJB 클라이언트에서 비교적 짧게 EJB를 호출하면 웹컨테이너 최대 스레드 수보다 작게 설정하고(1/2정도), 그렇지 않으면 웹컨테이너 최대 스레드 수보다 크지 않게 설정하면 됩니다.

팁을 하나 드리자면, 서버 설정 강좌에서 이미 설명드렸지만 해당 메뉴에 Pass by reference 라는 옵션이 있습니다. 해당 옵션은 ORB 가 파라미터를 어떠한 방식으로 보낼지를 설정하는 것으로 당연히 객체를 복사해서 전달하는 것보다는 당연히 메모리 레퍼런스를 통해서 전달하는 것이 성능상의 이점이 있습니다. 그렇기 때문에, EJB 어플리케이션에서 성능을 위해 많이 사용됩니다.

마지막으로는 DB 와의 실제 연결을 위한 데이터 소스 연결 풀입니다. 데이터베이스 사용 시 connection 재사용을 위한 connection 풀을 의미하며 웹 컨테이너의 최대 스레드 수보다 작게 설정하시는 것이 좋습니다. 참고로 해당 옵션에 대한 설명은 DB 연결 강좌에서 자세하게 되어 있으므로 좀 더 깊은 내용을 알고 싶으신 분들은 해당 내용을 다시 참조하시기 바라겠습니다.



여기서도 팁이 하나 더 있습니다. 관리콘솔에서 데이터 소스 > 해당 데이터 소스 > WebSphere Application Server data source properties 메뉴를 클릭하면 다음과 같은 옵션들을 볼 수 있습니다.

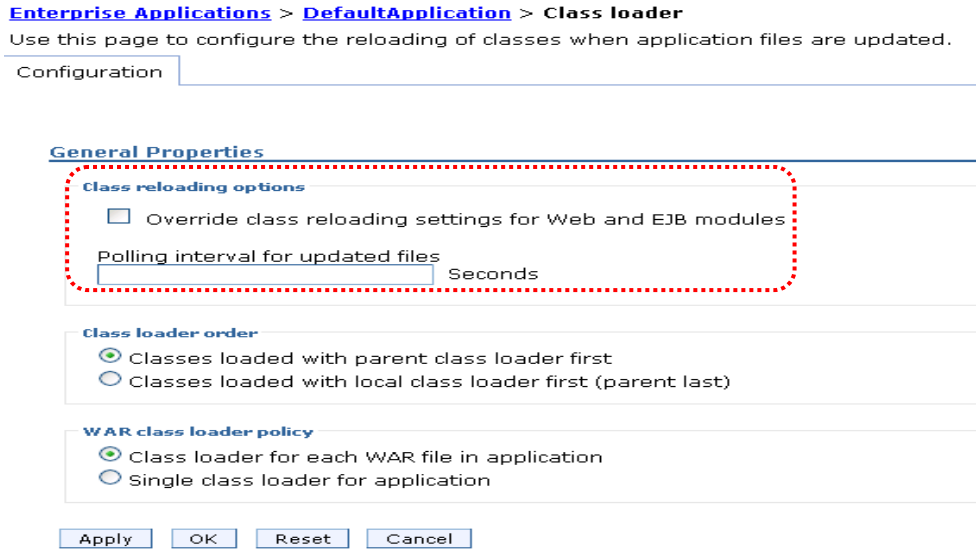


그 중에서 Statement cache size 옵션은 하나의 Connection 에서 캐시 가능한 Statement 를 의미합니다. 보통 Java 에서 많이 사용하는 PrepareStatement 를 생각하시면 됩니다. 호출되어진 PrepareStatement 를 캐시하여 재사용하므로 사용하지 않았을 때와 비교해서 테스트 해보면 10% 에서 20% 정도의 성능 향상이 있으므로 적절한 값을 입력하여 튜닝 하시기를 바라겠습니다. (단, 이 수치를 계산하실 때 주의점은 WAS 전체가 아니라 하나의 Connection 에 대한 수치라는 것입니다. 그렇기 때문에 한번의 요청에 얼마나 많은 PrepareStatement 문장을 이용해서 DB 에 쿼리를 보내는지를 계산하고 지정하시면 됩니다.)

