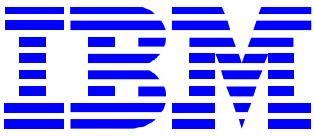
WebSphere Application Server v8.5

장애 사전 예방(Health Management)

(2013. 03.)

IBM SWG WebSphere CTP 이정운 과장(juwlee@kr.ibm.com)

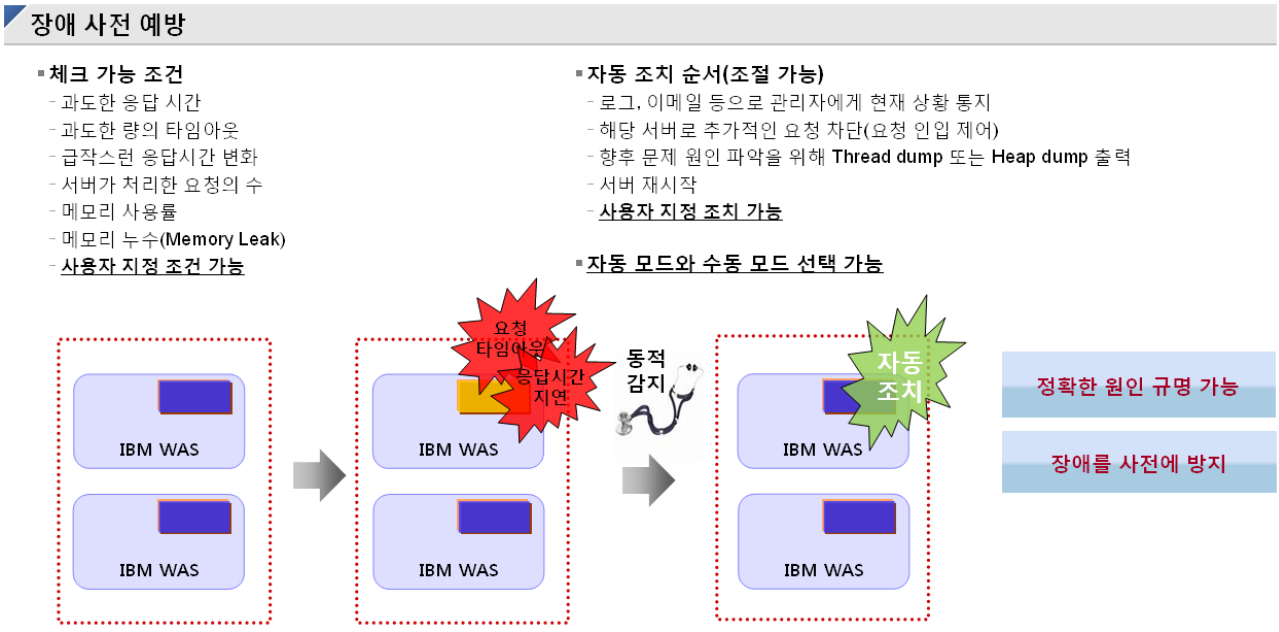


0) 장애 사전 예방(Health Management)에 대한 간략 소개

안녕하세요 freeman 입니다.

이번 강좌에서 소개하고자 하는 IBM WAS v8.5 기능은 장애 사전 예방이라는 기능입니다. 장애 사전 예방이라는 기능은 말 그대로 장애가 발생된 다음에 어떤 액션이나 조치를 뒤 늦게 수행하는 것이 아니라 장애가 발생하기 전에 장애 징후를 감지하여 미리 지정된 조치를 수행하여 가급적 장애를 예방하는 기능을 의미합니다.

이를 이해하기 쉽게 조금 더 보충 설명 하자면, 장애는 한번에 또는 순식간에 발생되지 않습니다. 기본적으로 많은 장애는 장애가 발생되기 전에 장애 징후라는 것을 먼저 나타냅니다. 예를 들어 잘 나오는 응답시간이 1초, 3초, 5초처럼 점점 느려진다는지, 아니면 메모리를 얼마 사용하지 않는 애플리케이션임에도 불구하고 점점 메모리가 해제되지 않고 80%, 90% 이상 지속적으로 사용된다는 것과 같은 장애는 일반적으로 이와 같은 징후를 먼저 나타내며 이런 징후만 사전에 잘 캐치하여 관리자가 원하는 조치를 사전에 취할 수 있다면 문제가 실제 장애가 되거나 큰 장애가 발생 전에 사전에 예방 할 수 있습니다.

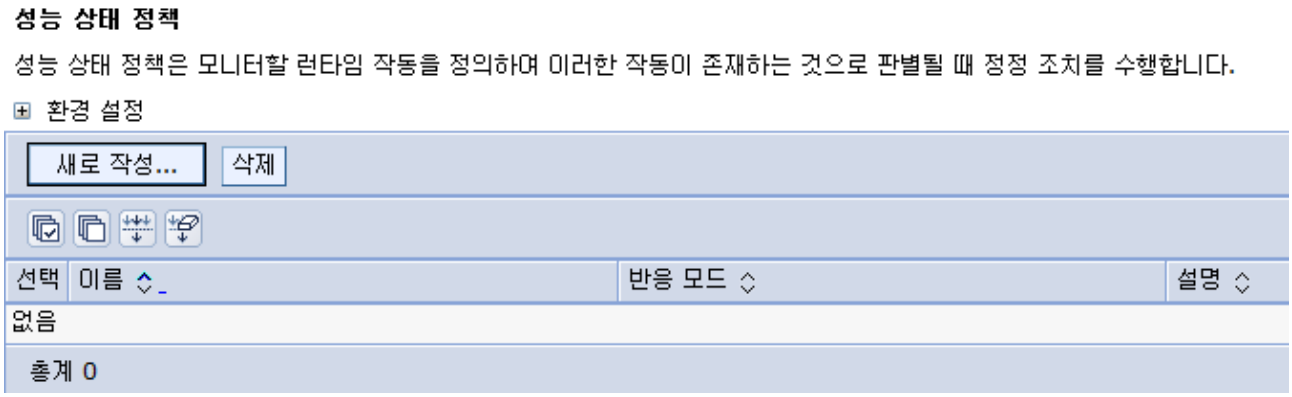


IBM WAS v8.5 는 이러한 장애 사전 예방을 위하여 다양한 체크 가능 조건을 가지고 있으며 이런 조건이 감지되면 관리자가 미리 지정된 조치를 만들어서 수행할 수 있는 기능을 가지고 있습니다. 또한, 이때 사용되는 조건과 조치 방법은 환경에 따라, 사용자가 원하는 형태로 직접 만들 수 있는 기능을 함께 제공합니다.

