ENPHAROS JAVA Administration Guide

ENPHAROS v5.1

DabomSoft

Pharos for one, Pharos for all

Copyright Notice

Restricted Rights Legend

Trademarks

안내서 정보

안내서 제목: ENPHAROS JAVA Administration Guide
발행일: 2015-12-14
소프트웨어 버전: ENPHAROS JAVA v5.1
안내서 버전: v5.1

목 차

그림 목차

표 목차

안내서에 대하여

안내서의 대상

본 안내서는 ENPHAROS JAVA®(이하 ENPHAROS JAVA)를 사용하여 어플리케이션 성능을 모니터링 하고자 하는 관리자를 위한 안내서이다. ENPHAROS JAVA를 이용해서 어플리케이션 성능 모니터링을 위한 기본 개념과 모니터링을 위해 필요한 설정 정보에 대해서 설명한다. 또한 ENPHAROS JAVA를 사용해서 어플리케이션 성능 모니터링의 다양한 구축 환경에 대한 설명과 해당 환경에서 적용을 위한 방법을 제시한다. 구축의 이해를 돕기 위해 다양한 예제를 수록해서 사용자의 이해를 돕고자 한다.

안내서의 전제 조건

본 안내서는 ENPHAROS JAVA 시스템에 대한 전반적인 이해와 ENPHAROS JAVA 시스템이 제공하는 각종 기능 및 특성에 대한 습득을 위한 기본서이다.

본 안내서를 원활하게 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- WAS에 대한 기본적인 이해
- Java 프로그래밍의 이해

관련 안내서

본 안내서는 기본적인 ENPHAROS JAVA 설치에 대한 내용을 주로 설명하며, ENPHAROS JAVA의 운영 및 사용자 안내서는 다루지 않는다.

- "ENPHAROS JAVA User Guide"

ENPHAROS JAVA의 각 기능설명 및 분석방법에 기술한다.

- WAS 관련 안내서

WAS 에 대한 안내서는 WAS Vendor 사에서 제공하는 안내서를 참고 한다.

안내서 구성

ENPHAROS JAVA Administration Guide는 총 3개의 장으로 구성되어 있고, 각 장의 주요 내용은 다음과 같다.

- 제1장: ENPHAROS JAVA 소개
  ENPHAROS JAVA의 개요 및 상세 구조 에이전트의 동작 원리를 기술 한다.

- 제2장: ENPHAROS 에이전트 디렉터리 구성
  ENPHAROS JAVA 에이전트의 디렉터리 구조와 주요 파일을 설명 한다.

- 제3장: ENPHAROS 에이전트 설정
  ENPHAROS JAVA 에이전트의 기본 설정 및 동적 반영 설정, 로그 설정 등을 설명 한다.

- 제4장: 부록
  ENPHAROS JAVA 에이전트의 주요 설정 파일에 대해 설명한다.

안내서 규약

표 기	의미
<AaBbCc123>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
[그림] 1-1	그림 이름
[표] 1-1	표 이름
AaBbCc123	명령어, 명령어 수행 후 화면에 출력된 결과물, 예제코드
[ ]	옵션 인수 값
|	선택 인수 값
「」	디렉터리 경로 또는 해당 파일 경로

시스템 환경

항목	요구사항
운영체제	IBM AIX 5.x, 6.x, 7.x, HP-UX 10, 11, Linux, Solaris, MS Windows 계열
JDK	JDK 1.5, JDK 1.6, JDK 1.7
DBMS	Oracle 9i, 10gR2, 11g 지원
	DB2

지원 WAS

제품명	요구사항
JEUS	JEUS 4.x, JEUS 5.x, JEUS 6.x
Weblogic	Weblogic 8.x, Weblogic 9.x, Weblogic 10.x, Weblogic 11.x
Tomcat	Tomcat 4.x, Tomcat 5.x, Tomcat 6.x
WebSphere	WebSphere 5.x, WebSphere 6.x, WebSphere 7.x
OC4J	OC4J 9i, OC4J 10g
JBoss	JBoss 4.x, JBoss 5.x, JBoss 6.x

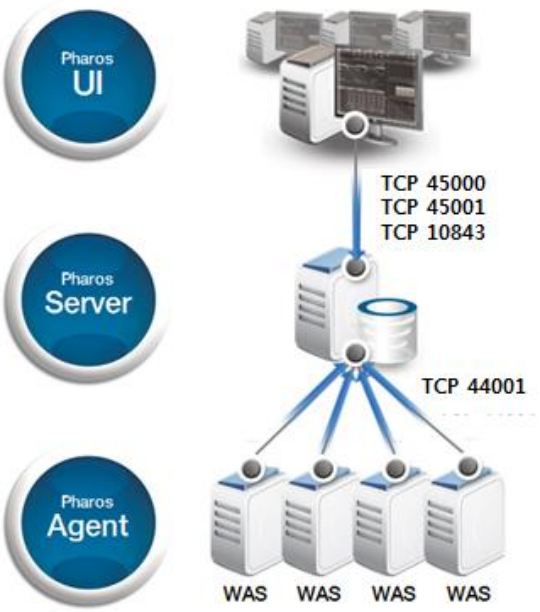
01

ENPHAROS JAVA 소개

1. ENPHAROS JAVA 소개

1.1. 개요

ENPHAROS JAVA는 모니터링 대상 WAS시스템 하에서 실행되는 서비스가 호출되는 시점부터 종료될 때까지 실시간으로 서비스 처리 상태에 대한 Call-Tree 및 데이터베이스 처리 정보를 수집하여 서비스의 현재상태에 대한 정보를 제공하는 시스템이다. 또한, 서비스 처리 시 CPU 사용량, 메모리 사용량, 경과 시간을 추적하여 서비스가 안정적으로 실행되는지 정보를 제공하는 시스템 이다.



[그림] 1-1 ENPHAROS JAVA 기본 구조

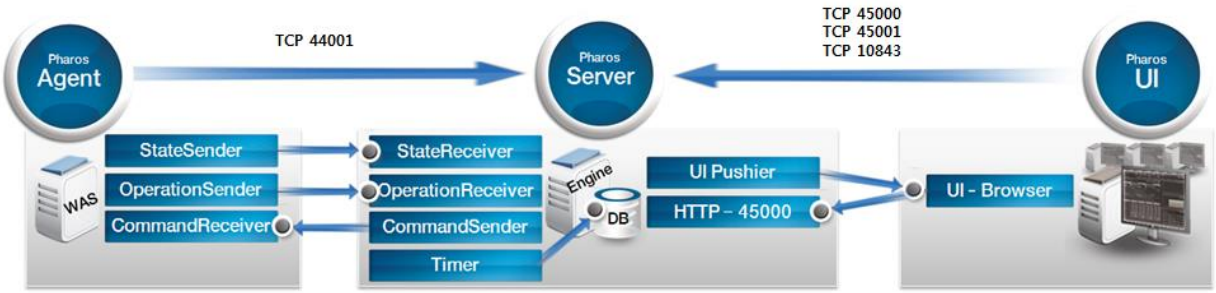
ENPHAROS JAVA는 ENPHAROS Agent(이하 에이전트), ENPHAROS Server(이하 수집서버), ENPHAROS UI(이하 UI)의 3-Tier 구조를 구현 되었으며 이는 모니터링 정보를 수집, 저장, 표현을 각각의 Tier에서 담당하여 분산 처리하여 엔터프라이즈 환경과 같은 대규모의 모니터링 환경에서 원활하게 동작할 수 있도록 구성되어 있다.

구 성	설 명
에이전트	모니터링 대상 WAS 내부에서 동작하여 성능 및 요청 처리 정보를 수집하여 수집서버로 전송 한다.
수집서버	에이전트가 수집한 성능 데이터를 수신하여 저장소에 저장하고 UI에서 요청하는 각종 성능 데이터를 전송 한다.
UI	클라이언트 환경으로 Web Browser에서 수집서버와 통신하여 모니터링 정보를 표현 한다.

[표] 1-1 ENPHAROS JAVA 구성 설명

1.2. ENPHAROS JAVA 상세 구조

ENPHAROS JAVA의 에이전트, 수집서버, UI는 각각 모니터링 정보를 수집, 저장, 표현하는 역할을 수행하며 각 Tier는 내부적으로 목적에 맞게 생성된 Thread를 이용하여 유기적으로 동작하도록 구성되어 있다.



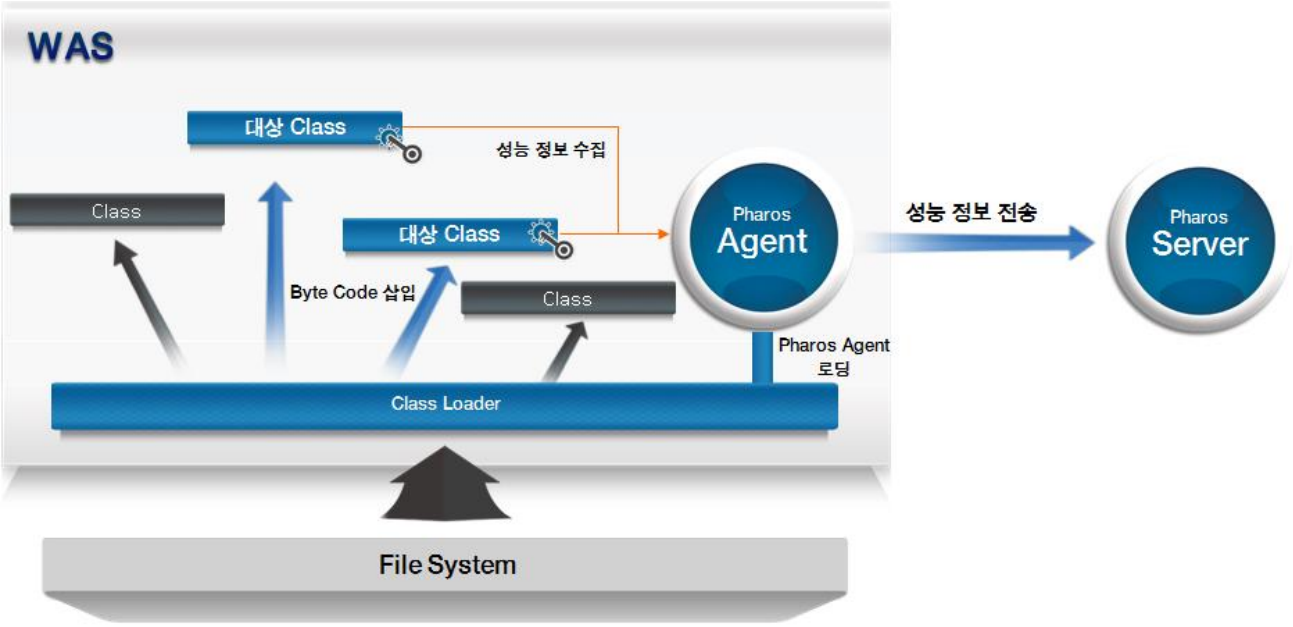
[그림] 1-2 ENPHAROS JAVA 상세 구조

Tier	모 듈	설 명
에이전트	StateSender	CPU, Memory, Active Thread 등과 같은 성능 정보를 주기적으로 수집하여 수집서버에 전송하는 Thread.
	OperationSender	사용자의 요청 정보를 수집하여 상시 전송하는 Thread.
	CommandReceiver	수집서버로부터 받은 명령을 실행하는 Thread.
수집서버	StateReceiver	에이전트의 StateSender가 주기적으로 전송하는 성능 정보를 수신하는 Thread.
	OperationReceiver	에이전트의 OperationSender가 상시 전송하는 요청 정보를 수신하는 Thread.
	CommandSender	UI에서 에이전트로 명령할 시에 이를 중간에서 에이전트로 전송하는 Thread.
	Timer	단위 시간 별 통계와 로그 생성 주기를 담당하는 Thread.
	UI-Pushier	처리된 사용자 요청 정보를 UI로 TCP 45001 포트를 이용하여 Push하는 Thread.
	HTTP-45000	UI를 통해 접근한 사용자가 주기적으로 호출하는 성능 정보를 응답 해주는 Thread.
UI	UI-Browser	TCP 45000으로 수집서버에 접근하여 모니터링 정보 및 통계 데이터를 조회하고 TCP 45001 포트를 이용하여 실시간 정보 데이터 및 Request Performance(응답시간 분포도)를 표시한다. TCP 10843 포트는 플래쉬의 보안을 위해 policy파일을 내려 받을 시 사용된다.