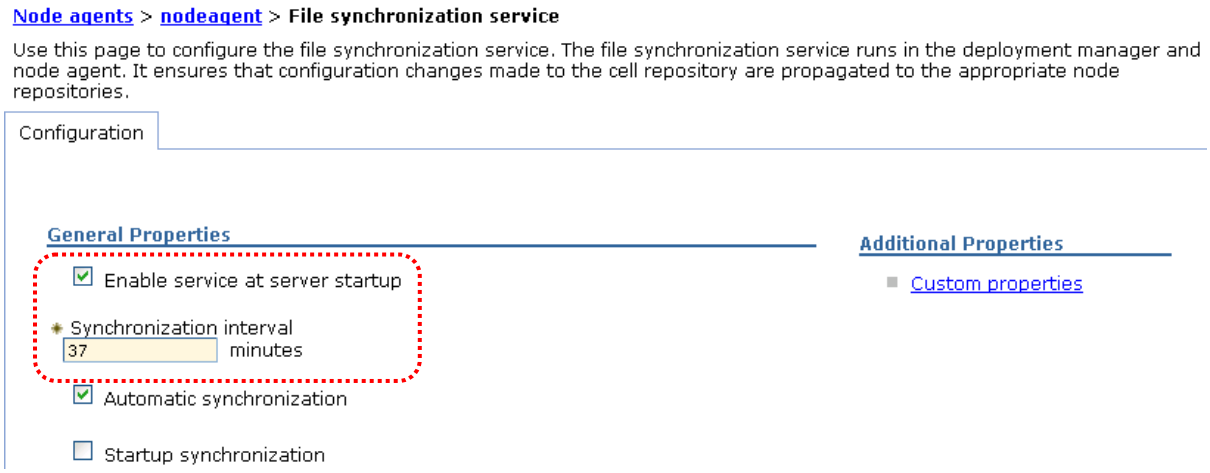
Part 5. 기타 튜닝에 도움이 되는 것

Part 2에서 4를 통해서 WAS 의 가장 중요한 부분에 대한 기본 튜닝에 대한 이야기는 마무리 하였습니다. 그러나 이대로 끝내기는 조금 아쉬워서 간단하게 몇가지 튜닝에 도움이 되는 옵션들을 언급하고 넘어가도록 하겠습니다.

먼저 하단에서 확인가능한 어플리케이션에 대한 class reloading 옵션은 가능한 꺼주는 것이 성능에 도움이 됩니다. 어플리케이션 마다 주기적으로 자기 클래스에 변화가 없는지 매번 체크하는 작업은 불필요한 CPU 작업이며 성능에 좋지 않습니다. 만약 반드시 이 기능을 쓰셔야 한다면 체크 주기를 늘리기를 성능의 입장에서 권장드립니다.



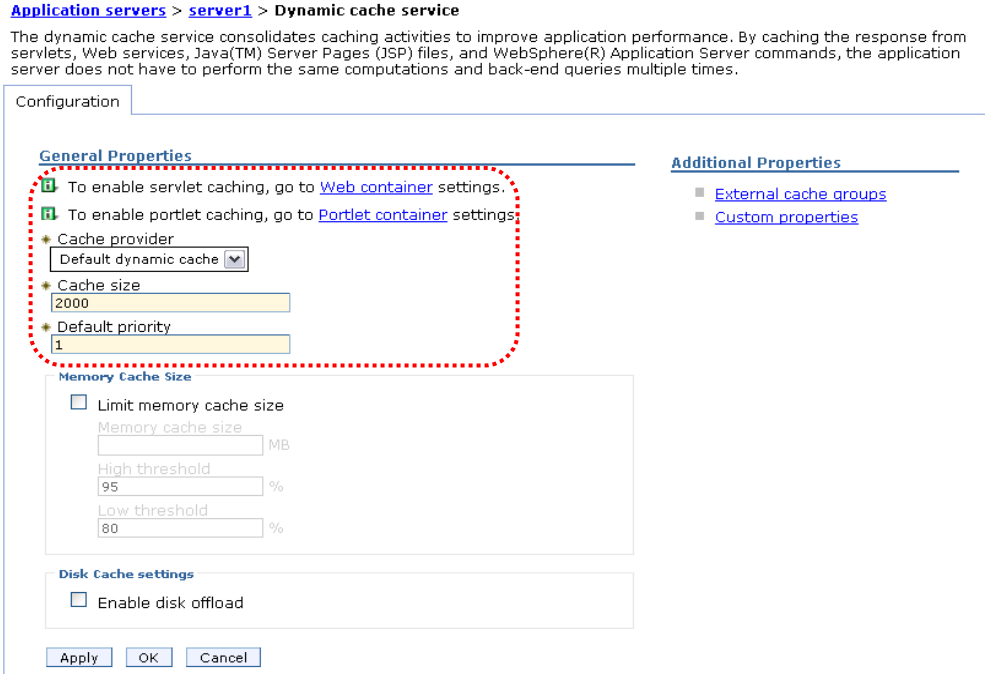
위의 옵션과 연이어서 성능의 관점에서 추가적으로 생각해 보실 것은 노드 에이전트의 파일 동기화 작업입니다. 기본 값이 1분으로 되어 있기 때문에 해당 노드 에이전트는 1분마다 주기적으로 Dmgr 을 체크하여 변경된 설정사항을 동기화 합니다. 관리콘솔에서 System administratioin > Node agents > 노드 에이전트 이름 > File synchronizatioin service 를 클릭하면 하단의 그림처럼 확인할 수 있으며 빈번하고 주기적인 작업입니다. 그렇기 때문에 해당 작업은 가급적 시간을 늘려주는 것이 비록 미미하다고 하여도 CPU 사용을 줄이고 전체 성능에 좋은 영향을 줄 수 있습니다.



