CA Application Performance Management

Java Agent 구현 안내서

릴리스 9.5

CA Technologies 제품 참조

이 문서에서는 다음과 같은 CA Technologies 제품과 기능을 참조합니다.

- CA Application Performance Management(CA APM)
- CA Application Performance Management ChangeDetector(CA APM ChangeDetector)
- CA Application Performance Management ErrorDetector(CA APM ErrorDetector)
- CA Application Performance Management for CA Database Performance(CA APM for CA Database Performance)
- CA Application Performance Management for CA SiteMinder®(CA APM for CA SiteMinder®)
- CA Application Performance Management for CA SiteMinder®Application Server Agents(CA APM for CA SiteMinder® ASA)
- CA Application Performance Management for IBM CICS Transaction Gateway(CA APM for IBM CICS Transaction Gateway)
- CA Application Performance Management for IBM WebSphere Application Server(CA APM for IBM WebSphere Application Server)
- CA Application Performance Management for IBM WebSphere Distributed Environments(CA APM for IBM WebSphere Distributed Environments)
- CA Application Performance Management for IBM WebSphere MQ(CA APM for IBM WebSphere MQ)
- CA Application Performance Management for IBM WebSphere Portal(CA APM for IBM WebSphere Portal)
- CA Application Performance Management for IBM WebSphere Process Server(CA APM for IBM WebSphere Process Server)
- CA Application Performance Management for IBM z/OS®(CA APM for IBM z/OS®)
- CA Application Performance Management for Microsoft SharePoint(CA APM for Microsoft SharePoint)
- CA Application Performance Management for Oracle Databases(CA APM for Oracle Databases)
- CA Application Performance Management for Oracle Service Bus(CA APM for Oracle Service Bus)

- CA Application Performance Management for Oracle WebLogic Portal(CA APM for Oracle WebLogic Portal)
- CA Application Performance Management for Oracle WebLogic Server(CA APM for Oracle WebLogic Server)
- CA Application Performance Management for SOA(CA APM for SOA)
- CA Application Performance Management for TIBCO BusinessWorks(CA APM for TIBCO BusinessWorks)
- CA Application Performance Management for TIBCO Enterprise Message Service(CA APM for TIBCO Enterprise Message Service)
- CA Application Performance Management for Web Servers(CA APM for Web Servers)
- CA Application Performance Management for webMethods Broker(CA APM for webMethods Broker)
- CA Application Performance Management for webMethods Integration Server(CA APM for webMethods Integration Server)
- CA Application Performance Management Integration for CA CMDB(CA APM Integration for CA CMDB)
- CA Application Performance Management Integration for CA NSM(CA APM Integration for CA NSM)
- CA Application Performance Management LeakHunter(CA APM LeakHunter)
- CA Application Performance Management Transaction Generator(CA APM TG)
- CA Cross-Enterprise Application Performance Management
- CA Customer Experience Manager(CA CEM)
- CA Embedded Entitlements Manager(CA EEM)
- CA eHealth® Performance Manager(CA eHealth)
- CA Insight™ Database Performance Monitor for DB2 for z/OS®
- CA Introscope®
- CA SiteMinder®
- CA Spectrum® Infrastructure Manager(CA Spectrum)
- CA SYSVIEW® Performance Management(CA SYSVIEW)

CA 에 문의

기술 지원팀에 문의

온라인 기술 지원 및 지사 목록, 기본 서비스 시간, 전화 번호에 대해서는 http://www.ca.com/worldwide에서 기술 지원팀에 문의하십시오.

목차

제 1 장: Java Agent 소개

이 섹션은 다음 항목을 포함하고 있습니다.

Introscope 및 Java Agent 정보

CA Introscope 는21 프로덕션 환경에서 복잡한 웹 응용 프로그램을 매일 24 시간 모니터링하고, 고객에게 영향을 미치기 전에 문제를 감지하고, 감지된 문제를 빠르게 공동 작업으로 해결할 수 있게 해 주는 엔터프라이즈 응용 프로그램 성능 관리 솔루션입니다. 이 솔루션의 핵심 아키텍처 요소는 오버헤드가 낮은 에이전트입니다.

에이전트는 Introscope 의 데이터 수집 구성 요소로, 트랜잭션이 실행되는 컴퓨팅 환경과 응용 프로그램에 대한 세부적인 성능 정보를 수집하는 기능을 합니다. Java Agent 는 JVM(Java Virtual Machine)에서 실행되는 응용 프로그램 및 리소스에서 이 정보를 수집한 다음 추가 처리를 위해 Enterprise Manager 로 보냅니다.

Java Agent 는 응용 프로그램에서 사용되는 JVM 을 구성하는 구성 요소의 바이트 코드에 *프로브*를 삽입합니다. 바이트 코드에 프로브를 삽입하는 것은 응용 프로그램을 모니터링할 수 있게 해 주는 *계측* 프로세스의 일부입니다.

응용 프로그램을 계측하려면 PBD(ProbeBuilder 지시문) 파일에 정의된 추적 프로그램도 필요합니다. PBD 파일의 명령 또는 지시문은 모니터링할 응용 프로그램 구성 요소를 식별합니다. 추적 프로그램은 응용 프로그램이 JVM 에서 실행될 때 에이전트가 어떤 프로브에서 어떤 메트릭을 수집해야 하는지 식별합니다. PBD 파일을 사용자 환경에 맞게 변경하여 모니터링되는 항목을 제어할 수 있습니다.

Java Agent 를 설치하면 사용자 환경에 대한 기본 모니터링 기능을 사용할 수 있도록 몇 개의 기본 PBD 파일이 배포됩니다. 이 기본 모니터링 기능을 수정하여 가시성과 성능 간의 균형을 필요에 맞게 조정할 수 있습니다. 응용 프로그램이 계측된 후 Java Agent 는 사용자가 원하는 데이터를 수집하여 Enterprise Manager 에 보고합니다. 그러면 Enterprise Manager 는 실시간 보고 및 기록 보고용으로 데이터를 처리하고 저장합니다. Introscope Workstation 을 사용하면 수집된 데이터를 보고 필요한 작업을 수행하여 경고를 생성하거나 적절한 조치를 수행할 수 있습니다.

응용 프로그램 관리를 위한 주요 동작은 다음과 같습니다.

- 응용 프로그램 서버의 성능 및 가용성을 모니터링할 수 있는 에이전트를 배포합니다.
- 응용 프로그램 구성 요소의 모니터링을 테스트, 조정 및 최적화합니다.
- 필요한 대로 에이전트 작업을 제어하도록 에이전트 프로필을 사용자 지정합니다.
- 응용 프로그램 또는 에이전트별로 메트릭 그룹, 대시보드, 경고 및 작업을 생성합니다.
- 응용 프로그램 문제를 조사, 심사 및 진단합니다.

Java Agent 배포 계획

배포를 계획할 때의 목적은 에이전트 오버헤드와 응용 프로그램 성능에 대한 가시성 간의 균형을 적절하게 맞추는 것입니다. 에이전트의 오버헤드가 낮으면 프로덕션 환경에서 모든 트랜잭션을 실시간으로 모니터링할 수 있습니다. 오버헤드를 낮게 유지하면 중요한 응용 프로그램 및 서버 리소스의 성능과 가용성이 향상됩니다. 그러나 오버헤드를 낮게 유지하면 문제가 발생할 때 문제를 진단하는 데 충분한 정보가 제공되지 않는다는 단점도 있습니다. 따라서 전체 응용 프로그램 수명 주기에 맞게 에이전트 구성을 배포하고 조정하는 것이 좋습니다. 또한 응용 프로그램을 개발하거나 테스트할 때 모니터링하는 구성 요소가 많을수록 응용 프로그램이 프로덕션에 릴리스된 후의 구성 요소 수를 줄일 수 있습니다.

기본 기능 설치 및 평가

에이전트를 배포하는 첫 번째 단계에서는 기본 에이전트 구성을 설치하고 평가합니다. 기본 에이전트 구성에서는 컴퓨팅 환경과 응용 프로그램의 일반적인 여러 구성 요소에 대한 데이터 수집을 보여 줍니다. 또한 기본 에이전트 구성에서 일부 기능은 사용하도록 설정된 반면 사용 빈도가 적은 다른 기능은 사용하지 않도록 설정되어 있습니다.

이 단계의 목적은 기본적으로 제공되는 데이터 수집의 수준 및 범위를 평가하고 Introscope 와 응용 프로그램 모니터링 방법에 익숙해지는 것입니다. 에이전트에서 기본적으로 제공되는 성능 메트릭에 익숙해진 후에는 필요에 따라 데이터를 수집하도록 에이전트를 사용자 지정할 수 있습니다.

기본 성능을 평가할 때는 에이전트에서 수집하는 메트릭이 많을수록 시스템 리소스가 많이 소비된다는 점에 유의하십시오. 반면 에이전트에서 수집하는 메트릭이 적을수록 잠재적 문제를 파악하기가 어려워집니다. 에이전트 구성을 미세 조정할 때는 데이터 수집 수준과 허용 가능한 성능 수준 간의 균형을 맞추도록 하십시오. 적절한 계측 수준은 일반적으로 에이전트가 배포되는 위치에 따라 달라집니다. 예를 들어 테스트 환경 내에서 모니터링하는 에이전트는 대개 많은 수의 메트릭을 수집하도록 구성됩니다. 그러나 프로덕션 서버의 에이전트는 대개 꼭 필요한 정보만 제공하도록 구성됩니다.

구성 요구 사항 확인

에이전트를 배포하기 전에 데이터 수집 요구 사항을 확인하십시오. 이 정보를 통해 에이전트의 데이터 수집 동작을 수정하고, 대체 에이전트 구성을 사용할 경우의 오버헤드 영향을 평가할 수 있습니다.

Introscope 는 응용 프로그램 수명 주기의 모든 단계에서 사용할 수 있습니다. 예를 들어 개발 단계부터 테스트, 부하 확인, 스테이징 및 프로덕션 단계에 이르는 전 과정에서 Introscope 를 사용할 수 있습니다. 수명 주기의 각 단계마다 모니터링 목적, 환경적 제약 및 서비스 수준 기대치가 서로 다를 수 있습니다. 이러한 차이를 해결하려면 에이전트가 모니터링 대상 환경의 유형에 따라 다르게 동작하도록 구성해야 합니다.

이 단계의 목적은 성능 세부 정보의 가시성과 리소스 오버헤드 사이에서 적절한 균형이 이루어지는 지점을 확인하는 것입니다. 또한 모니터링 대상 환경에 합리적인 비용으로 최적 수준의 가시성을 얻을 수 있어야 합니다.

개발과 같은 프로덕션 전 응용 프로그램 환경에서는 일반적으로 데이터 수집 수준을 높게 구성할수록 응용 프로그램 성능을 세부적으로 파악할 수 있습니다. 프로덕션 또는 고용량 트랜잭션 환경에서는 일반적으로 보고되는 메트릭 수를 줄여 에이전트 오버헤드를 제어합니다. 요구 사항에 따라 에이전트 속성을 구성하여 특정 에이전트 동작을 제어할 수도 있습니다. 예를 들어 수집되는 최대 개수의 메트릭을 추적하고 오래된 메트릭을 제거할 수 있습니다.

현재 환경을 고려하여 적절한 수준의 가시성과 허용 가능한 성능 오버헤드를 확인하고 그에 따라 요구 사항에 맞게 에이전트를 구성할 수 있습니다.

적절한 구성 속성을 사용하여 기준 에이전트 프로필 정의

모니터링할 응용 프로그램 환경의 유형을 확인한 후에는 "후보" 에이전트 구성을 생성할 수 있습니다. 대부분의 에이전트 작업은 에이전트 프로필(IntroscopeAgent.profile) 내의 속성을 사용하여 제어됩니다. 예를 들어 IntroscopeAgent.profile 파일에서는 에이전트가 사용하는 ProbeBuilder 지시문 파일과 ProbeBuilder 목록 파일을 정의합니다. 또한 에이전트 프로필에 나열된 파일은 에이전트가 수집하는 특정 메트릭을 제어합니다. IntroscopeAgent.profile 파일에서는 특정 기능을 사용 또는 사용하지 않도록 설정하는 속성이나 폴링 간격과 같이 작업을 조정하는 속성도 제공합니다.

사용 중인 구성 및 환경에 따라 에이전트 프로필의 속성을 조정하여 각 변경 사항의 영향을 평가할 수 있습니다. 예를 들어 처음에는 ChangeDetector 를 사용하도록 설정되지 않은 기본 에이전트 프로필을 사용하고, 나중에 프로필에서 ChangeDetector 속성을 사용하도록 설정하여 다른 기능을 추가하기 전에 변경 후의 에이전트 성능을 평가할 수 있습니다.

에이전트 성능 오버헤드 평가

에이전트 구성을 평가할 때는 수집된 메트릭을 통해 응용 프로그램의 성능과 가용성을 충분하게 파악할 수 있는지 확인하십시오. 또한 메트릭 수가 너무 많아 운영 환경에 지나친 부하가 발생하지는 않는지 확인하십시오. 에이전트에서는 성능 및 가용성 문제를 파악하고 지역화하는 데 필요한 것 이상의 메트릭을 보고할 수 없습니다.

응용 프로그램의 성능 특성을 이해하면 에이전트 성능을 효과적으로 평가할 수 있습니다. 예를 들어 영향을 확인하기 위해 기본 모니터링을 구현하기 전과 그 후에 응용 프로그램에 대한 부하 테스트를 실행할 수 있습니다.

데이터 수집을 제어된 방식으로 확장하려면 변경 사항을 구현하기 전후에 에이전트 작업 및 오버헤드를 확인하십시오. 예를 들어 각 추가 기능의 성능을 개별적으로 평가할 수 있도록 한 번에 한 응용 프로그램에 대한 모니터링 지원만 추가하십시오.

에이전트 구성 유효성 검사 및 배포

후보 에이전트 구성이 필요한 가시성을 제공하는 것으로 확인된 후에는 해당 환경에 구성을 배포하십시오.

대상 환경에 다음 구성 항목을 설치하면 유효성이 검사된 구성을 배포할 수 있습니다.

- IntroscopeAgent.profile 파일
- 수정되거나 사용자 지정된 PBD 파일

Java Agent 배포

에이전트를 배포하는 과정은 대략적으로 다음과 같은 단계로 이루어집니다.

1. 대상 컴퓨터에 에이전트를 설치합니다.
2. 응용 프로그램을 계측하도록 응용 프로그램 서버를 구성 (see page 41)합니다.
3. 에이전트 및 에이전트 프로필의 위치를 포함하도록 응용 프로그램 서버의 시작 스크립트 또는 Java 명령을 구성 (see page 42)합니다.
4. Enterprise Manager 에 대한 에이전트 연결을 구성 (see page 68)합니다.
5. 응용 프로그램 서버를 다시 시작하여 응용 프로그램을 계측하고 데이터 수집을 시작합니다.
6. 설치 중에 설정된 속성을 변경하거나 데이터 수집 또는 다른 에이전트 작업을 수정하려는 경우 IntroscopeAgent.profile 을 구성 (see page 77)합니다.

제 2 장: Java Agent 설치 및 구성

이 섹션은 다음 항목을 포함하고 있습니다.

에이전트를 설치하기 전에

에이전트를 설치하기 전에 다음 단계를 수행하십시오.

1. Java Agent 배포 과정을 검토합니다.
2. 다음과 같은 지원되는 버전의 응용 프로그램 서버가 있는지 확인합니다.

   참고: 에이전트를 설치할 서버에서는 지원되는 버전의 JVM 을 로컬로 사용할 수 있어야 합니다. 응용 프로그램 서버 및 JVM 요구 사항은 *Compatibility Guide*(호환성 안내서)를 참조하십시오.
3. 지원되는 버전의 JVM 이 있는지 확인합니다. 지원되는 버전을 사용할 수 없는 경우 이전 버전을 사용합니다.
   - Java 1.4.x 의 경우 8.x Java Agent 를 사용합니다.
   - Java 1.3.x 의 경우 7.2. Java Agent 를 사용합니다.

4. Introscope Enterprise Manager 및 Workstation 구성 요소가 설치되어 있는지 확인합니다. 에이전트와 Enterprise Manager 의 연결을 위한 호스트 이름 및 포트 번호를 확인합니다.

   ping 또는 텔넷을 사용하여 에이전트와 Enterprise Manager 간의 연결을 확인할 수 있습니다.

참고: Enterprise Manager, Workstation 및 WebView 구성 요소를 설치하는 방법에 대한 자세한 내용은 *CA APM 설치 및 업그레이드 안내서*를 참조하십시오.

추가 정보:

Java Agent 배포 계획 (페이지 22)
Java Agent 배포 (페이지 25)

Java Agent 설치 방법 선택

다음 방법 중 하나로 Java Agent 를 설치할 수 있습니다.

- GUI(그래픽 사용자 인터페이스) 또는 텍스트 기반 콘솔 설치 관리자를 대화식으로 사용
- 대화식 프롬프트 없이 편집된 응답 파일을 사용하여 자동으로 설치
- 수동으로 개별 응용 프로그램 서버용 파일 추출 및 구성

컴퓨터에 파일을 로컬로 설치하려는 경우 대개 대화식 설치 관리자를 사용합니다. 대화식으로 설치할 경우 표시되는 프롬프트는 그래픽 또는 텍스트 기반 인터페이스를 사용할 때와 동일합니다. 그러나 GUI 또는 텍스트 기반 설치 관리자를 선택하는 옵션은 설치 관리자를 실행하는 컴퓨터의 운영 체제에 따라 달라집니다. 예를 들어 대부분의 UNIX 환경에서는 GUI 또는 텍스트 기반 콘솔 설치 관리자가 지원되지만 기본적으로는 콘솔 설치 관리자가 시작됩니다.

컴퓨터에 파일을 원격으로 설치하거나 미리 구성된 일련의 설치 지침을 배포하려는 경우에는 편집된 응답 파일을 사용하여 자동으로 설치 관리자를 실행할 수 있습니다.

설치 관리자를 대화식이나 자동 방식으로 실행하지 않으려는 경우에는 설치 아카이브를 사용하여 에이전트를 수동으로 설치하고 구성할 수 있습니다. 수동 설치 방법을 사용할 경우 응용 프로그램 및 운영 체제별 아카이브에서 특정 에이전트 파일 집합을 추출한 다음 배포 옵션을 수동으로 구성할 수 있습니다. CA Technologies 에서는 응용 프로그램 서버 및 운영 체제 유형이 동일한 여러 시스템에 Java Agent 를 신속하게 배포할 수 있도록 이러한 아카이브를 제공합니다.

대화식, 자동 또는 수동 설치 방법에 대한 자세한 내용은 관련 단원을 참조하십시오.

대화식으로 Java Agent 설치

운영 체제에 적절한 Java Agent 설치 관리자를 선택하여 시작하고 프롬프트에 응답하는 대화식 방법으로 Java Agent 를 설치할 수 있습니다. GUI 설치 관리자를 사용하는 경우 드롭다운 메뉴를 사용하고 확인란을 선택하여 선택 항목을 지정할 수 있습니다. 텍스트 기반 콘솔 설치 관리자에서는 텍스트를 직접 입력하는 방법으로 옵션을 선택합니다.

다음 단계를 따르십시오.

1. 운영 체제에 적절한 설치 아카이브를 선택합니다. 예:
   - Microsoft Windows 에 설치하려면 IntroscopeAgentInstaller<*version*>windows.zip 을 선택합니다.
   - 지원되는 UNIX 또는 Linux 운영 체제에 설치하려면 IntroscopeAgentInstaller<*version*>unix.tar 를 선택합니다.
   - IBM z/OS 에 설치하려면 IntroscopeAgentInstaller<*version*>zOS.tar 를 선택합니다.
   - IBM OS/400(IBM i 운영 체제)에 설치하려면 IntroscopeAgentInstaller<*version*>os400.zip 을 선택합니다.
2. 운영 체제에 적절한 명령을 사용하여 설치 아카이브 파일을 추출합니다. 예를 들어 UNIX 또는 Linux 의 경우 다음 명령을 사용합니다.
   `tar –xvf IntroscopeAgentInstaller<version>unix.tar`
3. 메시지에 따라 설치를 시작합니다.
4. 응용 프로그램 서버 루트 디렉터리의 위치를 지정합니다.

   응용 프로그램 서버 루트 디렉터리를 지정하지 않으려는 경우에는 기본 응답(Windows 의 경우 C:\, UNIX 의 경우 /)을 그대로 사용합니다.

5. 유효한 응용 프로그램 서버 목록에서 응용 프로그램 서버 유형을 선택합니다.

   선택하는 응용 프로그램 서버에 따라 사용할 수 있는 추가 모니터링 옵션이 결정됩니다.

6. Java Agent 의 루트 설치 디렉터리를 지정합니다. 대부분의 경우 응용 프로그램 서버 루트 디렉터리를 사용합니다.

   설치 관리자는 루트 설치 디렉터리 내에 *<Agent_Home>* 디렉터리인 "wily" 디렉터리를 생성합니다.

7. 에이전트 프로필을 생성할지 기존 에이전트 프로필을 사용할지를 지정합니다.

   에이전트 프로필을 생성할 경우 다음을 선택해야 합니다.

   - 표준 또는 전체 계측
   - 에이전트 이름 및 프로세스 이름
   - Enterprise Manager 호스트 및 포트 번호

   기존 프로필을 선택할 경우 파일 위치를 지정해야 합니다. 파일의 정규화된 경로를 지정하십시오.

   지정한 설정은 필요한 경우 설치 후에 IntroscopeAgent.profile 파일을 편집하여 변경할 수 있습니다.

8. 응용 프로그램 서버를 시작하는 데 사용되는 Java 실행 파일의 경로를 지정합니다.

9. 응용 프로그램을 계측하기 위한 ProbeBuilding 메서드를 지정합니다. 대부분의 경우 JVM AutoProbe 를 사용합니다.

10. ChangeDetector 에이전트 확장을 사용할지 여부를 지정합니다.

    ChangeDetector 를 사용하도록 설정할 경우 ChangeDetector 에이전트의 이름을 지정해야 합니다.

    ChangeDetector 를 사용하도록 설정하지 않을 경우 ChangeDetector 파일은 *<Agent_Home>*/examples 디렉터리에 설치됩니다. 나중에 파일을 *<Agent_Home>*/core/ext 디렉터리에 복사하고 IntroscopeAgent.profile 파일을 수정하여 ChangeDetector 를 사용하도록 설정할 수 있습니다.

    참고: ChangeDetector 에 대한 자세한 내용은 *CA APM ChangeDetector 사용자 안내서*를 참조하십시오.

11. 설치할 추가 모니터링 옵션을 선택합니다. 예를 들어 웹 서비스 및 다른 SOA 환경 구성 요소를 모니터링하려면 CA APM for SOA 를 선택합니다.

    설치 중에 다른 모니터링 옵션을 사용하도록 설정하지 않으면 파일이 *<Agent_Home>*/examples 디렉터리에 설치되며 나중에 이를 사용하도록 설정할 수 있습니다.

12. 에이전트와 함께 설치할 추가 기능이 포함된 .zip 또는 .tar 파일에 대한 "픽업 폴더"의 위치를 지정합니다.

    지정한 위치에 있는 .zip 또는 .tar 파일은 *<Agent_Home>* 디렉터리에 추출됩니다.

13. 선택 항목을 검토하고 에이전트 설치를 계속하여 설치를 완료합니다.

참고: 에이전트를 설치할 때 응용 프로그램 서버가 중지 또는 시작되거나 응용 프로그램 서버 시작 스크립트가 구성되지는 않습니다. 설치 후 수동으로 이 작업을 수행하십시오. 구체적인 단계는 모니터링하는 응용 프로그램 서버의 유형에 따라 달라집니다.

자동으로 Java Agent 설치

자동 모드에서는 명령줄에서 에이전트 설치 관리자를 호출하고 설치 지침이 포함된 응답 파일을 지정합니다. 그러면 진행 상태에 대한 정보가 표시되지 않고 백그라운드에서 자동으로 설치가 실행됩니다. 이렇게 설치할 경우 사용자 상호 작용 없이 에이전트를 설치할 수 있으므로 원격 컴퓨터에 에이전트를 설치하거나 구성이 동일한 여러 에이전트를 설치하려는 경우에는 이 방법이 가장 일반적으로 사용됩니다.

이전에 Java Agent 를 설치한 경우 자동으로 생성된 응답 파일을 사용하여 추가 에이전트를 설치할 수 있습니다. 또는 설치 시 제공된 샘플 응답 파일을 수동으로 편집하거나 텍스트 편집기를 사용하여 사용자 고유의 응답 파일을 생성할 수 있습니다.

자동 생성 응답 파일 정보

대화식 또는 자동으로 Java Agent 설치 관리자를 실행할 때마다 선택한 설치 옵션을 기록하는 응답 파일이 생성됩니다. 이 자동 생성 응답 파일은 *<Agent_Home>/install* 디렉터리에 저장됩니다. 파일 이름은 다음과 같은 형식으로 응답 파일이 생성된 날짜 및 시간을 나타냅니다.

```
autogenerated.responsefile.<year>.<month>.<day>.<hour>.<minutes>.<seconds>
```

예를 들어 2005 년 4 월 30 일 오전 7 시 10 분 05 초에 설치한 경우 다음과 같은 이름의 응답 파일이 자동으로 생성됩니다.

```
autogenerated.responsefile.2005.4.30.7.10.05
```

동일한 설정의 후속 자동 설치에 이 자동 생성 응답 파일을 사용하거나, 새 설정을 사용하도록 응답 파일을 편집할 수 있습니다.

샘플 응답 파일 정보

이전에 Java Agent 를 설치하지 않았거나 이전에 선택한 설치 옵션 대신 기본값을 사용하려는 경우 Java Agent 설치 관리자에 포함된 기본 샘플 응답 파일을 편집할 수 있습니다. 기본 샘플 응답 파일은 *<Agent_Home>*\install 디렉터리에 다음 이름으로 포함되어 있습니다.

```
SampleResponseFile.Agent.txt
```

샘플 응답 파일은 대부분의 속성에 대한 기본 설정을 제공하지만 자동 설치에 이 파일을 사용하려면 먼저 파일을 수동으로 편집해야 합니다.

응답 파일 속성 구성 및 에이전트 설치

자동 모드에서 Java Agent 설치 관리자를 호출하기 전에 자동 생성 응답 파일을 사용할지, 기본 샘플 응답 파일을 사용할지, 아니면 사용자 지정 응답 파일을 생성할지에 따라 파일의 속성을 적절하게 구성해야 합니다. 응답 파일에서 설정하는 속성은 설치 관리자를 대화식으로 실행할 때 선택하는 속성과 같습니다.

참고: 속성 설정에 대한 자세한 내용은 *<Agent_Home>*/install/SampleResponse.Agent.txt 파일의 주석을 참조하십시오.

다음 단계를 따르십시오.

1. 응답 파일을 텍스트 편집기에서 엽니다.
2. 적절한 속성 값을 설정합니다. 구성할 속성은 다음과 같습니다.

USER_INSTALL_DIR=*<root_installation_directory>*

에이전트를 설치할 디렉터리를 지정합니다. 대부분의 경우 응용 프로그램 서버 루트 디렉터리를 사용해야 합니다.

silentInstallChosenFeatures=Agent

설치할 구성 요소를 지정합니다.

appServer=Default

모니터링할 응용 프로그램 서버의 유형을 지정합니다. 유효한 값은 Default, JBoss, Tomcat, WebLogic, WebSphere, Sun, Oracle 또는 Interstage 입니다. 값은 대/소문자를 구분합니다.

이 설정은 에이전트와 함께 설치되는 PBD(ProbeBuilder 지시문) 및 유효한 추가 모니터링 옵션을 제어합니다.

(선택 사항) appServerHome=

응용 프로그램 서버의 홈 디렉터리를 지정합니다. USER_INSTALL_DIR 속성을 응용 프로그램 서버 루트 디렉터리로 설정한 경우에는 이 속성이 필요하지 않습니다.

(선택 사항) appServerJavaExecutable=

응용 프로그램 서버를 시작하는 데 사용되는 Java 실행 파일의 경로를 지정합니다.

instrumentationLevel=Typical

전체 계측을 사용할지 표준 계측을 사용할지를 지정합니다. 값은 대/소문자를 구분합니다.

agentName=Default Agent

Workstation 에 표시할 에이전트 이름을 지정합니다.

processName=Default Process

Workstation 에 표시되는 프로세스 이름을 지정합니다.

emHost=localhost

에이전트가 기본적으로 연결해야 하는 Enterprise Manager 를 실행하는 컴퓨터의 호스트 이름을 지정합니다.

emPort=5001

에이전트가 Enterprise Manager 에 연결하는 데 사용해야 하는 포트 번호를 지정합니다.

(선택 사항) alternateAgentProfile=

기존 에이전트 프로필에 대한 절대 경로를 지정합니다.

(선택 사항) pickupFolder=

에이전트와 함께 설치할 추가 기능의 .zip 또는 .tar 파일이 포함된 "픽업 폴더"에 대한 절대 경로를 지정합니다.

(선택 사항) changeDetectorEnable=false

ChangeDetector 에이전트 확장을 사용할지 여부를 지정합니다. 이 속성을 false 로 설정해도 ChangeDetector 파일이 설치되지만 사용되지는 않습니다. 나중에 이를 사용하도록 설정할 수 있습니다.

(선택 사항) changeDetectorAgentID=

ChangeDetector 에이전트 확장의 이름을 지정합니다. ChangeDetector 를 사용하도록 설정할 경우 이 속성의 주석 처리를 제거하고 값을 설정하십시오.

(선택 사항) shouldEnable*

사용하도록 설정할 추가 CA APM 모니터링 솔루션을 지정합니다. 선택적 CA APM 모니터링 솔루션의 모든 속성은 기본적으로 false 로 설정됩니다. 사용하도록 설정할 수 있는 옵션은 응용 프로그램 서버 유형에 따라 달라집니다.

3. 응답 파일을 저장하고 텍스트 편집기를 닫습니다.
4. 설치 관리자 실행 파일의 경로와 응답 파일의 절대 경로를 지정하여 자동 모드에서 설치 관리자를 호출합니다.

   ```
   <path_to_installer> -f <absolute_path_to_response-file>
   ```

   설치 관리자를 호출하는 데 사용하는 명령은 명령을 실행할 운영 체제에 따라 달라집니다.

   예를 들어 Windows 의 경우 다음과 같은 형식의 명령을 사용합니다.

   ```
   IntroscopeAgent<version>windows.exe -f C:\temp\myResponseFile.txt
   ```

   Linux 또는 UNIX 의 경우 다음과 같은 형식의 명령을 사용합니다.

   ```
   ./IntroscopeAgent<version>unix.bin -f /
   ```

   운영 체제에 적절한 명령 형식을 선택하십시오.
5. *<Agent_Home>/install/Introscope_Agent_<version>_InstallLog.log 파일*을 검사하여 에이전트가 설치되었는지 확인합니다.

설치 아카이브를 사용하여 수동으로 설치

Java Agent 설치 관리자를 대화식으로 실행하거나 응답 파일을 구성하지 않고도 시스템에 에이전트 파일을 배치할 수 있습니다. 응용 프로그램 서버별 아카이브를 사용하여 에이전트를 설치할 수 있습니다.

설치 아카이브에는 에이전트 설치 관리자를 실행할 경우 설치되는 모든 파일이 포함되어 있습니다. 아카이브를 컴퓨터에 복사한 후 해당 내용을 추출하고 에이전트 프로필을 구성하여 연결할 Enterprise Manager 와 기타 속성을 지정할 수 있습니다. 이러한 파일을 사용하여 여러 에이전트를 배치 작업으로 배포하거나 해당 파일을 기본 에이전트 파일 집합의 아카이브로 보관하십시오.

아카이브에서 수동으로 Java Agent 를 설치하려면 설치할 컴퓨터의 사용 가능한 디스크 공간이 35 MB 이상인지 확인하십시오.

다음 단계를 따르십시오.

1. 응용 프로그램 서버 및 운영 체제에 적절한 설치 아카이브를 선택합니다.
2. 운영 체제에 적절한 명령을 사용하여 JVM 에서 액세스할 수 있는 위치에 아카이브의 내용을 추출합니다. 예를 들어 UNIX 또는 Linux 의 경우 다음 명령을 사용합니다.
   `tar -xvf  IntroscopeAgentFilesOnly-NoInstaller<version><app_server>.<os>.tar`
3. <*Agent_Home*>/core/config/IntroscopeAgent.profile 파일을 텍스트 편집기에서 열고 Enterprise Manager 에 대한 연결을 구성합니다.

   Enterprise Manager 와 통신할 수 있도록 속성을 설정하는 방법에 대한 자세한 내용은 Enterprise Manager 에 대한 연결 구성 (see page 68)을 참조하십시오.
4. 추가 속성을 구성한 다음 IntroscopeAgent.profile 파일을 저장하고 닫습니다.
5. Java Agent 시작 파일 및 에이전트 프로필의 위치로 응용 프로그램 서버를 구성합니다.
6. 응용 프로그램 서버를 다시 시작합니다.

Java Agent 디렉터리 구조 정보

에이전트를 설치하면 루트 설치 디렉터리에 다음과 같은 디렉터리 구조가 생성됩니다.

wily

이 디렉터리는 에이전트를 시작하는 데 사용되는 Agent.jar 가 포함된 *<Agent_Home>* 디렉터리입니다.

<Agent_Home> 디렉터리 내의 하위 디렉터리에는 Java Agent 의 다양한 기능을 사용할 수 있게 해 주는 라이브러리 및 확장 파일이 있습니다.

core

- config

  에이전트 작업, 메트릭 데이터 수집 및 계측 프로세스를 제어하는 IntroscopeAgent.profile, ProbeBuilder 지시문 파일(.pbd)과 ProbeBuilder 목록 파일(.pbl)이 포함되어 있습니다. IntroscopeAgent.profile 에 정의된 특정 속성과 에이전트 프로필에서 배포되고 참조되는 .pbd 및 .pbl 파일은 설치 중에 선택한 옵션에 따라 달라집니다.

  config 디렉터리 내의 hotdeploy 하위 디렉터리를 사용하면 IntroscopeAgent.profile 을 편집하거나 응용 프로그램을 다시 시작하지 않고도 새 지시문을 배포할 수 있습니다. hotdeploy 디렉터리에 있는 파일이 잘못된 구문을 포함하거나 너무 많은 메트릭을 포함하는 경우에는 계측이 실패하거나 응용 프로그램 성능이 저하될 수 있습니다.

- ext

  사용하도록 설정된 에이전트 확장 또는 기능에 대한 파일이 포함되어 있습니다. 예를 들어 이 디렉터리에는 응용 프로그램 심사 맵 및 LeakHunter 에 대한 파일이 포함되어 있습니다.

connectors

특정 환경에서 계측을 사용할 수 있도록 AutoProbe 커넥터를 구성하는 데 사용되는 CreateAutoProbeConnector.jar 유틸리티가 포함되어 있습니다.

common

확장에 대한 구성 및 속성 파일이 포함되어 있습니다.

예

CA APM for SOA 와 같은 선택적 에이전트 확장에 대한 폴더 및 파일이 포함되어 있습니다. 설치 시 확장을 사용하도록 설정하지 않은 경우 나중에 이 디렉터리의 파일을 사용하여 확장을 구성할 수 있습니다.

install

설치 프로세스를 기록하는 로그 파일과 자동 설치에 사용할 수 있는 파일이 포함되어 있습니다. 예를 들어 생성된 응답 파일과 SampleResponseFile.Agent.txt 파일이 이 디렉터리에 있습니다.

설치 아카이브에서 수동으로 Java Agent 를 설치할 경우에는 이 디렉터리가 생성되지 않습니다.

로그

에이전트 로그 파일이 저장됩니다.

tools

다음과 같은 내용을 포함합니다.

- 웹 서버 로그 파일을 분석하는 URLGrouper.jar 명령줄 유틸리티
- 확장 파일의 목록. 예를 들어 configurePMI.bat, CreateIU.jar, listServers.bat, MergeUtility.jar, NetInterface.jar 및 setPmiModules.jacl 파일이 이 디렉터리에 있습니다.
- TagScript 도구 파일: TagScript.jar, TagScript.bat 및 TagScript.sh 명령줄 스크립트

UninstallerData

에이전트 및 관련 리소스를 제거하는 데 사용되는 실행 파일과 관련 리소스를 포함하는 하위 디렉터리가 있습니다.

설치 아카이브에서 수동으로 Java Agent 를 설치할 경우에는 이 디렉터리가 생성되지 않습니다.

version

선택적 CA APM 모니터링 솔루션에 대한 버전 정보가 포함되어 있습니다. 이 정보는 설치 시 사용하도록 설정한 항목에 관계없이 설치됩니다.