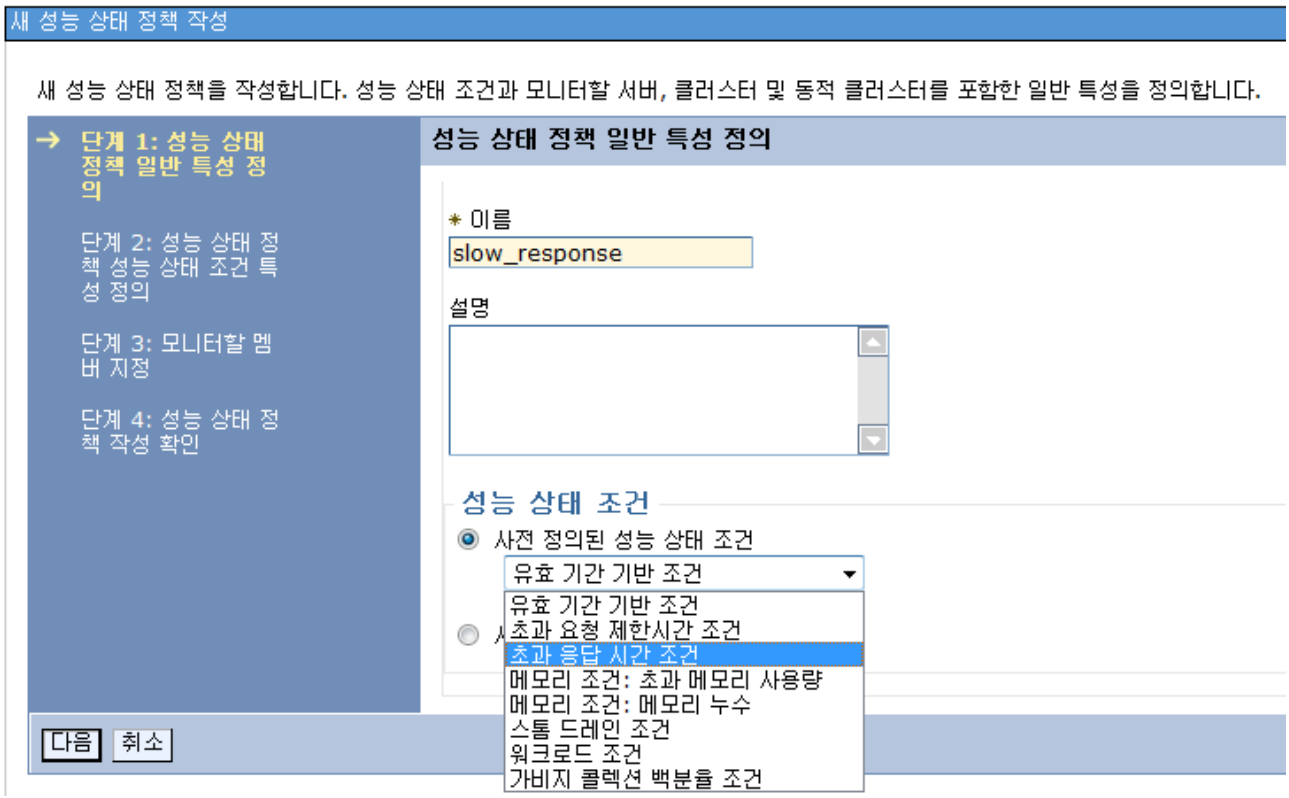
그럼 지금부터 강좌를 통해서 장애 사전 예방 기능을 직접 확인해볼 수 있는 시간을 가지도록 하겠습니다.

1) 장애 사전 예방 설정

1. 장애 사전 예방 설정을 위하여 관리콘솔에서 조작 정책 > 성능 상태 정책(Health Management) 메뉴를 선택합니다.

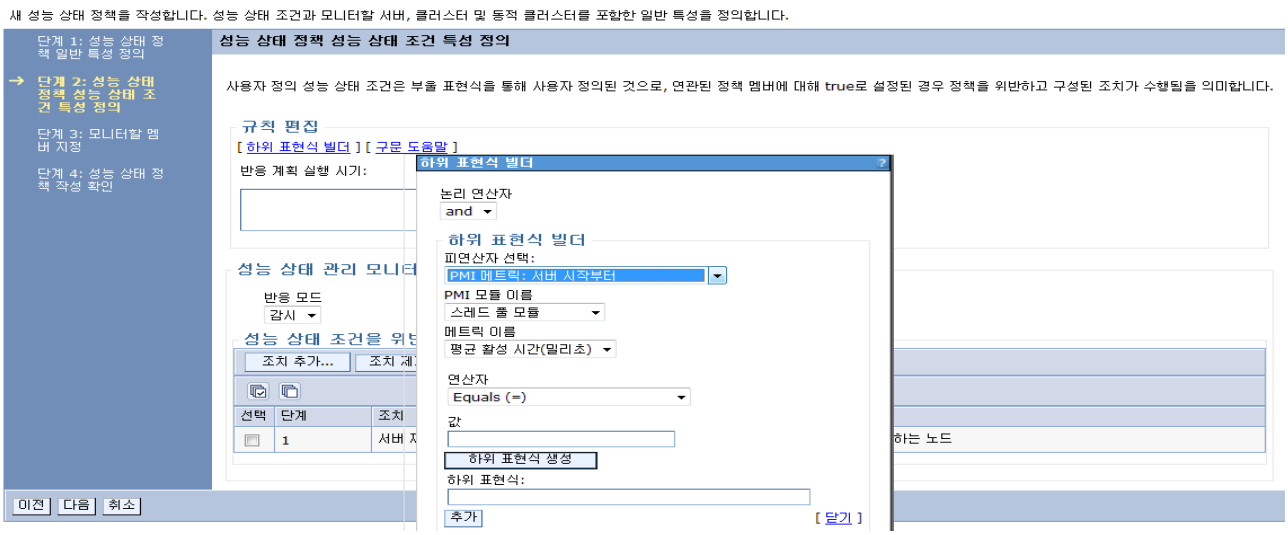


2. 해당 메뉴에서 새로 작성을 클릭 한 후 성능 상태 정책 이름 및 사용하고자 하는 조건을 선택합니다. (본 강좌에서는 샘플로 초과 응답 시간 조건에 대한 작업을 수행해보도록 하겠습니다.)



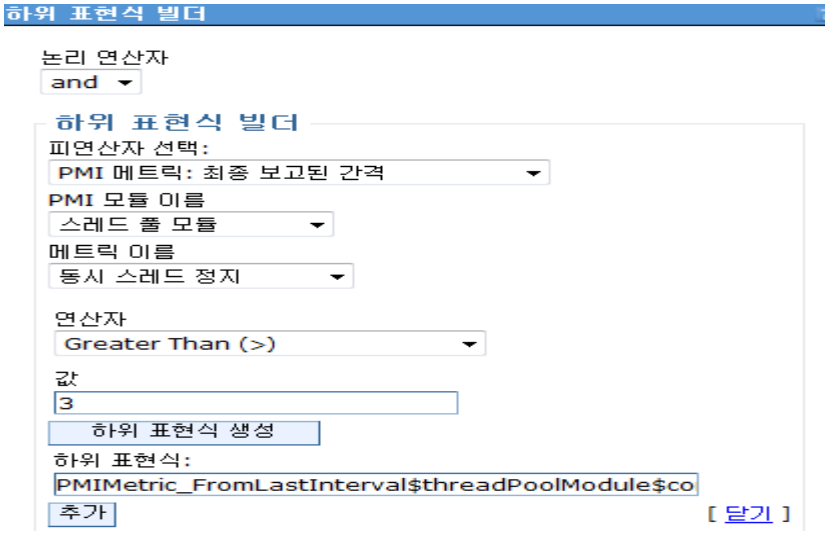
참고적으로 상단의 메뉴에서 보시는 것처럼 미리 만들어진 사전 정의된 성능 상태 조건을 선택해서 사용하실 수도 있으며 사용자 정의 조건을 직접 만들어서 사용할 수도 있습니다.