[표] 1-2 ENPHAROS JAVA의 Thread 정보

1.3. ENPHAROS JAVA 에이전트 동작 원리

Class Loader에 의해 에이전트 라이브러리가 로딩되면 성능 정보를 수집할 대상 Class에 수집용 Byte Code를 삽입하여 Class를 변형시키고 해당 Class가 호출될 시에 성능 정보를 수집하여 수집서버에 전송한다. 성능 수집용 Byte Code는 ASM 기반으로 제작되어 있다.



[그림] 1-3 ENPHAROS JAVA 에이전트 동작 원리

※ ENPHAROS JAVA 에이전트 동작 원리

- Class Loader에 의해 Class가 로딩 될 때 대상 Class의 Method에 성능정보 수집용 코드(Byte Code)가 추가된다.
- Class변형 대상 Class와 Method는 에이전트에서 기 정의한 J2EE Class/Method들과 criteria 설정을 통해 사용자가 지정한 Class/Method 이다
- -Xbootclasspath 옵션 혹은 -javaagent 옵션을 추가하여 에이전트를 기동하고 수집용 코드가 추가 되도록 한다.

02

에이전트 디렉터리 구성

2. 에이전트 디렉터리 구성

2.1. 에이전트 디렉터리 구조

일반적으로 ENPHAROS JAVA 에이전트가 설치될 파일 디렉터리 구성은 다음과 같다.

```
+---- bin
+---- config
+---- config_sample
+---- lib
+---- license
+---- MEAT-INF
+---- logs
+---- plugins
+---- readme
+---- xsds
+---- site
```

각 디렉터리 별로 저장하는 파일의 설명은 다음과 같다.

구 분	설 명
bin	에이전트를 실행하기 위한 스크립트 파일들이 저장된 디렉터리이다.
config	에이전트의 설정 값을 위한 파일들이 저장된 디렉터리이다.
config_sample	환경 설정 파일 Sample들이 저장된 디렉터리이다.
lib	에이전트에서 서비스 처리정보를 수집하기 위해 필요한 라이브러리 파일이 저장된 디렉터리이다.
license	License 파일 정보가 저장된 디렉터리이다.
META-INF	에이전트Build를 위한 manifest 파일이 저장된 디렉터리이다.
logs	에이전트별 로그 파일이 저장될 디렉터리이다.
plugins	에이전트에 plugin 형태로 기능을 추가할 때 사용되는 모듈이 저장된 디렉터리이다.
readme	제품의 릴리즈 노트가 저장된 디렉터리이다.
xsds	에이전트에서 사용하는 XML 스키마 파일들이 저장된 디렉터리이다. main.xsd의 헤더파일인 agent.xsd, common.xsd, custom.xsd, listener.xsd, scripts.xsd 파일이 존재한다.
site	사이트에서 사용하는 전문에서 데이터를 추출하기 위한 설정파일인 order.properties를 포함하고 있는 디렉터리이다.

[표] 2-1 에이전트 디렉터리 구조

2.1.1. bin 디렉터리

bin 디렉터리는 각종 명령어 스크립트 파일이 저장되어 있는 영역으로 각각의 파일에 대한 설명은 다음과 같다.

파일명	설명
agent.cmd(sh)	Web Server 모니터링 데몬 기동 명령 파일
checkconfig.cmd(sh)	에이전트의 설정 값의 문법오류 명령을 검증하는 파일
classfinder.cmd(sh)	classloader에 올라와 있는 class의 경로를 찾아주는 파일
ipcheck.cmd(sh)	라이선스 발급 시 에이전트가 설치된 장비의 IP 목록을 확인하는 파일
machine.cmd(sh)	시스템 리소스 정보를 모니터링하는 파일
make_rt.cmd(sh)	JDK1.4버젼 사용 시 rt.jar 파일을 생성하는 파일
native2_test.cmd(sh)	운영체제 별 JVM 리소스 모니터링 모듈 체크를 검증하는 파일
set_classpath.cmd(sh)	에이전트 classpath 를 설정하는 파일
set_pharos_env.cmd(sh)	에이전트의 JDK1.4사용 시 환경 정보를 실행하는 파일
set_pharos_env15.cmd(sh)	에이전트의 JDK1.5 이상 사용 시 환경 정보를 실행하는 파일
version.cmd(sh)	에이전트 Version 을 확인하는 파일

[표] 2-2 에이전트 bin 디렉터리 구성

명령어를 수행 하기 전,
$PHAROS_HOME/bin/set_classpath.cmd(sh), $PHAROS_HOME/bin/set_pharos_env.cmd(sh) 파일의 $PHAROS_HOME 경로를 수정 한다.
수정방법은 "2.1.1.8. set_classpath.cmd(sh)", "2.1.1.9. set_pharos_env.cmd(sh)" 을 참조한다.

2.1.1.1. agent.cmd(sh)

Web Server 모니터링 데몬 프로세스를 실행하는 명령어이다.
agent.cmd(sh) [Agent Name] ---------- "참고 ①, ②" 참조

2.1.1.2. checkconfig.cmd(sh)

에이전트 설정 파일인 main.xml안의 문법을 체크할 때 사용하는 명령어이다.
checkconfig.cmd(sh) [Agent Name] ---------- "참고 ①, ②" 참조

2.1.1.3. classfinder.cmd(sh)

classloader에 올라와 있는 class의 경로를 찾아주는 파일

2.1.1.4. ipcheck.cmd(sh)

라이선스 발급 시 에이전트가 설치된 장비의 IP 목록을 확인할 때 사용하는 명령어이다.

ipcheck.cmd(sh)

2.1.1.5. machine.cmd(sh)

시스템 리소스 모니터링 데몬 기동 명령어로서, 에이전트가 설치된 운영체제의 CPU, Memory 사용 정보를 수집한다.

machine.cmd(sh)

2.1.1.6. make_rt.cmd(sh)

모니터링 대상 WAS가 JDK1.4 버전을 사용 시 BCEL방식(참고 ⑤)의 수집모듈을 생성할 때 사용하는 명령어이다.

make_rt.cmd(sh) [Agent Name] [JAVA_HOME] ---------- "참고 ①, ③, ④" 참조

2.1.1.7. native2_test.cmd(sh)

모니터링 대상 WAS의 JVM에서 사용하는 CPU, Memory(Heap)정보를 운영체제 별로 수집하는 모듈을 테스트 하기 위한 명령어이다.

native2_test.cmd(sh) ---------- "참고 ⑥" 참조

2.1.1.8. set_classpath.cmd(sh)

에이전트에서 사용하는 Class Path를 정의하고 설정하는 명령어이다.

set_classpath.cmd(sh)

명령어 수행 전 파일 안의 $PHAROS_HOME의 경로를 수정 한다.

```
PHAROS_HOME=/test/was/pharos; export PHAROS_HOME

SYSTEM_CP=$CLASSPATH
CLASSPATH=

#### Agent Library
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-loader.jar
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-agent.jar
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-common.jar

---------- <이하 생략> ----------
```

[그림] 2-1 set_classpath.sh 파일 내용

2.1.1.9. set_pharos_env.cmd(sh)

모니터링 대상 WAS가 JDK1.4를 사용 시 에이전트의 환경변수를 설정하기 위한 명령어이다.

set_pharos_env.cmd(sh)

$PHAROS_HOME 경로를 에이전트 설치 경로로 수정 한다.

```
PHAROS_HOME=/test/was/pharos; export PHAROS_HOME

#### Agent specific library & JAVA Options ####
if [ -n "$PHAROS_AGENT" ]; then
        PHAROS_OPTIONS="-Dpharos.home=${PHAROS_HOME}"
        PHAROS_OPTIONS="${PHAROS_OPTIONS} -Dpharos.agent=${PHAROS_AGENT} -Dpharos.use.asm=false
        PHAROS_OPTIONS="${PHAROS_OPTIONS} -
        Xbootclasspath/p:${PHAROS_HOME}/config/${PHAROS_AGENT}/lib/rt.jar"
        # PHAROS_OPTIONS="${PHAROS_OPTIONS} -Duse.loader.logger=true"
        export PHAROS_OPTIONS
fi
```

[그림] 2-2 set_pharos_env.sh 파일 내용

2.1.1.10. set_pharos_env15.cmd(sh)

모니터링 대상 WAS가 JDK1.5 이상 사용 시 환경변수를 설정하기 위한 명령어이다.

set_pharos_env15.cmd(sh)

$PHAROS_HOME 경로를 에이전트 설치 경로로 수정 한다.

```
PHAROS_HOME=/test/was/pharos; export PHAROS_HOME

#### Agent specific library & JAVA Options ####
if [ -n "$PHAROS_AGENT" ]; then
    PHAROS_OPTIONS="-Dpharos.home=${PHAROS_HOME}"
    PHAROS_OPTIONS="${PHAROS_OPTIONS} -Dpharos.agent=${PHAROS_AGENT} -Dpharos.use.asm=true"
    # PHAROS_OPTIONS="${PHAROS_OPTIONS} -Xbootclasspath/p:${PHAROS_HOME}/lib/pharos-javax15.jar"
    # PHAROS_OPTIONS="${PHAROS_OPTIONS} -
Xbootclasspath/p:${PHAROS_HOME}/config/${PHAROS_AGENT}/lib/rt.jar"
    PHAROS_OPTIONS="${PHAROS_OPTIONS} -javaagent:${PHAROS_HOME}/lib/pharos-loader.jar"
    # PHAROS_OPTIONS="${PHAROS_OPTIONS} -Duse.loader.logger=true"
    export PHAROS_OPTIONS
fi
```

[그림] 2-3 set_pharos_env15.sh 파일 내용

2.1.1.11. version.cmd(sh)

에이전트의 버전을 확인할 수 있는 명령어이다.

version.cmd(sh)

참고

① [Agent Name] : 모니터링 대상 Web Server의 식별자

② [Agent Name] 인자 입력 시 config 디렉터리 내 해당 [Agent Name]으로 디렉터리와 환경설정 파일이 존재하여야 한다.