가상 트랜잭션 감지 구성

가상 트랜잭션 모니터링의 구성 설정은 introscope.agent.synthetic.header.names 매개 변수를 사용하여 지정할 수 있습니다.

introscope.agent.synthetic.header.names 매개 변수 값은 모니터링된 HTTP 요청이 가상 트랜잭션의 일부인지 여부를 확인하는 데 사용되는 HTTP 헤더 매개 변수를 나열합니다. 개별 매개 변수 이름은 쉼표로 구분됩니다. 이 매개 변수를 정의하지 않거나 값을 비워 두면 가상 트랜잭션이 감지되지 않습니다. HTTP 헤더 매개 변수 이름을 여러 개 정의할 경우, 매개 변수를 지정한 순서대로 검토가 이루어집니다. 값이 지정된 첫 번째 HTTP 매개 변수가 가상 트랜잭션 정의에 사용됩니다.

가상 트랜잭션이 보고되는 노드는 다음과 같이 각 트랜잭션을 감지하는 데 사용되는 특정 HTTP 헤더 매개 변수에 따라 결정됩니다.

- 매개 변수 값이 *lisaframeid* 또는 *x-wtg-info* 이외의 값이면 HTTP 매개 변수 값 자체가 노드 이름으로 사용됩니다. 유효한 노드 이름이 사용되도록 적절한 수정 작업이 수행됩니다.
- 매개 변수 값이 *lisaframeid* 인 경우 가상 노드 이름은 CA LISA 입니다.
- 매개 변수 값이 *x-wtg-info* 인 경우 HTTP 헤더 매개 변수 값에 이름 및 값 쌍 시퀀스가 포함된 것으로 간주됩니다. 각 쌍은 앰퍼샌드 기호로 구분되며 각 쌍 내에서 특성 이름과 값은 등호로 구분됩니다. 가상 트랜잭션 노드 이름은 *group, name, ipaddress* 및 *request_id* 의 값을 | 노드 구분 기호로 구분하여 구성됩니다.

예를 들어 매개 변수가 다음과 같은 경우:

```
introscope.agent.synthetic.header.names=Synthetic_Transaction,x-wtg-info,lisaframeid
```

다음의 *x-wtg-info* 헤더를 사용할 경우 SampleGroup|sample|192.168.193.1|start 라는 노드 아래에 메트릭이 보고됩니다.

```
clear
synthetic=true&instance=ewing&name=sample&group=SampleGroup&version=4.1.0&ipaddress=192.168.193.1&sequencenumber=1&request_id=start&executiontime=1226455047
```

x-wtg-info HTTP 헤더 매개 변수 값에 정의되지 않은 모든 특성에는 다음과 같은 기본값이 사용됩니다.

- *group*=unknownGroup
- *name*=unknownScript
- *ipaddress*=0.0.0.0
- *request_id*=Action

introscope.agent.synthetic.header.names 를 정의하지 않으면 다음과 같은 구성 매개 변수가 무시됩니다.

`introscope.agent.synthetic.node.name=Synthetic Users`

가상 트랜잭션으로 인식된 트랜잭션이 보고되는 노드입니다. 이 노드는 *Frontends(프런트엔드)|Apps(응용 프로그램)|<WebAppName>*에 위치하며, 여기서 *<WebAppName>*은 웹 응용 프로그램 이름입니다. 이 값은 기본적으로 Synthetic Users 입니다.

`introscope.agent.non.synthetic.node.name=Real Users`

가상 트랜잭션으로 인식되지 않은 트랜잭션이 보고되는 노드입니다. 이 노드는 *Frontends(프런트엔드)|Apps(응용 프로그램)|<WebAppName>*에 위치하며, 여기서 *<WebAppName>*은 웹 응용 프로그램 이름입니다. 정의하지 않을 경우 *<WebAppName>* 아래에 노드가 추가로 생성되지 않습니다.

`introscope.agent.synthetic.user.name=Synthetic_Trace_By_Vuser`

해당 값이 가상 사용자 이름으로 사용되는 HTTP 헤더 매개 변수의 이름입니다. 가상 사용자 이름은 서로 다른 가상 트랜잭션을 구분하기 위해 사용됩니다. 각 가상 사용자 이름에 해당하는 노드가 *Synthetic User* 노드 아래에 생성됩니다. 이 구성 매개 변수가 정의되고 동일한 이름의 HTTP 헤더 매개 변수가 존재하면 가상 트랜잭션 메트릭이 보고됩니다. 트랜잭션은 *<Synthetic Users>|<Synthetic User>* 노드 아래에 보고됩니다.

- *introscope.agent.synthetic.node.name* 구성 매개 변수가 *<Synthetic Users>* 노드 이름을 결정합니다.
- HTTP 헤더 매개 변수 값이 *<Synthetic User>* 노드 이름을 결정합니다.

참고: 이러한 속성의 변경 내용은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

TagScript 유틸리티 사용

CA TagScript 유틸리티를 HP Vugen 과 함께 사용하여 가상 사용자 정보의 추출을 지정할 수 있습니다.

TagScript 유틸리티를 사용하려면

1. TagScript 유틸리티를 엽니다.

   Windows:

   ```
   <Agent_Home>\wily\tools\TagScript.bat
   ```

   UNIX:

   ```
   <Agent_Home>/wily/tools/TagScript.sh
   ```

   어느 환경을 대상으로 스크립트를 수정할지 묻는 메시지가 표시됩니다.

2. 다음 옵션 중 하나를 클릭합니다.
   - "Performance Testing"(성능 테스트) - HP Loadrunner 스크립트
   - "프로덕션" - HP Business Process Monitor 또는 Sitescope 스크립트
   - "Un-tag"(태그 해제) - 태그 지정 프로세스 전환
3. HP Vugen 스크립트가 저장되어 있는 디렉터리로 이동한 후 각 .c 스크립트를 두 번 클릭하여 엽니다.

   HP Vugen 스크립트 .c 파일이 모두 백업되고 수정된 버전으로 대체됩니다.
4. HP Vugen 이 열려 있고 유틸리티가 실행 중이면 수정된 스크립트를 다시 로드하라는 메시지가 나타납니다. 이 메시지가 표시되면 "모두 예"를 클릭하십시오.
5. TagScript 유틸리티를 닫거나, 파일 선택 대화 상자에서 "취소" 단추를 클릭합니다. TagScript 유틸리티를 반드시 닫아야 하는 것은 아니며, 대부분의 사용자는 HP Vugen 을 사용하는 동안 이 유틸리티를 열어 둡니다. 스크립트가 수정되거나 새로운 스크립트가 생성된 경우, 유틸리티를 열어 두면 프로세스가 더 간단합니다.
6. 다음과 같은 위치에서 스크립트 태그가 지정되었는지 확인합니다.
   - 각 스크립트의 시작 부분에 HP Vugen 코드의 신규 단락이 삽입되었는지 확인합니다.
   - 모든 lr_start_transaction 및 lr_end_transaction 전과 스크립트 맨 마지막에 태그가 표시되는지 확인합니다.

7. (선택 사항) 개별 Blame 스택 집합을 사용하여 각 가상 사용자를 HP Loadrunner 성능 테스트를 통해 추적합니다. 각 사용자를 추적하려면 스크립트 시작 부분의 선언 단락에서 다음 행에 대한 주석 처리를 제거합니다.

   ```
   web_add_auto_header("Synthetic_Trace_By_Vuser",vuserOverview)
   ```

   참고: "프로덕션" 태그 지정 스크립트에 대해 이 옵션의 주석 처리를 제거하면 각 요소 또는 가상 생성기에 대해 개별 Blame 스택 집합이 생성됩니다.

Java 7 Autoprobe

Java 로 컴파일되어 실행되는 응용 프로그램을 사용하는 경우 -XX:-UseSplitVerifier 를 추가하십시오. 그렇지 않으면 다음과 유사한 바이트 코드 확인 오류가 발생할 수 있습니다.

```
java.lang.VerifyError: StackMapTable error: bad offset(java.lang.VerifyError:
StackMapTable 오류: 잘못된 오프셋)
```

또는

```
java.lang.ClassFormatError: Illegal local variable
table(java.lang.ClassFormatError: 잘못된 로컬 변수 테이블)
```

이 지침은 javaagent 또는 Xbootclasspath 를 통해 JVM AutoProbe 를 사용하는 경우에 적용됩니다.

응용 프로그램 계측 방법

에이전트를 설치한 후에는 응용 프로그램을 계측하도록 응용 프로그램 서버를 구성해야 합니다. 응용 프로그램을 계측하는 가장 일반적인 방법은 JVM AutoProbe 와 – *javaagent* 명령줄 옵션을 사용하는 것입니다. JVM AutoProbe 는 런타임에 동적으로 응용 프로그램을 계측합니다. JVM AutoProbe 는 Introscope 가 파일 시스템에서 로드되는 모든 바이트 코드를 볼 수 있도록 부트스트랩 또는 응용 프로그램 서버 클래스 로더에서 후크를 제공하는 모든 J2EE 응용 프로그램 서버에 적합합니다.

대부분의 JVM 공급자는 – *javaagent* 옵션을 지원합니다. 이 옵션을 지원하지 않는 JVM 을 사용하는 경우에는 대체 계측 방법을 사용해야 합니다.

ProbeBuilder 수동 실행은 응용 프로그램이 시작되기 전에 바이트 코드를 정적으로 계측해야 하는 경우에만 필요합니다. ProbeBuilder 마법사나 명령줄 프롬프트를 사용하여 ProbeBuilder 를 수동으로 실행할 수 있습니다. ProbeBuilder 는 바이트 코드를 계측하고 계측된 jar 또는 클래스 파일을 새 이름으로 출력합니다. 그런 다음 새로 계측된 이 바이트 코드가 응용 프로그램이 시작되기 전에 응용 프로그램의 클래스 경로 앞에 배치되거나 이름이 적절하게 바뀝니다.

JVM AutoProbe 를 사용하는 대체 방법에 대한 자세한 내용은 "부록 B: 대체 계측 방법"을 참조하십시오.

Java Agent 를 시작하도록 응용 프로그램 서버 구성

에이전트의 기본 *.jar* 파일과 에이전트 프로필에 대한 경로를 포함하도록 계측할 응용 프로그램 서버를 구성해야 합니다. 대부분의 경우 응용 프로그램 서버의 시작 스크립트를 편집한 다음 응용 프로그램 서버를 다시 시작하여 이 작업을 수행합니다. 응용 프로그램 서버가 다시 시작되면 Java Agent 는 JVM 및 응용 프로그램 환경의 기본 구성 요소에 대해 검색된 클래스를 계측합니다. 관련된 구체적인 단계는 응용 프로그램 서버에 따라 달라집니다.

Java Agent 를 사용하도록 Apache Tomcat 구성

Java Agent 를 사용하도록 Apache Tomcat 을 구성하려면 Tomcat 시작 스크립트를 편집해야 합니다. 기본적으로 Tomcat 시작 스크립트는 *$CATALINA_HOME/bin* 디렉터리의 *catalina.sh* 또는 *catalina.bat* 입니다. 웹 서버의 요구 사항에 따라 다른 시작 스크립트를 사용하거나 시작 스크립트에 다른 위치를 사용할 수 있습니다.

다음 단계를 따르십시오.

1. Tomcat 시작 스크립트가 있는 디렉터리로 이동합니다. 예:
   ```
   cd /apache-tomcat-6.0.18/bin
   ```
2. Tomcat 시작 스크립트를 텍스트 편집기에서 엽니다. 예:
   ```
   vi catalina.sh
   ```

3. Java 옵션을 설정할 수 있는 명령줄을 찾고 다음 명령줄 옵션을 추가하여 에이전트의 시작 *.jar* 파일과 에이전트 프로필의 경로를 지정합니다.

   ```
   -javaagent:<PathToAgentJar>
   -Dcom.wily.introscope.agentProfile=<PathToAgentProfile>
   ```

   예를 들어 *Agent.jar* 를 사용하려는 경우 서버를 시작하는 명령 앞에 다음과 유사한 코드를 추가합니다.

   ```
   set JAVA_OPTS=%JAVA_OPTS%  -javaagent:c:\apache-root\wily\Agent.jar
   -Dcom.wily.introscope.agentProfile=
   c:\apache-root\wily\core\config\IntroscopeAgent.profile
   ```

4. 시작 스크립트를 저장합니다.
5. (선택 사항) JMX 메트릭을 수집하도록 에이전트 프로필을 구성하여 Apache Tomcat JMX 메트릭을 보고하도록 설정합니다. IntroscopeAgent.profile 을 열고 다음 속성을 설정합니다.

   introscope.agent.jmx.enable=true

   참고: 플랫폼별 MBeanServer 와 함께 JMX 원격 관리 서버를 사용하여 콘솔에서 Apache Tomcat 의 JMX 메트릭을 확인하려면 IntroscopeAgent.profile 에 com.wily.use.platform.mbeanserver=true 를 추가해야 합니다. 이 구성은 플랫폼별 MBeanServer 사용 여부를 명령줄에서 설정했던 이전 버전의 Introscope 에서 변경되었습니다.
6. IntroscopeAgent.profile 을 저장하고 닫습니다.
7. Tomcat 서버를 다시 시작합니다.

데이터 수집 사용자 지정

Apache Tomcat 서버에 Java Agent 를 설치한 후에는 PBD 를 수정하여 데이터 수집을 사용자 지정할 수 있습니다.

다음 단계를 따르십시오.

1. <*Agent_Home*>\wily\core\config 디렉터리로 이동합니다.
2. tomcat.pbd 파일을 텍스트 편집기에서 엽니다.
3. 사용자 지정할 섹션을 수정합니다.
4. 파일을 저장하고 닫습니다.

Java Agent 를 사용하도록 JBoss 구성

Java Agent 를 사용하도록 JBoss 를 구성하려면 사용 중인 JBoss 버전에 해당하는 JBoss 시작 스크립트를 편집해야 합니다.

- JBoss 7 이상 버전의 JBoss 시작 스크립트

  독립 실행형 모드에 대한 기본 JBoss 시작 스크립트는 $JBOSS_HOME/bin 디렉터리의 standalone.sh 또는 standalone.bat 입니다. 도메인 모드에 대한 기본값은 $JBOSS_HOME/bin 디렉터리의 domain.sh 또는 domain.bat 입니다.

- JBoss 6 및 이전 버전의 JBoss 시작 스크립트

  기본적으로 JBoss 시작 스크립트는 $JBOSS_HOME/bin 디렉터리의 run.sh 또는 run.bat 입니다.

웹 서버의 요구 사항에 따라 다른 시작 스크립트를 사용하거나 시작 스크립트에 다른 위치를 사용할 수 있습니다.

다음 단계를 따르십시오.

1. 다음 예와 같이 JBoss 시작 스크립트가 있는 디렉터리로 이동합니다.

   cd /jboss.GA/bin

2. JBoss 시작 스크립트를 텍스트 편집기에서 엽니다. 예:

   vi run.sh / vi standalone.sh

3. Java 옵션을 설정하기 위한 명령줄을 찾습니다. 에이전트에 대한 시작 ..jar 파일의 경로와 에이전트 프로필의 경로를 지정하려면 다음 명령줄 옵션을 추가합니다.

   - JBoss 7 이상 버전

     -Djboss.modules.system.pkgs=com.wily,com.wily.*
     -javaagent:<PathToAgentJar>
     -Dcom.wily.introscope.agentProfile=<*PathToAgent*Profile>

   - JBoss 6 및 이전 버전

     -javaagent:<PathToAgentJar>
     -Dcom.wily.introscope.agentProfile=<*PathToAgentProfile*>

예를 들어, Agent.jar 를 사용하는 경우 서버를 시작하는 명령 앞에 다음 예와 유사한 코드를 추가합니다.

- JBoss 7 이상 버전

  set JAVA_OPTS= %JAVA_OPTS%
  -Djboss.modules.system.pkgs=com.wily,com.wily.*
  -javaagent:%JBOSS_HOME%\wily\Agent.jar
  -Dcom.wily.introscope.agentProfile=%JBOSS_HOME%\wily\core\config\
  IntroscopeAgent.profile

- JBoss 6 및 이전 버전

  set JAVA_OPTS= -javaagent:%JBOSS_HOME%\wily\Agent.jar
  -Dcom.wily.introscope.agentProfile=%JBOSS_HOME%\wily\core\config\
  IntroscopeAgent.profile %JAVA_OPTS%

4. run.bat 또는 standalone.bat 파일을 저장합니다.
5. (선택 사항) JMX 메트릭을 수집하도록 에이전트 프로필을 구성하여 JBoss JMX 메트릭을 보고하도록 설정합니다.

   a. 텍스트 편집기에서 IntroscopeAgent.profile 을 열고 introscope.agent.jmx.enable=true 로 설정합니다.

      참고: 플랫폼별 MBeanServer 와 함께 JMX 원격 관리 서버를 사용하여 JConsole 에서 JBoss 의 JMX 메트릭을 확인하려면 IntroscopeAgent.profile 에 com.wily.use.platform.mbeanserver=true 를 추가해야 합니다. 이 구성은 플랫폼별 MBeanServer 사용 여부를 명령줄에서 설정했던 이전 버전의 Introscope 에서 변경되었습니다.

   b. introscopeAgent.profile 파일을 저장하고 닫습니다.

   c. JBoss 7 이상 버전의 경우 설치된 JBoss 의 <Agent_Home>/wily/common 디렉터리로 이동하여 WebAppSupport.jar 파일을 <Agent_Home>/wily/core/ext 디렉터리로 옮깁니다.

   d. JBoss 6 및 이전 버전:

      - <Agent_Home>/wily/common 디렉터리로 이동하여 WebAppSupport.jar 파일을 /server/<server_configuration>/lib 디렉터리로 옮깁니다.
      - <Agent_Home>/wily/deploy 디렉터리로 이동하여 introscope-jboss-service.xml 파일을 /server/<server_configuration>/deploy 디렉터리로 옮깁니다.

      참고: 이러한 경로는 기본 구성을 사용하고 있다고 가정합니다. 그렇지 않은 경우 파일을 해당 JBoss 설치 디렉터리로 옮기십시오.

JBoss 응용 프로그램 서버 로깅 고려 사항

JBoss 응용 프로그램 서버를 사용하는 경우 다음 정보를 고려하십시오.

증상:

다음은 응용 프로그램 서버에서 로깅을 수행할 때 모든 에이전트에 공통되는 동작입니다.

- JBoss 6 시스템의 경우 응용 프로그램 서버는 boot.log 파일을 생성하지 않습니다.
- JBoss 7 시스템의 경우 에이전트가 시작되지 않거나 다음과 같은 오류 메시지가 나타납니다.

  `The LogManager was not properly installed(LogManager 가 제대로 설치되지 않았습니다.)`

해결책:

이 문제를 해결하려면 다음 방법 중 하나를 사용하십시오.

- 에이전트 프로필에서 에이전트의 로깅을 해제하고 서버를 시작합니다. 서버가 시작된 후 로깅이 사용되도록 설정합니다.
- JBoss 시작 스크립트를 텍스트 편집기에서 열고 다음으로 업데이트합니다.

```
set JAVA_OPTS= %JAVA_OPTS%
-Djboss.modules.system.pkgs=org.jboss.logmanager,com.wily,com.wily.*
-Djava.util.logging.manager=org.jboss.logmanager.LogManager
-javaagent:%JBOSS_HOME%\wily\Agent.jar
-Dcom.wily.introscope.agentProfile=%JBOSS_HOME%\wily\core\config\IntroscopeAg
ent.profile
-Xbootclasspath/p:%JBOSS_HOME%\modules\org\jboss\logmanager\main\jboss-logman
ager-1.2.2.GA.jar;%JBOSS_HOME%\modules\org\jboss\logmanager\log4j\main\jboss-
logmanager-log4j-1.0.0.GA.jar;%JBOSS_HOME%\modules\org\apache\log4j\main\log4
j-1.2.16.jar
```

JBoss-specific PBD 및 PBL

JBoss 응용 프로그램 서버에 Java Agent 를 설치할 경우 다음 JBoss 관련 PBD 및 PBL 을 사용하여 데이터 수집을 사용자 지정할 수 있습니다.

- *jboss4x.pbd*
- *jsf.pbd*
- *jsf-toggles-full.pbd*
- *jsf-toggles-typical.pbd*
- *jboss-full.pbl*
- *jboss-typical.pbl*

JBoss 7 자동 이름 지정 구성

자동 이름 지정 기능은 webappsupport.jar 파일을 appserver 배포 폴더에 복사하는 경우에만 동작합니다.

예

독립 실행형 서버의 경우 webappsupport.jar 를 JBOSS7_HOME/standalone/deployments/ 폴더에 복사하십시오.

Java Agent 를 사용하도록 Oracle WebLogic 구성

Java Agent 를 사용하도록 Oracle WebLogic 을 구성하려면 WebLogic 시작 스크립트를 편집합니다. 사용하는 시작 스크립트는 요구 사항에 따라 WebLogic 도메인에만 적용될 수 있습니다. 기본적으로 WebLogic 시작 스크립트는 *$WEBLOGIC_HOME*/samples/domain/*<domain_name>*/bin 디렉터리의 startWebLogic.sh 또는 startWebLogic.cmd 입니다. 다른 시작 스크립트를 사용할 수 있습니다. 예를 들어, 응용 프로그램 관련 시작 스크립트나 시작 스크립트의 다른 위치를 사용할 수 있습니다.

다음 단계를 따르십시오.

1. 수정할 WebLogic 시작 스크립트가 있는 디렉터리로 이동합니다. 예:
   ```
   cd $WL_HOME/samples/domain/wl_server
   ```
2. WebLogic 시작 스크립트를 텍스트 편집기에서 엽니다. 예:
   ```
   vi startWebLogic.sh
   ```

3. Java 옵션을 설정할 수 있는 명령줄이나 Java 명령줄을 찾고 다음 명령줄 옵션을 추가합니다. -javaagent:*<PathToAgentJar>* -Dcom.wily.introscope.agentProfile=*<PathToAgentProfile>*
4. WebLogic 시작 스크립트 또는 응용 프로그램별 시작 스크립트를 저장하고 닫습니다.

WebLogic 용 시작 클래스 생성

응용 프로그램 서버 또는 클러스터에 맞게 WebLogic 시작 클래스를 생성하면 Java Agent 가 응용 프로그램 서버에서 추가 정보를 수집할 수 있습니다. 시작 클래스를 구성하면 Java Agent 가 해당 이름을 자동으로 확인할 수 있습니다. 또한 Java Agent 는 시작 클래스를 사용하여 Workstation 에서 응용 프로그램 건전성을 확인하는 데 사용되는 JMX 메트릭을 보고할 수도 있습니다.

다음 단계를 따르십시오.

1. "WebLogic Administrative Console"(WebLog 관리 콘솔)을 엽니다.
2. 왼쪽 창에서 Environments 폴더를 확장합니다.
3. Startup & Shutdown 폴더를 클릭합니다.

   "Startup and Shutdown"(시작 및 종료) 페이지가 열립니다.
4. "Configure a New Startup Class"(새 시작 클래스 구성)를 클릭합니다.

   "Configuration"(구성) 탭이 표시됩니다.
5. "Name"(이름) 필드에 다음을 입력합니다.
   `Introscope Startup Class`
6. "ClassName"(클래스 이름) 필드에 다음을 입력합니다.
   `com.wily.introscope.api.weblogic.IntroscopeStartupClass`
7. "만들기"를 클릭합니다.

   "Target and Deploy"(대상 및 배포) 탭이 나타납니다.
8. 이 시작 클래스를 사용할 수 있도록 설정할 서버에 해당하는 상자를 선택합니다.
9. "Apply"(적용)를 클릭하고"Run before application deployments"(응용 프로그램 배포 전에 실행) 옵션을 선택합니다.
10. *<server or application server>* 시작 클래스 경로에 WebAppSupport.jar 의 위치를 추가합니다.
11. 응용 프로그램 서버를 다시 시작합니다.

WebLogic Server 에서 크로스 프로세스 추적 사용

트랜잭션 추적 세션을 사용하면 호환 가능한 JVM 버전이 설치된 컴퓨터에서 JVM 경계를 넘는 트랜잭션을 포함하여 트랜잭션에서 수행되는 모든 작업을 추적할 수 있습니다.

크로스 프로세스 트랜잭션 추적은 동기 트랜잭션(예: EJB 에 대한 서블릿)과 비동기 트랜잭션에 대해 지원됩니다.

WebLogic 에서 크로스 프로세스 트랜잭션 추적을 사용하도록 설정하려면

1. WebLogic 용 시작 클래스 생성 (see page 48)에서 설명한 대로 시작 클래스를 생성합니다.
2. WebLogic Server 를 시작하기 위한 java 명령줄에 "-Dweblogic.TracingEnabled=true"를 추가합니다.
3. IntroscopeAgent.profile 을 텍스트 편집기에서 엽니다.
4. introscope.agent.weblogic.crossjvm 속성을 찾아서 true 로 설정합니다. 예: `introscope.agent.weblogic.crossjvm=true`
5. IntroscopeAgent.profile 을 저장하고 닫습니다.

JMX 메트릭에 대한 Java Agent 지원

CA Introscope 는49 응용 프로그램 서버 또는 Java 응용 프로그램에서 JMX 규격 MBean 으로 표시하는 관리 데이터를 수집하고 JMX 데이터를 Investigator 메트릭 트리에 나타낼 수 있습니다. WebLogic 은 다음과 같은 JMX 메트릭 출처를 제공합니다.

- RuntimeServiceMBean: 서버별 런타임 메트릭(활성 유효 구성 포함)
- DomainRuntimeServiceMBean: 도메인 차원 런타임 메트릭
- EditServiceMBean: 영구 구성을 사용자가 편집할 수 있도록 함

CA Introscope 는49 메트릭에 대해 RuntimeServiceMBean 만 폴링합니다. RuntimeServiceMBean 은 로컬 액세스(효율성 측면)를 지원하고 관련된 것으로 예상되는 대부분의 데이터를 포함합니다. 그러나 CA Introscope 는49 Sun JMX 사양에 맞게 빌드된 모든 MBean 을 지원할 수 있습니다.

참고: Sun JMX 사양에 대한 자세한 내용은 http://java.sun.com/products/JavaManagement/를 참조하십시오.

JMX 메트릭의 수집을 지원하기 위해 WebLogic 용 IntroscopeAgent.profile 파일에는 기본적으로 다음 키워드가 정의되고 사용하도록 설정되어 있습니다.

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

CA Introscope 는50 JMX 메트릭을 Investigator 트리의 다음 노드 아래에 표시합니다.

<도메인>|<호스트>|<프로세스>|<에이전트>|JMX|

추가 정보:

JMX 보고를 사용하도록 설정 (페이지 223)

Weblogic 에서 리소스 메트릭을 설정하는 방법

다양한 에이전트가 CA Introscope?Workstation 에서 리소스 메트릭 범주를 보고할 수 있습니다. Java Agent 가 리소스 메트릭을 보고할 수 있도록 Oracle WebLogic Server 를 구성할 수 있습니다.

Oracle WebLogic 에 대한 리소스 메트릭을 설정하려면 다음 단계를 따르십시오.

1. *CA APM for Oracle WebLogic Server 안내서*의 지침에 따라 CA APM for Oracle WebLogic 을 설치합니다.
2. *CA APM for Oracle Databases 안내서*를 따라 CA APM for Oracle Databases 옵션을 사용하도록 설정합니다.
3. *CA APM 구성 및 관리 안내서*의 리소스 메트릭 구성 지침을 따릅니다.

WLDF(WebLogic Diagnostic Framework) 정보

WLDF(WebLogic Diagnostic Framework)는 WebLogic Server 프로세스 내에서 실행되어 표준 서버 수명 주기에 참여하는 일련의 서비스를 정의 및 구현하는 모니터링 및 진단 프레임워크입니다. WLDF 를 사용하여 실행 중인 서버와 해당 컨테이너 내에 배포된 응용 프로그램에서 생성된 진단 데이터를 생성, 수집, 분석, 아카이브 및 액세스할 수 있습니다. 이 데이터를 통해 서버 및 응용 프로그램의 런타임 성능을 파악하고 오류 발생 시 오류를 격리하고 진단할 수 있습니다.

WLDF 를 사용하면 표준 인터페이스를 통해 서버 데이터에 동적으로 액세스할 수 있으며, 서버를 종료했다가 다시 시작하지 않고도 특정 시간에 액세스되는 데이터의 양을 수정할 수 있습니다.

WLDF 데이터를 Introscope 메트릭으로 변환하는 방법

WLDF 에서 정보는 각각 여러 개의 열을 포함하는 일련의 데이터 접근자로 구성됩니다. Introscope 는 이 WLDF 정보를 Introscope 관련 메트릭 형식으로 변환하여 Investigator 의 다음 노드 아래에 표시합니다.

```
<Domain>|<Host>|<Process>|AgentName|WLDF|
```

데이터 접근자의 정보는 도메인 이름과 하나 이상의 키/값 쌍으로 정의됩니다. Introscope 는 데이터 접근자 열을 Introscope 에서 생성되는 메트릭에 사전순으로 표시되는 키 및 값 정보로 변환합니다. 다음 예에서는 사용되는 구문을 보여 줍니다.

```
<Domain>|<Host>|<Process>|AgentName|WLDF|<domain
name>|<key1>=<value1>|<key2>=<value2>:<metric>
```

예를 들어 다음 표에서는 HTTPAccessLog 데이터 접근자의 BYTECOUNT 열에 대한 정보를 보여 줍니다.

도메인 이름	키/값 쌍	메트릭 이름
WebLogic	Name=HTTPAccessLog, Type=WLDFDataAccessRuntime=Accessor, WLDFRuntime=WLDFRuntime	BYTECOUNT

위 표의 데이터 접근자 정보는 다음과 같은 Introscope 메트릭으로 변환됩니다.

```
<Domain>|<Host>|<Process>|AgentName|WLDF|Weblogic|Name=HTTPAccessLog
|Type=WLDFDataAccessRuntime=Accessor|WLDFRuntime=WLDFRuntime:BYTECOUNT
```

키/값 쌍은 Introscope 메트릭에 사전순으로 표시됩니다.

WLDF 보고 사용

기본적으로 Introscope 에서는 WLDF 보고 기능이 사용하도록 설정되어 있지 않습니다. WLDF 메트릭 보고 기능을 사용하도록 에이전트 프로필을 수정할 수 있습니다.

WLDF 보고 기능을 사용하도록 설정하려면

1. 관리되는 응용 프로그램이 실행 중인 경우 종료합니다.
2. WebLogic 시작 클래스를 구성합니다.
3. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
4. 다음 속성을 찾아서 설정합니다.
   `introscope.agent.wldf.enable=true`
5. *IntroscopeAgent.profile* 을 저장하고 닫습니다.

Java Agent 를 사용하도록 JRockit JVM 이 포함된 WebLogic 구성

JRockit JVM 1.5 이상이 있는 WebLogic 9.0 이상에 해당

참고: 지원되는 WebLogic 버전에 대해서는 *호환성 안내서*를 참조하십시오. 이전 버전의 WebLogic 또는 JRockit JVM 을 사용하는 경우에는 AutoProbe 커넥터 파일 (see page 396)을 생성하고 실행하여 응용 프로그램을 계측할 수 있습니다.

JRockit JVM 을 사용하여 WebLogic 을 사용하는 경우 다음 명령줄 옵션을 사용하여 JVM 을 시작하십시오.

```
JAVA_VENDOR=Bea
JAVA_OPTIONS=%JAVA_OPTIONS% -javaagent:PathToAgentJar
-Dcom.wily.introscope.agentProfile=PathToIntroscopeAgent.profile
```

소켓 메트릭을 볼 수 있도록 JRockit JVM 이 포함된 WebLogic 구성

JRockit JVM 1.5.0 이 있는 WebLogic 9.0 에 해당

증상:

타사 호환성 문제로 인해 소켓 클라이언트에 대한 메트릭을 보는 데 문제가 있을 수 있습니다.

해결책:

다음 관리되는 소켓 옵션을 사용하여 소켓 클라이언트에 대한 메트릭을 켭니다.

다음 단계를 따르십시오.

1. toggles_typical.pbl 또는 toggles_full.pbl 파일을 텍스트 편집기에서 엽니다.
2. 소켓 메트릭을 추적하도록 관리되는 소켓 옵션을 설정합니다. 예:

```
#####################
# Network Configuration
# ================
#TurnOn: SocketTracing
# NOTE: Only one of SocketTracing and ManagedSocketTracing should be 'on'.
ManagedSocketTracing is provided to
# enable pre 9.0 socket tracing.
TurnOn: ManagedSocketTracing
TurnOn: UDPTracing
```

3. 변경 사항을 저장합니다.

   관리되는 소켓 옵션이 설정되었습니다.

Java Agent 를 사용하도록 IBM WebSphere 구성

Java Agent 를 사용하도록 IBM WebSphere 를 구성하려면 WebSphere 시작 스크립트를 편집합니다. 사용하는 시작 스크립트는 요구 사항에 따라 응용 프로그램 서버 노드에만 적용될 수 있습니다.

다음 단계를 따르십시오.

1. $WEBSPHERE_HOME/profiles/<*App Server Name*>/config/cells/<*셀 이름*>/nodes/<*노드 이름*>/servers/server1directory 로 이동합니다.

2. server.xml 파일을 편집합니다. 이 파일은 WebSphere 기본 시작 스크립트 및 위치입니다.

   참고: 응용 프로그램 서버 요구 사항에 따라 다른 시작 스크립트를 사용할 수 있습니다. 예를 들어, 응용 프로그램 관련 시작 스크립트나 시작 스크립트의 다른 위치를 사용할 수 있습니다. 또한 운영 체제별로 서로 다른 WebSphere 조합을 사용하거나 여러 JVM 공급자 또는 JVM 버전을 사용하려는 경우에는 특별한 요구 사항이 있을 수 있습니다.

3. 파일을 저장하고 닫습니다.

참고: 자세한 내용은 사용 중인 WebSphere Application Server 환경과 가장 유사한 환경에 대한 단원을 참조하십시오.

UNIX, Windows, OS/400, z/OS, IBM JVM 1.5 에서 WebSphere Application Server 6.1 구성

UNIX, Windows, OS/400, z/OS, IBM JVM 1.5 의 WebSphere Application Server 6.1 에 해당

동적 계측 기능을 사용하려면 클래스 재정의 지원이 필요합니다. IBM JDK 버전 5 에서 실행 중인 경우 클래스 재정의를 사용하면 성능이 상당히 저하될 수 있습니다. 동적 계측을 사용하려는 CA Introscope 및 IBM JDK 버전 5 고객은 이 성능 오버헤드에 대해 인지하고 있어야 합니다. 이 구성을 적용할 경우에는 QA 환경에서만 동적 계측 기능을 사용하는 것이 좋습니다.

참고: 이 성능 오버헤드에 대한 자세한 내용은 IBM "*Java Diagnostics Guide*"(Java 진단 안내서)를 참조하십시오.

IBM JVM 1.5 를 사용하여 WebSphere 6.1 을 실행하는 경우에는 대체 버전의 Java Agent .jar 파일 및 Java Agent 프로필을 사용하십시오. 이러한 파일은 이름이 AgentNoRedef.jar 및 IntroscopeAgent.NoRedef.profile 이며 *<Agent_Home>*/core/config 디렉터리에 있습니다.

참고: AllAppServer 에이전트 배포를 사용하는 경우 대체 프로필의 이름은 IntroscopeAgent.websphere.NoRedef.profile 입니다.

이전 버전의 파일 및 구문을 사용할 경우 다음에 대한 메트릭이 더 이상 보고되지 않습니다.

- 시스템 클래스
- NIO(소켓 및 데이터그램)
- SSL

또한 다음 기능도 영향을 받습니다.

- 소켓 계측은 9.0 이전의 CA Introscope®에 대해 ManagedSocket 스타일을 사용합니다.
- 원격 동적 계측이 해제됩니다.
- PBD 파일에 대한 변경 사항을 적용하려면 계측되는 JVM 을 다시 시작해야 합니다.
- 세부 수준 상속 및 계층 구조 지원 계측이 해제됩니다.

이전 버전의 파일 및 구문을 사용하는 경우 에이전트는 다음과 같은 작업으로 클래스 재정의 상태를 보고합니다.

- Investigator 의 에이전트 노드 아래에 "*에이전트 클래스 재정의 사용*"이라는 메트릭을 추가합니다. 메트릭 값은 true 또는 false 입니다.
- 에이전트 로그 파일에 로그 메시지를 기록합니다.
  - 클래스 재정의가 사용하도록 설정된 경우 로그 메시지는 WARN 수준에 다음과 같이 기록됩니다.