사용자 정의 조건 생성 예제



사용자 정의 조건을 선택하고 다음을 클릭하면 상단과 같이 하위 표현식 빌더를 통해서 규칙을 만들 수 있는 위자드 형태의 하위 표현식 빌더를 사용할 수 있습니다. 이를 통해서 사용자 환경에 적합한, 원하는 규칙을 손쉽게 만들 수 있습니다.

예 #1 : ThredPool 에서 Hang 현상이 예상되는 Thread 가 3 개 이상일 경우를 감지



PMIMetric_FromLastInterval$threadPoolModule$concurrentlyHungThreads >= 3L

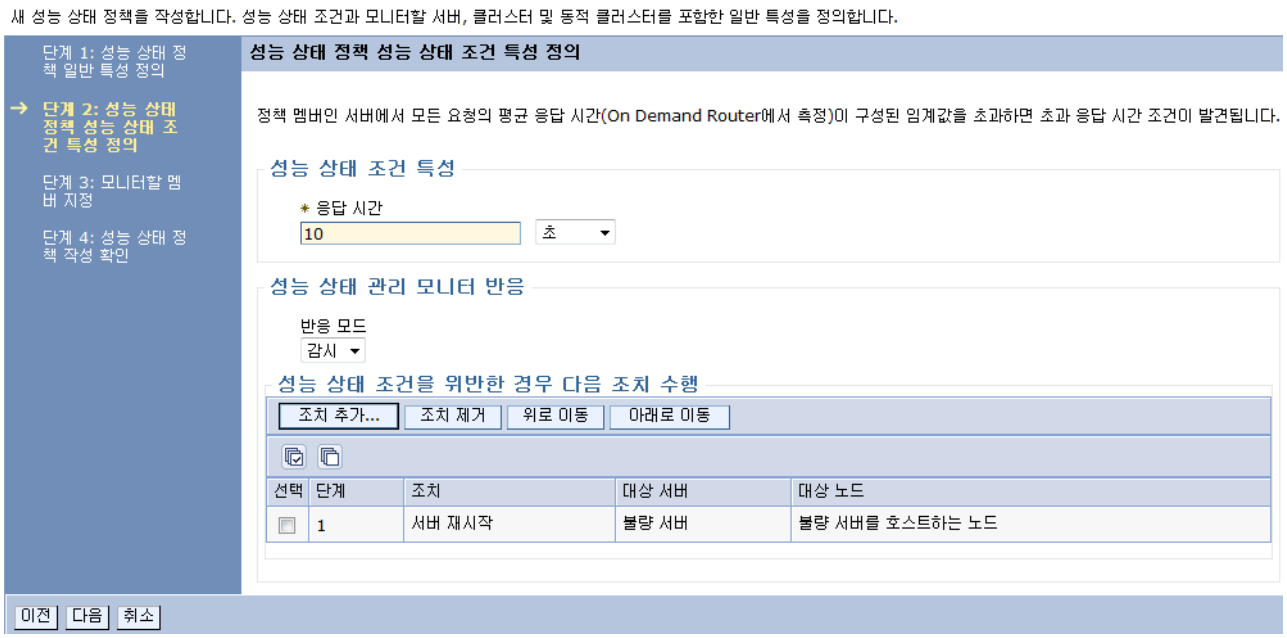
예 #2: Connection Pool 에서 5초이상 WaitTime 이 발생할 경우 감지

PMIMetric_FromLastInterval$connectionPoolModule$avgWaitTime > 5000L

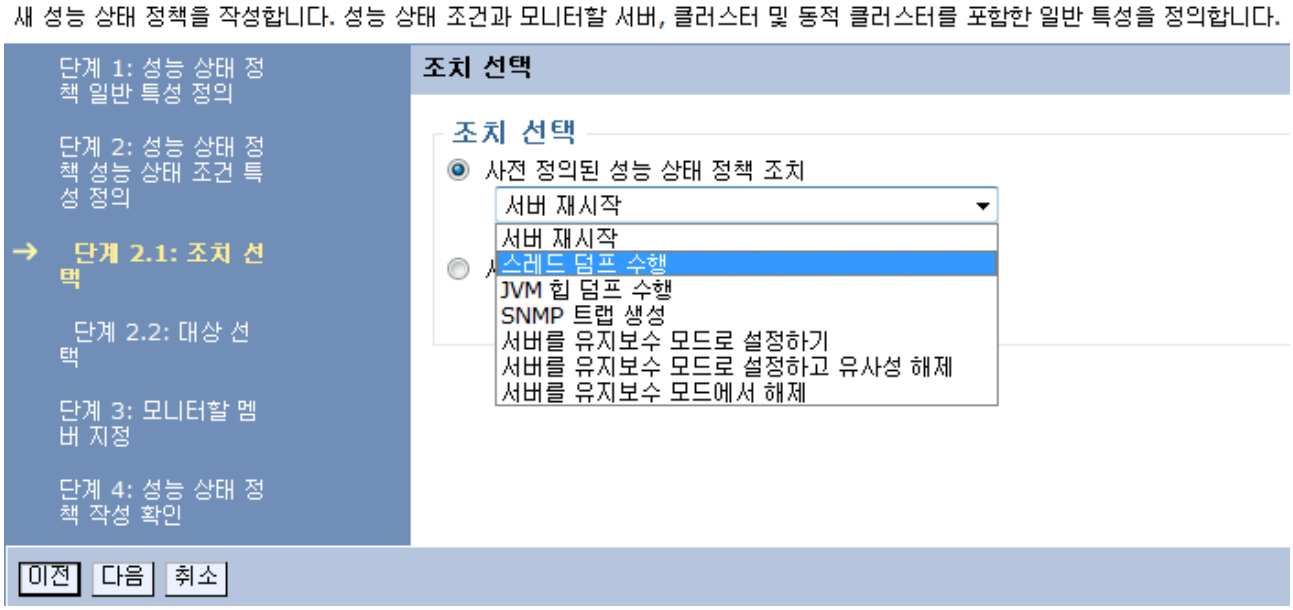
예 #3: 현재 프로세스의 CPU 사용률이 90% 이상일 경우 감지

PMIMetric_FromLastInterval$xdProcessModule$recentCPUUtilization > 90L

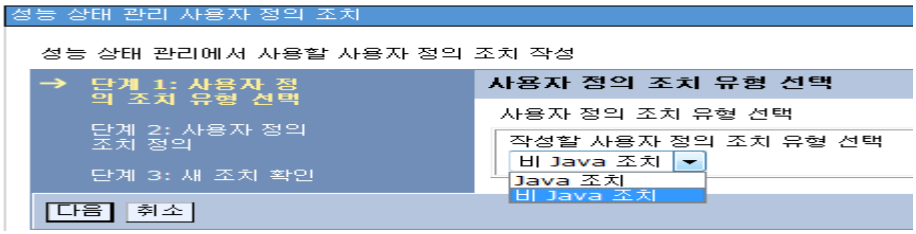
3. 초과 응답 시간 조건의 조건 특성을 입력한 후에 해당 장애 상황이 감지되었을 때 필요한 조치 사항을 확인합니다. 기본적으로 서버 재시작만이 작성되어 있는데 필요한 조치를 더 추가하기 위하여 조치 추가를 선택합니다.