③ [JAVA_HOME] : 모니터링 대상 WAS가 사용하는 JDK1.4의 설치 경로

④ [Agent Name]인자 입력 시 config 디렉터리 내 해당 [Agent Name]으로 디렉터리와 기본 환경설정 파일이 자동 생성 된다.

⑤ BCEL (Byte Code Engineering Library) : 자바언어에서 생성하는 클래스 파일을 분석하고 변경, 재조립하는 간편한 인터페이스를 제공 한다.

⑥ 명령어 수행 시 「lib/native」 디렉터리 안의 운영체제 디렉터리 별 libpharosutil.so 모듈이 존재해야 한다.

2.1.2. config / config_sample 디렉터리

config 디렉터리는 ENPHAROS JAVA 에이전트가 실행할 때 참조하는 설정 파일이 저장되어 있는 영역으로 각각의 파일에 대한 설명은 다음과 같다. config_sample 디렉터리 내 샘플을 이용하여 개별/종류별 모니터링 대상의 정보를 설정할 때 참고 한다.

디렉터리 명	설 명
Test-apache	Apache, WebtoB 설정 샘플
Test-iplanet	Iplanet 설정 샘플
Test-j2ee	개별 Web Application Server 설정 샘플
Test-javadaemon	개별 Java Daemon 설정 샘플

[표] 2-3 에이전트 config 디렉터리 구성

2.1.3. lib 디렉터리

lib 디렉터리는 ENPHAROS JAVA 에이전트에서 실행할 때 참조하는 라이브러리 파일이 저장되어 있는 영역으로 각각의 파일에 대한 설명은 다음과 같다.

<table>
<tr><th colspan="2">디렉터리 명</th><th colspan="2">파일 명</th><th>설 명</th></tr>
<tr><td rowspan="3">advices</td><td rowspan="2">adviceMaker</td><td rowspan="2">SampleAdvice</td><td>SampleApErrorAdvice.java
SampleApErrorAfterAdvice.java
SampleDataMakerAdvice.java
SampleEAIHashSequenceAdvice.java
SampleGuidMakerAdvice.java
SampleHashSequenceAdvice.java
SampleTxCodeMakerAdvice.java</td><td>advice sample code</td></tr>
<tr><td colspan="2">CompileHelper.cmd(sh)</td><td>compile 유틸</td></tr>
<tr><td colspan="3">Pharos-agent-advice-stacktrace.jar
pharos-agent-advice-common.jar</td><td>에이전트에서
사용되는 라이브러리</td></tr>
<tr><td rowspan="2">common-config</td><td>criteria</td><td colspan="2">criteria-main.xml
http-filter.xml
http-jsp.xml
http-servlet.xml
jdbc.xml
jdbc-proxy.xml
naming.xml
weblogic-jdbc.xml
pharos-default-exclude.xml
weblogic-jdbc.xml</td><td rowspan="2">config 디렉터리에
개별 [Agent Name]
식별자가 존재하지
않을 경우, 이곳이
공통 설정 장소로
인지된다.</td></tr>
<tr><td>properties</td><td colspan="2">loader.properties</td></tr>
</table>

		pharos.log4j.properties http_uri.exclude		
		main.xml		
common-config-profile		criteria	criteria-main.xml http.xml http-filter.xml http-jsp.xml http-servlet.xml naming .xml jdbc-proxy.xml jdbc-proxy.xml naming.xml weblogic-jdbc.xml	프로파일링을 위한 기본 설정이 되어 있는 common-config 디렉터리 (common-config 디렉터리와 교체해서 사용 한다.)
		properties	http_uri.exclude loader.properties pharos.log4j.properties	
			main.xml sample-main.xml	
jaxb1			javax-xml.jar jaxb-api.jar jaxb-impl.jar jaxb-libs.jar relaxngDatatype.jar xsdlib.jar	에이전트에서 사용되는 JDK1.4 라이브러리
jaxb2			activation.jar jaxb1-impl.jar jaxb-api.jar jaxb-impl.jar jaxb-xjc.jar jsr173_1.0_api.jar	에이전트에서 사용되는 JDK1.5이상 라이브러리
native	운영체제 별 디렉터리		libpharosutil.so librtmutil.so pharosutil.dll (Windows 계열) rtmutil.dll (Windows 계열)	운영체제 별 모니터링 대상 WAS의 리소스 데이터 수집 모듈
xmllib			xercesImple.jar xml-apis.jar	Xml Parsing 라이브러리
			pharos-agent-config.jar pharos-agent-config-2x.jar pharos-agent-ext.jar	에이전트에서 사용되는 라이브러리

		pharos-agent-http.jar pharos-loader.jar	

[표] 2-4 에이전트 lib 디렉터리 구성

참고

common-config 디렉터리는 여러 모니터링 대상 WAS의 설정을 개별로 config 디렉터리 내 [Agent Name] 식별자 디렉터리를 만들지 않고 공통으로 사용 시 활용 된다.

2.1.4. license 디렉터리

에이전트를 실행하기 위해 필요한 License 정보 파일이 저장된 디렉터리이다.

라이선스는 직접 ㈜다봄소프트(http://www.dabomsoft.com) 담당자에게 요청 한다.

(e-mail: jaehak.lee@dabomsoft.com)

2.1.5. logs 디렉터리

config 디렉터리 와 common-config로 생성 된 모니터링 대상 별로 디렉터리가 생성 되며 agent.log가 기록 된다. ENPHAROS JAVA 에이전트가 설치되어 있는 계정과 WAS 계정이 다른 경우 디렉터리에 저장 권한이 없을 경우 에이전트별 로그가 기록되지 않을 수 있다. 그렇기 때문에 logs 디렉터리에 아래 명령어를 이용하여 저장 권한을 주어야 한다.

```
AIX6:/home/pharosJava> chmod 755 logs
```

2.1.6. plugins 디렉터리

디렉터리 명	파일 명	설 명
available		현 설정이 저장되는 임시 디렉터리
common	pharos-agent-plugin-recorder.jar	설정변경 시 동적 반영을 위한 라이브러리

[표] 2-5 에이전트 plugins 디렉터리 구성

2.1.7. readme 디렉터리

에이전트 버전 별 History파일이 위치 한다.

2.1.8. xsds 디렉터리

에이전트에서 사용되는 xml 파일의 태그 정의 xsds 파일이 위치 한다.

2.2. 에이전트 주요 파일 설명

본 절에서는 에이전트에서 사용하는 주요 설정 파일에 대해 기술한다. 자세한 사항은 이후 "3. 에이전트 환경파일 설정"에서 다루고 있으니 관련 사항을 참조한다.

아래 [표] 2-6에서 관리자들이 사용할 수 있는 에이전트의 주요 파일들에 대해 설명한다.

파일명	설명
pharos-loader.jar	$PHAROS_HOME/lib
	에이전트의 기본 Class Loader
main.xml	$PHAROS_HOME/lib/common-config
	에이전트가 동작하는데 필요한 기본 설정 파일
loader.properties	$PHAROS_HOME/lib/common-config/properties
	동적 반영을 지원하는 설정 파일
criteria-main.xml	$PHAROS_HOME/lib/common-config/criteria
	수집 대상 Class와 Method를 정의하는 용도로 사용되는 설정 파일
agent.log	$PHAROS_HOME/logs/[Agent Name]/
	에이전트의 기본 로그 파일

[표] 2-6 에이전트 주요 파일 설명

03

에이전트 환경파일 설정

3. 에이전트 환경파일 설정

3.1. 에이전트 Class Loader

Java에서 성능 정보를 추출하기 위하여 Class가 로딩될 때 에이전트는 바이트 코드를 삽입하는데, 바이트 코드를 삽입하기 위하여 에이전트를 실행하는 옵션은 다음과 같다.

- JDK 버전 1.5 이상 –javaagent 옵션
- JDK 버전 1.4 이하 –Xbootclasspath 옵션

3.1.1. JDK ver. 1.4 under

「$PHAROS_HOME/bin」에 존재하는 make_rt.cmd(sh)를 실행하여 rt.jar파일을 생성한다.

```
make_rt.cmd(sh) [Agent_Name]
```

참고

make_rt.cmd(sh) 명령어를 수행 하기 전 아래 파일의 $PHAROS_HOME 경로를 수정 한다.
$PHAROS_HOME/bin/set_pharos_env.cmd(sh)
$PHAROS_HOME/bin/set_classpath.cmd(sh)

3.1.1.1. set_classpath.sh 설정

set_classpath.sh는 rt.jar파일 생성을 위해 classpath를 설정하는 파일이다. $PHAROS_HOME 경로를 에이전트 설치 경로로 수정 한다.

```
PHAROS_HOME=/test/was/pharos; export PHAROS_HOME

SYSTEM_CP=$CLASSPATH
CLASSPATH=

#### Agent Library
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-loader.jar
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-agent.jar
CLASSPATH=$CLASSPATH:$PHAROS_HOME/lib/pharos-common.jar

---------- <이하 생략> ----------
```

[그림] 3-1 set_classpath.sh 파일 내용

3.1.1.2. set_pharos_env.sh 설정

set_pharos_env.sh는 에이전트의 환경정보를 설정하는 파일이다. $PHAROS_HOME 경로를 에이전트 설치 경로로 수정 한다.

```
PHAROS_HOME=/test/was/pharos; export PHAROS_HOME

#### Agent specific library & JAVA Options ####
if [ -n "$PHAROS_AGENT" ]; then
        PHAROS_OPTIONS="-Dpharos.home=${PHAROS_HOME}"
        PHAROS_OPTIONS="${PHAROS_OPTIONS} -Dpharos.agent=${PHAROS_AGENT} -Dpharos.use.asm=false
        PHAROS_OPTIONS="${PHAROS_OPTIONS} -
        Xbootclasspath/p:${PHAROS_HOME}/config/${PHAROS_AGENT}/lib/rt.jar"
        # PHAROS_OPTIONS="${PHAROS_OPTIONS} -Duse.loader.logger=true"
        export PHAROS_OPTIONS
fi
```

[그림] 3-2 set_pharos_env.sh 파일 내용

3.1.1.3. make_rt.sh 실행

make_rt.sh는 앞에서 설정한 환경 변수를 사용하여 실제로 rt.jar 파일을 생성하는 명령이다.

```
#!/bin/sh
. ./set_pharos_env.sh

#export CLASSPATH=$PHAROS_HOME/lib/pharos-loader.jar

Java -Dpharos.home=$PHAROS_HOME -jar $PHAROS_HOME/lib/pharos-loader.jar $*
```

[그림] 3-3 make_rt.sh 파일 내용

$PHAROS_HOME/bin/make_rt.sh [Agent_Name] 명령어 수행 시 [그림] 3-4와 같은 결과를 출력한다.

```
PHAROS home : /home/jeus6/PharosAgent
PHAROS agent: agent_jeus6
JAVA Home   : /usr/j2sdk1.4.2_19/jre
JAVA Version: 1.4.2_19
JAVA Vendor : Sun Microsystems Inc.
RT JAR      : usr/j2sdk1.4.2_19/jre/lib/rt.jar

Added: pharos-loader.jar
Added: java.lang.ClassLoader

RESULT:
rt.jar was generated in /home/jeus6/PharosAgent/config/agent_jeus6/lib

============== PHAROS OPTIONS ==============
-Xbootclasspath/p:/home/jeus6/PharosAgent/config/agent_jeus6/lib/rt.jar -Dpharos.home=/home/jeus6/PharosAgent -Dpharos.agent=agent_jeus6
============================================
```

[그림] 3-4 rt.jar 생성 로그

생성된 rt.jar 파일은 「$PHAROS_HOME/config/[Agent_Name]/lib/rt.jar」로 저장 된다.

```
-Dpharos.home=/test/was/pharos
-Dpharos.agent=[Agent_Name]
-Xbootclasspath/p:/test/was/pharos/config/[Agent_Name]/lib/rt.jar
```

[그림] 3-5 rt.jar을 사용한 JVM 옵션

3.1.2. JDK ver. 1.5 over

JDK 1.5 버전 이상을 사용 시 아래와 같은 옵션을 사용하여 에이전트를 적용 한다.

```
-Dpharos.home=/test/was/pharos
-Dpharos.agent=[Agent_Name]
-javaagent:/test/was/pharos/lib/pharos-loader.jar
```

[그림] 3-6 pharos-loader.jar을 사용한 JVM 옵션

3.2. 에이전트 기본 설정

에이전트의 기본 설정 파일은 main.xml 파일이다.