    Introscope Agent Class Redefinition is enabled. Enabling class redefinition on IBM 1.5 JVMs is known to incur significant overhead.(Introscope Agent 클래스 재정의가 사용하도록 설정되어 있습니다. IBM 1.5 JVM 에서 클래스 재정의를 사용하면 상당한 오버헤드가 발생합니다.)
  - 클래스 재정의가 사용하지 않도록 설정된 경우 로그 메시지는 INFO 수준에 다음과 같이 기록됩니다.

    Introscope Agent Class Redefinition is disabled.((Introscope Agent 클래스 재정의가 사용하지 않도록 설정되어 있습니다.)
- 다음과 같이 표준 오류 메시지를 기록합니다(클래스 재정의가 사용하도록 설정된 경우에만 해당).

    Warning: Introscope agent has been configured to support class redefinition. IBM JVMs version 1.5 and higher are known to incur significant overhead with redefinition enabled. To avoid this overhead please use AgentNoRedef.jar instead of Agent.jar.(Introscope Agent 가 클래스 재정의를 지원하도록 구성되었습니다. IBM JVM 버전 1.5 이상에서는 재정의를 사용할 경우 상당한 오버헤드가 발생합니다. 이 오버헤드를 방지하려면 Agent.jar 대신 AgentNoRedef.jar 를 사용하십시오.)

IBM 이외의 JVM 을 사용하거나 1.5 버전 이외의 IBM JVM 을 사용하는 경우에는 앞의 메트릭 및 메시지가 출력되지 않습니다.

에이전트를 사용하도록 WebSphere Application Server 를 구성합니다.

다음 단계를 따르십시오.

1. "WebSphere Administrator Console"(WebSphere 관리자 콘솔)을 엽니다.
2. "Application Servers"(응용 프로그램 서버) > your server > "Server Infrastructure"(서버 인프라) > "Java and Process Management"(Java 및 프로세스 관리) > "Process Definition"(프로세스 정의) > "Java Virtual Machine"으로 이동합니다.
3. "Generic JVM Argument"(일반 JVM 인수) 필드를 다음과 같이 설정합니다.

   ```
   -javaagent:<Agent_Home>/AgentNoRedef.jar
   -Dcom.wily.introscope.agentProfile=<Agent_Home>/core/config/IntroscopeAgent.NoRedef.profile
   ```

   동일한 컴퓨터에 계측된 응용 프로그램과 계측되지 않은 응용 프로그램이 모두 있는 경우 "Generic JVM Argument"(일반 JVM 인수)에 -Xshareclasses:none 설정을 포함하십시오. 이 설정은 AIX 에서 오류를 방지합니다.

   참고: 동일한 에이전트 디렉터리를 사용하는 WebSphere 버전이 둘 이상인 경우에는 고유한 디렉터리가 필요합니다.

UNIX, Windows, Sun, HP, 기타 JVM 1.5 에서 WebSphere Application Server 6.1 구성

UNIX, Windows, Sun, HP, 기타 JVM 1.5 의 WebSphere Application Server 6.1 에 해당

참고: IBM 이외의 JVM 을 사용하거나 1.5 버전 이외의 IBM JVM 을 사용하는 경우에는 일부 메트릭 및 메시지가 출력되지 않습니다.

에이전트를 사용하도록 WebSphere Application Server 를 구성할 수 있습니다.

다음 예에서는 WebSphere 6.1 에 Java Agent 를 설치할 때 특정 JVM 과 함께 사용할 Java 인수 및 .jar 파일을 나타냅니다.

다음 단계를 따르십시오.

1. "WebSphere Administrator Console"(WebSphere 관리자 콘솔)을 엽니다.
2. "Application Servers"(응용 프로그램 서버) > your server > "Server Infrastructure"(서버 인프라) > "Java and Process Management"(Java 및 프로세스 관리) > "Process Definition"(프로세스 정의) > "Java Virtual Machine"으로 이동합니다.
3. "Generic JVM Argument"(일반 JVM 인수) 필드를 다음과 같이 설정합니다.

   ```
   -javaagent:<Agent_Home>/Agent.jar
   -Dcom.wily.introscope.agentProfile=<Agent_Home>/core/config/IntroscopeAgent.profile
   ```

4. WebSphere Application Server 를 다시 시작합니다.

UNIX, Windows, OS/400, JVM 1.5 에서 WebSphere Application Server 7.0 구성

UNIX, Windows, OS/400, JVM 1.5 이상의 WebSphere Application Server 7.0 에 해당

에이전트를 사용하도록 WebSphere Application Server 7.0 을 구성할 수 있습니다. WebSphere 7.0 을 모니터링하려면 Java Agent 를 시작하도록 서버를 구성하기 전에 빌드 7.0.0.8 이상이 설치되어 있는지 확인하십시오.

다음 단계를 따르십시오.

1. "WebSphere Administrator Console"(WebSphere 관리자 콘솔)을 엽니다.
2. "Application Servers"(응용 프로그램 서버) > your server > "Server Infrastructure"(서버 인프라) > "Java and Process Management"(Java 및 프로세스 관리) > "Process Definition"(프로세스 정의) > "Java Virtual Machine"으로 이동합니다.
3. "Generic JVM Argument"(일반 JVM 인수) 필드를 다음과 같이 설정합니다.

   ```
   -javaagent:<Agent_Home>/Agent.jar
   -Dcom.wily.introscope.agentProfile=<Agent_Home>/core/config/IntroscopeAgent.profile
   ```

4. WebSphere Application Server 를 다시 시작합니다.

z/OS 에서 WebSphere Application Server 7.0 구성 - JVM 1.5

z/OS 의 WebSphere Application Server 7.0 에 해당 - JVM 1.5 이상

에이전트를 사용하도록 WebSphere Application Server 7.0 을 구성할 수 있습니다. Java Agent 를 시작하도록 서버를 구성하기 전에 빌드 7.0.0.8 이상을 설치하는지 확인하십시오.

다음 단계를 따르십시오.

1. "WebSphere Administrator Console"(WebSphere 관리자 콘솔)을 엽니다.
2. "Application Servers"(응용 프로그램 서버) > your server > "Server Infrastructure"(서버 인프라) > "Java and Process Management"(Java 및 프로세스 관리) > "Process Definition"(프로세스 정의) > "Java Virtual Machine"으로 이동합니다.
3. "Generic JVM Argument"(일반 JVM 인수) 필드를 다음과 같이 설정합니다.

   ```
   -Xbootclasspath/a:<Agent_Home>/Agent.jar -javaagent:<Agent_Home>/Agent.jar
   -Dcom.wily.introscope.agentProfile=<Agent_Home>/core/config/IntroscopeAgent.p
   rofile
   ```

4. WebSphere Application Server 를 다시 시작합니다.

Java2 보안 정책 수정

Java2 보안이 사용되는 WebSphere 환경에서 AutoProbe 가 올바르게 실행되도록 하려면 Java2 보안 정책에 대한 사용 권한을 추가해야 할 수 있습니다.

Java2 보안 정책에 대한 사용 권한을 추가하려면

1. $*WebSphere home/properties/server.policy* 파일을 텍스트 편집기에서 엽니다.
2. 파일에 다음 사용 권한을 추가합니다.

```
// permissions for Introscope AutoProbe
grant codeBase "file:${was.install.root}/-" {
permission java.io.FilePermission "${was.install.root}${/
}wily${/}-", "read";
permission java.net.SocketPermission "*", "connect,resolve";
permission java.lang.RuntimePermission "setIO";
permission java.lang.RuntimePermission "getClassLoader";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
permission java.lang.RuntimePermission "loadLibrary.*";
permission java.lang.RuntimePermission "accessClassInPackage.*";
permission java.lang.RuntimePermission "accessDeclaredMembers";
};
grant {
permission java.util.PropertyPermission "*", "read,write";
};
```

   참고: 사용자가 쉽게 알아볼 수 있도록 줄 바꿈이 표시되었지만 *server.policy* 파일에 사용 권한을 추가할 때 줄 바꿈을 포함할 필요는 없습니다.

3. 파일을 저장하고 닫습니다.

WebSphere 에서 사용자 지정 서비스 구성

WebSphere Application Server 에서 사용자 지정 서비스를 만들거나 수정할 수 있습니다. 사용자 지정 서비스를 통해 Java Agent 는 응용 프로그램 서버에서 추가 정보를 수집할 수 있습니다. 사용자 지정 서비스를 구성할 경우 Java Agent 는 해당 서비스 이름을 자동으로 확인할 수 있습니다. 또한 사용자 지정 서비스를 통해 Java Agent 는 JMX 및 PMI(Performance Monitoring Infrastructure) 메트릭을 보고할 수 있습니다. 응용 프로그램 개요 탭의 Introscope Workstation 은 이 메트릭을 사용하여 응용 프로그램 상태를 확인합니다. 사용자 지정 서비스는 CA Introscope®에서 JMX 메트릭을 액세스하는 데 사용할 사용자 자격 증명을 암호화할 수 있습니다.

참고: WebSphere Application Server 에 추가된 SIBus 관련 메트릭 또는 새 PMI 모듈을 가져오려면 기존 사용자 지정 서비스를 비활성화한 다음 사용자 지정 서비스를 만드십시오.

다음 단계를 따르십시오.

1. "WebSphere Administrator Console"(WebSphere 관리자 콘솔)을 엽니다.
2. 구성할 서버를 선택하고 "Server Infrastructure"(서버 인프라) > "Administration"(관리) > "Custom Services"(사용자 지정 서비스)로 이동합니다.
3. 원하는 사용자 지정 서비스를 수정하거나 새로 만듭니다.
4. "구성" 페이지에서 다음 필드를 입력하고 "확인"을 클릭합니다.

   Enable service at server startup(서버 시작 시 서비스 사용)

   서버를 시작할 때 서비스가 시작되도록 지정합니다.

   External Configuration URL(외부 구성 URL)

   구성 속성 파일의 위치를 지정합니다. JMX 메트릭 구성의 경우 예를 들어 <*Agent_Home*>/wily/common/jmxconfig.properties 와 같은 jmxconfig.properties 파일을 사용합니다.

   Classname(클래스 이름)

   사용자 지정 서비스 클래스의 이름을 지정합니다. 예를 들면 다음과 같습니다.

   com.wily.introscope.api.websphere.IntroscopeCustomService

   com.wily.powerpack.websphere.agent.PPCustomService

   표시 이름

   CA Introscope®에 표시할 이름을 지정합니다(예: Introscope Custom Service).

   Classpath(클래스 경로)

   속성 파일의 정규화된 경로 이름을 지정합니다. 예를 들면 다음과 같습니다.

   ```
   <Agent_Home>/wily/common/WebAppSupport.jar
   <Agent_Home>/wily/common/PowerpackForWebSphere_Agent
   ```

5. JMX 메트릭에 액세스하기 위한 사용자 자격 증명을 구성합니다.
   a. 텍스트 편집기에서 <*Agent_Home*>/wily/common 으로 이동하여 jmxconfig.properties 파일을 엽니다.

b. 속성 설명에 따라 값의 주석을 해제하고 값을 설정합니다.

c. 파일을 저장하고 닫습니다.

6. 응용 프로그램 서버를 다시 시작합니다.

   서버가 시작되면 사용자 지정 서비스가 시작되고 사용자 자격 증명이 암호화됩니다. 따라서 서버가 시작될 때마다 서버는 암호화된 암호를 사용합니다.

예: 암호 암호화를 위한 jmxconfig.properties 파일 구성

다음 텍스트는 암호 암호화를 위해 설정된 jmxconfig.properties 의 예제를 보여 줍니다.

```
jmxUsername=dave
jmxPassword=mypassword
plainTextPassword=true
```

추가 정보:

JMX 보고를 사용하도록 WebSphere 및 WebLogic 구성 (페이지 228)

WebSphere PMI 메트릭 수집 사용

CA Introscope 는61 WebSphere 가 제공하는 PMI 인터페이스를 사용하여 PMI(Performance Monitoring Infrastructure) 메트릭을 추출함으로써 WebSphere 성능 데이터를 제공할 수 있습니다.

CA Introscope 에서61 PMI 데이터를 사용하려면 먼저 WebSphere 에서 PMI 데이터 수집 기능을 활성화해야 합니다. WebSphere 에서는 모든 성능 모니터링 설정이 기본적으로 해제되어 있습니다.

다음 단계를 따르십시오.

1. WebSphere 에서 CA Introscope®에 대한 사용자 지정 서비스를 구성합니다.
2. WebSphere 에서 PMI 데이터 수집 기능을 사용하도록 설정합니다.
3. IntroscopeAgent.profile 에서 PMI 데이터의 보고 기능을 사용하도록 설정합니다.

WebSphere 에서 PMI 데이터 수집 기능을 사용하도록 설정한 후에는 PMI 데이터가 CA Introscope 메트릭으로 표시될 수 있습니다. 필요에 따라 CA Introscope 로61 가져올 메트릭 범주를 필터링할 수 있습니다.

참고: z/OS 의 WebSphere 의 경우 CA Technologies 는 CA APM for WebSphere z/OS 를 사용할 것을 권장합니다. CA APM for WebSphere z/OS 는 WebSphere 관련 PBD 및 메트릭을 제공합니다. 이러한 PBD 및 메트릭은 오버헤드가 적은 추적 프로그램 기술을 사용하여 WebSphere 관련 메트릭을 가져옵니다. CA APM for WebSphere z/OS 를 사용할 경우 WebSphere for z/OS 에서 PMI 를 사용하도록 설정할 필요가 없습니다. WebSphere for z/OS 에서 PMI 보고 기능을 사용하도록 설정할 수도 있지만 이 경우 시스템 리소스가 많이 소비됩니다.

Introscope 에서 WebSphere PMI 메트릭 보고 구성

WebSphere 에서 성능 모니터링 설정을 사용하도록 설정한 후에는 Introscope 에서 PMI 데이터 수집 기능과 보고를 받으려는 메트릭 범주를 사용하도록 설정해야 합니다.

Introscope 에서 PMI 메트릭의 보고 기능을 구성하려면

1. 관리되는 응용 프로그램을 종료합니다.
2. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
3. WebSphere PMI Configurations 섹션에서 *introscope.agent.pmi.enable* 속성을 찾아 *true* 로 설정되어 있는지 확인합니다.

4. 상위 수준 PMI 메트릭 범주의 다음 속성을 찾습니다.

```
introscope.agent.pmi.enable.threadPoolModule=true
introscope.agent.pmi.enable.servletSessionsModule=true
introscope.agent.pmi.enable.connectionPoolModule=true
introscope.agent.pmi.enable.beanModule=false
introscope.agent.pmi.enable.transactionModule=false
introscope.agent.pmi.enable.webAppModule=false
introscope.agent.pmi.enable.jvmRuntimeModule=false
introscope.agent.pmi.enable.systemModule=false
introscope.agent.pmi.enable.cacheModule=false
introscope.agent.pmi.enable.orbPerfModule=false
introscope.agent.pmi.enable.j2cModule=true
introscope.agent.pmi.enable.webServicesModule=false
introscope.agent.pmi.enable.wlmModule=false
introscope.agent.pmi.enable.wsgwModule=false
introscope.agent.pmi.enable.alarmManagerModule=false
introscope.agent.pmi.enable.hamanagerModule=false
introscope.agent.pmi.enable.objectPoolModule=false
introscope.agent.pmi.enable.schedulerModule=false
# introscope.agent.pmi.enable.jvmpiModule=false
```

   네 개의 범주(*threadPool*, *servletSessions*, *connectionPool* 및 *j2c*)는 기본적으로 *true* 로 설정되어 있습니다. 적절한 속성을 *true* 로 설정하여 추가 메트릭 범주를 사용하도록 설정하거나, 범주를 주석 처리하여 보고되는 메트릭 수를 줄일 수 있습니다.

5. 변경 사항을 저장하고 파일을 닫습니다.
6. 관리되는 응용 프로그램을 다시 시작합니다.

Introscope 에서 PMI 수집 기능을 사용하도록 설정한 후에는 사용 가능한 PMI 메트릭이 Investigator 트리의 "WebSpherePMI" 노드 아래에 표시됩니다.

WebSphere 에 대한 리소스 메트릭 맵 데이터 보고 구성

다양한 에이전트가 CA Introscope?Workstation 에서 리소스 메트릭 범주를 보고할 수 있습니다. WebSphere 에서 PMI 데이터 수집을 사용하도록 설정한 후 리소스 메트릭 맵 데이터를 보고하도록 응용 프로그램 서버를 구성할 수 있습니다.

다음 단계를 따르십시오.

1. WebSphere 콘솔에 로그인합니다.
2. "Monitoring and Tuning"(모니터링 및 조정), "Performance Monitoring Infrastructure"를 선택합니다.
3. 보고에 사용할 서버를 선택합니다(예: server1).

4. "Configuration"(구성) 탭을 클릭하고 "Custom"(사용자 지정) 옵션을 선택합니다.
5. "Custom"(사용자 지정) 링크를 클릭합니다.

   옵션 구성 트리가 나타납니다.
6. 트리에서 "JDBC Connection Pools"(JDBC 연결 풀)를 선택하고 오른쪽 창에서 "WaitingThreadCount"를 사용하도록 설정합니다.
7. 트리에서 "ThreadPools"를 선택하고 오른쪽 창에서 "ActiveCount"를 사용하도록 설정합니다.
8. 구성을 저장하고 응용 프로그램 서버를 다시 시작합니다.

   응용 프로그램 서버가 리소스 메트릭을 보고하도록 구성되었습니다.

참고: 리소스 메트릭 맵 데이터에 대한 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

WebSphere 에서 크로스 프로세스 추적 사용

트랜잭션 추적 세션을 사용하면 호환 가능한 JVM 버전이 설치된 컴퓨터에서 JVM 경계를 넘는 트랜잭션을 포함하여 트랜잭션에서 수행되는 모든 작업을 추적할 수 있습니다. 크로스 프로세스 트랜잭션 추적은 동기 트랜잭션(예: EJB 에 대한 서블릿)과 비동기 트랜잭션에 대해 지원됩니다.

WebSphere 에서 크로스 프로세스 트랜잭션 추적 기능을 사용하도록 설정하려면

1. WebSphere 6.1 에서 사용자 지정 서비스 구성 (see page 59)에 설명된 대로 사용자 지정 서비스를 생성합니다.
2. 작업 영역 서비스를 켭니다.

   관리 페이지의 "Servers"(서버) > "Application servers"(응용 프로그램 서버)에서 "server1", "Business Process Services"(비즈니스 프로세스 서비스), "Work Area Service"(작업 영역 서비스)를 차례로 클릭한 다음 "Enable service at server startup"(서버 시작 시 서비스 사용) 상자를 선택합니다.
3. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
4. *introscope.agent.websphere.crossjvm* 속성을 찾아서 *true* 로 설정합니다. 예:

   ```
   introscope.agent.websphere.crossjvm=true
   ```
5. *IntroscopeAgent.profile* 을 저장하고 닫습니다.

WebSphere for z/OS 에서의 로깅 고려 사항

WebSphere z/OS 환경에서 로깅하는 경우 몇 가지 고려해야 할 사항이 있습니다. Java Agent 로깅 옵션에 대한 자세한 내용은 Java Agent 모니터링 및 로깅 (see page 155)을 참조하십시오

EBCDIC 로 로그 출력 태그 지정

WebSphere for z/OS 에서는 기본 인코딩이 EBCDIC CP1047 에서 ASCII ISO8859-1 로 변경되었습니다. z/OS 는 대개 EBCDIC 시스템이므로 Java Agent 또는 AutoProbe 에서 기록된 로깅 데이터는 최종 출력 스트림으로 ASCII 대신 EBCDIC 를 사용하도록 태그 지정되어야 합니다.

데이터를 ASCII 대신 EBCDIC 로 태그 지정하려면

1. *<Agent_Home>\wily\core\config* 디렉터리에 있는 *IntroscopeAgent.profile* 을 엽니다.
2. *IntroscopeAgent.profile* 에 다음 속성을 추가합니다.
   ```
   log4j.appender.console.encoding=IBM-1047
   log4j.appender.logfile.encoding=IBM-1047
   ```
3. *IntroscopeAgent.profile* 을 저장합니다.

로깅 기능의 시작 타이밍 문제 제거

WebSphere for z/OS 의 경우 속성을 사용하여 CA Introscope 로깅 기능에서 발생할 수 있는 시작 타이밍 창 노출을 제거할 수 있습니다.

다음 단계를 따르십시오.

1. *<Agent_Home>*\wily\core\config 디렉터리에 있는 IntroscopeAgent.profile 을 엽니다.
2. IntroscopeAgent.profile 에 다음 속성을 추가합니다.
   ```
   introscope.agent.logger.delay=100000
   ```
   값은 밀리초 단위이므로 이 예에서 기본 지연 시간은 100 초입니다.
3. IntroscopeAgent.profile 을 저장합니다.

Java Agent 를 사용하도록 Oracle Application Server 구성

OAS(Oracle Application Server)에서는 모든 응용 프로그램을 관리하는 데 하나의 구성 파일을 사용하므로 모든 JVM 도 Oracle 콘솔에서 관리됩니다. 이 파일을 대개 *opmn.xml* 이라고 합니다.

Java Agent 를 사용하도록 Oracle Application Server 를 수정하려면

1. 응용 프로그램 서버를 종료하고 *opmn.xml* 을 백업합니다.
2. *opmn.xml* 파일에서 계측할 응용 프로그램에 대한 섹션을 찾습니다. 이때 여러 응용 프로그램을 계측하도록 선택할 수 있습니다.
3. 선택한 응용 프로그램에 대한 *<category id="start-parameters">* 아래에서 *<data id="java-options"* 라는 섹션을 찾습니다.
4. *java-options* 행의 맨 끝에 있는 종료 *"/>* 앞에 사용자 환경에 적절한 경로를 사용하여 다음을 삽입합니다.

   ```
   -javaagent:<PathToAgentJar>
   -Dcom.wily.introscope.agentProfile=<PathToAgentProfile>
   ```

   예를 들어 응용 프로그램 하나에 대한 전체 항목은 한 행으로 표시됩니다.

   ```
   <data id="java-options" value="-server -XX:MaxPermSize=128M -ms512M -mx1024M
   -XX:AppendRatio=3
   -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
   -Djava.awt.headless=true -Dhttp.webdir.enable=false
   -javaagent:$ORACLE_HOME/wily/Agent.jar
   -Dcom.wily.introscope.agentProfile=$ORACLE_HOME/wily/core/config/IntroscopeAg
   ent.profile/>
   ```

Java Agent 를 사용하도록 GlassFish 구성

Java Agent 를 사용하도록 Sun GlassFish 오픈 소스 응용 프로그램 서버 프로젝트를 구성할 수 있습니다.

참고: 지원되는 Sun GlassFish 버전에 대해서는 *Compatibility Guide*(호환성 안내서)를 참조하십시오.

다음 단계를 따르십시오.

1. GlassFish 에서 다음 위치로 이동합니다.

   ```
   /appserver-install-dir/domains/domain1/config/
   ```

2. domain.xml 파일을 텍스트 편집기에서 엽니다.
3. Agent.jar 파일 및 IntroscopeAgent.profile 의 전체 경로를 domain.xml 파일의 java-config 요소에 JVM 옵션으로 추가합니다. 예:

```
<java-config ..........>
<jvm-options>-javaagent:/sw/wily/Agent.jar</jvm-options>
<jvm-options>-Dcom.wily.introscope.agentProfile=/sw/wily/core/config/IntroscopeAgent.profile</jvm-options>
.....>
```

4. 텍스트 편집기에서 구성 속성을 열고 wily 속성을 추가합니다.

   예를 들어, Glassfish 3.2 의 경우 속성 파일은 <glassfish_home>/glassfish/config/osgi.properties 입니다.

   편집할 속성:

```
eclipselink.bootdelegation=oracle.sql, oracle.sql.*, com.wily.*
org.osgi.framework.bootdelegation=sun.*,com.sun.*,com.wily.*
```

5. Webappsupport.jar 파일을 <Agent_Home>/wily 루트 디렉터리에서 <Agent_Home>/wily/ext 디렉터리로 복사합니다.
6. 응용 프로그램 서버를 시작합니다.

Java Agent 를 사용하도록 SAP Netweaver 구성

Java Agent 를 사용하려면 JVM AutoProbe for SAP NetWeaver 를 구성합니다.

참고: 지원되는 SAP NetWeaver 버전에 대해서는 *호환성 안내서*를 참조하십시오.

다음 단계를 따르십시오.

1. SAP Configtool(configtool.bat)을 시작합니다.
2. 수정할 인스턴스를 선택합니다.
3. 오른쪽 창에서 VM Parameters(VM 매개 변수) > System(시스템)을 선택합니다.
4. -D 를 제공하지 않고 매개 변수를 생성합니다. 예:

```
name: com.wily.introscope.agentProfile
value: <Agent_Home>/core/config/IntroscopeAgent.profile
```

5. Additional(추가) 탭을 클릭하고 다음과 같이 매개 변수를 생성합니다.
   ```
   name: -javaagent:<Agent_Home>\Agent.jar
   value: <leave empty>
   ```
6. 변경 사항을 저장합니다.
7. SAP 서버를 다시 시작합니다.
8. Configtool 을 사용하여 변경한 내용이 적용되었는지 확인하고 다음 파일을 엽니다.
   ```
   <drive>:\usr\sap\...\j2ee\cluster\instance.properties
   ```
9. ID<*server_id*>.JavaParameters 로 시작하는 행을 찾아 입력한 정보가 포함되어 있는지 확인합니다.

Enterprise Manager 에 대한 연결 구성

에이전트는 메트릭을 보고하기 위해 Enterprise Manager 에 연결해야 합니다. 기본 통신 설정을 사용할 경우 에이전트는 포트 5001 을 사용하여 로컬 Enterprise Manager 에 연결할 수 있습니다. 그러나 일반적으로 에이전트와 Enterprise Manager 는 동일한 시스템에 있지 않습니다. 에이전트를 설치할 때 또는 에이전트를 설치한 후에 *IntroscopeAgent.profile* 을 수정하여 기본 설정을 수정할 수 있습니다.

요구 사항에 따라 다음을 사용하도록 에이전트와 Enterprise Manager 간의 통신을 구성할 수 있습니다.

- 직접 소켓 연결
- HTTP 터널링 연결, 또는 프록시 서버를 통한 HTTP 터널링
- HTTPS(HTTP over Secure Sockets Layer) 연결
- SSL(Secure Socket Layer) 연결

직접 소켓 연결을 사용하여 Enterprise Manager 에 연결

에이전트에서 Enterprise Manager 에 연결하는 가장 일반적인 방법은 직접 소켓 연결을 사용하는 것입니다. 가능한 경우 직접 소켓 연결을 사용하여 Enterprise Manager 에 연결하는 것이 좋습니다.

에이전트와 Enterprise Manager 간의 직접 소켓 연결을 구성하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 속성을 찾아 에이전트에서 기본적으로 연결할 Enterprise Manager 의 호스트 이름 또는 IP 주소를 지정합니다. 예:

   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=sfcollect01
   ```

   Enterprise Manager 가 둘 이상 있는 클러스터를 사용하는 경우에는 Collector Enterprise Manager 를 지정해야 합니다.
3. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 속성을 Enterprise Manager 수신 포트로 설정합니다. 예:

   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001
   ```
4. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 속성을 Enterprise Manager 에 연결하는 데 사용되는 소켓 팩터리로 설정합니다. 예:

   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.DefaultSocketFactory
   ```
5. (선택 사항) 기본 Enterprise Manager 에 대한 연결이 끊어질 경우에 에이전트에서 연결할 백업 Enterprise Manager 를 하나 이상 지정합니다.
6. *IntroscopeAgent.profile* 파일을 저장하고 닫습니다.

HTTP 터널링을 사용하여 Enterprise Manager 에 연결

Enterprise Manager 에 대한 직접 소켓 연결이 적절하지 않은 경우 HTTP 를 통해 Enterprise Manager 에 연결하도록 에이전트를 구성할 수 있습니다. 이 구성을 사용하면 HTTP 트래픽만 허용하는 방화벽을 통과하여 통신할 수 있습니다.

참고: HTTP 터널링은 응용 프로그램 서버 및 Enterprise Manager 에 직접 소켓 연결보다 많은 CPU 및 메모리 오버헤드를 발생시킵니다.

에이전트에 대한 HTTP 터널링을 구성하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.

2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 속성을 에이전트에서 기본적으로 연결할 Enterprise Manager 의 호스트 이름 또는 IP 주소로 설정합니다. 예:
   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=webhost
   ```
3. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 속성을 Enterprise Manager 의 포함된 웹 서버에 대한 HTTP 수신 포트로 설정합니다. 예:
   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8081
   ```
   이 속성은 Enterprise Manager 의 *<EM_Home>/config/IntroscopeEnterpriseManager.properties* 파일에서 *introscope.enterprisemanager.webserver.port* 속성에 지정된 값과 일치해야 합니다. 기본적으로 이 포트는 8081 입니다.
4. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 속성을 HTTP 터널링 소켓 팩터리로 설정합니다. 예:
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory
   ```
5. *IntroscopeAgent.profile* 파일을 저장하고 닫습니다.

HTTP 터널링을 위한 프록시 서버 구성

HTTP 터널링 에이전트가 프록시 서버를 통해 Enterprise Manager 에 연결하도록 구성할 수 있습니다. 이 구성은 에이전트가 프록시 서버를 통해 라우트된 아웃바운드 HTTP 트래픽만 허용하는 방화벽 뒤에서 실행되는 정방향 프록시 서버에 필요합니다.

프록시 서버 구성 속성은 에이전트가 HTTP 를 통해 터널링되도록 구성된 경우에만 적용됩니다. 프록시 서버 구성은 단일 연결뿐 아니라 에이전트에 구성된 모든 HTTP 터널링 연결에 적용됩니다. 이 구성은 여러 Enterprise Manager 가 각각 HTTP 를 통해 연결되는 경우 Enterprise Manager 간의 장애 조치를 구성할 때 특히 고려해야 할 점입니다.

중요! 에이전트 HTTP 터널링을 사용하려면 HTTP/1.1 이 필요합니다. 또한 프록시 서버에서 HTTP Post 를 지원해야 합니다.

중요! 프록시에 연결할 수 없으면 에이전트는 이 프록시를 건너뛰고 Enterprise Manager 와 직접 연결합니다. 프록시에 연결할 수 있지만 인증이 실패하는 경우 에이전트는 이 프록시를 통해 계속 Enterprise Manager 에 연결을 시도합니다.

HTTP 터널링을 위해 프록시 서버를 구성하려면

1. IntroscopeAgent.profile 을 텍스트 편집기에서 엽니다.
2. *introscope.agent.enterprisemanager.transport.http.proxy.host* 속성을 프록시 서버의 호스트 이름 또는 IP 주소로 설정합니다.
3. *introscope.agent.enterprisemanager.transport.http.proxy.port* 속성을 프록시 서버의 포트 번호로 설정합니다.
4. (선택 사항) 프록시 서버에서 인증을 위한 사용자 자격 증명이 필요한 경우 다음 속성을 설정합니다.
   `introscope.agent.enterprisemanager.transport.http.proxy.username=<user_name>`
   `introscope.agent.enterprisemanager.transport.http.proxy.password=<user_password>`
5. IntroscopeAgent.profile 을 저장하고 닫습니다.

HTTPS 터널링을 사용하여 Enterprise Manager 에 연결

IntroscopeAgent.profile 파일의 속성을 구성하여 에이전트가 HTTPS(HTTP over Secure Sockets Layer)를 사용하여 Enterprise Manager 에 연결하도록 할 수 있습니다.

HTTPS 를 통한 연결을 구성하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 속성을 대상 Enterprise Manager 의 호스트 이름 또는 IP 주소로 설정합니다.
3. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 속성을 Enterprise Manager 의 포함된 웹 서버에 대한 HTTPS 수신 포트로 설정합니다. 예:
   `introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444`
4. HTTPS 터널링 소켓 팩터리를 사용하도록 *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 속성을 설정합니다. 예:
   `introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory`
5. *IntroscopeAgent.profile* 을 저장하고 닫습니다.

SSL 을 통해 Enterprise Manager 에 연결

직접 SSL(Secure Socket Layer) 연결을 사용하는 경우 HTTP 터널링 없이 SSL 을 사용하여 에이전트와 Enterprise Manager 의 통신을 구성할 수 있습니다.

SSL 을 사용하여 연결하도록 에이전트를 구성하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. SSL 소켓 팩터리를 사용하여 Enterprise Manager 의 SSL 수신 포트에 연결하도록 에이전트를 구성합니다.
3. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 속성을 대상 Enterprise Manager 의 호스트 이름 또는 IP 주소로 설정합니다.
4. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 속성을 Enterprise Manager 의 SSL 수신 포트로 설정합니다.
5. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 속성을 SSL 소켓 팩터리로 설정합니다. 예를 들어 이 속성을 다음과 같이 설정합니다.
   ```
   com.wily.isengard.postofficehub.link.net.SSLSocketFactory
   ```
6. 에이전트가 Enterprise Manager 를 인증해야 하는 경우 *introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT* 속성의 주석 처리를 제거하고 Enterprise Manager 의 인증서가 들어 있는 신뢰 저장소의 위치로 설정합니다. 신뢰 저장소를 지정하지 않으면 에이전트는 모든 인증서를 신뢰합니다. 절대 경로나 에이전트 프로필을 기준으로 한 상대 경로를 지정할 수 있습니다. 예:
   ```
   introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/certs
   ```
7. 필요한 경우 *introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT* 속성을 설정하여 신뢰 저장소 암호를 지정합니다.
8. Enterprise Manager 에서 클라이언트 인증이 필요한 경우 *introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT* 속성을 에이전트의 인증서가 들어 있는 키 저장소의 위치로 설정합니다. 절대 경로나 에이전트 프로필을 기준으로 한 상대 경로를 지정할 수 있습니다. Windows 에서는 백슬래시를 이스케이프해야 합니다. 예:
   ```
   introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=C:\\keystor
   e
   ```
9. 필요한 경우 *introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT* 속성을 키 저장소 암호로 지정합니다.

10. *introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT* 속성을 허용되는 암호 그룹의 쉼표로 구분된 목록으로 설정합니다.

    이 속성 값을 지정하지 않으면 기본 암호 그룹이 사용됩니다.

11. *IntroscopeAgent.profile* 을 저장하고 닫습니다.

에이전트 부하 분산 구성

작업 부하가 에이전트에 의해 보고되는 기본 메트릭인 클러스터에서는 MOM 에이전트 부하 분산을 구성하여 전체 클러스터 수용 능력을 최적화할 수 있습니다.

참고: MOM 에이전트 부하 분산에 대한 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

여러 에이전트 유형 업그레이드

일부 환경에는 수천 개의 에이전트가 서로 다른 여러 응용 프로그램 서버에 분산되어 있습니다. 예를 들어 WebLogic 에 3,000 개, WebSphere 에 2,000 개, JBoss 에 3,000 개로 분산된 8,000 개의 에이전트가 환경에 있을 수 있습니다. 여러 응용 프로그램 서버에서 에이전트를 손쉽게 업그레이드하기 위해 에이전트 상위 집합 패키지를 사용할 수 있습니다. 에이전트 상위 집합 패키지는 SAP NetWeaver 를 제외한 지원되는 응용 프로그램 서버에 사용되는 파일이 모두 포함된 운영 체제별 패키지입니다.

다음의 에이전트 상위 집합 패키지를 사용할 수 있습니다.

- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.windows.zip
- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.unix.tar
- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.zOS.tar
- IntroscopeAgentFiles-NoInstaller<version>allappserver.os400.zip

참고: 이러한 파일 이름에서 *<version>*은 Java Agent 의 최신 릴리스 번호를 나타냅니다.

각 패키지에는 다음과 같은 파일이 포함되어 있습니다.

- 모든 응용 프로그램 서버별 PBD 및 PBL

- 모든 응용 프로그램 서버 에이전트 프로필. 파일 이름에 응용 프로그램 서버 이름이 포함되어 있습니다. 예:
  - IntroscopeAgent.weblogic.profile
  - IntroscopeAgent.websphere.profile

  참고: 기본 IntroscopeAgent.profile 은 포함되어 있지 않습니다. 자세한 내용은 3 단계를 참조하십시오.
- 운영 체제 유형에 적합한 모든 에이전트 JAR 파일 및 플랫폼 모니터

다음 단계를 따르십시오.