4. 조치 추가를 선택하면 이전에 조건 선택과 비슷하게 미리 만들어진 사전 정의된 성능 상태 정책 조치 중에서 선택이 가능하며 추가적으로 사용자 정의 성능 상태 정책 조치를 직접 만드실 수 도 있습니다. (사용자 정의 성능 상태 정책 조치의 경우에는 Java 애플리케이션 수행이나 일반적인 Script 수행 등을 사용자가 원하는 형태로 만드실 수 있습니다.)

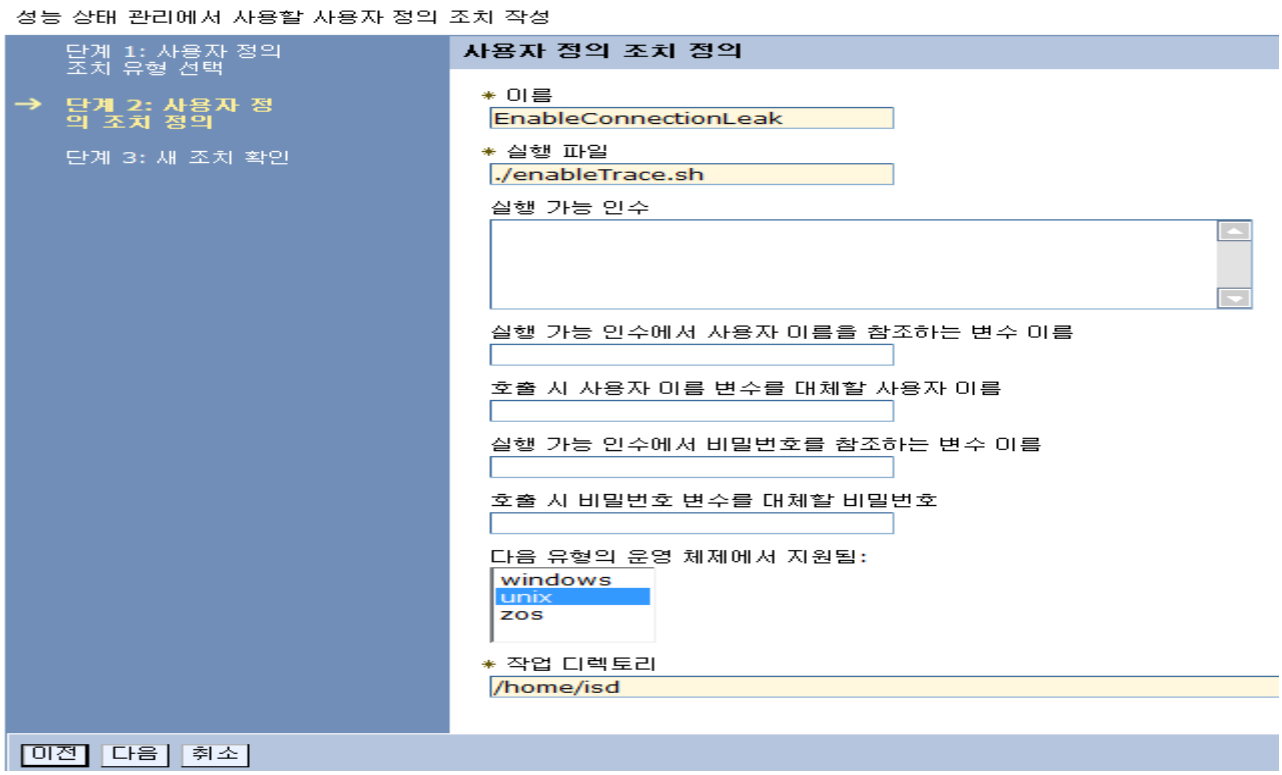


사용자 정의 조치 생성 예제



사용자 정의 조치를 선택하고 다음을 클릭하면 상단과 같이 Java 형태/비 Java 형태의 조치를 만들수 있습니다. 여기서 비 Java 형태의 조치란 일반적으로 수행되는 script 형태를 의미합니다.

예) “ConnLeakLogic=all” trace 옵션을 동적으로 추가하는 runTraceToServer.py 스크립트 수행



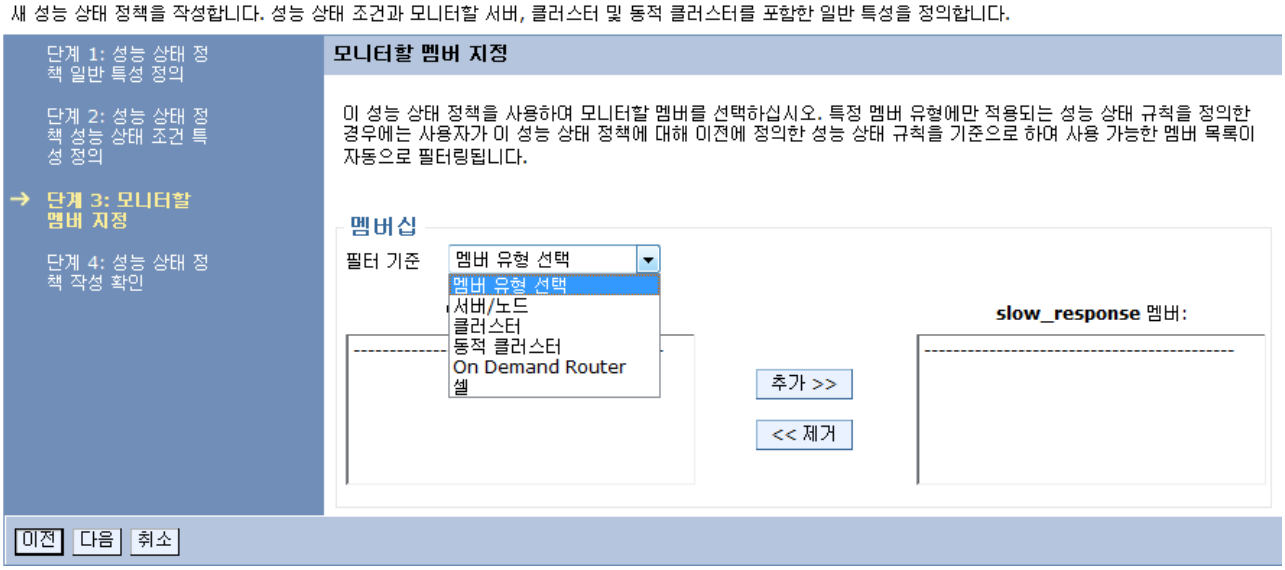
enableTrace.sh

```
echo ‘Node Name = ‘ $node
echo ‘Server Name = ‘ $server
WSADMIN_CMD="/opt/xd/profiles/dmgr/bin/wsadmin.sh -lang jython"
$WSADMIN_CMD -f runTraceToServer.py $server ConnLeakLogic=all
echo '********************************************************************'   >> PoolContents.txt
```