「$PHAROS_HOME/lib/common-config/main.xml」---------- 전체 에이전트 공통 설정을 사용 할 경우

「$PHAROS_HOME/config/[Agent_Name]/main.xml」--------- 에이전트 별로 개별 설정을 사용 할 경우

```
<?xml version="1.0" encoding="UTF-8"?>
<agent>
	<server>
		<ip>127.0.0.1</ip>
		<port>44001</port>
	</server>
	<tier>J2EE</tier>
	<j2ee>
		<!--
		<layers>
			<in>HTTP</in>
			<out>JDBC</out>
		</layers>
		-->
		<operation-config>
			<send-over-time>3000</send-over-time>
			<send-level>SMART</send-level>
			<force-send-issue>true</force-send-issue>
			<use-stack-trace>
				<connection>false</connection>
				<statement>false</statement>
				<result-set>false</result-set>
			</use-stack-trace>
			<condition>
				<min-stack-count>100</min-stack-count>
				<max-stack-count>1000</max-stack-count>
				<time-limit>1000</time-limit>
			</condition>
		</operation-config>
		<thread-config>
			<cpu>
				<enable>false</enable>
			</cpu>
		</thread-config>
		<web>
			<visitor-keygen-
class>com.kwise.pharos.agent.http.DefaultVisitorKeyGenerator</visitor-keygen-class>
		</web>
		<jdbc-classes>
			<driver-classes>
				<class-name>oracle.jdbc.driver.OracleDriver</class-name>
				<!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
			</driver-classes>
			<!--
			<connection-classes>
				<class-name>com.ibm.db2.jcc.am.df</class-name>
				<class-name>com.ibm.db2.jcc.t4.b</class-name>
			</connection-classes>
			-->
		</jdbc-classes>
	</j2ee>
</agent>
```

[그림] 3-7 j2ee main.xml 파일 내용

3.2.1. <server>

<server> 태그는 [표] 3-1과 같이 수집서버에 전송할 정보를 설정한다.

태그	설명
<ip>	에이전트에서 수집된 데이터를 전송할 수집서버의 IP 주소 (동적 반영 가능)
<port>	에이전트에서 수집된 데이터를 전송할 수집서버의 포트번호

[표] 3-1 <server>의 내부 태그

```
<server>
        <ip>127.0.0.1</ip>
        <port>44001</port>
</server>
```

3.2.2. <tier>

<tier> 태그는 데이터를 수집할 에이전트의 Tier를 정의 한다.

Tier	설명
JAVA	Java 데몬 에이전트인 경우 설정
J2EE	WAS 에이전트인 경우 설정
IPLANET	Iplanet WebServer의 에이전트인 경우 설정
APACHE	Apache Webserver의 에이전트인 경우 설정

[표] 3-2 <tier> 태그 설정 값

```
<tier>J2EE</tier>
```

3.2.3. <j2ee>

<j2ee> 태그는 "3.2.2.<tier>" 에서 설정한 값을 tag로 한다. <tier>JAVA</tier>라면 <Java>가 설정되며 <tier>J2EE</tier>라면 <J2EE>가 설정된다.

```
<tier>JAVA</tier> 일때
-----------------------------------
<Java>
        … (중략) …
</Java>
```

```
<tier>J2EE</tier> 일때
-----------------------------------
<j2ee>
        … (중략) …
</j2ee>
```

```
<tier>APACHE</tier> 일때
-----------------------------------
<apache>
        … (중략) …
</apache>
```

참고

에이전트의 메인 설정은 J2EE를 기본으로 설정한다.

3.2.3.1. <operation-config>

<operation-config> 태그는 에이전트에서 수집서버로 Operation 정보를 보내는 방법을 설정한다.

<table>
<tr><th>태그</th><th colspan="2">설명</th></tr>
<tr><td><send-over-time></td><td colspan="2">Operation에 대한 임계 시간을 지정하는 태그로 수집서버로 보낼 Operation 정보량을 조절한다. (millisecond)
지정된 임계 시간 미만인 Operation에 대해서는 <send-level> 태그에 설정한 정보에 따라서 Operation 정보를 수집서버에 전송한다.</td></tr>
<tr><td><send-level></td><td colspan="2"><send-over-time> 태그에 설정한 임계 시간 미만인 Operation에 대해서 수집서버로 Operation 정보를 전송할지 여부를 등록한다.
● 설정값
- SMART: Call-Tree의 첫 번째 Method만 서버로 전송한다.
- NORMAL: Call-Tree 정보를 전송하지 않는다.
㈜ Call-Tree 정보를 전송하지 않기 때문에 통계정보에 포함되지 않는다. 통계정보에 포함시키기 위해서는 "SMART"로 설정해야 한다.</td></tr>
<tr><td><force-send-issue></td><td colspan="2"><send-over-time> 태그에 설정한 임계 시간 미만인 Operation에 대해서 에러가 있는 경우 Operation 정보를 서버로 전송할지 여부를 등록한다.
● 설정값
- true : 에러를 서버에 전송함
- false : 에러를 서버에 전송하지 않음</td></tr>
<tr><td rowspan="4"><use-stack-trace></td><td colspan="2">자동으로 stacktrace를 생성할 것인지 여부에 대한 조건을 설정한다.</td></tr>
<tr><td><connection></td><td>connection leak이 있을 경우 stack trace를 생성한다.
● 설정값 – true : 생성함 / false : 생성하지 않음.</td></tr>
<tr><td><statement></td><td>statement leak이 있을 경우 stack trace를 생성한다.
● 설정값 – true : 생성함 / false : 생성하지 않음.</td></tr>
<tr><td><result-set></td><td>resultSet leak이 있을 경우 stack trace를 생성한다.
● 설정값 – true : 생성함 / false : 생성하지 않음.</td></tr>
<tr><td rowspan="4"><condition></td><td colspan="2">Request에 대한 Operation 정보를 생성할 때 Method Call-Tree가 많으면 수집서버에 부하를 줄 수 있고, 또한 처리시간 매우 짧은 Method도 Call-Tree에 추가함으로써 분석이 어려워지는 것을 방지하기 위하여 아래 3가지 태그 항목을 이용하여 Operation에 Call-Tree 개수를 유기적으로 줄일 수 있다.</td></tr>
<tr><td><min-stack-count></td><td><time-limit>의 임계값에 관계없이 Call-Tree에 포함될 최소한의 개수를 지정한다. 즉, <time-limit> 태그에 지정된 임계값 이하의 Method도 동 태그에 지정된 개수만큼은 Call-Tree에 포함된다.</td></tr>
<tr><td><max-stack-count></td><td>최대 Call-Tree 개수를 지정한다. 동 태그에 지정된 이상의 Call-Tree는 Operation에 포함되지 않는다.</td></tr>
<tr><td><time-limit></td><td>동 태그는 Method 처리 정보를 Operation에 포함시킬지 여부를 판단하기 위한 임계값이다. (millisecond)</td></tr>
</table>

		동 태그에 임계값을 지정하면 Operation 정보에 <min-stack-count> 태그에 지정된 개수까지는 임계값에 관계없이 Call-Tree에 포함되고, <min-stack-count> 태그에 지정된 개수를 초과하는 Method 정보는 <time-limit> 태그에 지정된 임계값을 초과하는 경우에만 Call-Tree에 포함한다. 그러나 Method 처리시간이 <time-limit> 태그에 지정된 임계값을 초과하더라도 Call-Tree 건수는 <max-stack-count> 태그에서 지정한 개수는 초과할 수 없다.
<jdbc> ※기본적으로 사용하지 않도록 되어있다.	<use-sql-helper>	기본적으로 설정에 없는 부분이다. 사용을 한다면 아래의 태그와 같이 추가한다. 어플리케이션의 DB가 Oracle인 경우 SQL PLAN을 지원한다. 이를 사용을 한다면 true로 설정한다. ORACLE에서 SQL PLAN설정은 *<본 문서 "부록 C">*의 내용을 참조한다.

[표] 3-3 <operation-config> 내부태그 설명

```
<operation-config>
        <send-over-time>3000</send-over-time>
        <send-level>SMART</send-level>
        <force-send-issue>true</force-send-issue>
        <use-stack-trace>
                <connection>false</connection>
                <statement>false</statement>
                <result-set>false</result-set>
        </use-stack-trace>
        <condition>
                <min-stack-count>100</min-stack-count>
                <max-stack-count>1000</max-stack-count>
                <time-limit>1000</time-limit>
        </condition>
        <jdbc>
                <use-sql-helper>true</use-sql-helper>
        </jdbc>
</operation-config>
```

위의 설정 예를 보면 <send-over-time> 태그가 3초로 되어있기 때문에 3초 이상의 Operation 데이터를 수집서버로 전송한다. <use-stack-trace> 태그는 모두 false로 되어 있어 자동으로 Stack Trace을 생성하지 않는다. <min-stack-count> 태그가 100으로 설정되어 있어 <time-limit> 태그가 1초로 설정되어 있더라도 1초 미만과 1초 이상의 Call-Tree는 최소한 100개의 Method 호출 정보는 Operation에 포함된다. 100개 이상의 Method 호출 정보는 <time-limit> 태그가 1초로 설정되어 있기 때문에 1초 이상의 Call-Tree 정보만 Operation에 포함된다. 마지막으로 <max-stack-count> 태그가 1000으로 설정되어 있기 때문에 최대 Call-Tree 개수는 1000개를 넘을 수 없다.

참고

1. TX trace연계를 위한 부분일 경우에는 무조건 기록한다. (link-point)
2. error가 발생하는 경우 무조건 기록한다.

3. 자식 operation이 존재하는 경우 무조건 기록한다.
4. min-stack-count보다 작을 경우는 무조건 기록한다. 0인 경우는 time-limit이상인 경우만 기록한다.
    * min-stack-count를 넘어서는 경우
        1. time-limit이상인 경우 기록한다.
        2. SQL수행이고, force-sql인 경우에는 무조건 기록한다.
5. max-stack-count를 넘어서는 경우에는 time-limit을 넘어서더라도 기록을 하지 않는다.
    * max-stack-count가 0인 경우는 모든 SQL을 수집대상으로 한다.
    * -1로 설정된 경우에는 time-limit조건만을 사용하는 것으로 간주한다.