1. 대상 운영 체제에 적절한 상위 집합 패키지를 선택합니다.
2. 선택한 에이전트 패키지를 응용 프로그램 서버의 홈 디렉터리에 추출합니다.

   참고: *<Agent_Home>*/core/config 디렉터리의 파일 중 사용하지 않을 응용 프로그램 서버를 참조하는 추가적인 PBD 및 PBL 은 무시해도 됩니다.
3. IntroscopeAgent.profile 을 아직 구성하지 *않은* 경우 다음 단계를 수행하십시오.
   a. 적절한 IntroscopeAgent.<*application_server_name*>.profile 을 선택합니다.
   b. 이름을 IntroscopeAgent.profile 로 변경합니다.
   c. 현재 환경에 맞게 파일을 구성합니다.
4. IntroscopeAgent.profile 을 이미 구성한 경우 다음 단계를 수행하십시오.
   a. 편집기에서 해당 IntroscopeAgent.<*application_server_name*>.profile 파일을 엽니다.
   b. 사용할 새 속성을 찾습니다.
   c. 기존 IntroscopeAgent.profile 로 전송하십시오.

추가 정보:

설치 아카이브를 사용하여 수동으로 설치 (페이지 35)

Java Agent 제거

Java Agent 를 제거하려면 모니터링 중인 각 응용 프로그램에 대해 Java Agent 가 설치된 위치를 알고 있어야 합니다.

Java Agent 설치 관리자를 사용하여 Java Agent 를 설치한 경우 제거 프로그램을 사용하여 설치된 에이전트 파일을 제거할 수 있습니다. 제거 프로그램을 시작하고 화면의 지시를 따르십시오.

지원되는 JVM 에서 Java Agent 를 제거하려면

1. 응용 프로그램 서버를 중지합니다.

   참고: 제거 프로그램을 실행하기 전에 모니터링되는 JVM 을 종료해야 합니다.

2. 응용 프로그램 서버 시작 스크립트를 열고 Java 명령줄에서 Java Agent 스위치를 제거합니다. 시작 옵션에는 응용 프로그램 서버에 따라 다음이 포함됩니다.

   - *-Xbootclasspath*
   - *-javaagent:<path_to_the_agent_jar>*
   - *-Dcom.wily.introscope.agentProfile=<path_to_the_agent_profile>*

3. 응용 프로그램 서버를 재부팅합니다.
4. *<Agent_Home>*/UnstallerData 디렉터리에 있는 *Uninstall_Introscope_Agent* 프로그램을 실행합니다.
5. *<Agent_Home>* 디렉터리를 수동으로 삭제합니다.

z/OS 에서 Java Agent 제거

z/OS 에서 Java Agent 를 제거하려면 *rm -rf* 명령을 사용하여 *<Agent_Home>* 디렉터리를 삭제하는 것이 좋습니다. 타사 버그로 인해 z/OS 에서는 실행 파일 제거 프로그램이 올바르게 실행되지 않기 때문입니다.

중요! 모니터링되는 JVM 을 종료한 후에만 파일을 제거하십시오.

z/OS 에 설치된 활성 Introscope 의 경우 *UninstallerData* 폴더를 그대로 유지해야 합니다. *UninstallerData* 폴더를 삭제하면 이후 버전의 Introscope 로 업그레이드할 수 없게 됩니다. 전체 인스턴스를 제거하려는 경우가 아니면 *UninstallerData* 폴더를 삭제하지 마십시오.

제 3 장: 에이전트 속성 구성

대부분의 에이전트 작업은 *IntroscopeAgent.profile* 파일 내의 속성을 사용하여 구성됩니다. 이 단원에서는 가장 일반적으로 설정하는 에이전트 속성에 대해 설명합니다. 사용 환경에 따라 추가 속성도 적합할 수 있습니다. 각 에이전트 버전에 대해 서로 다른 속성을 설정하거나 서로 다른 기본값을 사용할 수도 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

Enterprise Manager 와의 통신을 수정하는 방법

에이전트는 메트릭을 보고하기 위해 Enterprise Manager 에 연결해야 합니다. 에이전트를 설치한 후 *IntroscopeAgent.profile* 을 수정하여 사용되는 통신 채널을 수정할 수 있습니다.

요구 사항에 따라 다음을 사용하도록 에이전트와 Enterprise Manager 간의 통신을 구성할 수 있습니다.

- 직접 소켓 연결
- HTTP 터널링 연결, 또는 프록시 서버를 통한 HTTP 터널링
- HTTPS(HTTP over Secure Sockets Layer) 연결
- SSL(Secure Socket Layer) 연결

에이전트에서 Enterprise Manager 에 연결하는 가장 일반적인 방법은 직접 소켓 연결을 사용하는 것입니다. 가능한 경우 직접 소켓 연결을 사용하여 Enterprise Manager 에 연결하는 것이 좋습니다. 다른 유형의 연결을 사용하려면 해당 단원을 참조하십시오.

백업 Enterprise Manager 및 장애 조치 속성을 구성하는 방법

에이전트를 설치할 때는 에이전트에서 기본적으로 연결할 Enterprise Manager 의 호스트 이름 및 포트 번호를 지정합니다. 필요한 경우 백업 Enterprise Manager 를 하나 이상 지정할 수도 있습니다. 에이전트와 기본 Enterprise Manager 사이의 연결이 끊어지면 에이전트는 백업 Enterprise Manager 에 연결을 시도할 수 있습니다.

에이전트가 백업 Enterprise Manager 에 연결할 수 있게 하려면 에이전트 프로필에 Enterprise Manager 의 통신 속성을 지정하십시오. 기본 Enterprise Manager 를 사용할 수 없으면 에이전트는 허용되는 연결 목록의 다음 Enterprise Manager 에 연결을 시도합니다. 첫 번째 백업 Enterprise Manager 와 연결할 수 없으면 목록의 다음 Enterprise Manager 에 연결을 시도합니다. 에이전트는 연결에 성공할 때까지 이런 식으로 목록의 각 Enterprise Manager 에 대해 연결을 시도하며, 어떠한 Enterprise Manager 에도 연결할 수 없는 경우에는 10 초 동안 기다렸다가 다시 시도합니다.

다음 단계를 따르십시오.

1. *IntroscopeAgent.profile* 파일을 텍스트 편집기에서 엽니다.
2. 각 백업 Enterprise Manager 에 대해 다음 속성을 에이전트 프로필에 추가하여 대체 Enterprise Manager 통신 채널을 하나 이상 지정합니다.

   ```
   introscope.agent.enterprisemanager.transport.tcp.host.NAME
   introscope.agent.enterprisemanager.transport.tcp.port.NAME
   Introscope.agent.enterprisemanager.transport.tcp.socketfactory.NAME
   ```

   여기서 *NAME* 은 새 Enterprise Manager 채널의 식별자로 바꿉니다. 새 채널을 생성할 때는 DEFAULT 나 기존 채널 이름을 사용하지 마십시오. 다음은 백업 Enterprise Manager 를 두 개 생성하기 위한 예입니다.

   ```
   introscope.agent.enterprisemanager.transport.tcp.host.BackupEM1=paris
   introscope.agent.enterprisemanager.transport.tcp.port.BackupEM1=5001
   Introscope.agent.enterprisemanager.transport.tcp.socketfactory.BackupEM1=com.
   custom.postofficehub.link.net.DefaultSocketFactory
   introscope.agent.enterprisemanager.transport.tcp.host.BackupEM2=voyager
   introscope.agent.enterprisemanager.transport.tcp.port.BackupEM2=5002
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.BackupEM2=com.
   wily.isengard.postofficehub.link.net.DefaultSocketFactory
   ```

3. *introscope.agent.enterprisemanager.connectionorder* 속성을 찾아 기본 및 백업 Enterprise Manager 의 식별자를 포함하는 쉼표로 구분된 목록으로 설정합니다. 목록 내의 식별자 순서에 따라 Enterprise Manager 의 연결 순서가 정의됩니다. 예:
   ```
   introscope.agent.enterprisemanager.connectionorder=DEFAULT,BackupEM1,BackupEM2
   ```
4. *introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds* 속성을 찾아 에이전트가 기본 Enterprise Manager 에 연결을 시도할 빈도를 지정합니다. 기본 간격은 120 초(2 분)입니다. 예:
   ```
   introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120
   ```
5. 변경 내용을 저장하고 *IntroscopeAgent.profile* 파일을 닫습니다.
6. 응용 프로그램을 다시 시작합니다.

추가 GC 메트릭을 사용하도록 설정하고 사용하는 방법

가비지 수집 및 메모리 관리는 응용 프로그램 성능에 상당한 영향을 줄 수 있습니다. 기본 GC 힙 메트릭은 기본적으로 사용할 수 있습니다. 가비지 수집 처리 및 메모리 풀 사용량에 대한 추가 세부 정보를 제공하기 위해 선택적 메트릭을 사용하도록 설정할 수도 있습니다. 사용하도록 설정된 추가 메트릭은 Investigator 의 "GC Monitor"(GC 모니터) 노드 아래에 표시됩니다. "GC Monitor"(GC 모니터) 메트릭은 메모리 풀 할당 및 가비지 수집 처리를 최적화하는 데 유용한 정보를 보고합니다. 따라서 이러한 메트릭은 대개 응용 프로그램을 개발 또는 테스트하거나 응용 프로그램 성능 문제를 조사할 때 사용합니다. 대부분의 경우 프로덕션 환경에서는 이러한 메트릭이 실시간 응용 프로그램 관리에 사용되지 않으며 기본적으로 사용하지 않도록 설정되어 있습니다.

"GC Monitor"(GC 모니터) 메트릭을 사용하도록 설정하려면 에이전트 프로필을 텍스트 편집기에서 열고 *introscope.agent.gcmonitor.enable* 속성을 true 로 설정하면 됩니다. 이 속성을 사용하도록 설정한 후에는 모니터링 중인 JVM 의 가비지 수집기 및 메모리 풀에 대한 세부 정보를 볼 수 있습니다. 메트릭에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

스레드 덤프를 사용하도록 설정하고 구성하는 방법

스레드 덤프는 에이전트 JVM 내에서 발생하는 사항에 대한 유용한 세부 정보를 제공할 수 있습니다. 스레드 덤프 기능은 메트릭 브라우저 트리의 각 에이전트 노드와 연결된 "스레드 덤프" 탭에서 제공됩니다.

스레드 덤프 수집 및 분석에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오. Thread_Dump 권한을 설정하면 사용자가 "스레드 덤프" 탭을 보고 모든 기능을 사용할 수 있습니다. 자세한 내용은 *CA APM 보안 안내서*를 참조하십시오.

스레드 덤프를 사용하도록 설정하려면 *IntroscopeAgent.profile* 및 *IntroscopeEnterpriseManager.properties* 속성이 모두 필요합니다. 기본적으로 "스레드 덤프" 탭과 해당 기능은 사용하도록 설정되어 있습니다. 그러나 스레드 덤프 설정 속성 중 하나 또는 둘 모두가 false 로 설정되어 있는 경우에는 "스레드 덤프" 탭을 볼 수 없습니다.

MOM 에서 스레드 덤프를 사용하거나 사용하지 않도록 설정할 경우 해당 구성은 클러스터의 모든 수집기에 적용됩니다. 따라서 MOM 에서 스레드 덤프를 사용하지 않도록 설정하면 모든 수집기에서도 스레드 덤프가 사용하지 않도록 설정됩니다.

스레드 덤프를 사용하도록 설정하려면

1. <*Agent_Home*>/wily/core/config 디렉터리에 있는 *IntroscopeAgent.profile* 파일을 열고 다음 속성을 설정합니다.
   ```
   introscope.agent.threaddump.enable=true
   ```
2. *IntroscopeAgent.profile* 을 저장하고 닫습니다.
3. <*EM_Home*>/config 디렉터리에 있는 *IntroscopeEnterpriseManager.properties* 파일을 열고 다음 속성을 설정합니다.
   ```
   introscope.enterprisemanager.threaddump.enable=true
   ```
4. *IntroscopeEnterpriseManager.properties* 를 저장하고 닫습니다.

CA Introscope 사용자가 "Deadlock Count"(교착 상태 수) 메트릭을 볼 수 있도록 하려면 *IntroscopeAgent.profile* 을 구성합니다. 추가 구성을 수행하여 에이전트 "Thread"(스레드) 노드 메트릭을 표시할 수 있습니다.

"Deadlock Count"(교착 상태 수) 메트릭 수집 기능을 사용하도록 설정하려면

1. <*Agent_Home*>/wily/core/config 디렉터리에 있는 *IntroscopeAgent.profile* 파일을 엽니다.
2. 이 속성을 true 로 설정하여 수집할 "Deadlock Count"(교착 상태 수) 메트릭을 수집하도록 설정합니다.
   `introscope.agent.threaddump.deadlockpoller.enable=true`
3. (선택 사항) 전체 버전의 PBL 에서 메트릭을 에이전트 "Threads"(스레드) 노드에 표시하도록 설정합니다.
   - introscope.autoprobe.directivesFile 속성에서 PBL 파일의 이름을 지정합니다.

     예를 들어 WebLogic Server 를 위한 전체 버전의 표준 PBL 을 사용하려면 이 속성을 다음과 같이 설정합니다.
     `introscope.autoprobe.directivesFile=weblogic-full.pbl`

     사용자는 *AgentName* | "스레드"에서 활성 스레드 수 및 스레드 그룹과 같은 메트릭을 볼 수 있습니다.
4. *IntroscopeAgent.profile* 을 저장하고 닫습니다.

IntroscopeAgent.profile 및 *IntroscopeEnterpriseManager.properties* 속성을 모두 사용하여 스레드 덤프를 구성합니다.

스레드 덤프를 구성하려면

1. <*EM_Home*>/config 디렉터리에 있는 *IntroscopeEnterpriseManager.properties* 파일을 엽니다.
2. (선택 사항) 스레드 덤프 파일을 Enterprise Manager 의 특정 디렉터리에 저장하도록 다음 속성을 설정합니다. 예를 들어 TestThreadDumps 이라는 이름일 수 있습니다.
   `introscope.enterprisemanager.threaddump.storage.dir=TestThreadDumps`
3. (선택 사항) 지정된 기간(일)보다 오래된 스레드 덤프 파일을 삭제하도록 다음 속성을 설정합니다. 예를 들어 30 일보다 오래된 스레드 덤프 파일을 삭제합니다.
   `introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days=30`

4. (선택 사항) 지정된 기간(일) 후에 스레드 덤프 파일을 삭제하도록 다음 속성을 설정합니다. 예를 들어 2 일 후마다 스레드 덤프 파일을 삭제합니다.
   `introscope.enterprisemanager.threaddump.storage.clean.disk.freq.days=2`
5. (선택 사항) Enterprise Manager 에 저장할 수 있는 최대 스레드 덤프 파일 수를 제한하도록 다음 속성을 설정합니다. 예를 들어 파일 5,000 개로 제한합니다.
   `introscope.enterprisemanager.threaddump.storage.max.disk.usage=5000`

   참고: 다음 경우에 유의해야 합니다.

   * 저장되는 스레드 덤프 파일의 수가 introscope.enterprisemanager.threaddump.storage.max.disk.usage 속성에 설정된 제한을 초과하는 경우

   그리고

   * introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days 속성에 설정된 기간(일)보다 오래된 파일이 없는 경우

   이러한 경우에는 Enterprise Manager 가 스레드 덤프 파일을 저장하지 않습니다.
6. *IntroscopeEnterpriseManager.properties* 를 저장하고 닫습니다.
7. Enterprise Manager 를 다시 시작합니다.

Enterprise Manager 가 다운되는 경우 사용자가 스레드 덤프 데이터를 볼 수 있도록 스레드 덤프 파일을 다른 Enterprise Manager 에 복사할 수 있습니다.

중요! 스레드 덤프 디렉터리에 파일을 추가하거나 스레드 덤프 디렉터리에서 파일을 제거한 경우에는 Enterprise Manager 를 다시 시작하십시오. 스레드 덤프 파일은 Enterprise Manager 간에 이동하지 않는 것이 좋습니다.

스레드 덤프 파일을 Enterprise Manager 간에 복사하려면

1. 스레드 덤프 파일이 들어 있는 Enterprise Manager(EM1)의 <*EM_Home*>/threaddumps 디렉터리로 이동합니다.
2. 스레드 덤프 파일을 복사합니다.

3. 사용자가 스레드 덤프를 볼 수 있도록 하려는 Enterprise Manager(EM2)의 <*EM_Home*>/threaddumps 디렉터리에 해당 파일을 붙여넣습니다.
4. Enterprise Managers EM1 및 EM2 를 모두 다시 시작합니다.
5. 필요한 경우 에이전트 연결을 설정하고, EM2 에서 스레드 덤프를 사용하도록 설정하고 구성합니다.

   EM2 사용자가 에이전트 노드를 선택한 다음 "스레드 덤프" 탭의 "이전 항목 로드" 단추를 클릭하면 EM1 에서 이동된 스레드 덤프가 목록에 표시됩니다.

분포 통계 메트릭을 수집하도록 에이전트를 구성하는 방법

"평균 응답 시간" 메트릭을 생성하기 위해 BlamePointTracer 가 분석한 응답 시간 분포를 수집하도록 에이전트를 구성할 수 있습니다. 분포 통계 메트릭은 특정 작업이 어떻게 다른지 설명하는 자세한 설명 정보를 제공합니다. 모니터링된 응답 시간 값의 통계적 분포에 대한 추가적인 데이터는 getExtendedMetricData 웹 서비스를 통해 얻을 수 있습니다.

CA APM 에이전트는 특정 작업에 대한 응답 시간 정보를 수집하도록 구성할 수 있습니다. 응답 시간 정보는 선택한 작업에 대한 평균 응답 시간 메트릭과 쌍을 이루어 분포 통계 메트릭에 저장됩니다.

다음 단계를 따르십시오.

1. <*Agent_Home*>/wily/core/config 디렉터리에 있는 *IntroscopeAgent.profile* 파일을 엽니다.
2. introscope.agent.distribution.statistics.components.pattern 의 주석을 해제하고 편집하여 쌍을 이룬 분포 통계 메트릭을 생성할 평균 응답 시간 메트릭을 지정합니다. 예를 들어 이 식은 다음과 같습니다.

   ```
   Servlets\\|LoginServlet:.*
   ```

   LoginServlet 응답 시간에 대한 분포 통계가 생성되었습니다.
3. (선택 사항) 다음 지침을 따라 식을 만듭니다.
   a. 세로 막대와 마침표는 정규식 내에서 특수한 의미를 가지므로 메트릭 노드 구분 기호 앞에는 백슬래시를 사용합니다.
   b. 백슬래시는 에이전트 프로필에서 특수한 의미를 가지므로 백슬래시 앞에는 또 다른 백슬래시를 사용합니다.

      정규식은 에이전트 로그에서 특수 문자 뒤에 표시됩니다. 예:

      ```
      Servlets\|Login Servlet:.*
      ```
   c. 요약 수준 및 개별 메트릭 정규식을 매칭합니다. 메트릭이 매칭되지 않으면 요약 수준에 대한 분포 통계가 요약되거나 생성되지 않습니다.

      다음 예는 매칭하는 식을 나타냅니다.

      ```
      Servlets(\\|.*):.*
      ```

      ```
      Servlets.*
      ```
4. IntroscopeAgent.profile 파일을 저장하고 닫습니다.
5. 에이전트를 다시 시작합니다.

분포 통계 메트릭의 예

구성 속성을 설정한 방법에 따라 다음에 대해 분포 통계를 수집할 수 있습니다.

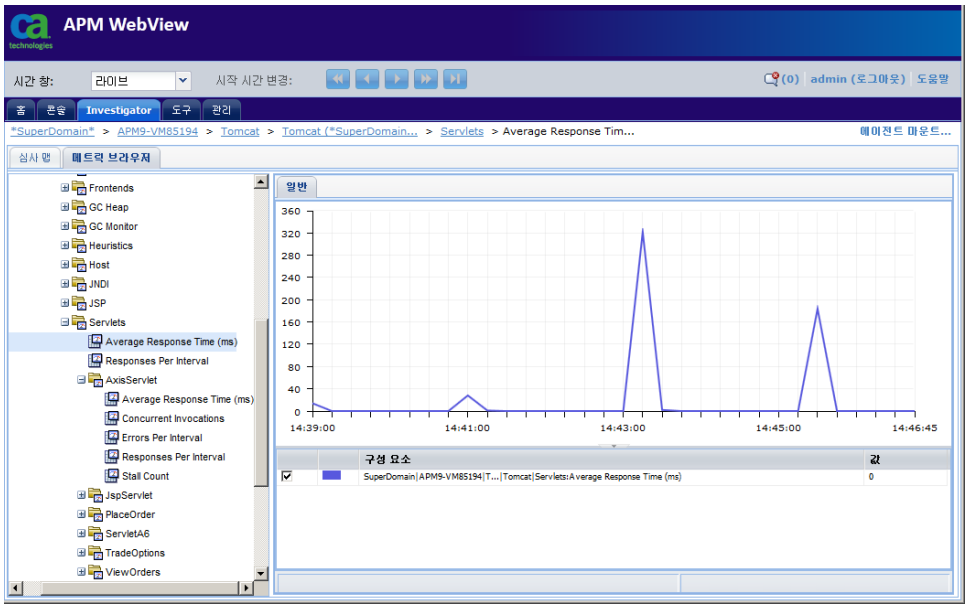
- 개별 및 요약 수준에 대한 서블릿 및 JSP 분포 메트릭의 예제 (see page 85)
- 서블릿 및 JSP 분포 메트릭의 예제 (see page 86)

개별 및 요약 수준에 대한 서블릿 및 JSP 분포 메트릭의 예제

서블릿 및 JSP 에 대한 개별 및 요약 수준의 분포 통계를 수집하려면 다음 패턴을 사용하십시오.

```
introscope.agent.distribution.statistics.components.pattern=(Servlets|JSP)(\\|.*):.*
```

다음 이미지는 서블릿에 대한 분포 통계 메트릭과 Java Agent 노드에 대한 예약 수준을 수집하는 Investigator 를 보여 줍니다.

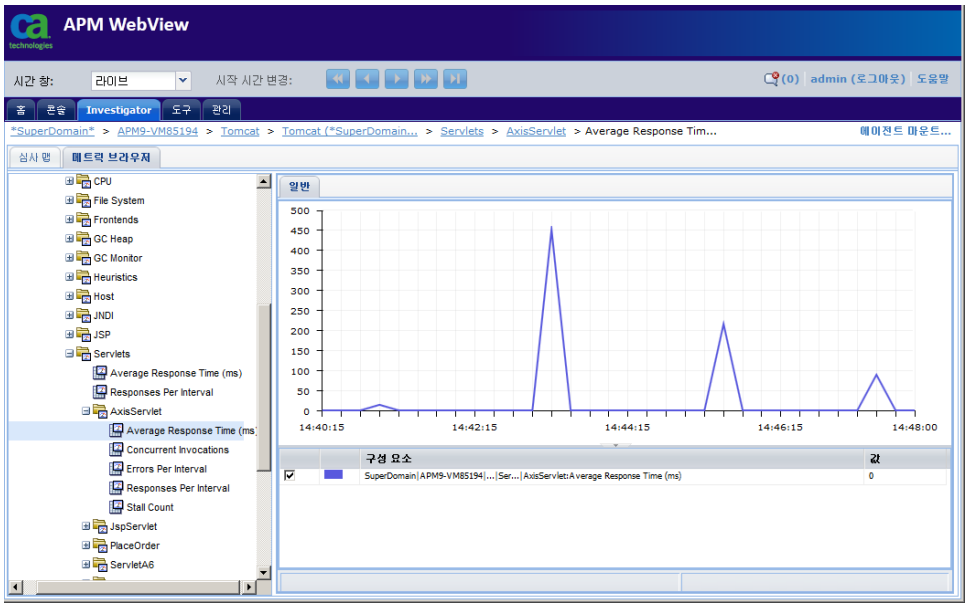


서블릿 및 JSP 분포 메트릭의 예제

서블릿 또는 JSP 요약 수준이 아닌 서블릿 및 JSP 수준에 대한 분포 통계를 수집하려면 이 패턴을 사용하십시오.

```
introscope.agent.distribution.statistics.components.pattern=(Servlets|JSP)\\|.*
```

다음 이미지는 Java Agent 노드에 대한 요약 수준이 아닌 서블릿에 대한 분포 통계 메트릭을 수집하는 Investigator 를 보여 줍니다.



제 4 장: AutoProbe 및 ProbeBuilding 옵션

이 섹션은 다음 항목을 포함하고 있습니다.

AutoProbe 및 ProbeBuilding 개요

Java Agent 는 모니터링할 응용 프로그램의 바이트 코드에 프로브를 삽입합니다. ProbeBuilding 은 PBD(ProbeBuilder 지시문) 및 PBL(ProbeBuilder 목록)을 사용하여 응용 프로그램에 삽입할 프로브를 결정하는 방법입니다. Java Agent 에 포함된 기본 PBD 및 PBL 파일은 기본 수준의 메트릭 수집 기능을 제공합니다. 이러한 파일의 기본 설정을 수정하여 환경에 더욱 적합하도록 메트릭 수집을 조정할 수 있습니다.

응용 프로그램과 환경에서 수집할 메트릭에 대한 프로브를 삽입하려면 PBD 와 PBL 을 구성합니다. 그런 다음 이러한 파일을 사용하여 JVM AutoProbe 를 사용한 자동 방법 또는 ProbeBuilder 를 사용한 수동 방법으로 응용 프로그램을 계측합니다. 응용 프로그램을 계측하려면 JVM AutoProbe 를 사용하십시오. 그러나 JVM AutoProbe 구성은 응용 프로그램 환경에 따라 달라집니다.

참고: 지원되는 JVM 버전에 대해서는 *호환성 안내서*를 참조하십시오.

JVM 에서 응용 프로그램을 계측하려면 다음 방법 중 하나를 사용하십시오.

- JVM AutoProbe(-javaagent 속성 사용). CA Technologies 는 JVM AutoProbe 를 JVM 에서 응용 프로그램을 계측할 것을 강력히 권장합니다.
- 수동 ProbeBuilding 은 고급 계측 기술입니다. 이 방법을 사용하려면 CA Support 에 문의하십시오.

중요! 응용 프로그램을 계측할 때는 한 가지 계측 방법만 사용해야 합니다.

Java Agent 를 설치 및 구성 (see page 27)했는지 확인하십시오.

지원되지 않는 계측 방법

AutoProbe for Application Servers 계측 방법은 Java Agent 에 대해 지원되지 않습니다. AutoProbe for Application Servers 를 사용하여 이전 JVM 버전(1.4 이하)과 이전 Java Agent 버전을 사용하는 응용 프로그램을 계측할 수 있습니다. CA Technologies 는 JVM 1.5 이상을 사용하는 응용 프로그램에 대해 JVM AutoProbe 를 사용할 것을 권장합니다.

참고: 지원되는 JVM 버전에 대한 자세한 내용은 *호환성 안내서*를 참조하십시오. *호환성 안내서*에 달리 설명되지 않은 한, 지원되는 JVM 은 응용 프로그램 서버 버전과 함께 제공된 JVM 입니다.

ProbeBuilding 구성

ProbeBuilding 기술은 계측 프로세스를 수행합니다. ProbeBuilder 지시문(PBD) 파일에서 정의된 프로브는 에이전트가 런타임에 웹 응용 프로그램과 가상 컴퓨터에서 수집하는 메트릭을 식별합니다.

기본적으로, AutoProbe 는 Java Agent 에서 제공되는 일반적인 PBD 를 사용합니다. 이 설정은 적당한 수의 메트릭을 수집합니다. 다음 옵션을 사용하여 메트릭 수집 수준을 사용자 지정하고 ProbeBuilding 동작을 구성할 수 있습니다.

- 전체 또는 표준 추적 옵션 (see page 89)
- 동적 ProbeBuilding (see page 89)
- ProbeBuilding 클래스 계층 (see page 94)
- 바이트 코드의 행 번호 제거 (see page 98)

전체 또는 표준 추적 옵션

Introscope 에서 PBL(ProbeBuilder 목록) 파일은 계측 프로세스에 사용되는 추적 프로그램 그룹을 제어합니다. *introscope.autoprobe.directivesFile* 속성은 PBL 파일을 하나 이상 지정합니다.

Introscope 에서는 각 기본 PBL 을 두 가지 버전으로 제공합니다. *전체* 버전은 표준 버전보다 많은 수의 추적 프로그램 그룹을 사용하여 메트릭을 세부적으로 보고하고, *표준* 버전은 보다 적은 수의 추적 프로그램 그룹을 사용합니다. 따라서 메트릭 보고의 세부 수준은 낮지만 오버헤드가 줄어듭니다. 기본적으로 *introscope.autoprobe.directivesFile* 은 표준 버전의 기본 PBL 파일을 지정합니다.

전체 및 표준 간에 추적 수준을 변경하려면

1. 관리되는 응용 프로그램을 중지합니다.
2. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
3. *introscope.autoprobe.directivesFile* 속성에서 사용할 PBL 파일의 이름을 지정합니다.

   예를 들어 WebLogic Server 를 위한 전체 버전의 표준 PBL 을 사용하려면 이 속성을 다음과 같이 설정합니다.

   ```
   introscope.autoprobe.directivesFile=weblogic-full.pbl
   ```
4. 관리되는 응용 프로그램을 다시 시작합니다.

동적 ProbeBuilding

CA Introscope®는 동적 ProbeBuilding 을 사용하여 관리되는 응용 프로그램 또는 에이전트를 다시 시작하지 않고 새 PBD 와 변경된 PBD 를 구현합니다. 동적 ProbeBuilding 은 PBD 를 수정하거나 심사 중에 응용 프로그램 서비스를 중단하지 않고 데이터 수집 수준을 변경하는 데 유용합니다.

중요! 동적 ProbeBuilding 은 Java 1.5 이상에서만 사용할 수 있습니다. 동적 ProbeBuilding 은 Java 1.5 기능과 -javaagent 명령을 사용합니다.

참고: Workstation 에서는 트랜잭션 추적 뷰어를 통해 동적 계측을 수행할 수 있습니다. 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

동적 ProbeBuilding 을 사용하면 CA Introscope®가 주기적으로 새 PBD 및 변경된 PBD 를 검색합니다. 오버헤드를 최소화하기 위해 CA Introscope®는 수정된 PBD 가 영향을 주는 클래스만 선별적으로 다시 계측합니다. 성능 향상을 위해 동적 에이전트 재계측 범위는 PBD 가 편집될 때 계측이 변경된 클래스만 다시 로드하도록 제한됩니다.

PBD 를 편집하거나 PBD 를 hotdeploy 디렉터리에 추가하는 경우 사용자 지시문(예: 클래스에 대한 추가 또는 제거 지시문, 추적 프로그램 그룹 전환)만 다시 계측됩니다.

중요! 추적 프로그램 그룹을 사용하는 지시문에 대한 변경 내용(예: IfFlagged 스위치가 있는 TraceAllMethods 와 같은 지시문의 변경 내용)만 지원됩니다. 하지만 CA Introscope®는 추적 프로그램 그룹 또는 플래그가 있는 즉시 사용 가능한 지시문만 제공합니다. 건너뛰기 또는 변환에 대한 변경 사항은 지원되지 않습니다.

다음과 같은 지시문은 다시 계측되지 않습니다.

- 추적 프로그램 추가 또는 새 추적 프로그램 매핑 변경과 같은 시스템 지시문
- Skip 지시문에 지정된 배열, 인터페이스, 클래스 및 모든 변환

다음을 실행하도록 다시 계측 프로세스를 구성할 수 있습니다.

- 특정 클래스 로더가 로드하는 모든 클래스 제외
- 특정 클래스 패키지로 범위 제한

참고: 동적 ProbeBuilding 은 기본적으로 사용하도록 설정되어 있지 않습니다.

클래스가 다시 계측되어 메트릭에 대한 데이터를 더 이상 보고하지 않는 경우에도 메트릭은 여전히 Investigator 에 표시됩니다. 클래스가 다시 계측되더라도 해당하는 기존 메트릭은 Investigator 창에서 사라지지 않습니다.

중요! Java 1.5 의 제한으로 인해 일부 클래스 바이트에는 액세스할 수 없으며, 그로 인해 다음과 같은 영향이 있습니다.

- j2ee.pbd 파일에 대한 수정 사항은 선택되지 않을 수 있으며 메트릭이 계속해서 이전 이름으로 게시될 수 있습니다.
- 일부 예외가 에이전트 로그에 나타날 수 있습니다.
- 이 문제를 방지하려면 j2ee.pbd 파일을 수정한 후 응용 프로그램 서버를 다시 시작하십시오.

동적 ProbeBuilding 을 구성할 경우 추적 프로그램 그룹을 기반으로 변경하는 것이 좋습니다.

예: 추적 프로그램 그룹 XYZ 에 대한 계측 수준 제어

이 예제는 추적 프로그램 그룹에 대한 계측 수준을 제어하는 방법에 대해 설명합니다.

다음 단계를 따르십시오.

1. 두 개의 계측 프로그램 그룹을 만듭니다.
   - XYZTracing - 일반 추적 옵션
   - XYZTracingLite - 더 적은 구성 요소가 추적됨
2. 둘 사이의 전환: XYZTracing 을 끄고 XYZTracingLite 를 켭니다.
3. 환경 성능에 대한 동적 ProbeBuilding 의 영향을 봅니다.
4. 추적 그룹을 적절히 조정합니다.

   조정은 각 추적 프로그램 그룹의 일부로 추적되는 모든 클래스에 영향을 줍니다.

동적 ProbeBuilding 구성

동적 ProbeBuilding 을 구성하려면 IntroscopeAgent.profile 을 편집하십시오.

다음 단계를 따르십시오.

1. <*Agent_Home*>/wily/core/config 디렉터리로 이동합니다.
2. IntroscopeAgent.profile 을 텍스트 편집기에서 엽니다.
3. introscope.autoprobe.enable 속성이 true 로 설정되어 있는지 확인합니다.

4. 다음 속성의 주석 처리를 제거하여 해당 속성을 설정합니다.

   - introscope.autoprobe.dynamicinstrument.enabled=true

     이 속성은 동적 ProbeBuilding 을 사용하도록 설정합니다. 관리되는 응용 프로그램을 다시 시작하면 이 속성이 적용됩니다.

   - introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1

     PBD 변경 사항을 확인할 폴링 간격(분)입니다. 기본값은 1 분 간격으로 설정됩니다. 관리되는 응용 프로그램을 다시 시작하면 이 속성이 적용됩니다.

   - introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1

     일부 클래스 로더 구현에서는 대량의 클래스 파일을 반환하는 것으로 확인되었습니다. 이 동작으로 메모리 오류를 방지할 수 있습니다. 관리되는 응용 프로그램을 다시 시작하면 이 속성이 적용됩니다.

   - introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10

     한 번에 너무 많은 클래스를 다시 정의하면 CPU 가 무리하게 사용될 수 있습니다. 이 속성을 사용하면 PBD 의 변경 사항으로 인해 많은 수의 클래스를 다시 정의해야 하는 경우 프로세스가 안정적인 속도로 일괄 처리됩니다.

5. IntroscopeAgent.profile 의 변경 사항을 저장하고 파일을 닫습니다.
6. 관리되는 응용 프로그램을 다시 시작합니다(해당하는 경우).

동적 계측이 IBM JDK 의 성능에 영향을 줌

IBM JDK 버전 5 를 사용하는 CA Introscope®에 해당

참고: CA Introscope®를 IBM JDK 버전 6 과 함께 사용할 경우에는 클래스 재정의를 사용하는 데 관련된 성능 오버헤드가 없습니다.

증상:

동적 계측 기능을 사용하려면 클래스 재정의 지원이 필요합니다. 클래스 재정의를 사용하면 성능에 큰 영향을 줄 수 있습니다. 이 성능 오버헤드에 대한 기술 정보는 IBM Java Diagnostics Guide (Java 진단 안내서)에 제공되어 있습니다. 동적 계측의 이점을 누리려는 CA Introscope 및 IBM JDK 고객은 이 성능 오버헤드를 염두에 두어야 합니다.

해결책:

CA Technologies 는 프로덕션 환경에서만 동적 계측 기능을 사용할 것을 권장합니다.

추가 정보:

UNIX, Windows, OS/400, z/OS, IBM JVM 1.5 에서 WebSphere Application Server 6.1 구성 (페이지 54)
소켓 메트릭 (페이지 156)

동적 ProbeBuilding 과 동적 계측 비교

동적 ProbeBuilding 과 동적 계측은 다음과 같은 차이가 있습니다.