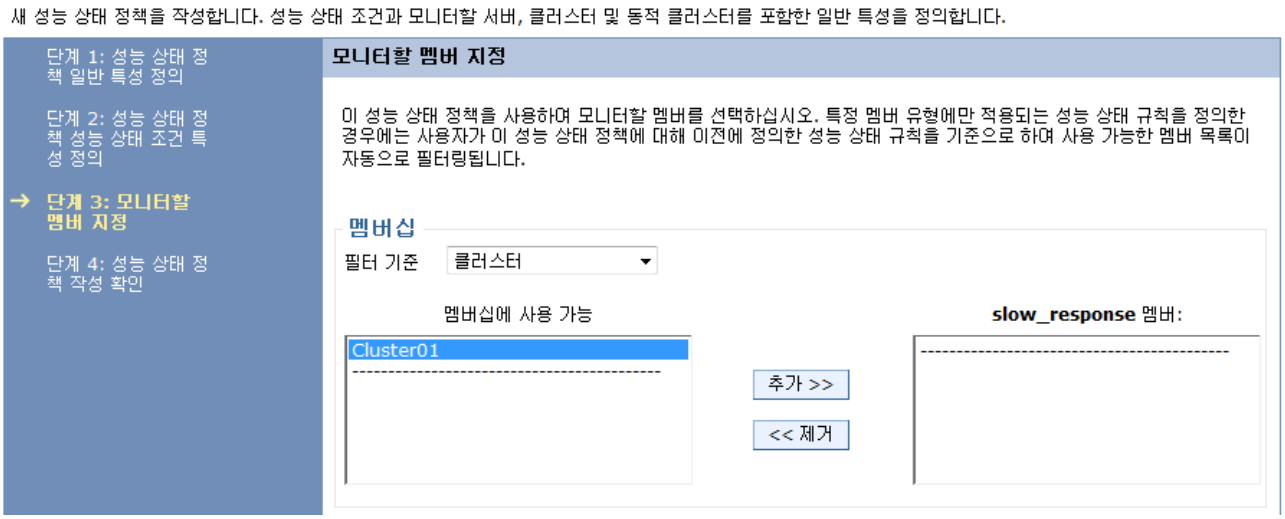
참조 : runTraceToServer.py on http://www.websphere.pe.kr/xe/2052

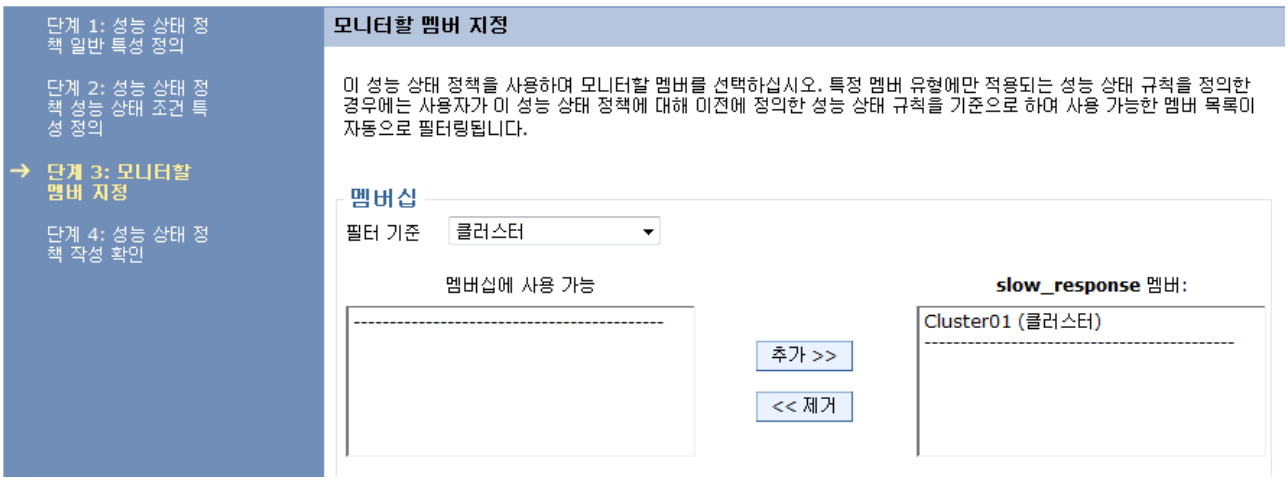
5. 사전 정의된 성능 상태 정책 조치 중에서 JVM 힙 덤프 수행을 선택하고 다음을 누르면 하단과 같이 새로운 조치 수행이 추가된 것을 확인할 수 있습니다. 이를 위로 이동, 아래로 이동 메뉴를 통해서 원하시는 형태의 순서로 변경도 가능합니다.
(추가적으로 성능 상태 관리 모니터링에 대한 반응 모드는 감시와 자동이 있으며 감시는 모니터링만 수행하는 것이며 자동으로 해두어야지만 실제 액션을 자동으로 수행할 수 있습니다.)

6. 다음을 클릭하면 하단과 같이 실제 모니터할 멤버 지정 메뉴로 가게 되며 원하시는 형태의 멤버 유형을 선택할 수 있습니다.

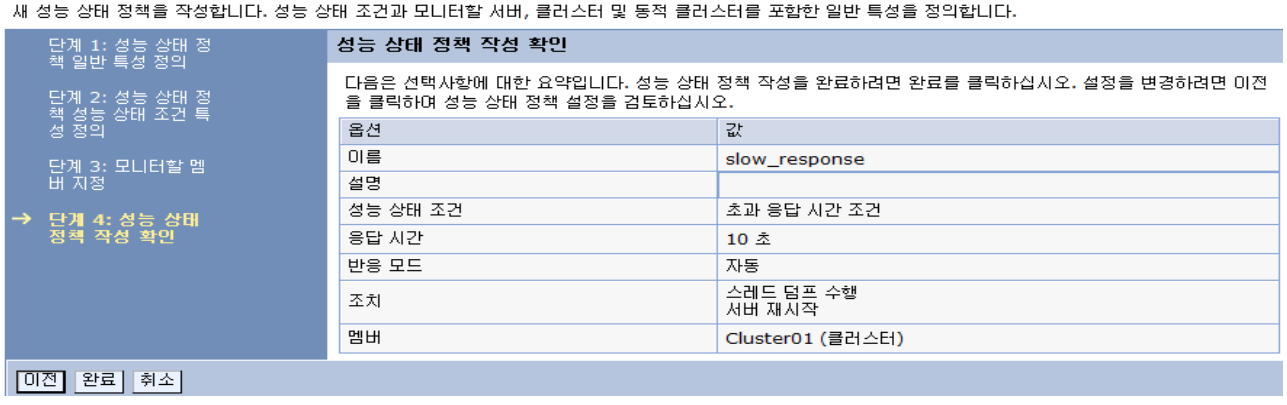


7. 모니터할 멤버 기준을 정하셨으면 사용 가능 멤버 리스트에 해당 서버 or 클러스터가 나오는 것을 확인할 수 있습니다. 해당 멤버를 선택하고 추가 버튼을 클릭하면 성능 상태 정책 멤버로 추가됩니다. (해당 테스트를 위해서 사전에 두 개의 일반적인 WAS 를 가진 클러스터를 하나 만들고 해당 클러스터를 멤버로 등록합니다.)





8. 마지막으로 요약정보에 이상이 없다면 완료 버튼을 클릭하시면 장애 사전 예방을 위한 성능 상태 정책 설정을 문제 없이 마무리 하신 것입니다.