3.2.3.2. <thread-config>

<thread-config> 태그는 Thread별 CPU 사용량 정보를 구할 것인지 여부를 설정한다. true/false의 값을 설정할 수 있으며 기본적으로 기능이 비활성화 되어 있다.

- 설정값 – true : CPU 사용량 정보를 구함
  false : CPU 사용량 정보를 구하지 않음

```
<thread-config>
        <cpu>
                <enable>false</enable>
        </cpu>
</thread-config>
```

3.2.3.3. <web>

<web> 태그의 <visitor-keygen-class>는 방문자 정보 생성시 사용할 Class이름을 설정한다.

```
<web>
        <visitor-keygen-class>com.kwise.pharos.agent.http.DefaultVisitorKeyGenerator</visitor-keygen-class>
</web>
```

3.2.3.4. <jdbc-classes>

<jdbc-classes> 태그의 <driver-classes>는 SQL 수행 정보를 추적하기 위한 설정으로 JDBC Driver 또는 JDBC Connection Class 를 지정하여 설정한다.

1. Oracle의 경우

```
<jdbc-classes>
        <driver-classes>
                <class-name>oracle.jdbc.driver.OracleDriver</class-name>
                <!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
        </driver-classes>
</jdbc-classes>
```

2. DB2의 경우

```
<jdbc-classes>
        <driver-classes>
                <class-name>com.ibm.db2.jcc.DB2Driver</class-name>
                <!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
        </driver-classes>
</jdbc-classes>
```

3. MySQL의 경우

```
<jdbc-classes>
        <driver-classes>
                <class-name>com.mysql.jdbc.Driver</class-name>
                <!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
        </driver-classes>
</jdbc-classes>
```

4. PostgreSQL의 경우

```
<jdbc-classes>
        <driver-classes>
                <class-name>org.postgresql.Driver</class-name>
                <!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
        </driver-classes>
</jdbc-classes>
```

참고

UI CallTree 에 SQL 문이 표시되지 않는 경우, <driver-classes> 태그 중, <class-name>의 해당 Driver 이외의 Driver 가 Agent.log 에 있는지 확인 후, 기록되지 않은 Driver 명을 추가하면 SQL문이 표시된다.

3.3. loader.properties 설정

loader.properties파일은 「$PHAROS_HOME/lib/common-config/properties」에 위치하며 동적 반영을 지원한다.

```
#### print 'PHAROS INFORMATION' to STDOUT at start time ####
# show.pharos.info=true

#### setting for was vendor checking order ####
# was.check.order=jeus,weblogic,JBoss,JBoss7

#### logging instrumented class in agent's log directory
# pharos.debugging.class=false

#### setting for gathering method statistics in memory
# use.memory.method.statistics=false

#### enable to collect JDBC leak information (default: false)
# use.jdbc.leak.check=false
# use.jdbc.leak.connection.check=true
# use.jdbc.leak.statement.check=true
# use.jdbc.leak.resultset.check=true

#### JDBC proxy type: jdk(default) / spy / context
# jdbc.proxy.type=context

#### use HTTP visitor collection on/off
# use.http.visitor=true
# use.http.visitor.cookie.expires=false
# use.http.visitor.cookie.secure=true

#### check remote ip
# http.remote.ip=HEADER:X-Forwarded-For

#### use proxy mode for JSP
# use.proxy.jsp=false

#### machine: sets the count of cores manually
# machine.cpu.ncores=

#### Send data to server with literal
# send.literal.always=false

#### WebLogic user ID. Need to collect connection pool information for WebLogic 8.x
# weblogic.username=

#### WebLogic user passord. Need to collect connection pool information for WebLogic 8.x
# weblogic.password=

#### When weaving class, exclude construct(ex: using JRebel, use this option, true)
# weave.exclude.construct=false

#### When weaving class, exclude method prefix(ex: using JRebel, use below option)
# weave.exclude.method.prefix=__jr,__JR,__rebel

#### Whether appending SOCKET I/O to call-tree
#use.recording.io=false

#### Collect Thread Information
# use.thread.monitor=true
# use.thread.cputime=true

#### Using PHAROS link with othre agents by stream, to read/write GUID
# ex) linkage.sockets.read.list=1344,1355
# linkage.sockets.read.list=
# ex) linkage.sockets.write.list=127.0.0.1:1344,192.168.1.150:1355
# linkage.sockets.write.list=
```

```
#### Active request config
active.request.interval=5000
active.request.time.threshold=0

#### Make call tree for priority
# priority.calltree.layer=

#### Agent lazy initializing after this class loaded. For JBoss OSGi version.
# agent.lazy.init.class.name=java.util.logging.Logger

#### JBoss 7.0 and JBoss 7.1 management setting (default 127.0.0.1,9990)
#JBoss.management.ip=
#JBoss.management.port=

#### use Pharos UI - ClassManager on/off
#use.ui.classmanager=true
```

[그림] 3-8 loader.properties 파일 내용

3.3.1. show.pharos.info

에이전트가 시작될 때 Pharos Information을 stdout으로 표시할 지 여부를 설정한다. 기본값은 true로 설정되어 있다. 특정 JEUS에서 해당 로그가 있을 경우 컨테이너가 실행되지 못하는 경우가 발생할 때 설정값을 false로 설정한다.

- 설정값 – true : stdout에 표시함.
- 설정값 – false : stdout에 표시하지 않음.

3.3.2. was.check.order

WAS 벤더를 체크하는 순서를 설정한다. 기본값은 jeus, weblogic, JBoss, JBoss7로 설정되어 있다.
설정 순서대로 체크한다.

- 설정값 – jeus,weblogic,JBoss,JBoss7

3.3.3. pharos.debugging.class

Class를 weaving할 때 수정된 Class를 「$PHAROS_HOME/logs/$PHAROS_AGENT」에 기록한다. 주로 디버깅 용도로 사용된다. 기본값은 false로 설정되어 있다.

- 설정값 – true : agent.log 에 표시함.
- 설정값 – false : agent.log 에 표시하지 않음.

3.3.4. use.memory.method.statistics

Class Manager에서 보여지는 Method 호출 통계를 생성하는 기능을 on/off 할 수 있는 설정이다. 기본값은 false로 설정되어 있다.

- 설정값 – true : Method 호출 통계를 생성함.
- 설정값 – false : Method 호출 통계를 생성하지 않음.

위 기능은 에이전트에서 Method 호출 별 통계를 생성하는 기능이다. 에이전트에서 통계데이터를 가지고 있으므로, JVM 재기동시 데이터가 초기화 된다. 주로, 프로파일링시 Method 호출빈도를 보여주는 용도로 사용되며, 성능의 영향을 줄 수 있으므로 부하 상황이나, CPU 사용율이 크게 증가하는 경우 옵션 값을 false로 설정한다.

3.3.5. use.jdbc.leak.check

해당 설정은 JDBC leak 정보를 추출할지 여부를 결정한다. 기본값은 true 로 설정되어 있으며 connection leak, statement leak, resultset leak 정보를 각각 추출할지 여부도 설정 가능하다.

- 설정값 – true : leak정보를 추출함.
- 설정값 – false : leak정보를 추출하지 않음.

참고

상관관계로 인해 statement는 connection이 true인 경우에 동작하며, resultset은 statement가 true인 경우에 동작한다.

3.3.5.1. use.jdbc.leak.connection.check

해당 설정은 Connection leak 정보를 추출할지 여부를 결정한다. 기본값은 true 로 설정되어 있다.

- 설정값 – true : Connection leak정보를 추출함.
- 설정값 – false : Connection leak정보를 추출하지 않음.

3.3.5.2. use.jdbc.leak.statement.check

해당 설정은 statement leak 정보를 추출할지 여부를 결정한다. 기본값은 true 로 설정되어 있다.

- 설정값 – true : statement leak정보를 추출함.
- 설정값 – false : statement leak정보를 추출하지 않음.

3.3.5.3. use.jdbc.leak.resultset.check

해당 설정은 resultset leak 정보를 추출할지 여부를 결정한다. 기본값은 true 로 설정되어 있다.

- 설정값 – true : result leak정보를 추출함.
- 설정값 – false : result leak정보를 추출하지 않음.

3.3.6. jdbc.proxy.type

해당 설정은 JDBC 수집방식을 proxy 방식으로 수집하는 설정이다. 기본값은 context 로 설정되어 있다.

- 설정값 – jdk : dynamic proxy를 이용함.
- 설정값 – spy : concrete proxy(delegator)를 이용함.
- 설정값 – context : InitialContext에 대한 proxy를 생성하는 방법을 이용함

3.3.7. use.http.visitor

COOKIE 중, PHAROS_VISITOR를 활성화/비활성화 여부를 설정한다. 기본값은 true 로 설정되어 있다.

- 설정값 – true : 활성화
- 설정값 – false : 비활성화

3.3.8. use.http.visitor.cookie.expires

COOKIE 중, PHAROS_VISITOR의 만료 기한을 설정한다. 기본값은 false 로 설정되어 있다.

- 설정값 – true : 브라우저 쿠키로 1년간 보존
- 설정값 – false : 브라우저를 닫을 때 제거됨

3.3.9. use.http.visitor.cookie.secure

HTTPS 요청에 대한 사용자 정보 수집 여부를 설정한다. 기본값은 true 로 설정되어 있다.

- 설정값 – true : 활성화. PHAROS_VISITOR 쿠키의 값 뒤에 "Secure" 문자열을 붙인다.
- 설정값 – false : 비활성화. "Secure" 문자열을 붙이지 않는다.

3.3.10. use.proxy.jsp

JSP 파일의 사이즈가 큰 경우 ClassFormatError가 발생할 수 있다. 해당 옵션은 jspService에 대해 proxy Method를 사용할 수 있도록 한다. 기본값은 false로 설정되어 있다.

- 설정값 – true : Proxy Method 사용가능
- 설정값 – false : Proxy Method 사용 불가능

3.3.11. machine.cpu.ncores

서버의 CPU (core) 개수를 설정한다. AIX서버의 경우 CPU 사용량이 잘못 표시되는 문제를 수정할 때 사용한다.

3.3.12. send.literal.always

서버에 동일 데이터를 보낼 때 해쉬 값이 아닌 실 데이터로의 전송 여부를 설정한다. 기본값은 false로 설정되어 있다.

- 설정값 – true : 실 데이터로 전송
- 설정값 – false : hash 값으로 전송

3.3.13. weblogic.username

WebLogic 8.x에서 JMX를 통해 ConnectionPool 정보를 추출할 때 사용한다.

- 설정값 – WebLogic Console 접속시의 ID를 설정한다. ID가 없는 경우 설정하지 않아도 된다.

3.3.14. weblogic.password

WebLogic 8.x에서 JMX를 통해 ConnectionPool 정보를 추출할 때 사용한다.

- 설정값 – WebLogic Console 접속시의 Password를 설정한다. Password가 없는 경우 설정하지 않아도 된다.

3.3.15. weave.exclude.construct

Weaving시에 생성자를 대상에서 제거하는 옵션이다. 기본값은 false로 설정되어 있다.

- 설정값 – true : 생성자를 제거함.
- 설정값 – false : 생성자를 제거하지 않음.