- 동적 ProbeBuilding (see page 89)은 사용자가 PBD 파일에 대해 직접 변경한 사항과 IntroscopeAgent.profile 에서 직접 구성한 사항을 기반으로 합니다. PBD 파일을 업데이트하거나 변경한 후 올바른 위치에 저장하면 동적 ProbeBuilding 이 변경 내용을 구현합니다. 동적 ProbeBuilding 을 사용하려면 IntroscopeAgent.profile 을 구성하고 업데이트할 PBD 파일을 변경해야 합니다. 모든 변경 내용은 파일을 다시 수동으로 업데이트하거나 변경하지 않는 한 영구적입니다.

- 동적 계측은 Workstation 트랜잭션 추적 뷰어에서 수행됩니다. 사용자가 인터페이스를 사용하여 선택한 계측에 대한 변경 작업은 자동으로 수행되며 일시적으로만 적용되는 경우가 많습니다. 메서드를 동적으로 계측하는 것은 런타임 동안 계측을 삽입하는 것을 의미합니다. 트랜잭션 추적 세션 중 하나 이상의 메서드 또는 모든 메서드를 동적으로 계측할 수 있습니다. 그런 다음 새로 계측된 메서드에서 반환되는 메트릭을 볼 수 있습니다. 이 메서드는 동적 응용 프로그램 성능 튜닝을 가능하게 합니다. 동적 계측은 IntroscopeAgent.profile 에 대한 변경이 필요 없습니다. 영구적으로 계측 변경 내용을 반영하도록 지정하는 경우 올바른 위치에 PBD 가 생성 및 저장됩니다.

  참고: Workstation 트랜잭션 추적 뷰어에서 동적 계측을 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

  중요! CA Introscope®는 동적 계측 캐시를 응용 프로그램 서버 호스트 컴퓨터에 저장합니다. 동적 계측을 사용하려면 응용 프로그램 서버가 Enterprise Manager 에 액세스하는 데 사용하는 사용자 계정에 응용 프로그램 서버 호스트 컴퓨터의 *<Agent_Home>* 및 *<Agent_Home>*/logs 디렉터리에 대한 쓰기 권한이 있어야 합니다.

ProbeBuilding 클래스 계층

1.5 이전의 JVM 의 경우 CA Introscope 는94 클래스 계층에서 더 깊은 수준에 있는 클래스를 자동으로 계측하지 않습니다. CA Introscope 는94 프로브된 클래스를 명시적으로 확장하는 클래스만 계측합니다.

지원되는 JVM 을 사용할 때 프로브된 클래스의 여러 하위 클래스 수준을 계측하도록 CA Introscope 를94 구성할 수 있습니다. 이 제품은 연결된 내부 지시문에서 추적 프로그램 그룹을 적절히 업데이트하고 클래스를 동적으로 계측합니다. 지시문 변경 내용은 로그 파일에 기록됩니다.

참고: 지원되는 JVM 에 대한 자세한 내용은 *호환성 안내서*를 참조하십시오.

PBD 를 수동으로 업데이트하는 것을 선호하는 경우, 지시문 업데이트를 비활성화하고 로그 파일을 사용하여 적절한 업데이트를 결정할 수 있습니다.

추가 정보:

계측 및 상속 (페이지 136)

여러 수준의 서브클래스에 대한 계측 사용

내부 지시문을 동적으로 업데이트하도록 Introscope 를 구성하려면 다음 단계를 따르십시오.

여러 수준의 서브클래스에 대한 계측을 사용하도록 설정하려면

1. 동적 ProbeBuilding (see page 89)에 설명된 대로 동적 계측을 사용하도록 설정되어 있는지 확인합니다.
2. *IntroscopeAgent.profile* 을 엽니다.
3. 여러 수준의 서브클래스에 대한 계측을 사용하도록 설정하려면 다음 속성 설정의 주석 처리를 제거합니다.
   `introscope.autoprobe.hierarchysupport.enabled=true`
4. *IntroscopeAgent.profile* 을 저장합니다.

여러 개의 상속, 인터페이스 및 추상 메서드에 대한 지원

Java Agent 에서는 인터페이스별 및 상속별 계측을 지원합니다. 이 기능은 동적 계측으로 확장됩니다.

Java Agent 는 API getMethodCalls 를 사용하여 하위 클래스를 호출함으로써 메서드의 계측을 지원합니다. getMethodCalls 를 사용하면 다음과 같은 정보가 제공되므로 상속된 메서드 또는 인터페이스 메서드의 계측 결과를 보다 잘 이해할 수 있습니다.

- 메서드를 정의하는 클래스가 인터페이스인지 여부
- 가능한 메서드 계측의 영향을 받는 클래스 수. 이 수는 하위 클래스의 수이거나 구현하는 클래스의 수입니다.
- 메서드가 특정 스택 추적 내에서 호출되는지 여부

다음 구문을 사용하여 Java Agent 에서 인터페이스와 추상 메서드를 계측하는 추적 프로그램을 사용할 수 있습니다.

```
TraceOneMethodWithlabelIfInherits: GHOST <method> <Label> <Tracer Group> <Tracer Type> <Resource>
```

이 추적 프로그램은 다음과 같은 경우에서 인터페이스를 구현하는 클래스 또는 상위 클래스를 확장하는 클래스의 메서드를 계측합니다.

- 메서드가 인터페이스에 정의된 경우
- 메서드가 상위 클래스에서 추상 메서드인 경우

중요! 이 추적 프로그램을 사용할 경우 시스템의 성능에 상당한 영향을 줄 수 있습니다. 규모가 큰 에이전트 구성에 배포하기 전에 시스템 시작 및 계측 프로세스 중에 이 추적 프로그램의 영향을 테스트하십시오.

추가 정보:

사용자 지정 추적 프로그램 생성 (페이지 125)

계측되지 않는 서브클래스에 대한 주기적 폴링 구성

여러 수준 서브클래스 계측이 사용되는 경우 Introscope 에서는 응용 프로그램 시작 시 계측되지 않는 서브클래스를 확인합니다.

계측되지 않는 서브클래스를 폴링하도록 Introscope 를 구성하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. 다음 속성 설정의 주석 처리를 제거합니다.
   ```
   introscope.autoprobe.hierarchysupport.runOnceOnly=false
   ```
3. Introscope 가 계측되지 않는 서브클래스를 폴링하는 빈도를 기본값인 5 에서 다른 값으로 변경하려면 이 속성의 주석 처리를 제거하고 값을 원하는 폴링 빈도로 설정합니다.
   ```
   introscope.autoprobe.hierarchysupport.pollIntervalMinutes
   ```
4. 필요한 경우 다음 속성의 주석 처리를 제거하고 값을 원하는 한도로 설정하여 Introscope 가 계측되지 않는 서브클래스를 폴링하는 횟수를 제한할 수 있습니다.
   ```
   introscope.autoprobe.hierarchysupport.executionCount
   ```
   이 속성의 기본값은 3 분입니다.
5. *IntroscopeAgent.profile* 을 저장합니다.

지시문 업데이트 비활성화

다중 수준 하위 클래스 계층이 활성화된 경우, Introscope 가 계측되지 않은 하위 클래스를 감지하면 기본적으로 클래스가 계측되도록 내부 지시문을 적절히 업데이트합니다. PBD 를 수동으로 업데이트하는 것을 선호하는 경우 *IntroscopeAgent.profile* 에서 이 속성의 주석 처리를 제거하여 내부 지시문 업데이트를 비활성화할 수 있습니다.

```
introscope.autoprobe.hierarchysupport.disableDirectivesChange=true
```

지시문 로깅 제어

여러 수준 서브클래스 계측이 사용되는 경우 여러 수준 서브클래스 계측 로그가 생성되도록 하려면 *IntroscopeAgent.profile* 에서 다음 속성의 주석 처리를 제거해야 합니다. 이러한 속성이 구성되어 있으면 pbdupdate.log 라는 로그 파일이 <*Agent_Home*>/wily 디렉터리(기본값)나 사용자 지정 위치(지정한 경우)에 생성됩니다. 여러 수준 계측에 대한 세부 정보가 이 에이전트 로그에 기록됩니다.

```
log4j.additivity.IntroscopeAgent.inheritance=false
log4j.logger.IntroscopeAgent.inheritance=INFO,pbdlog
log4j.appender.pbdlog.File=pbdupdate.log
log4j.appender.pbdlog=com.wily.introscope.agent.AutoNamingRollingFileAppender
log4j.appender.pbdlog.layout=com.wily.org.apache.log4j.PatternLayout
log4j.appender.pbdlog.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z} [%-3p]
[%c] %m%n_
```

이러한 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

바이트 코드의 행 번호 제거

응용 프로그램 바이트 코드를 계측할 때 바이트 코드 행 번호는 기본적으로 유지됩니다. 바이트 코드 행 번호 정보를 유지하면 디버거를 사용하거나 스택 추적 정보를 가져올 때 유용합니다.

Java 명령줄에서 시스템 속성을 추가하여 이 기능을 해제할 수 있습니다. 이 기능을 해제하면 AutoProbe 또는 ProbeBuilder 가 응용 프로그램 코드를 계측할 때 모든 행 번호가 제거됩니다.

AutoProbe 또는 ProbeBuilder 를 사용할 때 바이트 코드의 행 번호를 제거하려면

- Java 명령줄에서 **-D** 옵션을 사용하여 다음 시스템 속성을 정의합니다.

  com.wily.probebuilder.removeLineNumbers=true

제 5 장: ProbeBuilder 지시문

이 단원에서는 ProbeBuilder 지시문을 생성하고 수정하는 방법에 대해 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

ProbeBuilder 지시문 개요

Introscope ProbeBuilder 는 PBD(ProbeBuilder 지시문) 파일을 통해 응용 프로그램을 계측하기 위한 타이머 및 카운터 같은 프로브를 추가하는 방법을 알 수 있습니다. PBD 파일은 에이전트가 Introscope Enterprise Manager 에 보고하는 메트릭을 제어합니다.

참고: 모든 메트릭은 시스템 클록에 의해 설정된 시간을 사용하여 계산됩니다. 트랜잭션 도중 시스템 클록이 재설정되면 해당 트랜잭션에 대해 보고된 경과 시간이 잘못될 수 있습니다.

Introscope 에는 일련의 기본 PBD 파일이 포함되어 있습니다. 응용 프로그램에 대한 특정 정보를 얻기 위해 사용자 지정 Introscope PBD 파일을 생성하여 클래스 또는 메서드를 추적할 수도 있습니다.

ProbeBuilder 지시문을 지정하는 파일 종류에는 다음 두 가지가 있습니다.

- **PBD(ProbeBuilder 지시문) 파일**

  PBD(ProbeBuilder 지시문) 파일에는 ProbeBuilder 가 응용 프로그램을 계측하는 데 사용하는 지시문이 포함되어 있습니다. 이 파일은 에이전트가 Enterprise Manager 에 보고할 메트릭을 결정합니다.

- **PBL(ProbeBuilder 목록) 파일**

  PBL(ProbeBuilder 목록) 파일에는 여러 PBD 파일 이름의 목록이 포함되어 있습니다. 서로 다른 PBL 파일이 동일한 PBD 파일을 참조할 수 있습니다.

중요! PBD 와 PBL 은 ASCII 문자만 지원합니다. PBDs 또는 PBL 에 유니코드 문자 등의 다른 문자를 사용하면 AutoProbe 에 문제가 발생할 수 있습니다.

Java Agent 를 설치하고 Introscope AutoProbe 를 사용하려고 하면 특정 응용 프로그램 서버에 대한 관련 PBD 및 PBL 파일이 포함됩니다. 이러한 파일은 <*Agent_Home*>\wily\core\config 디렉터리에 있습니다.

추가 정보:

기본 PBD 파일 (페이지 101)
기본 PBL 파일 (페이지 106)
사용자 지정 추적 프로그램 생성 (페이지 125)

기본 PBD 에 의해 추적되는 구성 요소

기본 Introscope PBD 파일은 다음과 같은 Java 구성 요소의 추적 기능을 구현합니다.

- Oracle JDBC
- JSP 태그 라이브러리
- JSP IO 태그 라이브러리
- JSP DB 태그 라이브러리
- Struts
- 서블릿
- JSF(JavaServer Faces)
- JSP(JavaServer Pages)
- EJB(Enterprise JavaBeans)
- JDBC(Java Database Connectivity)

- 네트워크 소켓
- RMI(원격 메서드 호출)
- XML(Extensible Markup Language)
- JTA(Java Transaction API)
- JNDI(Java Naming and Directory Interface)
- JMS(Java Message Service)
- CORBA(Common Object Request Broker Architecture)
- UDP(사용자 데이터그램 프로토콜)
- 파일 시스템
- 스레드
- 시스템 로그
- throw 및 catch 된 예외(기본적으로 해제됨)

일부 경우에 PBD(ProbeBuilder 지시문) 파일에서 모니터링 대상으로 선택된 Java 클래스가 너무 많으면 Java Agent 가 올바르지 않게 시작되거나 "중단"될 수 있습니다. 이러한 경우 *AutoProbe.log* 파일을 사용하여 Java Agent 중단의 원인이 된 클래스를 확인하고 문제의 원인이 된 클래스를 건너뛰는 건너뛰기 지시문을 PBD 파일에 추가하십시오. 건너뛰기 지시문을 추가하는 방법에 대한 자세한 내용은 건너뛰기 지시문 (see page 89)을 참조하십시오.

기본 PBD 파일

Java Agent 에는 다음과 같은 기본 PBD 파일이 포함되어 있습니다.

appmap.pbd

응용 프로그램 심사 맵 계측을 위한 추적 프로그램 지시문을 제공합니다.

appmap-ejb.pbd

응용 프로그램 심사 맵 EJB 계측을 위한 추적 프로그램 지시문을 제공합니다.

appmap-soa.pbd

SPM 지원 Java SOAP 스택에 대한 응용 프로그램 심사 맵 SOA 계측을 위한 추적 프로그램 지시문을 제공합니다.

참고: 자세한 내용은 *CA APM for SOA 구현 안내서*를 참조하십시오.

bizrecording.pbd

추적 프로그램 정의와 에이전트 비즈니스 기록을 설정하기 위한 지시문을 제공합니다.

biz-trx-http.pbd

비즈니스 중심 HTTP 계측을 위한 추적 프로그램 지시문을 제공합니다.

di.pbd

Apache Derby 구현 클래스를 계측하지 않기 위한 지시문 ProbeBuilder 를 제공합니다.

errors.pbd

심각한 오류를 야기하는 코드 수준 이벤트를 지정하여 Error Detector 를 구성합니다. 기본적으로 프런트엔드 및 백엔드 오류만 심각한 오류로 간주됩니다. 즉, 사용자에게 표시되는 오류 페이지로 나타나는 오류나 백엔드 시스템(ADO.NET, Messaging 등)과 관련된 문제를 나타내는 오류만이 이러한 오류에 해당합니다.

j2ee.pbd

일반적인 Java Enterprise Edition 구성 요소에 대한 추적 프로그램 그룹을 제공합니다. 특정 추적 기능을 설정(TurnOn)하려면 toggles-full.pbd 또는 toggles-typical.pbd 를 사용합니다.

java2.pbd

일반적인 Java 2 구성 요소에 대한 추적 프로그램 그룹을 제공합니다. 특정 추적 기능을 *설정(TurnOn)*하려면 *toggles-full.pbd* 또는 *toggles-typical.pbd* 를 사용합니다.

jsf.pbd

일반적인 JSF(Java Server Face) 구성 요소에 대한 추적 프로그램 그룹을 제공합니다.

jsf-toggles-full.pbd

jsf.pbd 에서 제공되는 추적 기능에 대해 *TurnOn* 지시문 형태의 on/off 스위치를 제공합니다. 대부분의 추적 프로그램 그룹은 설정되어 있습니다.

jsf-toggles-typical.pbd

jsf.pbd 에서 제공되는 추적 기능에 대해 *TurnOn* 지시문 형태의 on/off 스위치를 제공합니다.

jvm.pbd

다양한 Java VM 에 대한 지원을 구현하는 지시문을 제공합니다. 이 속성은 Introscope 기본 파일과 함께 사용합니다.

leakhunter.pbd

누수 감지 유틸리티인 CA APM LeakHunter 에 대한 계측 설정을 제공합니다. 일반적으로 이 파일의 내용은 수정하지 않습니다.

lisa.pbd

CA APM 통합을 위한 CA LISA 계측에 사용되는 추적 프로그램 지시문을 제공합니다.

oraclejdbc.pbd

Oracle JDBC 구성 요소에 대한 추적 프로그램 그룹을 제공합니다. 추적되는 Oracle JDBC 구성 요소 집합을 변경하려면 *TurnOn* 지시문을 주석으로 처리하거나 주석 처리를 제거하십시오.

ServletHeaderDecorator.pbd

CA CEM 과의 통합에 포함된 서블릿 헤더 데코레이터를 사용하도록 설정합니다.

smwebagentext.pbd

SiteMinder 웹 에이전트 Introscope 플러그 인에 대한 추적 프로그램을 제공합니다.

soaagent.pbd

CA SOA Security Manager(웹 서버 및 응용 프로그램 서버용 SOA 에이전트)의 일부인 TransactionMinder 에이전트에 대한 추적 프로그램을 제공합니다.

spm-correlation.pbd

구성 요소 간의 트랜잭션 추적 상관 관계를 제어하는 지시문을 제공합니다. CA APM for SOA 를 사용하는 경우 크로스 프로세스 트랜잭션 추적 기능을 사용하려면 이 파일이 필요합니다.

struts.pbd

Apache struts 를 모니터링하는 지시문을 제공합니다. 이 속성은 Introscope 기본 파일과 함께 사용합니다.

summary-metrics-6.1.pbd

7.0 이전 버전의 Introscope 인스턴스에서 JSP 추적, 서블릿 추적 및 EJB 추적에 필요한 지시문을 제공합니다.

taglibs.pbd

JSP 태그 라이브러리, Jakarta I/O 라이브러리 및 DGTags 태그 라이브러리로 추적해야 하는 클래스를 모니터링하는 지시문을 제공합니다.

TIBCO pbd

Java Agent 는 SOA 확장을 통해 TIBCO 를 모니터링하는 데 관련된 몇 개의 PBD 와 함께 설치됩니다.

참고: 자세한 내용은 *CA APM for SOA 구현 안내서*를 참조하십시오.

toggles-full.pbd

다른 지시문 파일에서 제공되는 추적 기능에 대해 *TurnOn* 지시문 형태의 on/off 스위치를 제공합니다. 대부분의 추적 프로그램 그룹은 설정되어 있습니다.

toggles-typical.pbd

다른 지시문 파일에서 제공되는 추적 기능에 대해 *TurnOn* 지시문 형태의 on/off 스위치를 제공합니다. 추적 프로그램 그룹의 일부만이 설정되어 있습니다.

webMethods pbd

Java Agent 는 CA APM for webMethods Broker 를 통해 webMethods 를 모니터링하는 데 관련된 몇 개의 PBD 와 함께 설치됩니다.

참고: 자세한 내용은 *CA APM for SOA 구현 안내서*를 참조하십시오.

WebSphere MQ pbd

Java Agent 는 CA APM for IBM WebSphere MQ 를 통해 WebSphere MQ 커넥터 및 메시징 시스템을 모니터링하는 데 관련된 몇 개의 PBD 와 함께 설치됩니다.

참고: 자세한 내용은 *CA APM for IBM WebSphere MQ 안내서*를 참조하십시오.

또한 Java Agent 는 모니터링할 응용 프로그램 서버에 따라 달라지는 응용 프로그램 서버별 PBD 도 설치합니다.

추가 정보:

기본 추적 프로그램 그룹 및 toggles 파일 (페이지 106)
추적 프로그램 그룹 설정 또는 해제 (페이지 117)

이전 릴리스의 기본 PBD 파일

에이전트에서는 기본적으로 현재 릴리스의 PBD 및 PBL 파일을 사용합니다. 그러나 제품에서는 <*Agent_Home*>/wily/examples/legacy 디렉터리에 이전 릴리스의 PBD 및 PBL 파일을 제공합니다. 이 디렉터리의 각 파일 이름에는 -legacy 접미사가 사용됩니다(예: default-full-legacy.pbl).

기본 PBL 파일

각 에이전트와 함께 사용할 수 있는 PBL 파일에는 다음 두 가지가 있습니다.

default-full.pbl(기본값)

대부분의 추적 프로그램 그룹이 설정된 PBD 파일을 참조합니다. Introscope 는 기본적으로 이 집합을 사용하여 전체 Introscope 기능을 보여 줍니다.

default-typical.pbl

참조되는 PBD 파일의 추적 프로그램 그룹 중 일부가 설정되어 있습니다. 이 표준 집합에는 일반적인 설정이 포함되어 있으며 특정 환경에 맞게 이 집합을 사용자 지정할 수 있습니다.

또한 Java Agent 는 모니터링할 응용 프로그램 서버에 따라 달라지는 응용 프로그램 서버별 PBL 도 설치합니다.

기본 추적 프로그램 그룹 및 toggles 파일

추적 프로그램 그룹은 PBD 파일에 정의되어 있습니다. 추적 프로그램 그룹은 일련의 클래스에 대한 정보가 보고되도록 합니다. PBD 파일에서 *flag* 라는 용어는 추적 프로그램 그룹 정보를 나타냅니다. 예를 들어 TraceOneMethodIfFlagged 또는 SetFlag 는 추적 프로그램 그룹 정보를 정의합니다.

추적 프로그램 그룹은 일련의 클래스에 적용되는 일련의 추적 프로그램으로 구성됩니다. 예를 들어 모든 RMI 클래스의 응답 시간 및 속도를 보고하는 추적 프로그램 그룹이 있습니다.

일부 추적 프로그램 그룹을 설정하거나 해제하여 시스템에 대한 메트릭 수집을 미세 조정할 수 있습니다. 이 메서드는 추적 프로그램 그룹의 구성 방식에 따라 오버헤드 사용이 늘어나거나 줄어드는 영향이 있습니다.

추적 프로그램 그룹은 기본 default-full.pbl 및 default-typical.pbl 파일이 참조하는 toggles-full.pbd 및 toggles-typical.pbd 파일에서 수정합니다. 다음 목록에는 기본 추적 프로그램 그룹과 해당 기본 구성이 나열되어 있습니다.

AgentInitialization

에이전트 초기화 구성

전체: 설정

표준: 설정

ApacheStandardSessionTracing

HTTP 세션 구성

전체: 설정

표준: 설정

AuthenticationTracing

인증 구성

전체: 설정

표준: 설정

CorbaTracing

CORBA 메서드 호출

전체: 설정

표준: 설정

DBCPTracing

DBCP 구성

전체: 설정

표준: 설정

DBCPv55Tracing

DBCP 구성

전체: 설정

표준: 설정

EJB2StubTracing

EJB 2.0 구성

전체: 설정

표준: 설정

EJB3StubTracing

EJB 3.0 구성

전체: 설정

표준: 설정

EntityBean3Tracing

엔터티 EJB 3.0 메서드 호출

전체: 설정

표준: 설정

EntityBeanTracing

엔터티 EJB 메서드 호출

전체: 설정

표준: 설정

HTTPServletTracing

HTTP 서블릿 서비스 응답

전체: 설정

표준: 설정

Application Server AutoProbe 를 사용하는 경우에는 추적 프로그램 그룹 HTTPAppServerAutoProbeServletTracing 을 설정하십시오.

InstanceCounts

추적 프로그램 그룹에서 식별된 개체 유형의 인스턴스 수를 카운트합니다.

전체: 설정

표준: 설정

클래스가 이 추적 프로그램 그룹을 사용하여 식별되기 전까지는 어떤 항목도 추적되지 않습니다.

J2eeConnectorTracing

J2EE 커넥터 정보

전체: 설정

표준: 설정

JavaMailTransportTracing

메일 전송 시간

전체: 설정

표준: 설정

JDBCQueryTracing

JDBC 쿼리

전체: 설정

표준: 설정

JDBCUpdateTracing

JDBC 업데이트

전체: 설정

표준: 설정

JMSConsumerTracing

JMS 메시지 처리 시간

전체: 설정

표준: 설정

JMSListenerTracing

JMS 메시지 처리 시간

전체: 설정

표준: 설정

JMSPublisherTracing

JMS 메시지 브로드캐스트 시간

전체: 설정

표준: 설정

JMSSenderTracing

JMS 메시지 브로드캐스트 시간

전체: 설정

표준: 설정

JSPTracing

JSP 서비스 응답

전체: 설정

표준: 설정

MessageDrivenBean3Tracing

메시지 구동 EJB 3.0 메서도 호출

전체: 설정

표준: 설정

MessageDrivenBeanTracing

메시지 구동 EJB 메서도 호출

전체: 설정

표준: 설정

NIOSocketTracing

NIO|채널|소켓 노드 아래에 각 소켓 연결에 대한 메트릭 집합을 생성하고 "백엔드" 노드 아래에 추가 메트릭을 생성합니다.

전체: 설정

표준: 설정

NIOSocketSummaryTracing

모든 NIO 소켓 연결을 포함하는 단일 메트릭 집합을 생성합니다.

전체: 설정

표준: 설정

이러한 메트릭은 에이전트 속성에 의해 NIOSocketTracing 메트릭에서 제외된 연결을 포함합니다. 이러한 메트릭은 또한 NIOSocketTracing 메트릭에서 항상 제외되는 일부 내부 NIO 소켓도 포함합니다.

NIOSelectorTracing

NIO 채널의 특정 내부 JVM 사용이 NIO 채널 메트릭에서 카운트되지 않도록 방지합니다.

사용자는 이 옵션을 제어할 수 없습니다.

NIODatagramTracing

각 데이터그램 "연결"에 대한 메트릭 집합을 생성합니다.

전체: 설정

표준: 설정

자세한 내용은 269 페이지의 NIODatagramTracing 메트릭을 참조하십시오.

NIODatagramSummaryTracing

모든 NIO 데이터그램 동작을 측정하는 단일 메트릭 집합을 생성합니다.

전체: 설정

표준: 설정

이러한 메트릭은 에이전트 속성에 의해 NIODatagramTracing 메트릭에서 제외된 연결을 포함합니다. 이러한 메트릭은 또한 NIODatagramTracing 메트릭에서 항상 제외되는 일부 내부 NIO 소켓도 포함합니다.

PersistentSessionTracing

HTTP 세션 구성

전체: 설정

표준: 설정

RMIClientTracing

RMI 클라이언트 메서드 호출

전체: 설정

표준: 설정

RMIServerTracing

RMI 서버 메서드 호출

전체: 설정

표준: 설정

ServerInfoTracing

서버 정보 구성

전체: 설정

표준: 설정

SessionBean3Tracing

세션 EJB 3.0 메서드 호출

전체: 설정

표준: 설정

SessionBeanTracing

세션 EJB 메서드 호출

전체: 설정

표준: 설정

SocketTracing

네트워크 소켓 대역폭 및 SSL 추적

전체: 설정

표준: 설정

StrutsTracing

Struts 프레임워크에서의 작업 실행 시간

전체: 설정

표준: 설정

SuperpagesSessionTracing

HTTP 세션 구성

전체: 설정

표준: 설정

ThreadPoolTracing

스레드 풀 구성

전체: 설정

표준: 설정

UDPTracing

UDP(사용자 데이터그램 프로토콜) 소켓 대역폭

전체: 설정

표준: 설정

UnformattedSessionTracing

HTTP 세션 구성

전체: 설정

표준: 설정

EJB3MethodLevelTracing

메서드 수준에서의 EJB 3.0 동작

전체: 설정

표준: 해제

EJBMethodLevelTracing

메서드 수준에서의 EJB 동작

전체: 설정

표준: 해제

FileSystemTracing

쓰고 읽은 파일 시스템 바이트

전체: 설정

표준: 해제

JAXMListenerTracing

JAXM 메시지 전송

전체: 설정

표준: 해제

JNDITracing

JNDI 조회 시간

전체: 설정

표준: 해제

JSPDBTagsTagLibraryTracing

SQL 데이터베이스에서 읽고 쓰기 위한 Jakarta DB Tags 사용자 지정 태그 라이브러리

전체: 설정

표준: 해제

JSPIOTagLibraryTracing

다양한 입력 및 출력 작업을 위한 Jakarta IO 사용자 지정 태그 라이브러리

전체: 설정

표준: 해제

JTACommitTracing

JTA 를 사용한 커밋 시간

전체: 설정

표준: 해제

ThreadTracing

클래스별 활성 스레드 수

전체: 설정

표준: 해제

XMLSAXTracing

XML 문서를 구문 분석하는 데 소요된 시간

전체: 설정

표준: 해제

XSLTTracing

XML 변환 시간

전체: 설정

표준: 해제

CatchException

예외 구성

전체: 해제

표준: 해제

FormattedSessionTracing

HTTP 세션 구성

전체: 해제

표준: 해제

HTTPAppServerAutoProbeServletTracing

HTTP 서블릿 구성

전체: 해제

표준: 해제

HTTPSessionTracing

HTTP 세션 구성

전체: 해제

표준: 해제

JSPTagLibraryTracing

사용자 지정 JSP 태그의 처리 시간

전체: 해제

표준: 해제

ManagedSocketTracing

네트워크 구성

전체: 해제

표준: 해제

ThrowException

예외 구성

전체: 해제

표준: 해제

일반적으로 기본 *toggles* PBD 파일을 편집하지 마십시오. 그러나 일부 추적 프로그램 그룹을 설정하거나 해제하여 메트릭 수집을 미세 조정할 수는 있습니다. *toggles* 파일에서 다음을 수행하여 추적 프로그램 그룹을 수정할 수 있습니다.

- 시스템 오버헤드를 줄일 수 있도록 추적 프로그램 그룹 설정/해제
- 추적 프로그램 그룹에 클래스 추가

추적 프로그램 그룹은 설정된 상태(주석 처리가 제거된 상태)에서만 정보를 보고하며 *TurnOn* 키워드를 사용하여 활성화됩니다.

참고: Java Agent 는 EJB(2.0 이상) 계측을 지원합니다. 메트릭 수집을 조정하려면 EJB 에 연결된 추적 프로그램 그룹을 켜거나 끄십시오. 응용 프로그램 심사 맵의 EJB 지원은 세션 및 엔터티 Bean 까지만 확장됩니다. 메시지 구동 Bean 은 지원되지 않습니다.

추가 메트릭 정보를 수집하도록 설정 전환

다음 토글은 켜면 활성화된 CA Technologies 제공 추적 프로그램 그룹에 대한 모든 API 에서 추가 메트릭을 수집합니다. 구성을 변경하려면 전체 또는 일반 토글 파일에 이러한 토글을 추가해야 합니다.

DefaultStalledMethod 추적

중단된 메서드 추적

전체: 설정

표준: 설정

DefaultConcurrentInvocationTracing

동시 호출 정보

전체: 설정

표준: 해제

DefaultRateMetrics

호출 비율 메트릭

전체: 해제

표준: 해제

추적 프로그램 그룹 설정 또는 해제

일부 추적 프로그램 그룹을 설정하거나 해제하여 시스템에 대한 메트릭 수집을 미세 조정할 수 있습니다.

추적 프로그램 그룹을 설정하려면

1. AutoProbe 또는 Java Agent 에서 사용 중인 파일 유형(*<appserver>-full.pbl* 또는 *<appserver>-typical.pbl*)에 따라 toggles-full.pbd 또는 toggles-typical.pbd 파일을 찾습니다. 이러한 파일은 *<appserver home>*wily/core/config 디렉터리나 *<Introscope_Home>*/core/config/systempbd 디렉터리에 있습니다.
2. 추적 프로그램 그룹을 찾아 설정하고, 행의 맨 앞에 있는 파운드 기호를 제거하여 해당 행의 주석 처리를 제거합니다. 다음 예의 지시문은 설정된 상태이므로 모든 HTTP 서블릿이 추적됩니다.

   ```
   TurnOn: HTTPServletTracing
   ```

   참고: 추적 프로그램 그룹에 대해 주석 처리가 제거된(설정된) 지시문은 해당 추적 프로그램 그룹이 사용되도록 합니다.

추적 프로그램 그룹을 해제하려면

- 다음 예와 같이 추적 프로그램 그룹의 행 맨 앞에 파운드 기호를 추가하여 해당 추적 프로그램 그룹을 주석으로 처리합니다.

  ```
  #TurnOn: HTTPServletTracing
  ```

추적 프로그램 그룹에 클래스 추가

기존 추적 프로그램 그룹에 클래스를 추가하여 특정 클래스에 대한 추적을 설정할 수 있습니다. 클래스를 추적 프로그램 그룹의 일부로 식별하려면 식별 키워드 중 하나를 사용하십시오.

예를 들어 *com.myCo.ejbentity.myEJB1* 클래스를 추적 프로그램 그룹 *EntityBeanTracing* 에 추가하려면 다음을 사용하십시오.

```
IdentifyClassAs: com.myCo.ejbentity.myEJB1 EntityBeanTracing
```

식별 키워드는 다음과 같습니다.

- IdentifyInheritedAs
- IdentifyClassAs
- IdentifyCorbaAs

EJB 서브클래스 추적

기본적으로 엔터티 및 세션 EJB 관련 지시문은 엔터티, 세션 또는 메시지 구동 EJB 인터페이스를 직접적이고 명시적으로 구현하는 EJB 에 대한 프로브만 추가합니다.

응용 프로그램의 EJB 는 엔터티 또는 세션 EJB 인터페이스를 직접적이고 명시적으로 구현하는 클래스의 서브클래스인 경우도 있습니다. 이러한 EJB 는 기본적으로 Introscope 에서 추적되지 않습니다.

EJB 서브클래스를 Introscope 에서 추적하려면 해당 EJB 서브클래스를 적절한 추적 프로그램 그룹에 추가해야 합니다. 이렇게 하려면 추적할 EJB 서브클래스의 직접 상위 항목을 참조하는 항목을 추가하십시오.

다음 모델에서 *<entity.bean.ancestor.class>* 또는 *<session.bean.ancestor.class>*는 계측할 EJB 의 간접 상위 항목에 대한 정규화된 클래스 이름으로 바꾸십시오.

엔터티 EJB 의 경우:

```
IdentifyInheritedAs: <entity.bean.ancestor.class> EntityBeanTracing
```

세션 EJB 의 경우:

```
IdentifyInheritedAs: <session.bean.ancestor.class> SessionBeanTracing
```

아래 예는 다음과 같은 클래스 계층 구조를 기반으로 합니다.

```
mySessionEJB implements javax.ejb.SessionBean
    mySessionEJBsubclass1 extends mySessionEJB
```

mySessionEJBsubclass1a extends mySessionEJBsubclass1

mySessionEJBsubclass1b extends mySessionEJBsubclass1

```
    mySessionEJBsubclass2 extends mySessionEJB
```

추적 프로그램 그룹 *SessionBeanTracing* 은 *mySessionEJB* 가 추적되도록 합니다.

다음 추적 프로그램은 *mySessionEJBsubclass1* 및 *mySessionEJBsubclass2* 를 추적합니다.

IdentifyInheritedAs: mySessionEJB SessionBeanTracing

다음 추적 프로그램은 *mySessionEJBsubclass1a* 및 *mySessionEJBsubclass1b* 를 추적합니다.

IdentifyInheritedAs: mySessionEJBsubclass1 SessionBeanTracing

참고: 이 예에서는 패키지를 사용하지 않습니다. 코드가 패키지에 있는 경우에는 코드에 클래스 이름과 함께 패키지 이름을 포함해야 합니다.

EJB 3.0 주석

다음 지시문을 사용하면 특정 클래스 수준 주석이 포함된 클래스를 추적 프로그램 그룹으로 그룹화할 수 있습니다. 이 지시문은 EJB 3.0 을 지원합니다. 3.0 사양을 따르는 EJB 는 잘 알려진 인터페이스를 명시적으로 구현하지는 않지만, 대신 주석을 통해 완전히 활성화됩니다. EJB 3.0 클래스를 쉽게 식별하려면 다음 지시문을 사용하십시오.

```
IdentifyAnnotatedClass <annotation-name> <flag-name>
```

이 지시문을 사용하려면 새 지시문에 대한 지시문 클래스와 지시문 파서 클래스를 생성하십시오. 그런 다음 바이트 코드를 검사하여 클래스에 특정 주석이 포함되어 있는지 여부를 확인하기 위한 matcher 클래스를 추가해야 합니다.

참고: 이 지시문은 메서드 수준 주석을 지원하지 않습니다.