2) 장애 사전 예방 테스트

1. 장애 사전 예방 설정을 위하여 인자를 받아서 sleep 시간을 조절 할 수 있는 간단한 Java/JSP 샘플을 하나 만들어서 미리 만들어서 성능 상태 정책으로 지정한 클러스터로 배포합니다.
(해당 샘플은 단순하게 지정된 시간만큼 Thread.sleep() 하는 소스이며 테스트의 편의성을 위해서 받은 sleep 인자를 메모리에 담아서 해당 인자만큼 sleep() 을 수행합니다. 별도 인풋이 없는 초기의 경우에는 기본 500ms 를 sleep 합니다.)

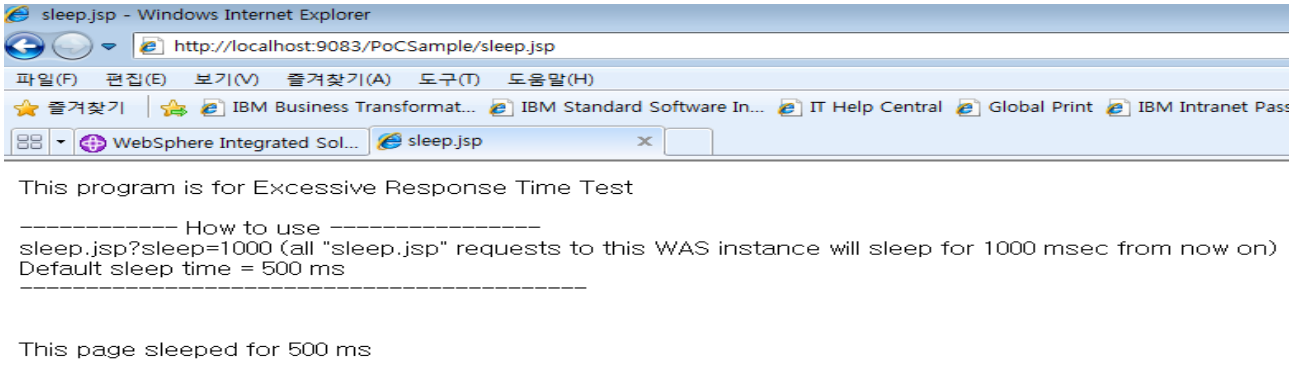
Sleep.jsp

```
<BODY><P>This program is for Excessive Response Time Test<BR>
<BR>------------ How to use -----------------<BR>sleep.jsp?sleep=1000 (all &quot;sleep.jsp&quot; requests to this WAS instance will sleep for
Default sleep time = 500 ms<BR>
-------------------------------------------<BR>

<%
  String sleepTimeParam = request.getParameter("sleep") ;
  if(sleepTimeParam != null)
  {
    long sleepTime = Long.parseLong(sleepTimeParam) ;
    SleepTimeInfo.getInstance().setSleepTime(sleepTime) ;
    out.println("<BR><BR>all &quot;sleep.jsp&quot; requests to this WAS instance will sleep for" + sleepTimeParam +  " msec from now on") ;
    return ;
  }
  else
  {
    long sleepTime = SleepTimeInfo.getInstance().getSleepTime() ;
    Thread.sleep(sleepTime) ;
    out.println("<BR><BR>This page sleeped for " + sleepTime + " ms <BR>") ;
    System.out.println("This page sleeped for " + sleepTime + " ms");
  }
%>
```

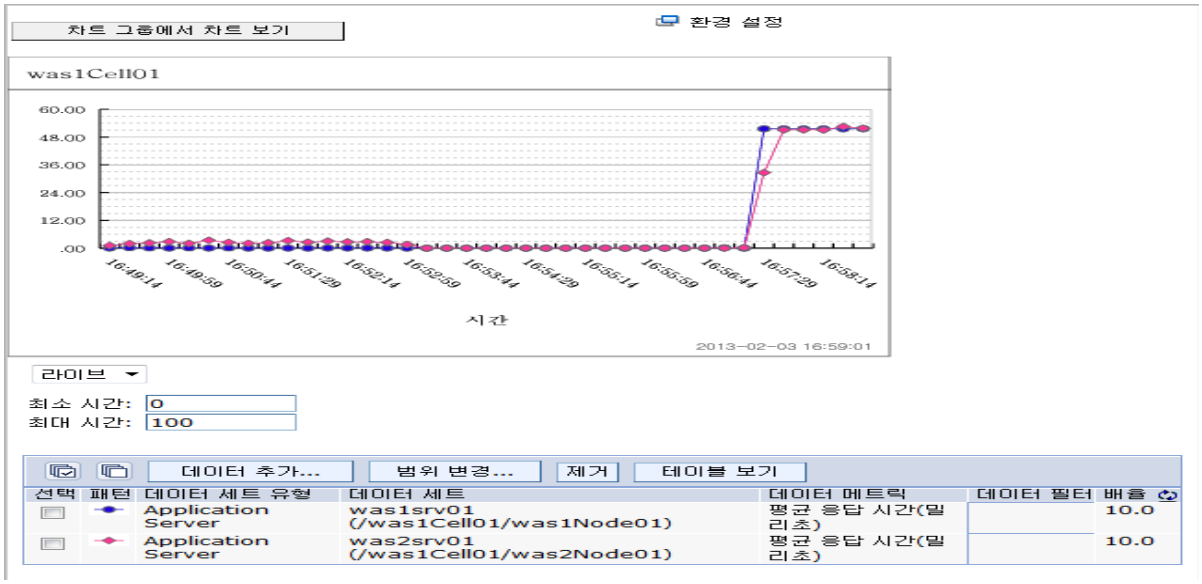
SleepTimeInfo.java

```
public class SleepTimeInfo {
        private long _sleepTime = 500L ;
        private static SleepTimeInfo _this = null ;

        private SleepTimeInfo()
        {
                super() ;
        }

        /**
         * @return Returns the _sleepTime.
         */
        public synchronized long getSleepTime() {
                return _sleepTime;
        }
        /**
         * @param count The _sleepTime to set.
         */
        public synchronized void setSleepTime(long count) {
                _sleepTime = count;
        }
        public synchronized static SleepTimeInfo getInstance()
        {
                if(_this == null)
                  _this = new SleepTimeInfo() ;
          return _this ;
        }
}
```