3.3.16. weave.exclude.method.prefix

Weaving시에 특정 Prefix를 가진 Method를 제거하는 옵션이다.

- 설정예 – __jr,__JR,__rebel

3.3.17. use.recording.io

Socket에 대한 read/write정보를 Call-Tree에 추가할 수 있도록 하는 기능이다. 기본값은 false로 설정되어 있다. 단, 에이전트 설정에 "-Dpharos.weave.io=true" 가 기술되어 있는 경우만 동작한다.

- 설정값 – true : Call-Tree에 표시함.
- 설정값 – false : Call-Tree에 표시하지 않음.

3.3.18. http.remote.ip

IP정보를 HTTP HEADER의 X-Forwarded-For키가 가진 값을 읽어 remote ip로 사용하는 기능이다.

- 설정예 – HEADER:X-Forwarded-For

3.3.19. use.thread.monitor

Thread 정보 수집여부를 설정한다. 기본값은 true로 설정되어 있다.

- 설정값 – true : Thread 정보 수집함.
- 설정값 – false : Thread 정보 수집하지 않음.

3.3.20. use.method.cputime

Thread 정보 중 cpu time 정보 수집여부를 설정한다. 기본값은 true로 설정되어 있다.

- 설정값 – true : cpu time 정보 수집함.
- 설정값 – false : cpu time 정보 수집하지 않음.

위 기능은 JAVA JMX 를 사용한 데이터 모니터링으로 사용시 CPU 자원의 사용률을 상승 시킬 수 있으므로 부하테스트 및 CPU 자원이 부족할 경우 false 로 설정한다.

3.3.21. linkage.sockets.read.list

agent 가 trace 연계를 위해 다른 agent 에서 stream 으로 들어온 APM Header 를 읽기 위해 사용된다.

- 설정예 – port,port : 1344,1355

3.3.22. linkage.sockets.write.list

agent 가 trace 연계를 위해 다른 agent 로 stream 을 통해 APM Header 를 쓰기 위해 사용된다.

- 설정예 – address:port : 127.0.0.1:1344,192.168.1.150:1355

3.3.23. active.request.interval

Active Request 전송 주기를 설정한다. 기본값은 5000ms 으로 설정되어 있다.

3.3.24. active.request.time.threshold

지정된 시간 이상인 경우 Request정보를 전송하고 지정된 시간 이하인 경우에는 Active Request는 count 만 전송한다. count만 전송될 경우에는 normal(파란색)로 간주된다. 기본값은 0으로 설정되어 있다.

3.3.25. priority.calltree.layer

설정된 layer일때 operation-config 필터링에서 제외시키는 기능

- 설정예 – MCA : MCA layer의 Operaton은 필터링에서 제외한다.

3.3.26. agent.lazy.init.class.name

JBoss LogManager 가 초기화되기 전까지 에이전트의 초기화를 지연시키는 옵션

- 설정값 – java.util.logging.Logger : java.util.logging.Logger class가 로딩된 이후에 에이전트가 초기화를 하게 됨

3.3.27. jboss.management.ip

JBoss AS7 버전 이후부터는 Connection Pool 정보를 jmx에서 취득할 수 없고, JBOSS의 Management에 접속해서 취득할 수 있도록 변경이 되었다. IP의 기본값은 127.0.0.1으로 되어있고, 이를 직접 설정하기 위해서는 주석을 제거하고 IP를 설정한다.

3.3.28. jboss.management.port

JBoss AS7 버전 이후부터는 Connection Pool 정보를 jmx 에서 취득할 수 없고, JBOSS 의 Management 에 접속해서 취득할 수 있다. PORT 의 기본값은 9990 으로 되어있고, 이를 직접 설정하기 위해서는 주석을 제거하고 PORT 를 설정한다.

3.3.29. use.ui.classmanager

ENPHAROSUI 의 ClassAnalyzer 화면에서 Criteria 에 설정된 클래스의 메소드 호출 카운트를 볼수가 있다. 이 화면은 엔지니어들이 criteria 설정을 위해서 참고할 수 있는 화면이다. 이러한 카운트정보는 에이전트에서 내부 메모리를 이용하기 때문에 많은 클래스를 대상으로 criteria 를 설정한다면 메모리누수가 발생될 여지를 가지고 있다. 이 때문에 에이전트 내부에서 메모리 통계를 하지 않도록 하기 위해 이 설정이 추가가 되었다. 기본값은 true 이고, 이를 false 를 하게되면 내부 통계를 하지 않게 되고 ENPHAROSUI 의 ClassAnalyzer 에 항목들이 표시가 되지않는다.

3.4. Criteria 설정

Criteria설정은 수집 대상 Class와 Method들을 정의하는 것을 말하며, 크게 ProfileSet이 정의된 criteria-main.xml과 실제 Profile 대상(Class/Method)이 정의된 xml 설정 파일로 나누어 진다.
설정 파일은 아래의 경로에 위치하며 순서대로 처음 발견된 criteria-main.xml파일이 적용된다.

「$PHAROS_HOME/lib/common-config/criteria」

3.4.1. criteria-main.xml 설정

기본적으로 [그림] 3-11과 같은 내용을 포함하고 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<criteria-config>
        <default-rule>include</default-rule>
        <file-list>
                <file>pharos-default-exclude.xml</file>
                <!—
                <file>weblogic-jdbc.xml</file>
                <file>jdbc.xml</file>
                <file>naming.xml</file>
                -->
                <file>jdbc-proxy.xml</file>
                <file>http-filter.xml</file>
                <file>http-jsp.xml</file>
                <file>http-servlet.xml</file>
        </file-list>
</criteria-config>
```

[그림] 3-9 criteria-main.xml 파일 내용

위에서 <file-list>의 <file> 태그를 이용해 수집 대상 Class와 Method가 정의된 파일 목록을 나열하면 된다. pharos-agent-config 모듈은 해당 파일을 읽어 들인 후 목록을 이용하여 같은 디렉터리에 존재하고 있는 수집 대상 설정 파일을 읽어 들이게 된다.

참고

파일의 나열 순서대로 Class를 로딩할 때 적용이 된다. Class의 정의는 중복이 될 수 있으므로 설정 시 반드시 순서를 고려하여야 한다.

3.4.2. 수집대상 Class와 Method의 정의

설정 파일은 xml 형식이며 파일 이름은 어떤 이름을 사용해도 상관없다. 여기서 정의된 파일 이름을 "3.5.1. criteria-main.xml 설정"의 criteria-main.xml 목록에 추가하면 된다.

아래 [그림] 3-13 의 설정 샘플을 참조한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
        <class-info>
                <rule>include</rule>
                <layer>HTTP</layer>
                <expression>javax.servlet.Filter+</expression>
                <instrument>
                        <startable>false</startable>
                        <recordable>true</recordable>
                        <recording-child>true</recording-child>
                        <capture-running>false</capture-running>
                </instrument>
                <methods>
                        <method>
                                <rule>include</rule>
                                <type>FILTER</type>
                                <expression>public void doFilter(..)</expression>
                                <instrument>
                                        <startable>true</startable>
                                        <recordable>true</recordable>
                                        <recording-child>true</recording-child>
                                        <capture-running>true</capture-running>
                                        <advice>
                                                ...중략...
                                        </advice>

---------------------------------------------------- <이하 생략> ----------------------------------------------------
```

[그림] 3-10 http.xml 의 설정 샘플

XML은 root태그로 <classes>를 가지며 하위에 대상 Class를 정의하는 <class-info> 태그를 가진다.

3.4.2.1. <class-info>

<class-info>는 수집 대상 Class 정보를 정의하는데 사용되며, 아래와 같은 태그들을 가진다.

태그	기본값	생략가능	설명
rule	include	O	include/exclude를 사용할 수 있다. include는 수집 대상을 의미하며, exclude는 제거대상을 의미한다.
layer	GENERAL	O	UI표현 시 보여줄 Class의 Layer명을 의미한다.
weaver	GENERAL	O	Class의 bytecode를 변경할 때 사용할 weaver를 정의한다. 특수 용도로 사용되며, 대부분의 경우 생략된다.
expression			Class 이름에 대한 표현 식
instrument		O	수집 시 필요한 각종 옵션들을 정의한다
methods		O	Method의 목록을 가진다. 만약 목록에 아무것도 정의되어 있지 않다면 해당 Class는 수집 대상이 아닌 것과 마찬가지이다.

[표] 3-4 class-info 태그의 설정

3.4.2.2. <expression>

Class 이름에 대한 표현은 일반적으로 Package 명을 포함한 Class의 전체 이름을 사용하면 된다. 한꺼번에 여러 Class를 지정하고자 하는 경우 '*'를 사용할 수 있으며 상속을 표현하고자 하는 경우 Expression의 마지막에 '+'를 사용하면 된다. 아래 [그림] 3-13의 설정 샘플을 참조한다.

```
oracle.jdbc.driver.OracleDriver
oracle*OracleDriver
oracle.*Driver
java.sql.Driver+
java.sql.Connection+
```

[그림] 3-11 Class 이름 설정 샘플

참고

'+'와 '*'를 동시에 사용할 수는 없다.

3.4.2.3. <instrument>

<instrument>는 Method 정보 수집 시 필요한 옵션들을 정의하는데 사용한다. 이 태그는 <class-info> 태그와 <methods> 태그, <method> 태그에서 모두 사용될 수 있으나 최종 적용되는 대상은 Method 이므로 세 가지 중 하나만 적용된다.

적용되는 순서는 <method> -> <methods> -> <class-info>의 순이며 앞 순서가 생략되는 경우 뒤 순서가 적용되는 방식이다. 아래와 같은 하위 태그를 지닌다.

태그	기본값	생략 가능	설명
stacktracable	true	O	현재는 자료의 표현에는 사용하지 않고 내부적으로만 사용하고 있다.
startable	false	O	Call-Tree 수집시 시작 Method로 사용할지를 설정하는데 사용된다. 시작 Method가 있어야만 Call-Tree가 수집되기 시작한다. HttpServlet의 service() Method가 좋은 예이다.
recordable	true	O	Call-Tree에 포함시킬지 여부를 설정한다. Startable Method가 시작된 이후에만 의미가 있다.
recording-child	true	O	하위의 Call-Tree를 계속 수집할지를 정의하는데 사용된다. false인 경우 하위의 Method가 recordable 일지라도 수집하지 않는다.
capture-running	false	O	해당 Method가 수행 상태일 때 서버로 상태정보를 전송 한다. 이 태그를 이용하여 HTTP/JDBC의 CAT차트가 만들어진다.
captures		O	Method에 대한 추가 정보를 수집하고자 할 경우 사용한다.
advice		O	특정 Method에서 사용자가 만든 코드를 수행하고자 할 경우 사용한다.
advices		O	여러 개의 advice를 등록하고자 하는 경우 사용한다. 내부에는 위에 있는 advice태그를 중복해서 나열하면 된다.

[표] 3-5 instrument 태그의 설정

3.4.2.4. <captures>

<captures> 태그는 <instrument>태그 내에 정의하여 Method 수행 시 참조되는 값들을 추출하는 용도로 사용된다. 수집된 정보는 Call-Tree에서 해당 Method (Operation)의 meta-data 정보로 저장된다.
여러 개의 값들을 추출하는 경우도 있으므로 <captures> 태그는 <capture> 태그를 목록으로 가진다.
<capture> 태그는 아래 [표] 3-6과 같은 하위 태그들을 가지고 있다.