응용 프로그램 심사 맵에 대한 EJB 지원

CA Introscope®는 (특히 Workstation 응용 프로그램 심사 맵에서의 사용에 대해) EJB(2.0 이상) 세션 및 엔터티 Bean 의 기본 추적을 지원합니다. 이 구성은 에이전트의 시작 시간에 영향을 주므로 CA Technologies 는 이 기본 기능을 테스트 환경에서만 사용할 것을 권장합니다.

프로덕션 환경에 이 기능을 배포하는 경우 특정 사항에 대해 EJB 추적 프로그램을 구성하십시오. 기본 기능은 너무 광범위할 수 있습니다.

ProbeBuilder 가 상위 클래스 또는 인터페이스로부터 상속하거나 이들을 구현하는 클래스에 플래그를 지정하도록 ProbeBuilder 에게 지시하려면 다음 지시문을 사용하십시오.

```
IdentifyDeepInheritedAs
```

EJB 2.0 응용 프로그램 심사 맵 지원을 위해 다음 지시문이 j2ee.pbd 파일에 있습니다.

```
IdentifyDeepInheritedAs: javax.ejb.EJBObject EJB2StubTracing
IdentifyDeepInheritedAs: javax.ejb.SessionBean SessionBeanTracing
IdentifyDeepInheritedAs: javax.ejb.EntityBean EntityBeanTracing
IdentifyDeepInheritedAs: javax.ejb.MessageBean MessageBeanTracing
```

이러한 지시문은 ProbeBuilder 가 클라이언트 측에서 EJB 스텁을 식별하고 서버 측에서 응용 프로그램 심사 맵에서 사용되는 Bean 을 식별할 수 있게 해 줍니다.

EJB 3.0 응용 프로그램 심사 맵 지원을 위해 다음 지시문이 j2ee.pbd 파일에 있습니다.

```
IdentifyInheritedAnnotatedClassAs
```

지시문은 직접, 또는 상위 인터페이스를 통해 인터페이스를 구현하는 모든 클래스와 일치합니다.

응용 프로그램 심사 맵의 컨텍스트에서 다음 추가 지시문이 j2ee.pbd 에 설정되어 있습니다.

```
IdentifyInheritedAnnotatedClassAs: javax.ejb.Remote EJB3StubTracing
```

EJB 이름 지정

EJB 를 다루는 호출되는 백엔드, 일반 프런트엔드 및 모니터링되는 구성 요소에 이름을 지정할 수 있습니다. 이름 포맷터를 사용하면 EJB(2.0 이상) 클라이언트 스텁 및 Bean 구현에 적절한 이름을 구성할 수 있습니다.

EjbNameFormatter 클래스는 EJB 관련 메트릭 이름, 응용 프로그램 심사 맵 응용 프로그램 이름 또는 노드 이름을 정의합니다. 다음과 같은 자리 표시자를 사용합니다.

- EJB 클라이언트 스텁의 경우: *{classname}*, *{interface}* 및 *{method}*
- EJB Bean 의 경우: *{classname}*, *{bean}*, *{interface}* 및 *{method}*

다음 메트릭 이름이 기본적으로 사용됩니다.

- EJB Bean 프런트엔드: *EJB|{interface}*
- EJB 클라이언트 스텁 백엔드: *EJB|{interface}*
- EJB Bean 의 응용 프로그램 심사 맵 소유자 이름: *{interface}*
- EJB 클라이언트 스텁: *Client {interface}*
- EJB Bean 의 응용 프로그램 심사 맵 노드 이름: *Server {interface}*

이러한 이름은 기본 EJB 이름 포맷터입니다. 이러한 이름은 j2ee.pbd 및 appmap-ejb.pbd 파일에서 사용됩니다. 동일한 이름 포맷터를 사용하지만 다른 메트릭 이름을 사용할 수 있습니다. 예를 들어 보다 적절한 이름을 사용하도록 기존 추적 프로그램 지시문을 수정하면서 플래그는 동일하게 유지할 수 있습니다.

```
...
# Default commented out:
#TraceComplexMethodsIfFlagged: EJB2StubTracing EJB2BackendTracer "{interface}"
#Add the EJB application name to backend marker as well as called method
TraceComplexMethodsIfFlagged: EJB2StubTracing EJB2BackendTracer
"MyCustomerBeanApp-{interface}-{method}"
...
SetTracerClassMapping: EJB2BackendTracer
com.wily.introscope.agent.trace.BackendTracer
com.wily.introscope.probebuilder.validate.ResourceNameValidator
SetTracerParameter: EJB2BackendTracer nameformatter
com.wily.introscope.agent.trace.ejb.Ejb2StubNameFormatter
```

참고: EJB 컨텍스트 추적 프로그램은 EJB 2.0 Bean 의 setContext() 메서드에서 설정됩니다. 이 추적 프로그램은 EJB 2.0 Bean 이름 포맷터에 대한 내부 CA Introscope® 추적 프로그램으로, 이름 포맷터가 올바르게 작동할 수 있도록 합니다.

IntroscopeAgent.profile, PBL 및 PBD 를 함께 사용

Java Agent 를 처음 설치할 때는 계측 수준(전체 또는 표준)을 설정합니다. 이 설정은 PBL(ProbeBuilder 목록) 파일의 *default-typical.pbl* 및 *default-full.pbl* 을 나타냅니다(자세한 내용은 기본 PBL 파일 (see page 106) 참조).

ProbeBuilder 지시문 적용

PBD 를 적용하는 방식은 사용하도록 선택한 메서드에 따라 다릅니다. CA Technologies 는 JVM AutoProbe 를 사용하여 PBD 를 구현할 것을 권장합니다. ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 를 사용하여 PBD 를 구현할 수도 있습니다.

JVM AutoProbe 사용

PBD 파일을 구현할 준비가 되었으면 이 파일을 *hotdeploy* 디렉터리에 추가하십시오. AutoProbe 는 *IntroscopeAgent.profile* 파일이 들어 있는 디렉터리(기본적으로 *<Agent_Home>*/wily/core/config 디렉터리)와 *<Agent_Home>*/wily/core/config/hotdeploy 디렉터리에서 PBD 파일을 찾습니다. AutoProbe 는 이러한 디렉터리를 기준으로 파일 이름을 확인합니다. *wily* 디렉터리를 이동한 경우에는 파일 경로를 올바른 디렉터리로 매핑해야 합니다.

AutoProbe 를 사용하여 PBD 를 구현하려면

1. 수정한 표준 PBD 또는 PBL 을 *<Agent_Home>*/wily/core/config 디렉터리에 저장합니다
2. 사용자 지정 PBD *여러 개*를 *<Agent_Home>*/wily/core/config/hotdeploy 디렉터리에 복사합니다. 이 디렉터리에 추가된 PBD 는 *IntroscopeAgent.profile* 에서 *introscope.autoprobe.directivesFile* 속성을 업데이트하거나 수정하지 않고도 구현됩니다.

   참고: 동적 계측을 사용하도록 설정한 경우 *hotdeploy* 디렉터리의 PBD 가 폴더에서 라이브로 선택됩니다. 재부팅은 필요하지 않습니다. 동적 계측에 대한 자세한 내용은 동적 ProbeBuilding (see page 89)을 참조하십시오.
3. *IntroscopeAgent.profile* 을 저장합니다.
4. 응용 프로그램을 다시 시작합니다.

ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 사용

PBD 파일을 구현할 준비가 되었으면 이 파일을 *hotdeploy* 디렉터리에 추가하십시오. 명령줄 ProbeBuilder 는 ProbeBuilder 가 실행된 디렉터리와 <*Agent_Home*>/wily/core/config/hotdeploy 디렉터리에서 사용자 지정 지시문 파일을 찾습니다. 명령줄 ProbeBuilder 는 이러한 디렉터리를 기준으로 파일 이름을 확인합니다.

ProbeBuilder 마법사나 명령줄 ProbeBuilder 를 사용하여 ProbeBuilder 지시문을 구현하는 단계는 JVM AutoProbe 를 사용할 때와 동일합니다. 자세한 내용은 JVM AutoProbe 사용 (see page 122)을 참조하십시오.

변경되거나 새로운 PBD 를 사용한 계측

변경되거나 새로운 지시문을 적용하려면 최신 PBD 를 사용하여 응용 프로그램이 계측되어야 합니다. 이 프로세스는 사용하는 ProbeBuilding 메서드에 따라 다릅니다.

-javaagent 를 통해 JVM AutoProbe 를 사용하는 JVM 1.5 시스템

응용 프로그램 또는 Java Agent 를 다시 시작하지 않고도 변경된 PBD 가 적용될 수 있도록 동적 계측을 구성할 수 있습니다. 이렇게 하면 PBD 수정을 수행하거나 응용 프로그램 서비스를 방해하지 않고 심사 기반 계측을 수행할 수 있습니다. 자세한 내용은 동적 ProbeBuilding (see page 89)을 참조하십시오.

신규 및 변경된 ProbeBuilder 파일

1.5 이전의 JVM 또는 –Xbootclasspath 를 사용한 설치에 해당

변경된 ProbeBuilder 지시문 파일 또는 ProbeBuilder 목록 파일은 응용 프로그램 서버가 응용 프로그램 클래스를 다음에 로드할 때 반영됩니다.

지시문을 추가 또는 변경할 때 관리되는 응용 프로그램이 실행되지 않으면 다음에 응용 프로그램을 시작할 때 업데이트된 지시문을 사용하여 응용 프로그램이 계측됩니다.

관리되는 응용 프로그램이 실행 중인 경우 관리되는 응용 프로그램 클래스를 로드 또는 다시 로드해야 합니다.

클래스가 다시 로드되도록 하는 것은 사용하는 응용 프로그램 서버에 따라 다릅니다. 대부분의 응용 프로그램 서버에서는 재시작이 필요합니다.

ProbeBuilder 마법사 사용

ProbeBuilder 마법사를 사용하려면

1. "사용자 지정 지시문" 화면에는 ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 사용 (see page 123)에서 설명한 대로 *hotdeploy* 디렉터리에 추가된 PBD 파일이 나열됩니다.
2. 사용할 사용자 지정 지시문 파일을 선택합니다.

명령줄 ProbeBuilder 사용

중요! 명령줄 ProbeBuilder 는 최신 PBD 를 Introscope 에서 사용할 수 있도록 하기 위한 마지막 옵션으로 사용하는 것이 좋습니다.

명령줄 ProbeBuilder 를 사용하려면

1. 관리되는 응용 프로그램을 중지합니다.
2. 명령줄에서 사용자 지정 PBD 및 PBL 파일을 지정하여 명령줄 ProbeBuilder 나 ProbeBuilder 마법사를 실행합니다.
3. 응용 프로그램에서 새 파일을 사용하도록 구성합니다.
4. 관리되는 응용 프로그램을 시작합니다.
5. 관리되는 응용 프로그램이 이미 실행 중인 경우에는 Enterprise Manager 와 Workstation 을 시작합니다.

사용자 지정 추적 프로그램 생성

사용자 지정 PBD 파일을 생성하여 메트릭 수집을 보다 세부적으로 조정할 수 있습니다. 응용 프로그램별 측정을 추적하는 추적 프로그램을 생성하여 사용자 지정 지시문을 생성하려면 특정 구문 및 키워드를 사용해야 합니다. 사용자 지정 추적 프로그램을 작성하려면 다음을 정의해야 합니다.

- 지시문 유형(일반적으로 추적할 클래스 또는 메서드 수를 나타냄)
- 추적할 특정 클래스 또는 메서드
- 클래스 또는 메서드에서 추적할 정보 유형(예, 시간, 속도 또는 개수)
- 이 정보를 나타낼 정규화된 메트릭 이름(리소스 경로 포함)

사용자 지정 PBD 는 *<Agent_Home>*/wily/core/config/hotdeploy 디렉터리에 저장됩니다. 이 디렉터리에 추가된 PBD 는 *IntroscopeAgent.profile* 에서 *introscope.autoprobe.directivesFile* 속성을 업데이트하거나 수정하지 않고도 구현됩니다.동적 계측을 사용하도록 설정한 경우 *hotdeploy* 디렉터리의 PBD 가 폴더에서 라이브로 선택됩니다. 재부팅은 필요하지 않습니다. 동적 계측에 대한 자세한 내용은 동적 ProbeBuilding (see page 89)을 참조하십시오.

사용자 지정 PBD 가 생성되었으면 Introscope 에서는 이를 기본 제공 PBD 인 것처럼 처리합니다. 생성된 메트릭에 대한 경고를 설정하거나, 이 메트릭을 SmartStor 에 저장하거나, Introscope Workstation 에서 사용자 지정 대시보드를 생성할 때 이 메트릭을 사용할 수 있습니다.

참고: 추적되는 메서드가 많을수록 오버헤드가 증가하므로 추적할 메서드는 신중하게 선택해야 합니다.

일반 메트릭에 대해 사용자 지정 BlamePointTracer 추적 프로그램 사용

BlamePointTracer 는 가장 많이 사용되는 추적 프로그램입니다. 이 추적 프로그램은 연결된 메서드나 클래스에 대해 다음과 같은 다섯 개의 개별 메트릭을 생성합니다.

- 평균 응답 시간(ms)
- 동시 호출
- 간격당 오류 수
- 간격당 응답 수
- 중단 수

다음은 *BlamePointTracer* 의 예제입니다. *BlamePointTracer* 가 *petshop.catalog.Catalog* 클래스에서 *search* 란 이름의 메서드에 대해 설정되었습니다. *PetShop|Catalog|search* 는 Introscope Investigator 에서 BlamePoint 메트릭이 표시될 노드의 이름입니다.

```
TraceOneMethodOfClass: petshop.catalog.Catalog search BlamePointTracer
"PetShop|Catalog|search"
```

추적 프로그램 구문에서 사용되는 지시문 이름 및 인수

PBD 파일에는 추적 프로그램을 그룹에 연결하거나 그룹이 사용 또는 사용되지 않도록 설정하는 단순 키워드 외에 추적 프로그램 정의도 포함됩니다. Introscope 가 추적 프로그램을 인식하고 처리하도록 하려면 사용자 지정 추적 프로그램을 구성할 때 특정 구문을 사용해야 합니다. 추적 프로그램은 지시문과 추적할 메서드 또는 클래스에 대한 정보로 구성되며 형식은 다음과 같습니다.

```
<directive>: [arguments]
```

여기서 *[arguments]*는 목록으로, 지시문에 따라 다릅니다.

참고: 사용되는 지시문에 따라 이러한 매개 변수 중 일부만이 필요합니다.

<directive>

가장 일반적으로 사용하는 지시문은 다음과 같은 추적 지시문입니다.

TraceOneMethodOfClass

지정된 클래스의 지정된 메서드를 추적합니다.

TraceAllMethodsOfClass

지정된 클래스의 모든 메서드를 추적합니다.

TraceOneMethodIfInherits

지정된 클래스나 인터페이스의 모든 직접 서브클래스 또는 직접 인터페이스 구현에서 메서드 하나를 추적합니다.

TraceAllMethodsIfInherits

지정된 클래스나 인터페이스의 모든 직접 서브클래스 또는 직접 인터페이스 구현에 포함된 모든 메서드를 추적합니다.

TraceOneMethodIfFlagged

지정된 클래스가 *TurnOn* 키워드로 사용하도록 설정된 추적 프로그램 그룹에 포함된 경우 메서드 하나를 추적합니다.

TraceAllMethodsIfFlagged

지정된 클래스가 *TurnOn* 키워드로 사용하도록 설정된 추적 프로그램 그룹에 포함된 경우 모든 메서드를 추적합니다.

참고: 구체적으로 구현된 메서드만이 추적 가능하며 실행 중에 메트릭 데이터를 보고할 수 있습니다. 사용자 지정 추적 프로그램에 추상 메서드를 지정하면 메트릭 데이터가 보고되지 않습니다.

추적 지시문에 필요한 구문은 일반적으로 다음 인수로 구성됩니다.

<Tracer-Group>

추적 프로그램을 연결할 그룹입니다.

GHOST

추적할 클래스 또는 인터페이스의 정규화된 이름입니다. 정규화된 클래스 이름에는 다음과 같이 클래스 이름뿐 아니라 클래스의 전체 어셈블리 이름도 포함됩니다.

[MyAssembly]com.mycompany.myassembly.MyClass

어셈블리 이름은 대괄호([])로 묶어야 합니다.

<method>

메서드 이름(예: *MyMethod*)입니다.

또는

반환 유형 및 매개 변수가 포함된 전체 메서드 서명(예: *myMethod;[mscorlib]System.Void([mscorlib] System.Int32)*입니다. 메서드 서명에 대한 자세한 내용은 서명 구별 (see page 131)을 참조하십시오.

<Tracer-name>

사용할 추적 프로그램 유형을 지정합니다. 예를 들어 *BlamePointTracer* 입니다. 추적 프로그램 이름에 대한 설명은 아래의 추적 프로그램 이름 표를 참조하십시오.

<metric-name>

수집된 데이터가 Introscope Workstation 에 표시되는 방식을 제어합니다.

다음 예에서는 메트릭 트리의 서로 다른 수준에서 메트릭 이름 및 위치를 지정하는 세 가지 방법에 대해 설명합니다.

metric-name - 메트릭이 에이전트 노드 내에 바로 표시됩니다.

resource:metric-name - 메트릭이 에이전트 노드에서 한 수준 아래의 리소스(폴더) 내에 표시됩니다.

resource|sub-resource|sub-sub-resource:metric-name - 메트릭이 에이전트 노드에서 두 수준 이상 아래의 리소스(폴더)에 표시됩니다. 파이프 문자(|)를 사용하여 리소스를 구분합니다.

일반적으로 사용되는 추적 프로그램 이름 및 예제

다음 목록에서는 가장 일반적으로 사용되는 추적 프로그램 이름과 추적 프로그램이 추적하는 항목을 설명합니다.

BlamePointTracer

Blame 관련 구성 요소의 평균 응답 시간, 간격당 수, 동시성, 중단 및 오류를 포함하는 표준 메트릭 집합을 제공합니다.

ConcurrentInvocationCounter

시작되었지만 아직 완료되지 않은 메서드의 횟수를 보고합니다. 결과는 Investigator 트리에서 추적 프로그램에 지정된 메트릭 이름 *<metric-name>* 아래에 보고됩니다. 예를 들어 이 추적 프로그램을 사용하여 동시 데이터베이스 쿼리의 수를 셀 수 있습니다.

DumpStackTraceTracer

이 추적 프로그램이 적용되는 메서드에 대해 계측된 응용 프로그램의 표준 오류로 스택 추적을 덤프합니다. 덤프 스택 추적 프로그램에서 throw 되는 예외 스택 추적은 실제 예외가 아니라 메서드 스택 추적을 출력하기 위한 메커니즘입니다.

이 기능은 메서드의 호출 경로를 확인하는 데 유용합니다.

중요! 이 기능은 과다한 시스템 오버헤드를 발생시킵니다. 따라서 이 추적 프로그램은 급격한 오버헤드 증가가 허용되는 진단 컨텍스트에서만 사용하는 것이 좋습니다.

MethodCPUTimer

메서드 실행 중에 사용되는 평균 CPU 시간(밀리초)으로, 메트릭 트리의 *<metricname>* 아래에 보고됩니다.

참고: 이 추적 프로그램을 사용하려면 지원되는 플랫폼의 플랫폼 모니터가 필요합니다.

MethodTimer

평균 메서드 실행 시간(밀리초)으로, 메트릭 트리에서 추적 프로그램에 지정된 메트릭 이름 *<metric-name>* 아래에 보고됩니다.

PerIntervalCounter

간격당 호출 수입니다. 이 간격은 데이터 소비자(예: Investigator 의 뷰 창)의 표시 기간에 따라 변경됩니다. 결과는 Investigator 트리에서 추적 프로그램에 지정된 메트릭 이름 *<metric-name>* 아래에 보고됩니다.

사용자 지정 추적 프로그램 정의에 따옴표 사용

사용자 지정 추적 프로그램에는 공백이 있는 메트릭 이름이 포함될 수 있습니다. 사용자 지정 메트릭 이름에 공백을 사용할 경우에는 모든 메트릭 이름을 따옴표("")로 묶는 것이 좋습니다.

중요! 클래스 이름은 따옴표로 묶지 마십시오. 사용자 지정 추적 프로그램이 오작동하는 원인이 될 수 있습니다. 예:

올바른 이름

```
IdentifyClassAs: MyClass MyTracers
```

올바르지 않은 이름

```
IdentifyClassAs: "MyClass" MyTracers
```

클래스 이름이 포함된 메트릭 이름을 생성하는 경우에는 전체 메트릭 이름을 따옴표로 묶어야 합니다. 메트릭 이름에 공백을 사용할 수 있으며, 이 경우 메트릭 이름의 모든 공백은 따옴표 내에 포함되어야 합니다. 예를 들어 메트릭 이름 "{classname}|Test One Node"는 다음과 같이 나타내야 합니다.

올바른 이름

```
TraceOneMethodIfFlagged: MyTracers AMethod BlamePointTracer "{classname}|Test One
Node"
```

올바르지 않은 이름

```
TraceOneMethodIfFlagged: MyTracers AMethod BlamePointTracer {classname}|Test One Node
```

중요! Introscope 에서는 다음과 같이 클래스 파일 이름이 올바르지 않은 클래스는 모니터링하지 않습니다.

```
org/jboss/seam/example/seambay/AuctionImage$JaxbAccessorM_getData_setData_[B:
```

*_[B:*로 인해 클래스 파일 이름이 올바르지 않습니다. Java 클래스 파일 이름의 일부로 여는 대괄호([)를 사용할 수 없습니다. 이와 같이 클래스 이름이 올바르지 않은 클래스가 있을 경우 Introscope 는 해당 클래스를 계측하지 못하며 이를 에이전트 로그에 오류 메시지로 보고합니다.

다음 단원에서는 메서드 추적 프로그램의 예를 보여 줍니다. 다음 예에서는 메트릭 이름을 따옴표("")로 묶었습니다. 사용자 지정 메트릭 이름을 생성할 때는 모든 메트릭 이름을 따옴표로 묶는 것이 좋습니다.

평균 추적 프로그램 예제

이 추적 프로그램은 지정된 메서드의 평균 실행 시간을 밀리초 단위로 추적합니다.

```
TraceOneMethodOfClass: com.sun.petstore.catalog.Catalog search BlamedMethodTimer "Petstore|Catalog|search:Average Method Invocation Time (ms)"
```

속도 추적 프로그램 예제

이 추적 프로그램은 초당 메서드 호출 횟수를 계산하여 지정된 메트릭 이름 아래에 이 속도를 보고합니다.

```
TraceOneMethodOfClass: com.sun.petstore.catalog.Catalog search BlamedMethodRateTracer "Petstore|Catalog|search:Method Invocations Per Second"
```

간격당 카운터 추적 프로그램 예제

이 메서드 추적 프로그램은 메서드의 간격당 호출 수를 계산하고 이 간격당 수를 지정된 메트릭 이름으로 보고합니다.

```
TraceOneMethodOfClass: com.sun.petstore.catalog.Catalog search PerIntervalCounter "Petstore|Catalog|search:Method Invocations Per Interval"
```

이 간격은 그래프 빈도와 같은 Enterprise Manager 의 모니터링 로직에 의해 결정됩니다.

Investigator 의 미리 보기 창은 기본적으로 15 초 간격입니다.

카운터 추적 프로그램 예제

이 추적 프로그램은 총 메서드 호출 횟수를 계산합니다.

```
TraceOneMethodOfClass: com.sun.petstore.cart.ShoppingCart placeOrder
BlamedMethodTraceIncrementor "Petstore|ShoppingCart|placeOrder:Total Order Count"
```

결합된 카운터 추적 프로그램 예제

이러한 추적 프로그램은 수를 계산하기 위해 incrementor 및 decrementor 추적 프로그램을 결합합니다.

```
TraceOneMethodOfClass: com.sun.petstore.account.LoginEJB login
MethodTraceIncrementor "Petstore|Account:Logged In Users"
TraceOneMethodOfClass: com.sun.petstore.account.LogoutEJB logout
MethodTraceDecrementor "Petstore|Account:Logged In Users"
```

고급 단일 메트릭 추적 프로그램

지시문 및 추적 프로그램은 메서드, 클래스 및 클래스 집합을 추적합니다. Introscope 가 추적할 수 있는 최소 단위는 메서드이고, 단일 메트릭 추적 프로그램은 특정 메서드에 대한 특정 메트릭을 보고합니다. 메서드 서명을 사용하거나, 키워드를 대체하거나, 메트릭 이름 매개 변수를 조작하는 등 여러 방법으로 단일 메트릭 추적 프로그램을 만들 수 있습니다.

서명 구별

메서드 서명을 기준으로 메서드에 추적 프로그램을 적용할 수 있습니다.

특정 서명으로 메서드의 단일 인스턴스를 추적하려면 내부 메서드 설명자 형식을 사용하여 지정된 메서드 이름에 서명을 추가합니다(반환 유형 포함).

예를 들어, *myMethod(Ljava/lang/String;)V* 는 문자열 인수와 void 반환 형식이 있는 메서드의 인스턴스를 추적합니다.

이 형식에 대한 자세한 내용은 *Sun Java Virtual Machine 설명서*를 참조하십시오.

메트릭 이름 키워드 기반 대체

키워드 기반 대체를 사용하면 런타임에 값을 메트릭 이름으로 대체할 수 있습니다.

추적 프로그램에서 메트릭 이름의 매개 변수가 런타임에 실제 값으로 대체되어 메트릭 이름이 됩니다. 이 기능은 모든 지시문에서 사용할 수 있습니다.

{method}

추적할 메서드 이름

{classname}

추적할 클래스의 런타임 클래스 이름

{packagename}

추적 중인 클래스의 런타임 패키지 이름

{packageandclassname}

추적 중인 클래스의 런타임 패키지 및 클래스 이름

참고: Introscope 가 패키지가 없는 클래스를 처리하는 경우 {packagename}를 문자열 "<Unnamed Package>"로 대체합니다.

키워드 기반 대체: 예제 1

pbd 파일의 추적 프로그램에 대한 메트릭 이름이 다음과 같고:

"{packagename}|{classname}|{method}:Response Time (ms)"

그리고 추적 프로그램이 패키지 myPackage 에 있는 myClass 의 런타임 클래스가 있는 메서드 myMethod 에 적용되는 경우, 메트릭 이름은 다음과 같습니다:

"myPackage|myClass|myMethod:Response Time (ms)"

키워드 기반 대체: 예제 2

.pbd 파일에서 다음과 같은 메트릭 이름을 갖는 추적 프로그램이 있다고 가정합니다.

"{packageandclassname}|{method}:Response Time (ms)"

이 추적 프로그램이 동일한 메서드에 적용된 경우 결과 메트릭 이름은 다음과 같이 됩니다.

"myPackage.myClass|myMethod:Response Time(ms)"

참고: 패키지와 클래스 사이에 첫 번째 예의 | 대신 .(마침표)가 사용되었습니다.

메트릭 이름 기반 매개 변수

다음 형식으로 *TraceOneMethodWithParametersOfClass* 키워드를 사용하여 메서드에 전달된 매개 변수를 기반으로 메트릭 이름을 생성하는 단일 메서드 추적 프로그램을 생성할 수 있습니다.

TraceOneMethodWithParametersOfClass: <class-name> <method> <tracer-name> <metric-name>

메트릭 이름에 매개 변수를 사용할 수 있습니다. 이렇게 하려면 메트릭 이름의 자리 표시자 문자열을 매개 변수 값으로 대체합니다. 사용할 자리 표시자 문자열은 "{#}"입니다. 여기서 #은 대체할 매개 변수의 인덱스입니다. 인덱스는 0 부터 시작됩니다. 대체 매개 변수는 원하는 수만큼 원하는 순서대로 사용할 수 있습니다. 모든 매개 변수는 메트릭 이름으로 대체되기 전에 문자열로 변환됩니다. 문자열 이외의 개체 매개 변수는 toString() 메서드를 사용하여 변환되므로 주의해서 사용해야 합니다.

중요! 매개 변수가 어떤 문자열로 변환될지 확실히 알 수 없는 경우에는 매개 변수를 메트릭 이름에 사용하면 안 됩니다.

메트릭 이름 기반 예제

웹 사이트에서는 order 라는 클래스와 process 라는 메서드를 사용합니다. 이 메서드에는 서로 다른 종류의 주문(책 또는 음악)에 대한 매개 변수가 있습니다.

다음과 같은 추적 프로그램을 생성할 수 있습니다.

```
TraceOneMethodWithParametersOfClass: order process(LJava/lang/string;)V
MethodTimer "Order|{0}Order:Average Response Time (ms)"
```

이 추적 프로그램은 다음과 같은 메트릭을 생성합니다.

Order

BookOrder

- 평균 응답 시간(ms)

MusicOrder

- 평균 응답 시간(ms)

TraceOneMethodWithParametersIfInherits 키워드를 사용할 수도 있습니다.

건너뛰기 지시문

AutoProbe 또는 ProbeBuilder 에서는 건너뛰기 지시문을 사용하여 일부 패키지, 클래스 또는 메서드를 건너뛸 수 있습니다. 기본적으로 AutoProbe 또는 ProbeBuilder 에서는 Java Agent 와 기본적인 Java 클래스 및 패키지를 건너뜁니다.

계산 개체 인스턴스

InstanceCounts 추적 프로그램 그룹은 관련된 특정 개체 유형의 인스턴스 수를 계산합니다(표준 IdentifyClassAs 및 IdentifyInheritedAs 지시문을 사용하여 InstanceCounts 추적 프로그램 그룹과 개체 유형을 연계하는 방법에 대한 자세한 내용은 추적 프로그램 그룹에 클래스 추가 (see page 117) 참조). 코드에 명시적으로 할당된 모든 인스턴스가 계산됩니다. 하위 유형 또한 계산됩니다. deserialization 또는 cloning 과 같은 다른 메커니즘을 통해 생성된 개체는 계산되지 않을 수 있습니다. 이 추적 프로그램 그룹을 추적하면 전적으로 계산된 인스턴스 수에 따라 잠재적으로 성능(또는 메모리)에 대한 영향을 증가시킬 수 있습니다.

참고: CA Technologies 의 테스트에 따르면 인스턴스의 수가 매우 큰 경우에만 성능에 실질적인 영향이 있습니다.

InstrumentPoint 지시문 켜기

키워드 *InstrumentPoint* 에 의해 식별되는 지시문에는 두 가지 유형이 있습니다. 하나는 예외를 추적하고, 다른 하나는 응용 프로그램 시작할 때(첫 번째 프로브가 실행될 때가 아닌) 에이전트 초기화시킵니다.

예외

다음 지시문은 throw 또는 catch 된 예외의 추적을 활성화하는 데 사용됩니다. 성능에 영향을 줄 수 있으므로 기본적으로는 활성화되어 있지 않습니다. 이 둘 중 하나를 활성화하려면 적절한 줄의 주석 처리를 제거하십시오.

```
#InstrumentPoint: ThrowException
#InstrumentPoint: CatchException
```

에이전트 초기화

에이전트 초기화 계측 지점 지시문은 추가적인 오버헤드를 발생시키지 않으며, 전체 또는 표준 PBD 집합 모두에서 기본적으로 활성화되어 있습니다.

```
InstrumentPoint: AgentInitialization
```

여러 ProbeBuilder 지시문 파일이 사용되는 경우 각 파일에서 활성화되어 있는 각 설정(예: 추적 프로그램 그룹, Skip, InstrumentPoint, Custom Method Tracer)이 적용됩니다.

사용자 지정 추적 프로그램 결합

동일한 메트릭에 적용되는 여러 추적 프로그램을 사용하여 결합 효과를 낼 수 있습니다. 이러한 결합은 대개 incrementor 및 decrementor 추적 프로그램과 함께 사용됩니다.

이 예제는 *Total Purchases* 란 이름의 메트릭을 생성합니다. 클래스 *cart* 와 메서드 buyBook 및 buyCD 를 사용하여 다음 추적 프로그램을 생성하십시오.

```
TraceOneMethodOfClass cart buyBook PerIntervalCounter "Total Purchases"
TraceOneMethodOfClass cart buyCD PerIntervalCounter "Total Purchases"
```

이렇게 하면 누군가 상품을 구매할 때 메트릭 *Total Purchases* 가 증가합니다.

계측 및 상속

1.5 이전의 JVM 에 해당

CA Introscope®는 1.5 이전 버전의 JVM 에서 클래스 계층의 더 깊은 수준에 있는 클래스는 자동으로 계측하지 않습니다. 두 수준 이상 깊이에 있는 프로브된 클래스의 하위 클래스가 로드될 경우 새 메서드와 재정의된 메서드는 자동으로 계측되지 않습니다. 구현될 때 프로브된 인터페이스를 명시적으로 명명하는 클래스는 인터페이스를 간접적으로 구현하는 경우라도 계측됩니다.

예를 들어 다음과 같이 ClassB 가 ClassA 를 확장하고 ClassC 가 ClassB 를 확장하는 클래스 계층 구조가 있다고 가정합니다.

```
Interface/ClassA
    ClassB
        ClassC
```

ClassA 를 계측하면 ClassB 는 ClassA 를 명시적으로 확장하므로 함께 계측됩니다. 그러나 ClassC 는 ClassA 를 명시적으로 확장하지 않으므로 CA Introscope®는 ClassC 를 계측하지 않습니다. ClassC 를 계측하려면 ClassC 를 명시적으로 식별해야 합니다.

1.5 이전의 Java 환경에서 하위 클래스가 계측되었는지 확인하려면 EJB 하위 클래스 추적 (see page 118) 지침을 따르십시오.

JVM 1.5 를 사용하는 경우 프로브된 클래스 (see page 94)의 여러 하위 클래스 수준을 계측하도록 CA Introscope®를 구성할 수 있습니다.

Java 주석

CA Introscope®는 사용자 지정 메트릭을 만들 때 Java 1.6 주석의 사용을 허용합니다.

- **참고:** Java 주석에 대한 자세한 내용은 Java 개발자 웹 사이트의 설명서를 참조하십시오.

클래스의 메서드를 계측하려면 IdentifyAnnotatedClassAs 를 사용하여 클래스를 추적 프로그램 그룹에 배치한 다음 TraceAllMethodsIfFlagged 지시문을 사용하십시오. 예:

```
SetFlag: AnnotationTracing TurnOn: AnnotationTracing
IdentifyAnnotatedClassAs: com.test.MyAnnotation AnnotationTracing
TraceAllMethodsIfFlagged: AnnotationTracing BlamePointTracer
"Target|MyTarget|{classname}"
```

이 예제에서 com.test.MyAnnotation 은 주석 이름입니다. 사용자 고유의 주석을 생성하려면 코드에 용어를 사용하십시오. 주석 이름이 들어 있는 클래스는 식별됩니다.

Blame 추적 프로그램을 사용하여 Blame 지점 표시

CA Introscope®용 Blame 기술은 관리되는 Java 응용 프로그램에서 작동하여 응용 프로그램 계층(응용 프로그램의 프런트엔드 및 백엔드)에서 메트릭을 볼 수 있게 해 줍니다. 경계 Blame 이라고 하는 이 기능을 통해 사용자는 응용 프로그램 프런트엔드 또는 백엔드에 대한 문제를 심사할 수 있습니다. 이 정보는 Workstation 응용 프로그램 심사 맵에서 응용 프로그램의 가장자리를 표시하는 데에도 사용됩니다.

CA Introscope®에서 프런트엔드 및 백엔드를 확인하는 방법과 프런트엔드에 대한 메트릭이 집계되는 방식을 제어하도록 URL 그룹을 구성하는 옵션에 대한 자세한 내용은 경계 Blame 구성 (see page 189)을 참조하십시오.

다음 단원에서는 추적 프로그램을 사용하여 응용 프로그램의 프런트엔드 및 백엔드를 명시적으로 표시하는 방법에 대해 설명합니다.

Blame 추적 프로그램

Introscope 는 프런트엔드 및 백엔드 메트릭을 캡처하기 위한 추적 프로그램인 FrontendMarker 및 BackendMarker 를 제공합니다. 이러한 추적 프로그램은 각각 프런트엔드와 백엔드를 명시적으로 표시합니다.

FrontendMarker 와 BackendMarker 를 사용하여 백엔드에 액세스하는 코드와 같은 사용자 고유의 코드를 계측하면 Introscope 가 사용자 지정 구성 요소에 대한 메트릭을 캡처하여 Investigator 트리에 나타낼 수 있습니다.

FrontendMarker 추적 프로그램(또는 해당 서브클래스 HttpServletTracer 및 PageInfoTracer)으로 구성 요소가 계측되지 않으면 프런트엔드 메트릭이 생성되지 않으며 구성 요소가 트랜잭션의 프런트엔드로 표시되지 않습니다.