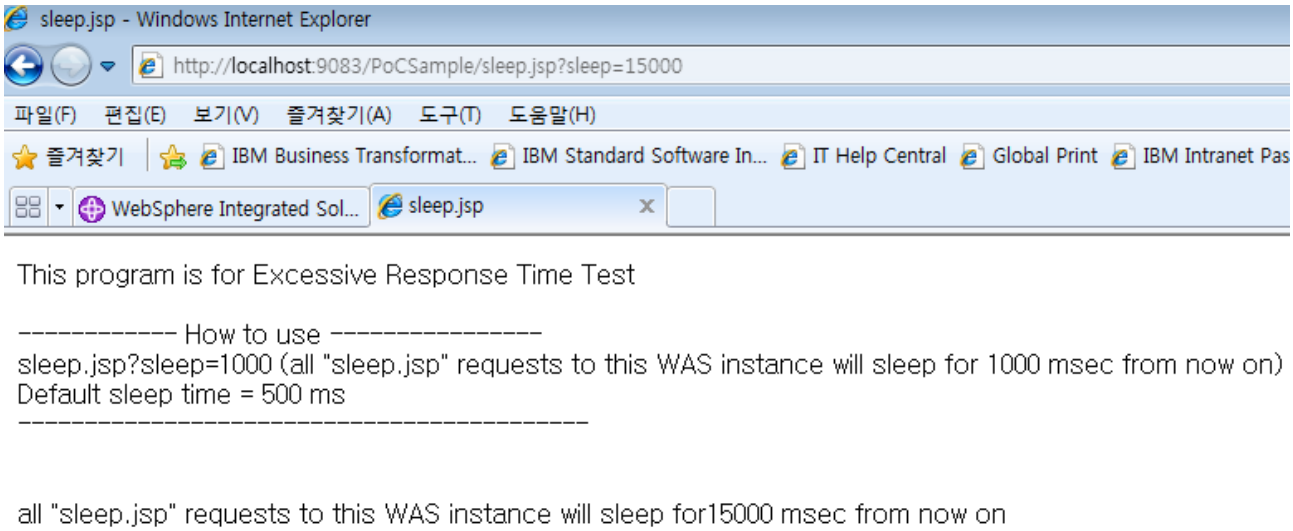
샘플 JSP 수행 화면



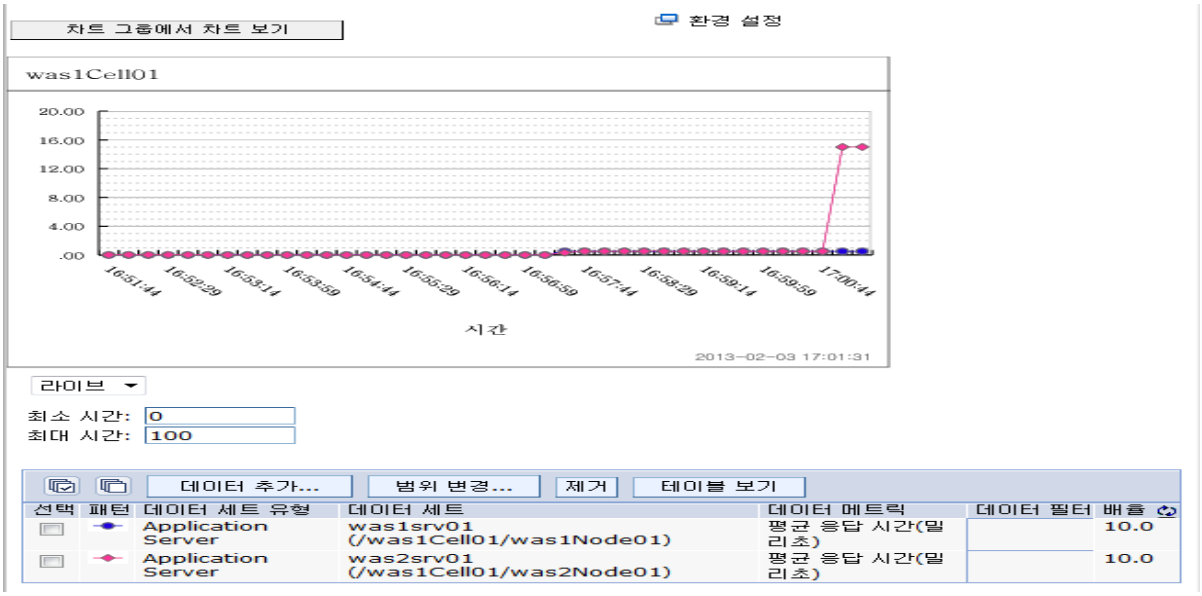
2. JMeter 와 같은 간단한 부하기를 통해서 지속적으로 지정된 Cluster 로 샘플 부하를 발생시키고 응답 시간을 관리콘솔에서 런타임 조작 > 보고서를 통해서 모니터링 합니다.
(지정된 Cluster 에 속해 있는 서버별로 모니터링 하게 되면 하단과 같이 확인하실 수 있습니다. JMeter 사용이 어려우신 분은 클릭을 반복하는 것이 조금 번거롭기는 하지만 브라우저를 이용해서 진행하셔도 됩니다.)



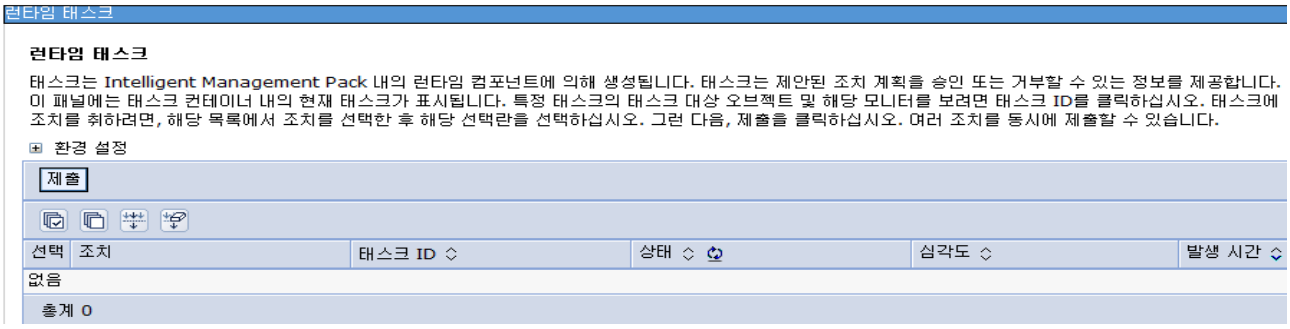
3. JMeter 를 통한 지속적인 부하가 들어오고 있는 중에 클러스터 중의 특정 WAS 를 선택하여 장애 상황을 가정하고 응답시간을 15초로 변경합니다. 하단과 같이 브라우저를 통해서 요청을 보내면 이후에 해당 서버로 들어오는 모든 동일 요청은 15초의 sleep 을 통해서 응답시간 지연을 발생하게 됩니다.
(http://localhost:9083(특정 서버의 서비스 포트)/PoCSample(컨텍스트 루트명)/sleep.jsp?sleep=15000)

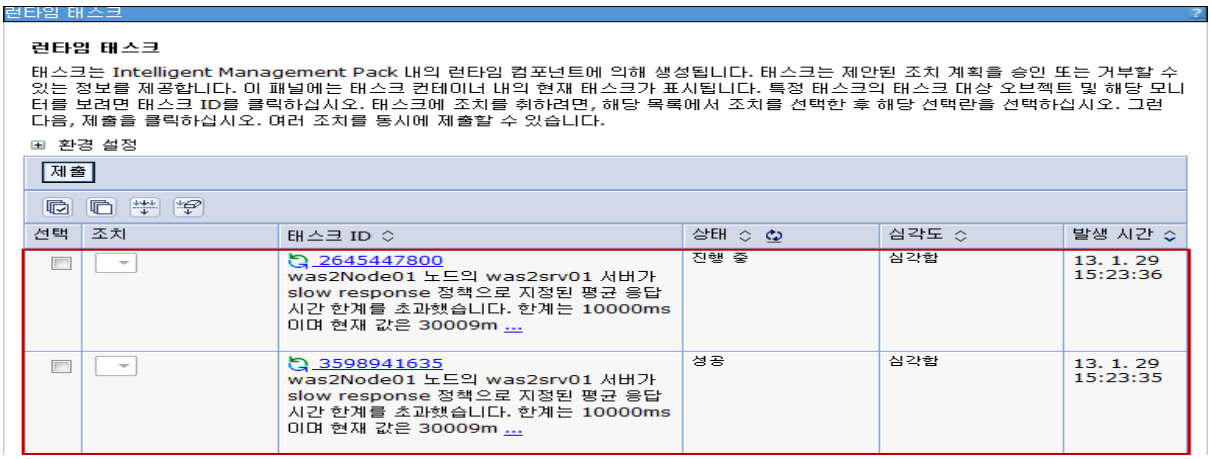


4. 이전에 사용한 보고서 기능을 통해서 확인해보면 하단과 같이 특정 서버의 응답시간이 15초로 변경된 것을 확인하실 수 있습니다.