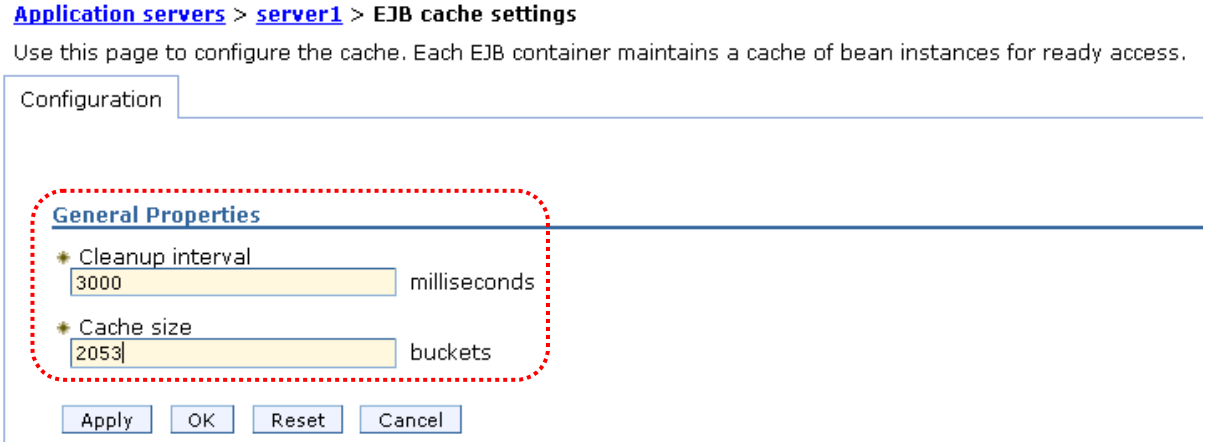
다음으로 가급적 로그들을 적게 남길수록 성능에 더 도움이 됩니다. 당연히 문제를 분석해야하는 특수한 상황이 아니라면 운영중에 트레이스는 남기지 않는게 좋으며, 필요없으면 로그를 남기지 않게 하는 것도 성능에 좋은 방안입니다. 하단의 로그 리스트들과 내용을 확인해보세요.^^&;