FrontendMarker 추적 프로그램(또는 해당 서브클래스)으로 트랜잭션 내의 구성 요소가 두 개 이상 계측될 경우 첫 번째로 지정된 구성 요소만 프런트엔드 메트릭을 생성합니다.

참고: 프런트엔드 추적 프로그램을 사용하는 경우 프런트엔드 추적 프로그램에서 지정된 응용 프로그램 이름은 응용 프로그램 심사 맵 추적 프로그램에 대해 지정된 이름과 일치해야 하며 두 이름은 모두 대/소문자를 구분합니다. 예를 들어 프런트엔드 추적 프로그램의 이름을 **AppOne** 으로 지정한 경우 응용 프로그램 심사 맵 추적 프로그램에서 이 추적 프로그램을 **APPONE** 으로 참조하면, AppOne 에 대한 정보가 Workstation 응용 프로그램 심사 맵에 올바르게 표시되지 않습니다.

특정 클래스가 프런트엔드로 표시되지 않도록 하려면 PBD 매개 변수 **is.frontend.unless** 를 지정합니다. PBD 지시문 *is.frontend.unless* 에 대한 자세한 내용은 사용자 지정 FrontendMarker 지시문 (see page 139)을 참조하십시오.

BackendMarker 가 구성되어 있지 않은 경우 Introscope 는 백엔드를 유추합니다. 즉, 명시적으로 표시되지 않은 경우 클라이언트 소켓을 여는 모든 구성 요소가 기본 백엔드가 됩니다.

다음과 같은 경우 BackendMarker 를 사용하면 유용합니다.

- Introscope 가 백엔드로 감지하는 항목에 원하는 이름을 지정하려는 경우
- Introscope 가 계측하지 않는 사용자 지정 Java 소켓을 표시하려는 경우
- JNI(Java Native Interface)를 통해 호출되는 기본 소켓에 대해 Java/JNI 브리지 메서드를 백엔드로 식별하려는 경우

FrontendMarker 와 BackendMarker 는 Blame 관련 구성 요소의 평균 응답 시간, 간격당 수, 동시성, 중단 및 오류와 같은 메트릭을 제공하는 BlamePointTracer 의 인스턴스입니다. BlamePointTracer 는 보다 세분화된 Blame 스택을 위해 중간 구성 요소에 적용할 수 있습니다.

깊게 중첩된 프런트엔드 트랜잭션의 높은 에이전트 CPU 오버헤드

서블릿은 Introscope 에 의해 프런트엔드로 표시되도록 구성됩니다. 일반적인 트랜잭션은 서블릿으로 시작하며, 이때 서블릿은 EJB 를 호출할 수 있고 EJB 는 백엔드를 호출합니다. 서블릿이 중첩된 방식으로 다른 서블릿을 호출하는 것은 가능하며, Introscope 는 이 경우 중첩된 프런트엔드로 간주합니다. 대부분의 경우 이것은 에이전트 CPU 오버헤드를 증가시키지 않습니다.

하지만 깊게 중첩된(예를 들어 40 개 수집 깊이) 프런트엔드 수준이 있는 트랜잭션의 경우 높은 CPU 오버헤드를 발생시킬 수 있습니다. 예를 들어, 트랜잭션에서 한 서블릿이 자신을 반복적으로 호출하거나(지속적 되풀이 호출) 다른 여러 서블릿을 호출하는 경우 에이전트 CPU 오버헤드가 증가할 수 있습니다. 이러한 오버헤드가 지나치게 높다고 판단되는 경우 CA Support 에 문의하십시오.

사용자 지정 FrontendMarker 지시문

PBD 매개 변수 is.frontend.unless 는 일부 클래스가 프런트엔드 구성 요소로 표시되지 않도록 해 줍니다. FrontendMarker(또는 HttpServletTracer 와 같은 그 하위 클래스)는 이러한 클래스를 계측합니다. 이 매개 변수는 쉼표로 구분된 절대 클래스 이름 목록으로 설정하십시오. 이 매개 변수는 최초 구성 요소가 일반 발송자인 경우 유용할 수 있습니다. 그리고 이 발송자는 받은 요청 유형을 처리하는 더 구체적인 구성 요소로 이 요청을 전달합니다. 따라서 두 번째 구성 요소는 보다 적절한 프런트엔드 표식이 됩니다. 기본값은 빈 목록입니다. PBD 매개 변수는 동적이지 않습니다. 이 매개 변수의 값이 변경되는 경우 계측된 응용 프로그램 서버를 다시 시작해야 합니다.

중요! 클래스 이름은 공백이 아닌 쉼표로 구분하십시오. 공백을 사용하면 SetTracerParameter 지시문이 무효화됩니다.

추적 프로그램에서 계측되고 이 매개 변수가 적용되는 매개 변수 목록에 지정된 모든 클래스는 다음과 같은 특성을 갖습니다.

- 일부 클래스가 프런트엔드 구성 요소로 표시되지 않는 한 PBD 매개 변수 is.frontend.unless 는 사용되도록 설정됩니다.
- Investigator 의 "프런트엔드" 노드 아래에 메트릭을 생성하지 마십시오.

예를 들어 NotAFrontend 및 AnotherNonFrontend 클래스가 com.ABCCorp 패키지에서 프런트엔드로 처리되지 않도록 해야 할 수 있습니다. 이러한 클래스는 MyFrontendTracer 라는 이름의 FrontendMarker 로 계측됩니다. 다음 PDB 지시문을 사용하십시오.

```
SetTracerParameter: MyFrontendTracer is.frontend.unless
com.ABCCorp.NotAFrontend,com.ABCCorp.AnotherNonFrontend
```

표준 PBD 의 Blame 추적 프로그램

Introscope 에서 제공되는 *j2ee.pbd* 및 *sqlagent.pbd* 의 두 개의 표준 PBD 가 경계 Blame 추적 기능을 구현합니다.

- *j2ee.pbd* 의 *HttpServletTracer* 는 FrontendMarker 의 인스턴스입니다.
- *sqlagent.pbd* 의 *SQLBackendTracer* 는 BackendMarker 의 인스턴스입니다.

이전 버전의 Introscope 에서 사용된 다음의 Blame 추적 프로그램은 여전히 존재하지만 Introscope PBD 에서 일반적으로 사용되지 않습니다.

- BlamedMethodTimer
- BlamedMethodRateTracer
- BlamedMethodTraceIncrementor
- BlamedMethodTraceDecrementor

경계 Blame 및 Oracle 백엔드

현재 버전의 Introscope 에서 Oracle 데이터베이스는 소켓 연결을 기준으로 감지되지 않습니다. Oracle 백엔드를 자동으로 감지하려면 Introscope 에 대해 사용 가능한 SQL 에이전트가 있어야 합니다.

SQL 에이전트가 없을 때 Introscope 가 Oracle 백엔드를 감지할 수 있도록 하려면 oraclejdbc.pbd 에서 다음과 같이 수정하십시오.

oraclejdbc.pbd 의 아래 부분에서:

```
#Socket data from the Oracle driver reports too many metrics
SkipPackagePrefixForFlag: oracle.jdbc. SocketTracing
SkipPackagePrefixForFlag: oracle.net. SocketTracing
```

다음 예와 같이 건너뛰기를 주석 처리합니다.

```
#Socket data from the Oracle driver reports too many metrics
#SkipPackagePrefixForFlag: oracle.jdbc. SocketTracing
#SkipPackagePrefixForFlag: oracle.net. SocketTracing
```

참고: 자세한 내용은 http://ca.com/support에서 기술 자료 문서 *Disabling Database Name Formatting in 7.1(7.1 에서 데이터베이스 이름 형식 지정 비활성화)(KB 1240)*을 참조하십시오.

제 6 장: Java Agent 이름 지정

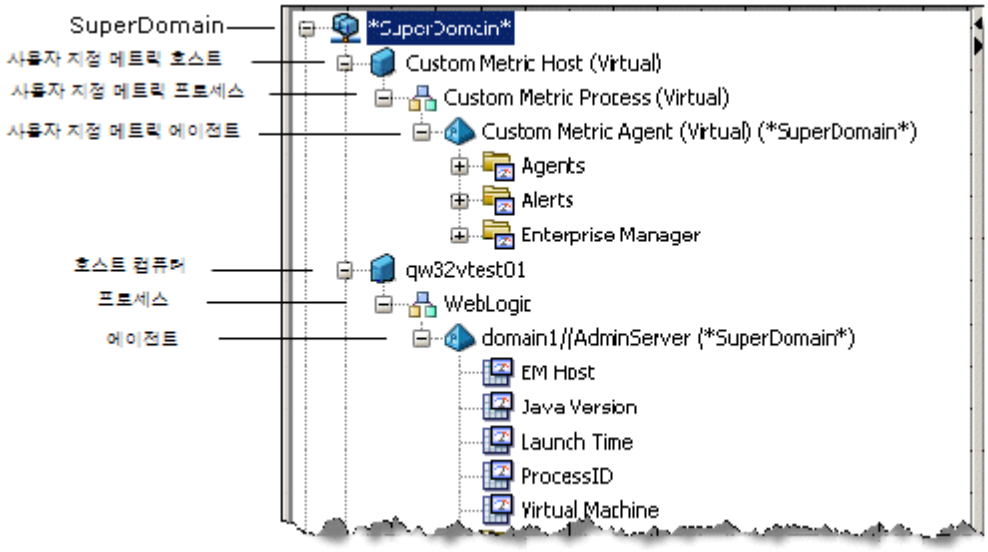
이 단원에서는 에이전트 이름 지정, 관련 환경 및 배포 고려 사항, 에이전트의 이름을 자동으로 지정하기 위한 옵션에 대한 정보를 제공합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

Java Agent 이름 이해

사용자가 명시적으로 이름을 할당했든, 이름을 자동으로 할당하는 메서드를 구성했든, 단순히 Java Agent 에서 모니터링하는 계측된 응용 프로그램을 시작했든 관계없이, Introscope 환경에서 실행되는 각 Java Agent 에는 이름이 있습니다. Java Agent 이름은 Introscope Workstation 과 Investigator 의 여러 뷰에서 중요한 요소이며, 대상 응용 프로그램과 모니터링 논리를 연결하는 프로세스에 핵심적인 역할을 합니다.

에이전트가 Enterprise Manager 에 메트릭을 보고하면 Investigator 트리에 해당 에이전트에 대한 노드가 생성됩니다. Workstation 에서 관리 논리(예: 대시보드, 경고 및 작업)를 구성할 경우 에이전트 이름은 관리 논리가 적용되는 응용 프로그램을 식별하는 정규식의 구성 요소입니다. 아래의 Investigator 트리에서는 *qw32vtest01* 호스트의 WebLogic 프로세스 아래에서 실행되는 *domain1//Adminserver* 라는 에이전트를 보여 줍니다.



에이전트가 이름을 파악하는 방법

Java Agent 는 다음과 같은 시퀀스를 사용하여 이름을 결정합니다.

1. Java Agent 가 첫 번째 방법을 사용하여 이름을 결정하는 경우 해당 이름을 사용하여 Enterprise Manager 에 연결합니다.
2. Java Agent 가 첫 번째 방법을 사용하여 이름을 결정하지 못한 경우 두 번째, 세 번째 방법을 차례로 시도합니다.
3. Java Agent 가 어떤 방법으로도 이름을 결정하지 못하는 경우 이름을 "UnnamedAgent"로 지정합니다.

방법 1: Java 시스템 디렉터리에 지정된 에이전트 이름

에이전트 이름은 명령줄에서 Java 시스템 속성을 사용하여 정의됩니다. 이 방법을 사용하여 사용하면 다른 모든 에이전트 이름 지정 방법이 무시됩니다. Java 시스템 속성을 사용하여 에이전트 이름 지정 (see page 147)을 참조하십시오.

방법 2: IntroscopeAgent.profile 의 시스템 속성 키에 지정된 에이전트 이름

에이전트 이름은 *IntroscopeAgent.profile* 의 속성에 지정된 Java 시스템 속성에서 가져옵니다. 시스템 속성 키를 사용하여 에이전트 이름 지정 (see page 147)을 참조하십시오.

방법 3: 응용 프로그램 서버에서 자동으로 가져오는 에이전트 이름

WebLogic 또는 WebSphere 의 특정 버전을 사용하는 경우 자동 에이전트 이름 지정 기능을 사용하여 응용 프로그램 서버로부터 에이전트 이름을 자동으로 가져올 수 있습니다. Enterprise Manager 에 연결하기 전에 이름을 결정하도록 에이전트에게 필요한 만큼의 시간을 주기 위해 시간 지연을 구성할 수 있습니다. 응용 프로그램 서버로부터 에이전트 이름 가져오기 (see page 148)를 참조하십시오.

방법 4: 에이전트 프로필에 명시적으로 지정된 에이전트 이름

에이전트 이름은 *IntroscopeAgent.profile* 프로필에서 *introscope.agent.agentName* 속성에 정의되어 있습니다. 이것은 이전 버전의 Introscope 에서 에이전트의 이름을 지정하는 표준 방법이었습니다. 모든 응용 프로그램에 대해 에이전트 프로필이 이미 있는 경우 이 옵션을 사용하십시오.

방법 5: "알 수 없는 에이전트"로 결정된 에이전트 이름

에이전트가 위에 나열된 방법 중 하나를 사용하여 이름을 결정할 수 없는 경우 에이전트는 자신의 이름을 "UnnamedAgent"로 지정합니다.

Introscope 가 에이전트 이름 지정 충돌을 해결하는 방법

정규화된 에이전트 이름(호스트 이름, 프로세스 이름, 에이전트 이름으로 구성)은 Introscope 환경의 각 에이전트에 대해 일반적으로 고유합니다. 동일한 에이전트 이름을 사용하는 서로 다른 에이전트는 호스트 이름과 프로세스 이름이 대개 다르므로 일반적으로 고유한 *정규화된* 에이전트 이름을 사용합니다. 여러 에이전트는 동일한 호스트에 존재하고, 동일한 프로세스를 모니터링하고, 동일한 에이전트 이름을 갖는 경우에만 동일한 정규화된 에이전트 이름을 갖습니다.

동일한 정규화된 이름을 갖는 에이전트가 이미 연결되어 있는 Enterprise Manager 에 에이전트가 연결을 시도하면 Enterprise Manager 는 새로 연결하는 에이전트의 이름에 고유 식별자를 추가합니다. 식별자는 퍼센트 문자(%)와 숫자로 구성됩니다. 이 메커니즘은 동일한 정규화된 이름을 사용하여 연결하는 여러 에이전트가 연결 중 고유하게 식별될 수 있도록 해 줍니다. Enterprise Manager 는 "%1"을 에이전트 이름에 추가하여 첫 번째 중복된 에이전트를 연결합니다.

예를 들어, 정규화된 에이전트 이름을 갖는 다음과 같은 두 개의 에이전트가 있다고 가정합니다.

```
hostPA|processNIM|PodAgent
```

이 에이전트들은 하나씩 Enterprise Manager 에 연결합니다. Enterprise Manager 는 두 번째 에이전트의 이름을 변경합니다.

```
PodAgent%1
```

동일한 정규화된 이름을 갖는 다른 에이전트가 연결하는 경우 *PodAgent%2*, *PodAgent%3*, *PodAgent%4* 등과 같이 연이어 이름이 변경됩니다. 여기서 퍼센트 문자 뒤의 숫자는 순서를 나타내는 숫자입니다.

이름이 변경된 에이전트의 연결이 해제되면 할당된 접미사가 재사용될 수 있습니다. 예를 들어, *PodAgent* 가 연결된 중에 *PodAgent%1* 의 연결이 해제되면 정규화된 이름 *hostPA|processNIM|PodAgent* 를 갖는 연결하려는 다음 에이전트의 이름이 *PodAgent%1* 로 변경됩니다.

이 접미사 식별자를 재사용하면 Enterprise Manager 가 연결마다 특정 에이전트 이름에 동일한 접미사를 할당할 수 있게 됩니다. 하지만 이후 연결에서 해당 에이전트는 다르게 이름이 변경될 수 있습니다. 연결마다 에이전트의 이름이 변경되면 기록 데이터를 쿼리할 때 문제가 될 수 있으므로 Enterprise Manager 에 의한 에이전트 이름 변경을 방지하는 이름 지정 전략을 구성하는 것이 좋습니다.

클러스터된 응용 프로그램에 대한 에이전트 이름 지정 고려 사항

동일한 응용 프로그램의 여러 인스턴스를 실행하는 경우 Introscope 는 문자와 무작위 숫자를 사용한 에이전트 이름을 추가하여 동일한 에이전트(사용자 지정 메트릭 에이전트 *포함*) 이름을 확인하려고 시도합니다. 하지만 CA Technologies 는 Introscope 에 이름 지정 확인 방법을 지정할 것을 권장합니다.

동일한 에이전트 이름 지정 확인을 위한 옵션:

- 복제된 에이전트 이름 지정을 활성화하여 해당 에이전트가 복제된 에이전트임을 Introscope 에 지정합니다(클러스터 환경에서 복제된 에이전트 이름 지정 사용 (see page 152) 참조).
- 사용자 스스로 고유한 에이전트 이름을 정의하고 각 에이전트에 대해 별도의 에이전트 프로필을 만듭니다(응용 프로그램 인스턴스의 고유 이름 구성 (see page 153) 참조).
- Introscope 가 자체적인 이름 지정 체계를 사용하여 각 에이전트를 고유하게 이름 지정하게 합니다(Introscope 가 에이전트 이름 지정 충돌을 해결하는 방법 (see page 145) 참조).

Java 시스템 속성을 사용하여 에이전트 이름 지정

Java 시스템 속성을 사용하여 에이전트 이름을 지정하려면

- Java 명령줄에서 다음 속성을 사용하여 원하는 이름을 지정합니다.
  `-Dcom.wily.introscope.agent.agentName=`

시스템 속성 키를 사용하여 에이전트 이름 지정

이 방법은 에이전트가 해당 이름을 찾는 데 사용하는 두 번째 방법입니다. 배포의 기존 Java 시스템 속성 값을 사용하여 에이전트의 이름을 지정하려면 이 방법을 사용하십시오.

시스템 속성 키를 사용하여 에이전트 이름을 지정하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. Agent Name 섹션의 다음 속성에서 에이전트 이름을 제공할 Java 시스템 속성을 지정합니다.
   `introscope.agent.agentNameSystemPropertyKey`

   참고: 여기에 지정된 Java 시스템 속성이 없으면 이 속성은 무시됩니다.
3. 응용 프로그램 서버를 다시 시작합니다.

응용 프로그램 서버로부터 에이전트 이름 가져오기

응용 프로그램 서버로부터 응용 프로그램 서버 인스턴스 이름을 자동으로 추출한 후 이 정보를 사용하여 자신의 이름을 지정하도록 에이전트를 구성할 수 있습니다. 이렇게 하면 별도의 에이전트 프로필 파일에 각각의 에이전트 이름을 구성할 필요가 없습니다. 또한 응용 프로그램 서버 환경에 변경 사항이 있는 경우 에이전트가 자신의 이름을 변경할 수도 있습니다. 이렇게 하면 여러 응용 프로그램 서버 플랫폼이 섞여 있을 수 있는 많은 수의 환경 전체에서 에이전트 프로필을 배포할 수 있습니다.

에이전트 이름 지정을 지원하는 응용 프로그램 서버

다음과 같은 지원되는 응용 프로그램 서버 버전에서 Introscope 를 사용할 경우 자동 에이전트 이름 지정이 지원됩니다.

- JBoss
- WebLogic 9.x
- WebSphere 6.1.x 분산
- WebLogic 10.0
- WebSphere 7.0.x 분산
- WebLogic 10.3

Introscope Workstation 에 표시되는 응용 프로그램 서버의 이름은 Java J2EE API 에 의해 결정됩니다. 모든 응용 프로그램 서버는 API 를 다르게 구현하므로 이로 인해 Workstation 에 표시되는 응용 프로그램 서버의 이름이 달라지기도 합니다. Workstation 에서는 여러 응용 프로그램 서버의 이름에 서로 다른 형식이 사용될 수 있으며, 동일한 응용 프로그램 서버 이름이라도 릴리스에 따라 다른 형식이 사용될 수 있습니다.

자동 에이전트 이름 지정

자동 에이전트 이름 지정이 사용되는 경우 에이전트는 시작 시 응용 프로그램 서버에서 이름 정보를 찾습니다. 에이전트는 에이전트 이름을 얻을 때까지 기다렸다가 Enterprise Manager 에 연결을 시도합니다.

에이전트가 이름 지정 정보를 찾은 경우 Introscope 에서는 에이전트 이름이 에이전트 명명 규칙을 따르도록 해당 정보를 편집합니다.

지원되는 응용 프로그램 서버에서의 에이전트 이름은 정보의 여러 부분으로 구성되며 이러한 부분은 응용 프로그램 서버에 따라 다릅니다.

- JBoss 의 경우 에이전트 이름은 서버가 시작될 때 지정된 구성 이름을 기반으로 합니다.
- WebLogic 의 경우 에이전트 이름은 다음과 같이 구성됩니다.

  도메인(데이터 센터) + 클러스터 + 인스턴스(WLS 의 인스턴스)
- WebSphere 의 경우 에이전트 이름은 다음과 같이 구성됩니다.

  셀(도메인) + 프로세스(WAS 의 인스턴스)

정보를 얻은 경우 각 세그먼트는 다음과 같이 슬래시로 구분됩니다.
`medrec/MyCluster/MedRecServer`

세그먼트 이름의 슬래시는 밑줄로 변환됩니다. 예를 들어 도메인 이름이 Petstore/West 로 지정된 경우 이 이름은 Petstore_West 로 변환됩니다.

참고: Introscope 는 다음 규칙에 따라 에이전트 이름 구성에 사용되는 정보를 편집합니다.

- 파이프, 콜론 또는 퍼센트 기호와 같은 문자는 밑줄로 바뀝니다.
- 영문자 이외의 문자로 시작하는 이름 앞에는 문자 "A"가 붙습니다.
- 이름이 지정되지 않은 경우에는 "UnknownAgent" 조건과 구별할 수 있도록 "UnnamedAgent"로 바뀝니다.

자동 에이전트 이름 지정을 사용하도록 설정하려면

1. *IntroscopeAgent.profile* 에서 *introscope.agent.agentAutoNamingEnabled* 를 *true* 로 설정합니다.
2. 응용 프로그램 서버에 따라 다음과 같이 변경합니다.
   - WebLogic 의 경우 Introscope 시작 클래스를 생성합니다. 자세한 내용은 WebLogic 용 시작 클래스 구성 (see page 48)을 참조하십시오.
   - WebSphere 의 경우 Introscope 사용자 지정 서비스를 생성합니다. 자세한 내용은 WebSphere 에서 사용자 지정 서비스 구성 (see page 59)을 참조하십시오.
   - JBoss 의 경우 XML 파일을 생성합니다. 자세한 내용은 JBoss 구성 (see page 44)을 참조하십시오.

자동 에이전트 이름 지정 및 이름 변경된 에이전트

자동 에이전트 이름 지정을 사용하여 에이전트는 항상 최신 응용 프로그램 서버 관련 에이전트 이름을 가져오려고 시도합니다. 에이전트는 주기적으로 새 이름을 확인합니다.

응용 프로그램 서버 구성에 대한 변경으로 인해 에이전트 이름이 변경되면 에이전트가 자동으로 자신의 이름을 변경합니다. Investigator 트리에서 이 에이전트는 연결이 해제된 것으로 표시됩니다. 연결이 해제된 에이전트는 Investigator 트리에 그대로 유지되며, 마운트 해제 시간 간격이 경과한 후 자동으로 마운트 해제되거나 수동으로 마운트 해제할 수 있습니다.

이름이 변경된 에이전트가 Enterprise Manager 에 다시 연결되고 Investigator 트리에 표시됩니다. 에이전트는 이러한 변경 사항을 로깅합니다.

Enterprise Manager 연결 지연을 위해 자동 에이전트 이름 지정 속성을 구성하는 방법과 이런 변경 검사 시간 간격에 대한 자세한 내용은 고급 자동 에이전트 이름 지정 옵션 (see page 150)을 참조하십시오.

고급 자동 에이전트 이름 지정 옵션

환경에 대한 자동 에이전트 이름 지정을 제어하기 위해 변경할 수 있는 여러 속성이 있습니다.

초기 Enterprise Manager 연결 지연

자동 에이전트 이름 지정 기능을 사용하는 경우 에이전트는 Enterprise Manager 에 연결하기 전에 일정 시간(구성 가능) 동안 에이전트 이름 정보를 찾으며 대기합니다. 기본 지연 시간은 120 초입니다.

지연 값을 변경하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. **Agent Name** 섹션의 *introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds* 속성에서 원하는 지연 시간을 구성합니다.
3. 응용 프로그램 서버를 다시 시작합니다.

에이전트 이름 변경 확인 간격

자동 에이전트 이름 지정 기능을 사용하는 경우 에이전트는 응용 프로그램 서버의 이름 지정 정보가 변경되었는지 여부를 주기적으로 확인합니다. 기본 간격은 10 분입니다.

이 간격을 변경하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. **Agent Name** 섹션의 *introscope.agent.agentAutoRenamingIntervalInMinutes* 속성에서 원하는 간격을 구성합니다.
3. 응용 프로그램 서버를 다시 시작합니다.

에이전트 로그 파일 자동 이름 지정 해제

기본적으로 Java 시스템 속성이나 응용 프로그램 서버에서 제공된 정보로 에이전트 이름이 자동으로 확인된 경우, 해당 에이전트와 관련된 로그 파일의 이름은 동일한 정보를 사용하여 자동으로 지정됩니다. 그러나 이 자동 로그 이름 지정을 해제하고 *IntroscopeAgent.profile* 에 지정된 에이전트 로그 이름을 계속 사용할 수 있습니다.

에이전트 로그 파일 자동 이름 지정을 해제하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. *introscope.agent.disableLogFileAutoNaming* 속성 값을 *true* 로 설정합니다.
3. *IntroscopeAgent.profile* 을 저장합니다.
4. 응용 프로그램 서버를 다시 시작합니다.

클러스터 환경에서 복제된 에이전트 이름 지정 사용

동일한 호스트 및 프로세스를 모니터링하며 이름이 동일한 에이전트가 두 개 있고 사용자가 에이전트에 고유 이름을 지정하지 않은 경우 에이전트 이름에는 번호가 추가됩니다. 복제된 에이전트 이름 지정을 사용하면 에이전트를 클러스터된 응용 프로그램의 특정 응용 프로그램 인스턴스와 서로 연결할 수 있습니다.

다음과 같은 경우에 복제된 에이전트가 실행됩니다.

- 호스트, 프로세스 또는 Java Agent 이름을 하나 이상의 다른 에이전트와 공유하는 에이전트를 실행하는 경우
- 동일한 에이전트 프로필을 사용하는 에이전트를 둘 이상 실행하는 경우

복제된 에이전트 이름 지정을 사용하도록 설정하려면

1. 관리되는 응용 프로그램과 Java Agent 를 중지합니다.
2. *IntroscopeAgent.profile* 을 열고 다음 속성을 true 로 설정합니다.
   `introscope.agent.clonedAgent=true`
3. *IntroscopeAgent.profile* 을 저장합니다.
4. 관리되는 응용 프로그램과 Java Agent 를 다시 시작합니다.

복제된 에이전트 이름 지정 시나리오

Java Agent 복제 속성이 켜져 있을 때 이름이 모두 *AgentX* 인 4 개의 Java Agent 가 있는 경우 Enterprise Manager 는 *AgentX-1*, *AgentX-2*, *AgentX-3*, *AgentX-4* 로 에이전트의 이름을 지정합니다. *AgentX-1* 의 연결이 해제되고 다시 연결되는 경우 이름 *AgentX-1* 이 계속 사용됩니다. 이 이름 지정을 사용하는 경우 원래 복제된 Java Agent 의 수보다 더 많은 Java Agent 이름이 데이터베이스에 있을 수 없습니다.

응용 프로그램 인스턴스의 고유 이름 구성

동일한 컴퓨터에서 응용 프로그램의 여러 인스턴스를 모니터링하는 경우 고유한 에이전트 이름을 명시적으로 구성할 수 있습니다.

고유한 에이전트 이름을 구성하려면

1. 각 응용 프로그램에 대해 개별 에이전트 프로필을 생성합니다.
2. 에이전트 프로필에서 각 에이전트의 이름을 고유하게 지정합니다.
3. 각 응용 프로그램에서 사용해야 하는 에이전트 프로필을 지정합니다.

응용 프로그램 심사 맵 및 에이전트 이름

Workstation 의 응용 프로그램 심사 맵은 응용 프로그램의 프런트엔드 및 백엔드를 정의하고 Introscope 데이터베이스에 응용 프로그램 구성 요소에 대한 정보를 저장할 때 여러 고유 식별자의 일부로서 에이전트 이름을 사용합니다. 에이전트 이름이 변경되는 경우 응용 프로그램 심사 맵의 일부 요소도 변경될 수 있습니다. 예를 들어, 에이전트의 최초 등록 중에 Enterprise Manager 는 에이전트 이름에 *%<시퀀스 번호>*(예: *MyAgent%1*)를 추가하여 중복된 에이전트 이름에 고유한 이름을 할당할 수 있습니다. 응용 프로그램 심사 맵의 일부가 파악하기 쉽도록 중복된 에이전트 이름에 의존하는 경우 맵의 요소가 변경될 수 있습니다.

이렇게 해도 에이전트나 Enterprise Manager 의 올바른 기능에 영향을 주지는 않지만 시스템의 전체 성능을 저하시킬 수 있으므로 사용자들이 Introscope 환경에서 구현된 이름 지정 방식을 인지하고 있는 것이 좋습니다. 사용자들은 이 문제를 피하기 위해 특정 에이전트 이름을 지정할 수 있습니다.

제 7 장: Java Agent 모니터링 및 로깅

Introscope 에서는 응용 프로그램을 모니터링하는 동안 Java Agent 자체의 건전성 및 동작도 모니터링할 수 있습니다. 이 단원에는 에이전트 건전성 모니터링 및 Java Agent 의 로깅 옵션에 대한 정보가 포함되어 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

에이전트 연결 메트릭 구성

기본적으로 Introscope 에서는 Enterprise Manager 에 연결된 에이전트의 연결 상태에 대해 모니터링 가능한 메트릭을 생성합니다. 에이전트 연결 메트릭을 모니터링하여 에이전트와 Enterprise Manager 간의 현재 연결 상태를 확인할 수 있습니다.

에이전트 연결 메트릭은 Workstation Investigator 에서 다음과 같이 Enterprise Manager 프로세스(사용자 지정 메트릭 호스트) 아래에 나타납니다.

```
Custom Metric Host (Virtual) \ Custom Metric Process(Virtual) \ Custom Metric Agent (Virtual) (*SuperDomain*) \ Agents \ <HostName> \ <Agent Process Name> \ <Agent Name> \ ConnectionStatus
```

연결 메트릭의 값은 다음과 같습니다.

- 0 - 에이전트에 대한 데이터를 사용할 수 없음
- 1 - 에이전트가 연결됨
- 2 - 에이전트 보고 시간이 느림
- 3 - 에이전트의 연결이 끊어짐

에이전트의 연결이 끊어진 경우 "주목할 사항" 이벤트도 생성됩니다. 사용자는 다른 이벤트와 마찬가지로 기록 쿼리 인터페이스를 사용하여 에이전트 연결 끊김을 쿼리할 수 있습니다. 에이전트 연결 끊김 이벤트는 Workstation 콘솔의 "개요" 탭에서 응용 프로그램 건전성을 평가하는 데 사용되는 데이터의 일부입니다.

에이전트와 Enterprise Manager 의 연결이 끊기고 나면 Introscope 에서는 에이전트가 시간 초과될 때까지 계속해서 연결 끊김 상태 메트릭을 생성합니다. 에이전트가 시간 초과되면 더 이상 연결 메트릭이 생성되거나 Enterprise Manager 에 보고되지 않습니다.

다음 단계를 따르십시오.

1. Enterprise Manager 가 설치된 컴퓨터에서 *<Introscope_Home>/config* 디렉터리에 있는 *IntroscopeEnterpriseManager.properties* 파일을 엽니다.
2. 다음 속성을 수정합니다.
   `introscope.enterprisemanager.agentconnection.metrics.agentTimeoutInMinutes`
   시간 단위는 분입니다.
3. *IntroscopeEnterpriseManager.properties* 를 저장합니다.

참고: Enterprise Manager 속성에 대한 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

소켓 메트릭

에이전트에는 기본적으로 소켓 및 SSL(Secure Sockets Layer) 메트릭 수집이 사용하도록 설정되어 있습니다.

참고: *Agent.NoRedef.jar* 를 사용하는 JVM 에서는 소켓 메트릭이 보고되지 않습니다. 자세한 내용은 AutoProbe for WebSphere 6.1 (see page 54)을 참조하십시오.

소켓 및 SSL 메트릭 수집 제한

소켓 및 SSL 메트릭 수집은 기본적으로 사용하도록 설정되어 있지만 보다 관련성이 높은 특정 정보를 수집하거나 오버헤드를 줄이도록 일부 메트릭 수집을 제한할 수 있습니다.

소켓 및 SSL 메트릭 수집을 제한하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. Agent I/O Socket Metrics 섹션에서 메트릭이 필요한 호스트 또는 포트 목록을 포함하도록 속성 값을 편집합니다.

   올바르지 않은 호스트 또는 포트가 매개 변수 값에 포함되어 있으면 에이전트 로그에 경고 메시지가 기록되고 해당 값은 무시됩니다. 따라서 목록에 항목이 포함되어 있지 않으면 제한이 적용되지 않습니다.

   - *introscope.agent.io.socket.client.hosts*
     쉼표로 구분된 호스트 목록으로, 지정된 호스트에 대한 '클라이언트' 소켓 메트릭만 생성됩니다. 호스트는 이름이나 IP 주소(IPv4 또는 IPv6 형식)의 텍스트 표현으로 지정할 수 있습니다.

     참고: 중복 호스트 이름은 삭제됩니다. 여러 개의 호스트 이름이 단일 IP 에 매핑되는 경우에는 하나의 이름만 유지됩니다. 그러나 이 속성은 클라이언트 연결을 동의어 이름 집합의 임의 항목과 일치시킵니다.

   - *introscope.agent.io.socket.client.ports*
     쉼표로 구분된 포트 번호 목록으로, 지정된 포트에 대한 '클라이언트' 소켓 메트릭만 생성됩니다.

     참고: 중복 포트는 삭제됩니다.

   - *introscope.agent.io.socket.server.ports*
     쉼표로 구분된 포트 번호 목록으로, 지정된 포트에 대한 '서버' 소켓 메트릭만 생성됩니다.

   위의 속성은 동적 속성이므로 변경 사항을 적용하기 위해 응용 프로그램을 다시 시작할 필요가 없습니다.
3. *IntroscopeAgent.profile* 을 저장합니다.

소켓 및 SSL 메트릭 수집 세부 조정

소켓 및 SSL 메트릭 수집 제한 (see page 157)에 설명된 대로 소켓 및 SSL 메트릭 수집을 제한할 수 있을 뿐 아니라, 특정 추적 프로그램 그룹을 설정 및 해제하여 메트릭 수집을 세부적으로 조정할 수도 있습니다. 이렇게 하면 원하는 특정 정보를 수집하고 오버헤드 비용을 줄이는 데 유용합니다.

소켓 및 SSL 메트릭 수집을 세부적으로 조정하려면

1. *<Agent_Home>*/wily/core/config 디렉터리에 있는 *java2.pbd* 파일을 엽니다.
2. *java2.pbd* 의 I/O Socket Tracer Group 또는 Network Tracer Group 섹션에서 설정 또는 해제할 추적 프로그램을 결정한 다음 이를 주석으로 처리하거나 주석 처리를 제거합니다. 예를 들어 입력 대역폭에 대한 메트릭을 수집하지 않으려면 다음과 같이 주석 처리합니다.

   ```
   #TraceOneMethodWithParametersIfFlagged: SocketTracing read
   InputStreamBandwidthTracer "Input Bandwidth (Bytes Per Second)"
   ```

   참고: 이름이 *MappingTracer* 로 끝나는 추적 프로그램은 주석으로 처리하면 안 됩니다.
3. *java2.pbd* 파일을 저장합니다.

응용 프로그램 심사 맵에서 SSL, NIO 및 소켓 추적

Workstation Investigator 의 응용 프로그램 심사 맵에는 계측된 클라이언트 소켓 연결이 표시될 수 있습니다. 에이전트는 트랜잭션에 사용되는 외부 시스템에 대한 세부 정보를 기록하고 이 정보를 Enterprise Manager 에 보냅니다. 그러면 이 정보가 응용 프로그램 심사 맵에 그래픽으로 표시됩니다.