태그	기본값	생략가능	설명
Name			추출된 값을 meta-data에 저장할 때 사용할 키 이름
Value			추출 대상. "[표] 3-7" 참조
is-global	false	O	추출된 meta-data를 해당 Method(Operation)과 함께 Start Method의 meta-data에도 등록한다. UI에서 쉽게 확인을 하고자 하거나 이 데이터를 이용하여 서버에서 특정 작업을 수행하고자 하는 경우 사용될 수 있다.

[표] 3-6 captures 태그의 설정

위에서 설명한 태그 중 <value> 태그에서 사용할 수 있는 값들은 [표] 3-7에 설명한다.

설정 가능한 값	결과값(예제)	설명
인자번호 ex)3		설정 값에 위치한 인자의 값을 추출한다. Object인 경우는 toString() 값을 표현한다.
3(key1, key2)	[value1, value2]	해당 위치의 인자가 Map타입인 경우 괄호 속에 나열된 키 값에 대한 데이터를 검출하여 나열한다.
return		Method의 수행 결과를 얻어낸다. Object인 경우 toString() 값을 표현한다.
this.classname	a.b.ClassName	현재 수행 중인 Class에 대한 전체 이름을 추출한다.
This.simple-classname	ClassName	현재 수행 중인 Class의 Package를 제외한 Class 이름만을 추출한다.
methodname	connect	현재 수행 중인 Method의 이름만을 추출한다.
full-classname-and-methodname	a.b.ClassName.connect()	Class 이름과 Method 이름을 추출한다. 인자는 표현하지 않는다.
simple-classname-and-methodname	ClassName.connect()	Class 이름과 Method 이름을 추출한다. Package 명과 인자는 표현하지 않는다.
full-signature		Call-Tree에서 표현되는 것처럼 완전한 이름을 얻어낸다.

[표] 3-7 captures에 대한 정의

3.4.2.5. <advice>

advice 태그는 Method 수행 시점에 사용자가 생성한 특정 advice class를 수행시키고자 하는 경우에 사용된다. 특수한 기능을 추가하거나 특정한 정보를 추출하기 위한 용도로 제공된다.

<advice>는 아래 [표] 3-8과 같은 하위 태그를 가지고 있다.

태그	기본값	생략 가능	설명
key-name		O	advice에 의해 리턴되는 값이 있는 경우 해당 이름으로 meta-data에 저장된다. 생략되는 경우 저장하지 않는다.
class-path		O	수행하고자 하는 Class가 있는 위치(dir/jar). 생략될 경우 에이전트가 가지고 있다고 간주한다.
classname			실제 수행을 하고자 하는 advice class의 이름

[표] 3-8 advice에 대한 정의

3.4.2.6. <methods> <method>

Method 이름에 대한 정의를 표현한다. 리턴 타입이나 인자에서 Class 이름이 사용될 경우에는 Package를 포함한 전체 이름을 사용해야 한다. Method는 크게 아래와 같이 몇 가지의 단위로 구성되어 있다.

[접근자] [리턴타입] [이름] ([인자],...)

목록	설명
접근자	public, private, static, final 등을 지칭하며, 생략하거나 '*'로 모든 접근자를 허용
리턴타입	primitive type이나 배열, Class 이름으로 사용 가능 '*'를 사용하여 모든 타입을 허용
이름	Method의 이름을 의미한다. '*'를 사용하여 여러 Method를 한번에 지정할 수 있다.
인자	primitive type 이나 배열, Class 이름을 사용하여 나열하며 구분자는 콤마(,)를 사용. 만약 인자를 조건으로 사용하지 않고자 한다면 '..'를 사용

[표] 3-9 Method 이름 설정

아래 [그림] 3-14 의 설정 샘플을 참조한다.

```
public java.lang.Connection connect(java.lang.String,java.lang.String,java.lang.String)
public * connect(..)
public * conn*(java.lang.String,..)
* * connect(..)
public static final int getNumber()
```

[그림] 3-12 Method 이름 설정 샘플

3.5. Criteria 설정 샘플

3.5.1. 특정 Class의 모든 method를 프로파일링 하는 예제

프로파일링의 대상으로 특정 클래스의 모든 method를 지정하기 위한 방법으로 아래 [그림] 3-15와 같이 설정한다. <class-info> 하단 <expression>에 package 명을 포함한 대상 Class명을 입력하고, <method> 하단 <expression>에 * *(..)와 같이 입력한다. 이와 같은 설정으로 대상 Class의 모든 method에 대한 프로파일링 할 수 있다. 본 예제는 ParamCaptureController 컨트롤러 Class에서 사용하는 모든 method를 모니터링 하기 위한 프로파일링이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
        <class-info>
            <rule>include</rule>
            <layer>HTTP</layer>
            <expression>com.rubymaster.codej.controller.ParamCaptureController</expression>
                <methods>
                        <method>
                                <rule>include</rule>
                                <type>CONTROLLER</type>
                                <expression>* *(..)</expression>
                                <instrument>
                                        <startable>false</startable>
                                        <recordable>true</recordable>
                                        <recording-child>true</recording-child>
                                        <capture-running>false</capture-running>
                                </instrument>
                        </method>
                </methods>
         </class-info>
</classes>
```

[그림] 3-13 특정 Class의 모든 Method를 프로파일링으로 등록하는 예제

3.5.2. 특정 Class의 특정 method를 프로파일링 하는 예제

프로파일링의 대상으로 특정 Class의 method를 지정하기 위한 방법으로 아래 [그림] 3-16과 같이 설정한다. <class-info> 하단 <expression>에 package 명을 포함한 대상 Class명을 입력하고, <method> 하단 <expression>에 public void service(java.lang.String)를 입력한다. 이와 같은 설정으로 대상 Class의 특정 method에 대한 프로파일링 할 수 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
        <class-info>
            <rule>include</rule>
            <layer>HTTP</layer>
            <expression>com.rubymaster.codej.controller.ParamCaptureController</expression>
                <methods>
                        <method>
                                <rule>include</rule>
                                <type>CONTROLLER</type>
                                <expression>public void service(java.lang.String)</expression>
                                <instrument>
                                        <startable>false</startable>
                                        <recordable>true</recordable>
                                        <recording-child>true</recording-child>
                                        <capture-running>false</capture-running>
                                </instrument>
                        </method>
                </methods>
         </class-info>
</classes>
```

[그림] 3-14 특정 Class의 특정 method를 프로파일링으로 등록하는 예제

3.5.3. 생성자(Constructor)를 프로파일링 하는 예제

프로파일링의 대상으로 특정 Class의 생성자를 별도로 프로파일링 하기 위해 아래 [그림] 3-17과 같은 방법으로 설정한다. Classloader에 <init> 로 보여지는 부분을 xml로 표현하기 위해 &ltinit&gt를 사용하기 때문에 아래와 같이 설정해야 생성자 프로파일링이 가능하다. 생성자 method에 parameter 값으로 특정 값이 존재할 경우 기존의 프로파일링 방법과 동일하게 &ltinit&gt(java.lang.String, java.lang.String 와 같이 설정할 경우 대상 parameter를 포함한 생성자를 프로파일링 할 수 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
        <class-info>
            <rule>include</rule>
            <layer>HTTP</layer>
            <expression>com.rubymaster.codej.controller.InitTest</expression>
                <methods>
                        <method>
                                <rule>include</rule>
                                <type>CONTROLLER</type>
                                <expression>public &ltinit&gt(..)</expression>
                                <instrument>
                                        <startable>false</startable>
                                        <recordable>true</recordable>
                                        <recording-child>true</recording-child>
                                        <capture-running>false</capture-running>
                                </instrument>
                        </method>
                </methods>
        </class-info>
</classes>
```

[그림] 3-15 생성자(Constructor)를 프로파일링 하는 예제

3.5.4. 특정 Class를 Start Point(StartOperation)으로 설정하는 방법

TCP통신과 같이 HTTP 를 통해서 들어오는 요청이 아닌 경우나 multi thread 방식으로 새로운 Class를 실행하는 경우 혹은, JOB Thread로 새로운 작업을 시작하는 경우에는 새로운 Operation으로 Agent가 서버에게 전송해 줘야 한다. 이를 위한 방법으로 아래 [그림] 3-18 과 같이 설정할 수 있다.

아래의 예제는 Excutor Class에서 invoke() Method로 시작되는 Thread를 startOperation으로 설정하고 Server에 전송하게 하는 예제이다. <startable>은 시작 Operation으로 등록한다는 의미 이며, <capture-running>은 이 Operation을 CAT(Current Active Thread)로 Server에 전송한다는 의미이다. 그 이외의 부분은 Criteria 설정에 상세히 설명되어 있으므로 참조한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
        <class-info>
            <rule>include</rule>
            <layer>JOB</layer>
            <expression>com.rubymaster.codej.job.Excutor</expression>
                <methods>
                        <method>
                                <rule>include</rule>
                                <type>INVOKER</type>
                                <expression>public void invoke(..)</expression>
                                <instrument>
                                        <startable>true</startable>
                                        <recordable>true</recordable>
                                        <recording-child>true</recording-child>
                                        <capture-running>true</capture-running>
                                </instrument>
                        </method>
                </methods>
        </class-info>
</classes>
```

[그림] 3-16 특정 Class를 Start Point로 설정하는 방법

3.5.5. 특정 Class의 특정 Method에 advice를 적용하는 방법

프로파일링 대상이 되는 Class의 Method에서 데이터를 추출하기 위해서는 Adivce를 사용해야 한다. 기본적으로 제공하는 기본 Advice 및 Site에 맞게 설정된 Advice를 적용하는 방법은 아래 [그림] 3-19와 같다. 아래의 예제는 ServiceInvoker Class의 process(java.lang.String) mehtod에서 데이터를 추출하기 위해

${pharos.home}/lib/advices/paa-biz-advice.jar 파일의 com.pj.advices.ServiceInvokerAdvice를 사용하겠다는 의미를 가진다. 이 기능을 설정하는 <advice> 태그는 <instrument>아래에 위치해야 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
  <class-info>
    <rule>include</rule>
    <layer>HTTP</layer>
    <expression>com.rubymaster.codej.biz.ServiceInvoker</expression>
        <methods>
          <method>
            <rule>include</rule>
            <type>PROCESS</type>
            <expression>public void process(java.lang.String)</expression>
            <instrument>
              <startable>false</startable>
              <recordable>true</recordable>
              <recording-child>true</recording-child>
              <capture-running>false</capture-running>
              <advice>
                <class-path>${pharos.home}/lib/advices/paa-biz-advice.jar</class-path>
                <classname>com.pj.advices..ServiceInvokerAdvice</classname>
              </advice>
            </instrument>
          </method>
        </methods>
  </class-info>
</classes>
```

[그림] 3-17 특정 Class의 특정 Method에 Adivce를 적용하는 방법

3.5.6. 다중 advice를 적용하는 방법

특정 Class의 특정 Method에 여러 개의 Advice를 적용하는 방법은 아래 [그림] 3-20과 같다.

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
  <class-info>
    <rule>include</rule>
    <layer>HTTP</layer>
    <expression>com.rubymaster.codej.biz.ServiceInvoker</expression>
        <methods>
          <method>
            <rule>include</rule>
            <type>PROCESS</type>
            <expression>public void process(java.lang.String)</expression>
            <instrument>
              <startable>false</startable>
              <recordable>true</recordable>
              <recording-child>true</recording-child>
              <capture-running>false</capture-running>
              <advices>
                <advice>
                  <class-path>${pharos.home}/lib/advices/paa-biz-advice.jar</class-path>
                  <classname>com.pj.advices..ServiceInvokerAdvice</classname>
                </advice>
                <advice>
                  <class-path>${pharos.home}/lib/advices/paa-biz-advice.jar</class-path>
                  <classname>com.pj.advices..ServiceInvokerAdvice</classname>
                </advice>
              <advices>
            </instrument>
          </method>
        </methods>
  </class-info>
</classes>
```