로그 파일명	내용	활용성
access_log	웹서버를 통해 접근한 사용자들에 관한 내용을 포함한 텍스트 파일	서버별, 페이지별 사용 내역 및 사용량에 대한 통계적 자료 또는 수행성능 관련 데이터 수집
plugin.log	웹서버 플러그인의 동작중 발생한 행위와 Error 메시지등이 기록된 텍스트 파일	웹 서버와 WAS 간 동작 상태 상세 모니터링, 장애 원인 분석, 튜닝 요소 분석
SystemOut.log	WAS 자체의 동작 메시지, 애플리케이션의 Console Out, Exception 및 Error 메시지 등이 기록된 텍스트 파일	WAS의 동작 상태 상세 모니터링, 장애 원인 분석, 튜닝 요소 분석
SystemErr.log	Exception 및 Error 내역만 기록된 텍스트 파일	WAS의 장애 원인 분석
native_out.log	WAS 운영 중 OS가 던진 output 들이 기록된 텍스트 파일	WAS의 장애 원인 분석, 튜닝 요소 분석
native_err.log	WAS 운영 중 OS가 던진 error 메시지 들이 기록된 텍스트 파일	WAS의 장애 원인 분석, 튜닝 요소 분석
activity.log	WAS에서 일어나는 모든 행위들을 자세히 기록한 바이너리 로그로써 Log Analyzer 라는 툴로 분석	WAS의 동작 상태 상세 모니터링, 장애 원인 분석, 튜닝 요소 분석
tranlog	WAS 자신이 자동 트랜잭션 복구를 위해 사용하는 바이너리 로그	Transaction auto recovery
HeapDump	특정 시점에서의 JVM Heap Memory의 내용과 운용 상황을 보여주는 바이너리 형식 덤프 파일로써, HeapRoots 라는 툴로 분석	WAS의 Memory leak 분석
javacore.txt	특정 시점에서의 JVM 내 스레드들의 운용 상황을 보여주는 텍스트 형식 덤프 파일	WAS의 장애 원인 분석, 튜닝 요소 분석
FFDC	First Failure Data Captuer – 장애시의 WAS의 설정 및 환경, 운용 상황 등을 수집하여 만든 로그 파일들의 모음. IBM 서비스 센터에서 사용하기 위한 로그	WAS의 장애 원인 분석, 튜닝 요소 분석

또한, 성능 튜닝에서 고려해 보실 수 있는 것은 가급적 캐시를 사용하는 것입니다. 캐시라는 것은 이미 아시겠지만, 한번 호출해서 가지고 온 정보를 메모리에 임시로 저장해 두었다가 다음번 요청이 오면 다시 처리하는 것이 아니라 미리 캐시해둔 결과 정보를 중간에서 반환해 주는 작업입니다. 한 번 캐시된 요청인 경우 처리 작업 없이 바로 결과를 반환하기 때문에 성능에 아주 많은 도움이 됩니다. WAS 관리콘솔에서 Application servers > 해당 서버 > Dynamic cache services 를 클릭하면 하단처럼 WAS 에서 제공하는 캐시 설정화면을 확인할 수 있습니다. 해당 설정을 이용하여 캐시를 사용하면 보다 더 나은 성능을 제공할 수 있습니다.



위의 내용에 덧붙여, Application servers > 해당 서버 > EJB cache settings 메뉴에서 EJB cache 관련 부분도 조정가능하다는 것도 참고하시기 바라겠습니다.



마지막으로 IBM WAS 에서는 튜닝 및 분석을 위해서 여러가지 다양한 툴을 제공하므로 해당 툴을 효과적으로 이용하시는 것이 반복작업을 막고 적절한 튜닝을 하는 지름길 중의 하나입니다. 하단의 WAS의 실시간 모니터링 및 진단 툴 리스트를 참고하시기 바라겠습니다.
(보통 해당 분석 툴들은 IBM Alphaworks 에서 무료로 다운로드 가능합니다.
http://www.alphaworks.ibm.com/java)

WAS의 실시간 모니터링및 진단 툴

툴 이름	역할	비고
Tivoli Performance Viewer	실시간 모니터링을 위한 GUI 툴	WAS Admin Console 에 내장
Tivoli Performance Advisor	실시간 모니터링 결과를 토대로 WAS 자원들의 적정한 설정값을 제안해 주는 GUI 툴	WAS Admin Console 에 내장
Log Analyzer	activity.log 라는 WAS의 바이너리 로그를 분석해 주는 GUI 툴	번들로 제공되는 GUI Application
Thread Analyzer	JVM의 현재 스레드들의 동작 상태를 보여주는 코어 덤프 파일을 분석할 수 있게 해 주는 GUI 툴.	상동
Heap Analyzer	JVM의 Heap (메모리) 사용 상태를 담은 HeapDump 파일을 분석하기 위한 GUI 툴	상동
GC Analyzer	JVM의 GC(Garbage Collection) 상태를 분석하기 위해 사용되는 GUI 툴	상동
IBM Support Assistant	기존에 Heap Analyzer, GC Analyzer 등으로 분리되어 있던 분석 툴을 ISA 라는 이름으로 통합하여 편의성을 꾀하고 Lab 에 바로 덤프를 보내거나 분석 리포트 제공이 가능한 툴	상동