5. 이렇게 지연되는 응답시간이 발생하고 어느 정도 시간이 소요되고 나서 (약 30초에서 60초 사이 정도 – 통계를 통해서 평균 응답시간이 구해지는 시간 정도) 관리콘솔 에서 시스템 관리 > 태스크 관리 > 런타임 태스크 메뉴를 확인해보면 하단과 같이 아무것도 없는 상태였다가 장애 징후를 사전에 인지하고 하단과 같이 지정된 평균 응답시간 한계를 초과했다는 메시지가 생성되는 것을 확인하실 수 있습니다.

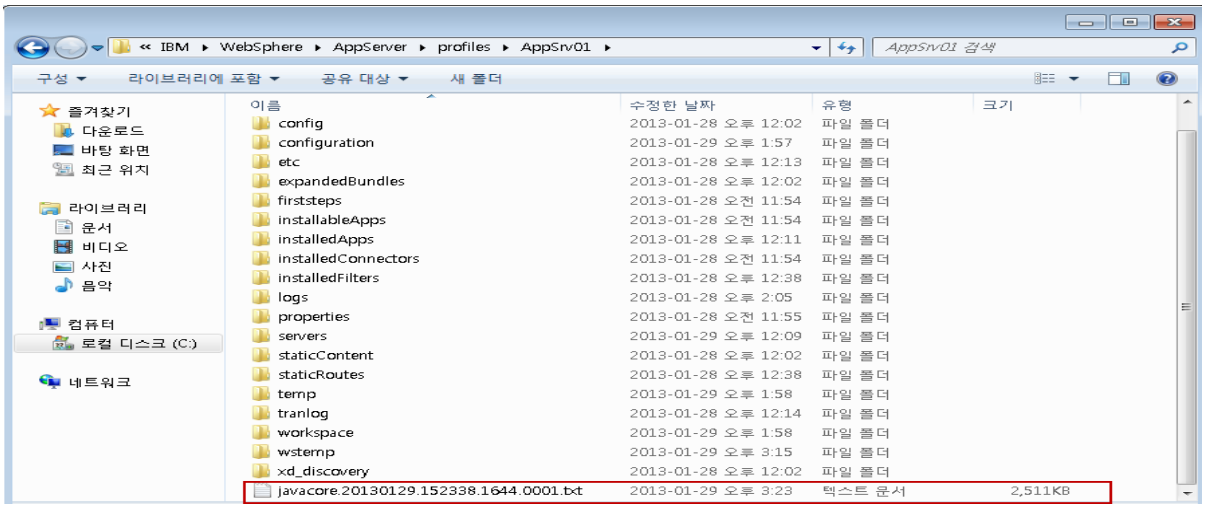




또한, Dmgr 의 SystemOut.log 를 확인해 보셔도 하단과 같이 해당 시점에 장애 징후를 감지하는 것을 확인할 수 있습니다.

```
[13. 2. 3   16:59:37:934 KST] 00000137 ResponseSubSy W   WXDH3010W: was2Node01 노드의 was2srv01 서버에 대한 서버 응답 시간 변경 지점이 발견되었습니다. 평균 응답 시간이 21ms에서
[13. 2. 3   17:00:18:042 KST] 0000002c ResponseSubSy W   WXDH3010W: was1Node01 노드의 was1srv01 서버에 대한 서버 응답 시간 변경 지점이 발견되었습니다. 평균 응답 시간이 0ms에서
[13. 2. 3   17:01:30:988 KST] 0000019e ResponseSubSy W   WXDH3005W: was2Node01 노드의 was2srv01 서버가 slow response 정책으로 지정된 평균 응답 시간 한계를 초과했습니다.  한계는
[13. 2. 3   17:01:31:019 KST] 0000019e FfdcProvider  W com.ibm.ws.ffdc.impl.FfdcProvider logIncident FFDC1003I: C:\IBM\WebSphere\AppServer\profiles\Dmgr01\logs\ffdc\dmgr_1d08db
[13. 2. 3   17:01:31:752 KST] 00000135 ThreadDumpTas I   WXDH1010I: 서버 was1Cell01/was2Node01/was2srv01의 스레드 덤프가 발행되는 중입니다.
[13. 2. 3   17:01:33:156 KST] 00000135 ReactionManag I   WXDH1011I: 성능 상태 제어기가 서버 was1Cell01/was2Node01/was2srv01에 대한 스레드 덤프를 발행했습니다.
[13. 2. 3   17:01:33:172 KST] 00000135 RestartTask   I   WXDH1007I: 서버 was1Cell01/was2Node01/was2srv01 useForceToManualMode=false clusterName=Cluster01을(를) 자동으로 다시 시
```

6. 이와 같이 미리 지정된 감지 조건에 따른 장애 징후가 감지되지 마자 관리자가 지정된 조치가 수행되며 본 강좌에서 테스트로 지정했던 것처럼 해당 프로파일의 root 폴더(스레드 덤프가 생성되는 기본 폴더) 에서 하단과 같이 스레드 덤프가 생성된 것을 확인하실 수 있습니다.



7. 또한, 스레드 덤프가 생성된 이후에 조금의 시간이 지난 후에 해당 서버가 재시작 되는 것을 해당 서버의 SystemOut.log 를 통해서 하단과 같이 확인할 수 있습니다.

8. 이처럼 WAS 가 가진 성능 상태 정책을 이용하면 장애가 발생되기 전에 장애의 징후를 포착하여 관리자가 원하는 작업이 수행 가능한 것을 확인할 수 있으며 이를 통해서 다양한 사전 조치를 취할 수 있습니다.

참고적으로 해당 기능의 작업에 대한 제어 주기나 사용 여부등의 세부 설정은 관리콘솔에서 조작 정책 > 자동 관리자 > 성능 상태 제어기 메뉴를 통해서 제어 가능합니다.

9) 참고 자료

1. 이 가이드는 IBM WAS v8.5 최초 사용자를 위한 기본 가이드 입니다.

2. IBM WAS 자체에 아직 익숙하지 않으신 분들은 가급적 기본강좌인 '하나씩 쉽게 따라 해보는 IBM WAS v7' 강좌를 먼저 읽고 이 강좌를 읽으시는 것이 훨씬 이해에 됩니다.
(http://www.websphere.pe.kr/xe/?mid=was_info_re&page=3&document_srl=800)

3. 가급적 IBM WAS v8.5 InfoCenter 의 해당 카테고리를 한 번 읽어보고 난 후에 작업하시기 바라겠습니다.

4. InfoCenter – WebSphere Application Server v8.5
(http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp)

5. InfoCenter – Health management
(http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/cwve_odhealth.html?resultof=%22%68%65%61%6c%74%68%22%20%22%6d%61%6e%61%67%65%6d%65%6e%74%22%20%22%6d%61%6e%61%67%22%20)