[그림] 3-18 다중 Adivce를 적용하는 방법

04

부록

4. 부록

A. 에이전트 주요 설정 파일

A.1. main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<agent>
	<server>
		<ip>127.0.0.1</ip>
		<port>44001</port>
	</server>
	<tier>J2EE</tier>
	<j2ee>
		<!--
		<layers>
			<in>HTTP</in>
			<out>JDBC</out>
		</layers>
		-->
		<operation-config>
			<send-over-time>0</send-over-time>
			<send-level>SMART</send-level>
			<force-send-issue>true</force-send-issue>
			<use-stack-trace>
				<connection>false</connection>
				<statement>false</statement>
				<result-set>false</result-set>
			</use-stack-trace>
			<condition>
				<min-stack-count>100</min-stack-count>
				<max-stack-count>1000</max-stack-count>
				<time-limit>1000</time-limit>
			</condition>
		</operation-config>
		<thread-config>
			<cpu>
				<enable>false</enable>
			</cpu>
		</thread-config>
		<web>
			<visitor-keygen-
class>com.kwise.pharos.agent.http.DefaultVisitorKeyGenerator</visitor-keygen-class>
		</web>
		<jdbc-classes>
			<driver-classes>
				<class-name>oracle.jdbc.driver.OracleDriver</class-name>
				<!-- class-name>com.informix.jdbc.IfxDriver</class-name -->
			</driver-classes>
			<!--
			<connection-classes>
				<class-name>com.ibm.db2.jcc.am.df</class-name>
				<class-name>com.ibm.db2.jcc.t4.b</class-name>
			</connection-classes>
			-->
		</jdbc-classes>
	</j2ee>
</agent>
```

A.2. criteria-main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<criteria-config>
	<default-rule>include</default-rule>
	<file-list>
		<file>pharos-default-exclude.xml</file>
		<!—
		<file>weblogic-jdbc.xml</file>
		<file>jdbc.xml</file>
		<file>naming.xml</file>
		-->
		<file>jdbc-proxy.xml</file>
		<file>http-filter.xml</file>
		<file>http-jsp.xml</file>
		<file>http-servlet.xml</file>
	</file-list>
</criteria-config>
```

A.3. loader.properties

```
#### print 'PHAROS INFORMATION' to STDOUT at start time ####
# show.pharos.info=true

#### setting for was vendor checking order ####
# was.check.order=jeus,weblogic

#### logging instrumented class in agent's log directory
# pharos.debugging.class=false

#### setting for gathering method statistics in memory
# use.memory.method.statistics=false

#### enable to collect JDBC leak information (default: false)
# use.jdbc.leak.check=false
# use.jdbc.leak.connection.check=true
# use.jdbc.leak.statement.check=true
# use.jdbc.leak.resultset.check=true

#### JDBC proxy type: jdk(default) / spy / context
# jdbc.proxy.type=context

#### use HTTP visitor collection on/off
# use.http.visitor=true

#### check remote ip
# http.remote.ip=HEADER:X-Forwarded-For

#### use proxy mode for JSP
# use.proxy.jsp=false

#### machine: sets the count of cores manually
# machine.cpu.ncores=

#### Send data to server with literal
# send.literal.always=false

#### WebLogic user ID. Need to collect connection pool information for WebLogic 8.x
# weblogic.username=

#### WebLogic user passord. Need to collect connection pool information for WebLogic 8.x
# weblogic.password=

#### When weaving class, exclude construct(ex: using JRebel, use this option, true)
# weave.exclude.construct=false

#### When weaving class, exclude method prefix(ex: using JRebel, use below option)
# weave.exclude.method.prefix=__jr,__JR,__rebel

#### Whether appending SOCKET I/O to call-tree
#use.recording.io=false

#### Collect Thread Information
# use.thread.monitor=true
# use.thread.cputime=true

#### Using PHAROS link with othre agents by stream, to read/write GUID
# ex) linkage.sockets.read.list=1344,1355
# linkage.sockets.read.list=
# ex) linkage.sockets.write.list=127.0.0.1:1344,192.168.1.150:1355
# linkage.sockets.write.list=

#### Active request config
active.request.interval=5000
active.request.time.threshold=0

#### Make call tree for priority
# priority.calltree.layer=

#### Agent lazy initializing after this class loaded. For JBoss OSGi version.
# agent.lazy.init.class.name=java.util.logging.Logger
```

B. 과부하 및 매크로 차단기능 설정

일정 시간 동안 과다한 요청을 발생시킬 경우 해당 사용자를 감지하여 특정 페이지로 redirect하는 기능은 아래와 같다. 설정법은 main.xml에서 설정하며, 아래의 기능은 during의 시간 동안 count이상의 호출을 실행한 사용자의 경우 delete-time동안 reject-page를 주는 기능이다.

```
<web>
    ....
    <macro>
        <enable>true</enable>
        <during>60000</during>
        <count>20</count>
        <delete-time>60000</delete-time>
        <reject-page>http://xxx.xxx.xxx.xxx/macro.html</reject-page>
        <key>
            <cookie>JSESSIONID</cookie>
        </key>
    </macro>
</web>
```

각 태그의 의미는 아래와 같다.

태그	설명
enable	사용여부
during	체크 시간
delete-time	차단 해제 시간
count	호출 횟수
reject-page	차단 조건이 되었을 경우 보여줄 페이지
key	사용자를 구분하는데 사용하는 값

B.1. SPC 기능

위와 같이 설정을 하면 동시 수행 thread가 30개 이상일 경우 busy-page를 보여준다.

```
<web>
    ....
    <spc>
        <enable>true</enable>
        <thread-limit>30</thread-limit>
        <busy-page>http://xxx.xxx.xxx.xxx/busy.html</busy-page>
    </spc>
</web>
```

각 태그의 의미는 아래와 같다.

태그	설명
enable	사용여부
thread-limit	제한 thread 갯수
busy-page	조건이 되었을 때 보여줄 페이지
group	그룹별 설정(생략가능)

C. Pharos Java SQL PLAN 정보 생성 및 확인

C.1. SQL plan 정보 추출 전 확인 사항

<수집서버에서 대상 SQL문이 실행되는 DB로의 접근 가능 여부>

실제 plan정보를 추출하는 것은 에이전트가 아닌 수집서버에서 직접 대상 DB로 접근을 하여 수행하기 때문이다.

<대상 user가 dictionary에 접근할 수 있는 권한을 가지고 있어야 한다.>

Plan정보를 추출할 때 'EXPLAIN PLAN ~'문을 사용하는데 이 때 dictionary정보에 접근을 해야하기 때문이다. 해당 권한은 DBA가 아래와 같은 쿼리를 이용해 권한을 할당하면 된다.

```
GRANT SELECT ANY DICTIONARY TO USER;
```

만약 모든 유저에게 권한을 할당하고 싶다면 아래 구문을 사용하면 된다.

```
GRANT SELECT ANY DICTIONARY TO PUBLIC;
```

<대상 SQL문을 수행하는 user가 테이블을 생성할 권한이 있는지 여부>

수집서버에서 plan정보를 추출할 때 특정 테이블을 필요로 한다. 만약 특정 테이블이 존재하지 않는다면 생성하는 작업을 하기 때문에 user가 테이블을 생성할 수 있는 권한을 지니고 있어야 한다.
만약 테이블 생성권한을 제공하지 않는다면 아래 쿼리문을 사용하여 테이블을 생성해 주어도 된다.

```
CREATE TABLE PHAROS_PLAN_TABLE(
        STATEMENT_ID            VARCHAR2(30),
        PLAN_ID                 NUMBER,
        TIMESTAMP               DATE,
        REMARKS                 VARCHAR2(4000),
        OPERATION               VARCHAR2(30),
        OPTIONS                 VARCHAR2(255),
        OBJECT_NODE             VARCHAR2(128),
        OBJECT_OWNER            VARCHAR2(30),
        OBJECT_NAME             VARCHAR2(30),
        OBJECT_ALIAS            VARCHAR2(65),
        OBJECT_INSTANCE         NUMERIC,
        OBJECT_TYPE             VARCHAR2(30),
        OPTIMIZER               VARCHAR2(255),
        SEARCH_COLUMNS          NUMBER,
        ID                      NUMERIC,
        PARENT_ID               NUMERIC,
        DEPTH                   NUMERIC,
        POSITION                NUMERIC,
        COST                    NUMERIC,
```

```
        CARDINALITY                 NUMERIC,
        BYTES                       NUMERIC,
        OTHER_TAG                   VARCHAR2(255),
        PARTITION_START             VARCHAR2(255),
        PARTITION_STOP              VARCHAR2(255),
        PARTITION_ID                NUMERIC,
        OTHER                       LONG,
        DISTRIBUTION                VARCHAR2(30),
        CPU_COST                    NUMERIC,
        IO_COST                     NUMERIC,
        TEMP_SPACE                  NUMERIC,
        ACCESS_PREDICATES           VARCHAR2(4000),
        FILTER_PREDICATES           VARCHAR2(4000),
        PROJECTION                  VARCHAR2(4000),
        TIME                        NUMERIC,
        QBLOCK_NAME                 VARCHAR2(30)
);

CREATE UNIQUE INDEX PHAROS_PLAN_TABLE_IDX ON PHAROS_PLAN_TABLE(STATEMENT_ID, ID);
```

C.2. SQL plan 정보 정상적으로 추출되는지 확인 방법

"B.1. SQL plan 정보 추출 전 확인 사항" 모두 확인 후 아래와 같은 방법으로 plan 정보가 정상적으로 추출되는지 확인해 볼 수 있다.

① 테스트는 SQLPlus를 이용한다.

```
sqlplus user/password
```

② 아래 쿼리문을 수행하여 plan정보를 생성한다.

```
explain plan set statement_id = '1' into pharos_plan_table for
select * from dual;
```

③ 결과를 확인한다.

```
SELECT decode(id,0,operation||'(OPTIMIZER='||optimizer||')',operation) OPERATION,
level,
position,
options,
decode(object_name,null,null,object_owner||'.'||object_name) OBJECT,
object_type,
```

```
object_instance ,
cost,
cardinality,
bytes,
cpu_cost,
io_cost,
access_predicates,
filter_predicates
FROM PHAROS_PLAN_TABLE
START WITH ID= 0 and STATEMENT_ID = '1'
CONNECT by prior ID=PARENT_ID and STATEMENT_ID='1'
ORDER BY ID;
```