팁1) IBM WAS v7 의 fixpack 9 번부터 WAS 튜닝을 손쉽게 할 수 있는 스크립트를 선보이고 있습니다. applyPerfTuningTemplate.py 라는 jython 스크립트로 하단과 같이 실행하면 테이블에 표시된 Best Practice 값으로 자동 튜닝됩니다.

```
wsadmin -f applyPerfTuningTemplate.py
[-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile production.properties
```

Parameter	Server default (default.properties template file)	Production environment (production.properties template file)	Development environment (development.properties template file)
JVM Heap Size (MB)	50 min / 256 max	512 min / 512 max	256 min / 512 max
Verbose GC	disabled	disabled	enabled
JVM Diagnostic Trace (Generic JVM Arguments)		-Xtrace:none -Dcom.ibm.xml.xlxp.jaxb.opti.level=3	-Xtrace:none -Dcom.ibm.xml.xlxp.jaxb.opti.level=3
HTTP (9080) and HTTPS (9443) Channel maxKeepAliveRequests	100	10000	10000
Development Mode	disabled		enabled
Server Component Provisioning	disabled	enabled	enabled
PMI	enabled	disabled	disabled
Authentication Cache Timeout*	10 minutes	60 minutes	60 minutes
Data Source Connection Pool Size*	1 min / 10 max	10 min / 50 max	
Data Source Prepared Statement Cache Size*	10	50	
ORB Pass-by-Reference	disabled	enabled	enabled
Thread Pools (Web Container, ORB, Default)	50 min / 50 max, 10 min / 50 max, 20 min / 20 max		5 min / 10 max

참조: IBM WAS v7 Information Center

여기까지 잘 이해가 되시나요? 드디어 20번째 강좌의 마지막 까지 달려왔습니다. 이제는 WAS 를 이해하는 것을 넘어서 각 설정 및 성능에 대해 한번 훑어 보기까지 했으니, 당신은 WAS 전문가.... 라고 말해주고 싶네요..^^& 그러나, 아직은 이른 감이 좀 있죠. 보다 더 중요한 것은 강좌를 따라만 하는 것이 아니라 여기서 배운 내용들을 실제로 써보고 몸으로 배우는 것입니다. 그래야만 진정한 WAS 전문가 반열에 오를 수 있겠죠. 또한, 추가적으로 튜닝 가능한 부분과 옵션을 스스로 찾아 볼 수 있게 되기를 바라겠습니다.

제가 이 강좌를 시작했던 이유는 처음 접하는 사람들에게 쉽고 친숙하게 IBM WAS 를 접하게 해보자는 의도 였으며, 이를 통해 실제 많이 써보게 되기를 바라는 마음이였습니다. 실제로 써봐야 지만 부족한 것이 무엇인지 알고 더 필요한게 무엇인지 알 수 있거든요. 비록 제 생각이지만 처음 강의부터 지금까지 잘 따라오셨다면 제 의도에 맞게 잘 진행된 것이라고 생각하구요. 이제는 슬슬 배운 내용을 실전에서 써볼 수 있게 되는 단계가 되기를 바라겠습니다. 그리고 이번 강좌를 통해서 하나씩 쉽게 따라 해보는 IBM WAS v7 의 2부 강좌도 종료하도록 하겠습니다. 어느덧 횟수로 2년, 만 1년 정도가 지났고 강좌도 20강까지 진행되어서 WAS에 대해서 다루고 싶던 많은 부분을 다룰 수 있었고 소개할 수 있었습니다. 여하튼, 제 강좌로 인해서 IBM WAS 랑 친해지셨는지 모르겠네요? 처음에 이야기 했던 제 강좌의 본래 목적이 잘 달성되었기를 바래봅니다. 그래

도 완전하게 이별이라는 이야기는 좀 그렇구요. 현재 3부를 구상중입니다. 그러나, 좀 처럼 쉽게 나올 것 같지는 않네요. ^^& 그래도 지속적으로 3부 강좌를 진행하도록 노력해 보도록 하겠습니다. 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
- Tuning the application serving environment
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/tprf_tuneprf.html