<Agent_Home>/wily/core/config 디렉터리에 있는 appmap.pbd 의 추적 프로그램은 기존 SSL, NIO 및 소켓 계측을 확장합니다. 추적 프로그램을 통해 에이전트는 더 많은 구성 요소 정보를 응용 프로그램 심사 맵에 보낼 수 있습니다. 추적 프로그램은 기본적으로 사용되도록 설정됩니다. appmap.pbd 의 Trace Sockets 및 Trace NIO Sockets 섹션에서 특정 추적 프로그램을 주석 처리하여 사용되지 않도록 설정할 수 있습니다.

응용 프로그램 심사 맵에서 구성 요소 이름 변경

응용 프로그램 심사 맵에 표시되는 구성 요소 이름에는 대상 호스트 및 포트 ID 가 포함되며, 이는 소켓 클라이언트 메트릭 이름이 표시되는 방식과 동일합니다. 구성 요소 이름의 호스트 ID 는 호스트 이름이나 호스트 IP 주소로 구성될 수 있습니다. 기본값은 호스트 이름입니다. 표시된 구성 요소 이름을 변경할 수 있습니다.

다음 단계를 따르십시오.

1. *<Agent_Home>*/wily/core/config 디렉터리에 있는 appmap.pbd 를 엽니다.
2. Trace Sockets 및 Trace NIO Sockets 섹션에서 수정할 추적 프로그램을 선택합니다.
3. 관련 추적 프로그램의 *{hostname}*을 *{hostip}*로 변경합니다.

   예를 들어 원래 추적 프로그램에서는 다음과 같이 기본 *{hostname}*을 사용합니다.

   ```
   TraceOneMethodWithParametersIfFlagged: SocketTracing read AppMapSocketTracerBT
   "System {hostname} on port {port}"
   TraceOneMethodWithParametersIfFlagged: SocketTracing write
   AppMapSocketTracerBT "System {hostname} on port {port}"
   ```

   호스트 IP 를 표시하려면 다음과 같이 *{hostip}*를 대신 사용하십시오.

   ```
   TraceOneMethodWithParametersIfFlagged: SocketTracing read AppMapSocketTracerBT
   "System {hostip} on port {port}"
   TraceOneMethodWithParametersIfFlagged: SocketTracing write
   AppMapSocketTracerBT "System {hostip} on port {port}"
   ```

4. appmap.pbd 파일을 저장합니다.

소켓 및 SSL 메트릭 수집 해제

소켓 및 SSL 메트릭을 수집할 필요가 없는 경우 이러한 메트릭의 수집을 완전히 해제할 수 있습니다.

소켓 및 SSL 메트릭 수집을 해제하려면

1. *toggles-full.pbd* 또는 *toggles-typical.pbd* 파일 중 배포에서 사용하는 파일을 엽니다.
2. 다음과 같이 행의 시작 부분에 파운드 또는 해시 기호(#)를 추가하여 *SocketTracing* 을 주석으로 처리합니다.
   ```
   #TurnOn: SocketTracing
   ```
3. 수정한 파일을 저장합니다.

toggles-full.pbd 및 *toggles-typical.pbd* 파일에서 추적 프로그램 그룹을 설정 및 해제하는 방법에 대한 자세한 내용은 기본 추적 프로그램 그룹 및 toggles 파일 (see page 106)과 추적 프로그램 그룹 설정 또는 해제 (see page 117)를 참조하십시오.

이전 버전과의 호환성

릴리스 9.0 이전의 추적 프로그램에 해당

Java Agent 소켓 추적 프로그램은 9.0 이전 릴리스의 추적 프로그램보다 세부적으로 구성할 수 있습니다. 그러나 이전 추적 프로그램으로 되돌리고 소켓 추적 기능 및 구성 옵션이 사용되지 않도록 설정할 수 있습니다.

릴리스 9.0 이전의 추적 프로그램을 사용하고 있는 경우 다음을 수행하도록 Java Agent 를 구성할 수 있습니다.

- 소켓 메트릭 및 이전 추적 프로그램 수집 (see page 161)
- 입력 및 출력 대역폭 메트릭 수집 (페이지 161)

소켓 메트릭 수집

릴리스 9.0 이전의 추적 프로그램에 해당

추적 프로그램을 사용하여 소켓 메트릭을 수집하려면 Java Agent 를 구성하십시오.

다음 단계를 따르십시오.

1. toggles-full.pbd 또는 toggles-typical.pbd 파일 중 배포에서 사용되는 파일을 엽니다.
2. 다음과 같이 행의 시작 부분에 파운드 또는 해시 기호(#)를 추가하여 SocketTracing 을 주석으로 처리합니다.
   ```
   #TurnOn: SocketTracing
   ```
3. 다음과 같이 ManagedSocketTracing 의 주석 처리를 제거합니다.
   ```
   TurnOn: ManagedSocketTracing
   ```
4. 파일을 저장합니다.

입력 및 출력 대역폭 메트릭 수집

릴리스 9.0 이전의 소켓 추적 프로그램에 해당

소켓 추적 프로그램을 사용하는 경우 입력 및 출력 대역폭 메트릭이 필요하다면 Java Agent 를 구성하십시오.

다음 단계를 따르십시오.

1. IntroscopeAgent.profile 을 엽니다.
2. Agent Socket Rate Metrics 섹션을 찾아 다음 속성을 true 로 변경합니다.
   ```
   introscope.agent.sockets.reportRateMetrics=true
   ```
   참고: 이 속성은 ManagedSocketTracing 이 사용하도록 설정되고 SocketTracing 은 사용하지 않도록 설정된 경우에만 작동합니다.
3. IntroscopeAgent.profile 을 저장합니다.

로깅 옵션 구성

응용 프로그램 서버에 Java Agent 가 설치되어 있으면 서버 시작 후에 로그 디렉터리 *<Agent_Home>*/wily/logs 가 생성됩니다. 응용 프로그램 서버 프로세스에는 *<Agent_Home>* 디렉터리에 대한 읽기/쓰기/실행 권한이 모두 있어야 합니다. 이렇게 하려면 응용 프로그램 서버 프로세스를 실행하는 사용자와 동일한 운영 체제에 Java Agent 를 설치하십시오. 또는 다른 사용자로 Java Agent 를 설치한 다음 *chmod* 명령을 사용하여 필요한 사용 권한을 부여하십시오.

Java Agent 를 세부 정보 표시 모드에서 실행할 수 있습니다. 세부 정보 표시 모드에서는 에이전트와 사용자 환경의 상호 작용 및 작업에 대한 정보가 보다 세부적으로 기록됩니다. 이 정보는 사용자 환경 또는 에이전트 기능의 문제를 해결하는 데 유용합니다.

Introscope 에서는 이러한 기능에 Log4J 기능을 사용합니다. 다른 Log4J 기능을 사용하려면 Log4J 설명서를 참조하십시오.

세부 정보 표시 모드에서 에이전트 실행

세부 정보 표시 모드에서 에이전트를 실행하면 보다 세부적인 정보가 에이전트 로그에 기록됩니다.

세부 정보 표시 모드에서 에이전트를 실행하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. 다음 속성에서 기존 *INFO* 를 *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel* 로 대체하여 수정합니다.

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel, console, logfile
   ```

3. *IntroscopeAgent.profile* 을 저장합니다.

   참고: 이 속성의 변경 사항은 1 분 내에 적용해야 하며 관리되는 응용 프로그램을 다시 시작할 필요는 없습니다.

에이전트 출력을 파일로 리디렉션

세부 정보 표시 모드에서 에이전트 로깅을 제어하는 속성은 에이전트 로그의 출력 위치와 이 로그 파일의 위치도 제어합니다. 자세한 내용은 세부 정보 표시 모드에서 에이전트 실행 (see page 162)을 참조하십시오.

에이전트 출력을 파일로 리디렉션하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. *log4j.logger.IntroscopeAgent* 속성을 찾습니다.

   이 속성에 대한 옵션은 다음과 같습니다.

   - *console*: 로그 파일의 정보를 콘솔로 보냅니다.
   - *logfile*: 로그 파일의 정보를 로그 파일로 보냅니다. 이 옵션을 선택할 경우 로그 파일의 위치는 *log4j.appender.logfile.File* 속성을 사용하여 구성합니다. 자세한 내용은 에이전트 로그 파일의 이름 또는 위치 변경 (see page 163)을 참조하십시오.

   예를 들어 에이전트가 세부 정보 표시 모드에서 로그 파일에 로그를 보고하도록 설정한 경우 이 속성을 다음과 같이 설정할 수 있습니다.

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel,logfile
   ```

   에이전트가 로그 파일과 콘솔 모두에 로그를 보고하도록 설정한 경우에는 이 속성에 *logfile* 과 *console* 을 모두 포함하십시오.

   참고: 기본적으로 에이전트 로그 *IntroscopeAgent.log* 는 *<Agent_Home>*/wily/logs 디렉터리에 기록됩니다. 에이전트 자동 이름 지정 옵션을 구성한 경우에는 에이전트 로그 파일 및 자동 에이전트 이름 지정 (see page 164)에 설명된 대로 에이전트 로그 파일의 이름도 자동으로 지정됩니다.
3. *IntroscopeAgent.profile* 을 저장합니다.

에이전트 로그 파일의 이름 또는 위치 변경

속성을 수정하여 로그 파일의 위치 및 이름을 변경할 수도 있습니다.

로그 파일의 이름 또는 위치를 변경하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.

2. *log4j.appender.logfile.File* 속성을 찾습니다.

   log4j.logger.IntroscopeAgent 속성에 *logfile* 이 지정된 경우 로그 파일의 위치는 *log4j.appender.logfile.File* 속성을 사용하여 구성합니다. 자세한 내용은 에이전트 출력을 파일로 리디렉션 (see page 163)의 2 단계를 참조하십시오.

   참고: 시스템 속성(Java 명령줄 -D 옵션)을 파일 이름의 일부로 포함할 수 있습니다. 예를 들어 Java 명령이 *-Dmy.property=Server1* 로 시작하는 경우 *log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* 는 *log4j.appender.logfile.File=../../logs/Introscope-Server1.log* 로 확장됩니다.

3. 새 위치 및 파일에 대한 정규화된 경로를 사용하여 로그 파일의 위치와 이름을 설정합니다. 예:
   `log4j.appender.logfile.File=C:/Logs/AgentLog1.log`

4. *IntroscopeAgent.profile* 을 저장합니다.

에이전트 로그 파일 및 자동 에이전트 이름 지정

자동 에이전트 이름 지정 기능을 사용할 경우 기본적으로 에이전트와 연결된 로그 파일의 이름은 에이전트의 이름을 지정하는 데 사용된 것과 동일한 정보를 사용하여 자동으로 지정됩니다.

자동 에이전트 이름 지정은 로그 파일에 다음과 같은 영향을 줍니다.

- 로그 파일의 원래 이름이 *.log* 로 끝나지 않는 경우 마침표와 *log* 가 추가됩니다.
- 영문자나 숫자가 아닌 모든 문자는 밑줄로 바뀝니다.
- 고급 Log4J 기능이 사용되는 경우에는 에이전트 로그 파일 자동 이름 지정 기능이 작동하지 않을 수 있습니다.

다음 예제에서는 에이전트 로그 파일의 이름이 지정되는 방식을 보여 줍니다. 이 예제에서는 *DOM1//ACME42* 라는 에이전트 이름을 사용합니다. 여기서 *DOM1* 은 WebLogic 도메인이고 *ACME42* 는 에이전트의 인스턴스입니다.

에이전트 로그 파일(기본적으로 *AutoProbe.log*)이 생성될 때 에이전트 이름을 아직 사용할 수 없으면 다음과 같이 파일 이름에 타임스탬프가 포함됩니다.

```
AutoProbe.20040416-175024.log
```

에이전트 이름을 사용할 수 있게 되면 다음과 같이 에이전트의 자동 이름을 사용하여 로그 파일의 이름이 변경됩니다.

```
AutoProbe.DOM1_ACME42.log
```

자동 로그 이름 지정 기능을 사용하지 않도록 설정할 수도 있습니다. 자세한 내용은 고급 자동 에이전트 이름 지정 옵션 (see page 150)을 참조하십시오.

날짜 또는 크기별로 로그 롤업

크기 또는 날짜를 기준으로 로그를 롤업하여 지정된 기간(일)의 정보만 보관하고 나머지는 삭제할 수 있습니다.

로그 파일을 롤업하려면

1. *IntroscopeAgent.profile* 을 열고 Logging Configuration 섹션을 찾습니다.
2. 다음 속성을 수정합니다.

   ```
   log4j.logger.IntroscopeAgent
   log4j.appender.logfile.File
   log4j.appender.console.layout
   log4j.appender.console.layout.ConversionPattern
   log4j.appender.logfile
   log4j.appender.logfile.MaxFileSize
   log4j.appender.logfile.MaxBackupIndex
   ```

   참고: 이 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

3. *IntroscopeAgent.profile* 을 저장합니다.

   예를 들어 다음 구성을 사용하면 최대 3 개의 백업/롤업된 로그가 보관되며 각 로그의 최대 크기는 2 킬로바이트입니다.

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLeve
   l, console, logfile
   log4j.appender.logfile.File=logs/IntroscopeAgent.log
   log4j.appender.console.layout=com.wily.org.apache.log4j.PatternLayout
   log4j.appender.console.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z} [%-3p]
   [%c] %m%n
   log4j.appender.logfile=com.wily.introscope.agent.AutoNamingRollingFileAppende
   r
   log4j.appender.logfile.MaxFileSize=2KB
   log4j.appender.logfile.MaxBackupIndex=3
   ```

ProbeBuilder 로그 관리

ProbeBuilder 는 ProbeBuilder 가 표시 및 계측하는 모든 클래스를 기록할 뿐 아니라, 계측 프로세스 중에 추가한 프로브에 대해 계측을 추가하지 않은 모든 클래스와 ProbeBuilder 가 사용한 PBD 도 기록합니다. 또한 건너뛰기로 인해 계측하지 않은 클래스도 기록합니다.

명령줄 ProbeBuilder 및 ProbeBuilder 마법사 로그 이름 및 위치

명령줄 ProbeBuilder 및 ProbeBuilder 마법사 로그 파일 위치는 ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 를 사용하여 Java 클래스를 지정하는 곳에 따라 결정됩니다. 디렉터리의 경우 로그 파일은 대상 디렉터리 안에 위치합니다. 파일의 경우 로그 파일은 대상 파일 옆에 위치합니다.

ProbeBuilder 로그 파일의 이름:
`<original-directory-or-original-file>.probebuilder.log`

<original-directory> 또는 *<original-file>*은 ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 를 사용하여 지정하는 Java 클래스 위치입니다.

가장 최근 로그 파일만 유지되고 모든 이전 로그 파일은 덮어쓰기됩니다.

AutoProbe 로그 이름 및 위치

AutoProbe 는 항상 변경한 사항을 기록하려고 합니다. 기본적으로 AutoProbe 로그 파일의 이름은 *AutoProbe.log* 로 지정됩니다.

AutoProbe 로그의 이름 또는 위치를 변경하려면

1. *IntroscopeAgent.profile* 을 텍스트 편집기에서 엽니다.
2. *introscope.autoprobe.logfile* 속성을 찾고 정규화된 파일 경로를 사용하여 로그 이름 및 위치를 수정합니다. 절대 이름이 아닌 이름은 *IntroscopeAgent.profile* 파일의 위치를 기준으로 확인됩니다.

   참고: 클래스 경로의 리소스에서 에이전트 프로필을 로드할 경우 *IntroscopeAgent.profile* 파일은 리소스 내에 있으므로 AutoProbe 가 AutoProbe 로그 파일에 기록할 수 없습니다.

   이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
3. *IntroscopeAgent.profile* 을 저장합니다.

제 8 장: LeakHunter 및 ErrorDetector 구성

이 단원에서는 LeakHunter 및 ErrorDetector 를 활성화하고 구성하는 방법에 대해 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

LeakHunter

Introscope LeakHunter 는 시간이 지남에 따라 크기가 늘어나는 컬렉션 인스턴스(즉, 컬렉션에 저장된 개체 수가 시간이 지남에 따라 늘어나는 경우)를 조사하여 잠재 메모리 누수의 원인을 파악할 수 있도록 설계된 추가 기능 구성 요소입니다.

몇 분 또는 몇 시간 정도로 짧게 실행되는 프로그램에서 발생하는 메모리 누수는 큰 문제가 되지 않을 수 있습니다. 그러나 웹 사이트와 같이 하루 24 시간 실행되는 응용 프로그램의 경우에는 사소한 메모리 누수라도 곧바로 큰 문제로 확대될 수 있습니다.

Introscope LeakHunter 를 설정하고 컬렉션 클래스를 검색한 다음 정보가 수집된 후에는 Introscope LeakHunter 를 해제하는 방법으로 발견된 메모리 누수에 대한 정보를 추적할 수 있습니다. 이 방법으로 LeakHunter 를 사용하면 일시적으로만 약간의 오버헤드가 발생합니다.

LeakHunter 작동 방식

LeakHunter 를 사용하도록 설정한 후 LeakHunter 가 새 잠재 누수를 찾는 동안 적용되는 시간 초과 기간을 정의할 수도 있습니다. AutoProbe 를 사용하는 경우에는 관리되는 응용 프로그램을 다시 시작하기만 하면 됩니다. ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 를 사용하는 경우에는 이전에 사용된 PBD 파일 외에도 leakhunter.pbd 를 사용하여 응용 프로그램을 다시 계측해야 합니다.

LeakHunter 는 시간에 지남에 따라 크기가 증가하는 컬렉션을 찾은 경우 다음 작업을 수행합니다.

- 컬렉션을 고유 ID 로 식별합니다.
- 컬렉션에 대한 정보를 Enterprise Manager 에 메트릭 데이터로 보고합니다.
- 컬렉션에 대한 정보를 에이전트 컴퓨터의 로그 파일에 보고합니다.
- 추적을 계속하여 해당 컬렉션에 대한 데이터를 보고합니다.

LeakHunter 는 컬렉션에 더 이상 누수가 없는 것으로 확인될 경우 해당 사실을 Enterprise Manager 와 로그 파일 모두에 보고하지만, 해당 컬렉션에 대한 추적과 데이터 보고는 계속합니다.

LeakHunter 는 시간 초과 기간이 만료될 때까지 계속해서 잠재 누수를 찾고 이미 식별된 잠재 누수를 모니터링합니다. 시간 초과 기간이 만료되면 LeakHunter 는 새로 할당된 컬렉션에서 잠재 누수 찾기를 중지하고 이미 잠재 누수로 식별된 컬렉션만 계속 검사합니다. 따라서 LeakHunter 오버헤드가 크게 줄어들며 잠재 누수를 추가적으로 모니터링할 수 있습니다. LeakHunter 는 관리되는 응용 프로그램이 종료될 때까지 식별된 잠재 누수를 계속 모니터링합니다.

메모리 누수의 원인을 찾으려면 Introscope Investigator 에서 메트릭 데이터를 탐색하거나 로그 파일을 확인하면 됩니다.

Java 에서 LeakHunter 가 추적하는 항목

Introscope LeakHunter 는 Java 구현에서 다음 컬렉션을 추적합니다.

- java.util.Collection 구현
  - java.util.ArrayList
  - java.util.LinkedList
  - java.util.TreeSet
  - java.util.HashSet
  - java.util.LinkedHashSet
  - java.util.Vector
  - java.util.Stack
- java.util.Map 구현
  - java.util.Map
  - java.util.SortedMap
  - java.util.HashMap
  - java.util.TreeMap
  - java.util.IdentityHashMap
  - java.util.Hashtable
  - java.util.Properties
  - java.util.LinkedHashMap

LeakHunter 가 추적하지 않는 항목

Introscope LeakHunter 는 다음을 추적하지 않습니다.

- 컬렉션에 의해 발생하지 않은 누수
- 참조 수가 증가하는 사용자 지정 컬렉션 구현 또는 기타 데이터 구조
- 계측되지 않는 누수 컬렉션

위에 더해, LeakHunter 는 Java 구현에서 다음을 추적하지 않습니다.

- Java 에서 LeakHunter 가 추적하는 항목 (see page 171)에 설명된 수집을 위해 생성된 모든 하위 클래스 하지만 ProbeBuilder 지시문(PBD) 파일을 업데이트하여 이 정보를 얻을 수 있습니다. 자세한 내용은 *leakhunter.pbd* 파일을 참조하십시오.

참고: Application Server AutoProbe 를 사용하는 경우 LeakHunter 는 응용 프로그램 서버에 의해 할당된 수집을 자동으로 추적하지 않습니다. 이러한 수집을 추적하려면 응용 프로그램 서버를 통계적으로 계측하거나 JVM AutoProbe 를 사용해야 합니다.

시스템 및 버전 요구 사항

Introscope LeakHunter 에는 Java Agent 와 동일한 시스템 요구 사항이 있습니다.

기본적으로 LeakHunter 는 설치 후 사용하도록 설정되지 않습니다. 따라서 해당 기능을 사용하려면 LeakHunter 를 사용하도록 설정해야 합니다.

LeakHunter 를 사용 또는 사용하지 않도록 설정

LeakHunter 는 에이전트 확장으로 실행되므로 클래스 경로를 업데이트할 필요가 없습니다. 기본적으로 LeakHunter 는 설치 후 사용하도록 설정되지 않습니다. 따라서 해당 기능을 사용하려면 LeakHunter 를 사용하도록 설정해야 합니다.

LeakHunter 를 사용하도록 설정하려면

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.

2. *LeakHunter Configuration* 제목 아래에서 *introscope.agent.leakhunter.enable* 속성을 찾아 값을 *true* 로 입력합니다.
3. 에이전트 프로필을 저장합니다.
4. *IntroscopeAgent.profile* 속성 *introscope.autoprobe.directivesFile* 에 나열한 **.typical.pbl* 또는 **.full.pbl* 파일을 열고 *leakhunter.pbd.*의 주석 처리를 제거합니다.

   참고: ProbeBuilder 마법사를 사용하는 경우 *leakhunter.pbd* 파일을 *<Agent_Home>\wily\core\config\hotdeploy* 디렉터리에 복사하십시오.
5. 응용 프로그램을 다시 시작합니다.

중요! 기본적으로 LeakHunter 와 같은 에이전트 확장은 *<Agent_Home>*\wily\core\ext 디렉터리에 있으며 이 디렉터리에서 참조됩니다. 그러나 에이전트 프로필에서 에이전트 확장 디렉터리의 위치를 변경할 수 있습니다. *\ext* 디렉터리의 위치를 변경할 경우에는 *\ext* 디렉터리의 내용도 이동해야 합니다.

LeakHunter 를 사용하지 않도록 설정하려면

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.
2. *LeakHunter Configuration* 제목 아래에서 *introscope.agent.leakhunter.enable* 속성을 찾아 값을 *false* 로 입력합니다.
3. 에이전트 프로필을 저장합니다.
4. 응용 프로그램을 다시 시작합니다.

LeakHunter 속성 구성

LeakHunter 구성 속성은 에이전트 프로필 *IntroscopeAgent.profile* 에 있습니다.

LeakHunter 를 구성하려면

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.

2. 다음 LeakHunter 속성을 원하는 대로 구성합니다.

introscope.agent.leakhunter.logfile.location

LeakHunter.log 파일의 위치를 지정합니다. 파일 이름은 *<IntroscopeAgent.profile>* 디렉터리를 기준으로 합니다. 이 속성이 주석으로 처리되어 있거나 비어 있는 경우에는 로그 파일이 기록되지 않습니다.

기본값은 *logs/LeakHunter.log* 입니다.

introscope.agent.leakhunter.logfile.append

응용 프로그램을 다시 시작할 때 로그 파일을 바꿀지(false 값) 기존 로그 파일을 추가할지(true 값)를 지정합니다.

기본값은 false 입니다.

introscope.agent.leakhunter.leakSensitivity

메모리 누수를 감지하기 위한 민감도 수준을 지정합니다. 누수 민감도 설정이 높으면 보고되는 잠재 누수가 많아지고, 민감도가 낮으면 보고되는 잠재 누수가 적어집니다.

속성 값은 1 에서 10 사이의 정수여야 합니다.

기본 민감도 수준은 5 입니다.

introscope.agent.leakhunter.timeoutInMinutes

LeakHunter 가 새 잠재 누수를 찾는 데 소요할 수 있는 시간(분)을 지정합니다. 이 속성 값은 음이 아닌 정수여야 합니다. 값 0 은 제한 시간이 없음을 나타냅니다.

기본값은 120 분입니다.

introscope.agent.leakhunter.collectAllocationStackTraces

할당 스택 추적 정보를 수집할지 여부를 지정합니다. 이 옵션을 설정하면 시스템 CPU 사용량 및 메모리 사용량이 높아질 수 있습니다. 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

기본값은 false 입니다.

introscope.agent.leakhunter.ignore.*n*

LeakHunter 에서 무시할 특정 컬렉션을 지정합니다.

일반 컬렉션의 경우 일반 유형 한정자를 포함하는 구문(예: System.Collections.Generic.List`1)을 사용합니다.

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

이러한 속성의 기본값은 다음과 같습니다. 여기서 *n* 은 0 에서 4 사이의 정수입니다.

```
introscope.agent.leakhunter.ignore.0=org.apache.taglibs.standard.lang.jstl.*
introscope.agent.leakhunter.ignore.1=com.bea.medrec.entities.RecordEJB_xwcp6o__WebLogic_CMP_RDBMS
introscope.agent.leakhunter.ignore.2=net.sf.hibernate.collection.*
introscope.agent.leakhunter.ignore.3=org.jnp.interfaces.FastNamingProperties
introscope.agent.leakhunter.ignore.4=java.util.SubList
```

3. 에이전트 프로필의 변경 사항을 저장합니다.

중요! *IntroscopeAgent.profile* 에는 LeakHunter 에서 무시되는 패키지를 제어하는 속성이 포함되어 있습니다. 이러한 속성은 기본적으로 사용되도록 설정되어 있습니다. 이러한 속성을 주석으로 처리할 경우 예외가 에이전트 로그에 보고되지 않습니다.

성능 저하의 원인이 되는 컬렉션 무시

컬렉션의 *size()* 메서드가 수집의 개체 수에 비례하는 시간 동안 실행되면 성능이 저하됩니다. 즉, 컬렉션의 *size()* 메서드가 실행되는 시간이 오래 걸릴수록(예: 잘못 구현된 LinkedList 에서 목록의 각 요소를 트래버스하여 목록의 크기 및 요소의 수를 가져오는 경우) 응용 프로그램 성능에 부정적인 영향을 줍니다.

이러한 수집은 LeakHunter 속성 구성에 설명된 것처럼 IntroscopeAgent.profile 의 무시 속성을 사용하여 무시되어야 합니다.

LeakHunter 실행

LeakHunter 는 에이전트 확장으로 실행됩니다.

LeakHunter 를 실행하려면

- JVM AutoProbe 사용: LeakHunter 를 사용하도록 설정한 후 응용 프로그램을 다시 시작합니다.
- 명령줄 ProbeBuilder 사용: *leakhunter.pbd* 와 이전에 사용한 .pbd 를 사용하여 관리되는 응용 프로그램을 다시 계측합니다. 관리되는 응용 프로그램을 다시 시작합니다.
- ProbeBuilder 마법사 사용: ProbeBuilder 마법사를 실행하고 사용자 지정 pbd 목록에서 *leakhunter.pbd* 를 선택합니다. 사용할 새 .jar 를 구현한 후 관리되는 응용 프로그램을 다시 시작합니다.

컬렉션 ID 를 사용하여 잠재 누수 확인

LeakHunter 는 Investigator 트리의 메트릭 데이터를 로그 파일의 데이터와 연결하고 응용 프로그램 전체에서 안정적인 이름을 제공할 수 있는 고유한 컬렉션 ID 로 잠재 누수 각각을 식별합니다.

고유 수집 ID 는 다음 유형 하나를 기초로 특정 구문을 갖습니다.

<method>-<4 digit hash code>#<unique number>

<field>-<4 digit hash code>#<unique number>

- *<method>*: 컬렉션이 할당된 메서드의 이름
- *<field>:* 컬렉션이 할당된 필드 이름
- *<4 digit hash code>*: 메서드 또는 필드 이름을 포함하는 클래스의 전체 이름을 나타내는 해시 코드
- *#<unique number>:* 에이전트를 실행하는 동안 컬렉션 ID 를 고유하게 식별할 수 있도록 메서드 및 해시 코드가 동일한 잠재 누수에 추가되는 번호

잠재 누수에 대한 수집 ID 는 전체 응용 프로그램 실행에서 안정적입니다.

다음은 이러한 컬렉션 ID 의 예입니다.

```
theLookupTable-6314#1
getLoginID-1234#1
getLoginID-1234#2
getLoginID-1234#3
verifyCart-5678#1
verifyCart-0012#1
```

LeakHunter 로그 파일

LeakHunter 로그 파일은 관리되는 응용 프로그램에서 Introscope LeakHunter 에 의해 식별된 잠재 누수에 대한 정보를 수록하고 있습니다. 각 항목에는 잠재 누수에 대한 정보가 수록되어 있습니다. LeakHunter 로그 파일에는 네 가지 시나리오에 대한 항목이 들어 있습니다.

- 잠재 누수가 처음 식별된 경우 - 처음 식별된 잠재 누수 로그 항목 (see page 178) 참조
- 식별된 누수가 누수를 멈춘 것으로 나타나는 경우 - 누수를 멈춘 식별된 잠재 누수 로그 항목 (see page 179) 참조
- 이전에 식별된 누수가 다시 누수되기 시작한 것으로 나타나는 경우 - 다시 누수되기 시작한 식별된 잠재 누수 로그 항목 참조
- LeakHunter 시간 만료가 발생한 경우 - LeakHunter 시간 만료 로그 항목 참조

처음 식별된 잠재 누수 로그 항목

이 유형의 LeakHunter 로그 항목에는 처음 식별된 잠재 누수에 대한 다음과 같은 정보가 포함됩니다.

- 현재 타임스탬프(로그에 기록된 시간)
- 컬렉션 ID
- 컬렉션 클래스
- 컬렉션 할당 방법
- 컬렉션 할당 시간
- 컬렉션 할당 스택 추적
- 컬렉션이 할당된 필드 이름
- 컬렉션의 현재 크기

참고: LeakHunter 로그 파일에 기록된 누수된 컬렉션의 현재 크기는 동적으로 업데이트되지 않습니다. 로그 파일에서는 누수가 처음 식별되었을 때의 누수 크기를 식별할 수 있습니다. 누수된 연결의 크기에 대한 최신 정보를 확인하려면 Introscope Workstation 에서 "누수" 탭을 클릭하십시오.

예제: 잠재 누수가 감지된 경우의 로그 파일 항목

다음 예제에서는 잠재 누수가 처음 식별된 경우 Java 로그 파일의 항목을 보여 줍니다.

```
5/2/09 9:55:06 AM PDT
Potential leak identified
Assigned ID: testInst-2604#1
Collection Class: java.util.Vector
Allocation Method: sonOfLH_test.testInst()
Allocation Timestamp: 5/2/09 9:54:21 AM PDT
Allocation Stack Trace:
Unknown(알 수 없음)
Field Name(s):
sonOfLH_test.v3
sonOfLH_test.v4
sonOfLH_test.v5
Current Size: 44
```

누수를 멈춘 식별된 잠재 누수 로그 항목

이 유형의 LeakHunter 로그 항목에는 누수를 멈춘 잠재 누수에 대한 다음과 같은 정보가 포함됩니다.

- 현재 타임스탬프(로그에 기록된 시간)
- 컬렉션 ID
- 컬렉션 클래스
- 컬렉션의 현재 크기

예제: 잠재 누수의 누수가 멈춘 경우의 로그 파일 항목

다음 예제에서는 잠재 누수의 누수가 멈춘 것처럼 보일 때 Java 로그 파일의 항목을 보여 줍니다.

```
4/27/10 1:18:12 PM PDT
Potential leak no longer appears to be leaking
Assigned ID: createNewInstance-2815#3
Collection Class: java.util.HashMap
Current Size: 70
```

다시 누수되기 시작한 식별된 잠재 누수 로그 항목

이 유형의 항목에는 다시 누수를 시작한 잠재 누수에 대한 다음과 같은 정보가 포함됩니다.

- 현재 타임스탬프(로그에 기록된 시간)
- 컬렉션 ID
- 컬렉션 클래스
- 컬렉션의 현재 크기

예제: 잠재 누수의 누수가 다시 시작된 경우의 로그 파일 항목

다음 예제에서는 잠재 누수의 누수가 다시 시작된 것처럼 보일 때 Java 로그 파일의 항목을 보여 줍니다.

```
4/27/10 1:21:42 PM PDT
Potential leak appears to be leaking again
Assigned ID: createNewInstance-2815#3
Collection Class: java.util.HashMap
Current Size: 79
```

LeakHunter 시간 만료 로그 항목

이 유형의 LeakHunter 로그 항목에는 계속 추적할 잠재 누수의 수가 포함됩니다.

예: 시간 만료가 발생할 때 로그 파일 항목

```
LeakHunter timeout occurred at 4/27/10 1:32:12 PM PDT
LeakHunter will only continue to track the 3 potential leaks
```

LeakHunter 사용

LeakHunter 를 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

ErrorDetector

Introscope ErrorDector 를 통해 응용 프로그램 지원 담당자는 사용자가 웹 트랜잭션을 완료하는 데 문제가 되는 심각한 오류를 확인하고 오류 원인을 진단할 수 있습니다. 이러한 종류의 응용 프로그램 가용성 문제가 발생하면 대개 "404 찾을 수 없음"을 비롯한 오류 메시지가 표시되지만 이러한 오류 메시지에는 IT 담당자가 문제의 근본적인 원인을 찾는 데 필요한 구체적인 정보가 들어 있지 않습니다. Introscope ErrorDetector 를 사용하면 이러한 심각한 오류를 응용 프로그램에서 실시간으로 모니터링하고, 오류의 빈도와 원인을 확인할 수 있으며, 근본적인 원인에 대한 구체적인 정보를 개발자에게 전달할 수 있습니다.

ErrorDetector 는 심각한 응용 프로그램 오류의 근본적인 원인 분석을 통해 뛰어난 트랜잭션 무결성을 제공하고 최고의 사용자 환경을 보장하는 유일한 응용 프로그램 관리 솔루션입니다.

Introscope ErrorDetector 를 통해 IT 팀은 다음과 같은 작업을 수행할 수 있습니다.

- 비정상적인 트랜잭션의 빈도 확인
- 기록된 예외가 사용자에게 영향을 주는지 여부 확인
- 트랜잭션 경로 중 오류가 발생한 정확한 위치 확인
- 심각한 오류를 재현, 진단 및 제거하는 데 필요한 정보 파악

Introscope ErrorDetector 는 Introscope 와 통합되어 Introscope Workstation 에서 오류를 모니터링할 수 있는 기능을 제공합니다. 응용 프로그램 오류가 발생하면 라이브 오류 뷰어를 사용하여 각 오류에 대한 상세한 정보를 검토할 수 있습니다.

오류의 유형

CA Technologies 는 J2EE 사양에 수록된 정보에 기초하여 "심각한" 오류를 기술하는 일련의 조건을 정의했습니다. ErrorDetector 는 오류와 예외를 모두 오류로 간주합니다. 가장 일반적인 유형의 오류는 throw 된 예외입니다.

일반적인 오류의 예는 다음과 같습니다.

- HTTP 오류(404, 500 등)

  참고: 경우에 따라 HTTP 404 오류는 응용 프로그램 서버가 아니라 웹 서버에서 발생합니다. 이 경우 ErrorDetector 는 에이전트를 통해 웹 서버 오류를 감지할 수 없습니다.

- SQL 문 오류
- 네트워크 연결 오류(시간 만료 오류)
- 백엔드 오류(예: JMS 를 통해 메시지를 보낼 수 없거나 메시지 큐에 메시지를 쓸 수 없는 오류)

CA Technologies 에서 중요한 오류라고 간주하는 항목이 사용자가 중요한 오류라고 생각하는 항목과 다를 수 있습니다. ErrorDetector 가 추적하는 오류 중에 중요하지 않다고 간주하는 오류가 있는 경우 무시하도록 선택할 수 있습니다. 추적하고자 하는 추가 오류가 있는 경우 오류 추적 프로그램을 사용하여 추가 오류를 추적하는 새 지시문을 만들 수 있습니다.

ErrorDetector 작동 방식

Introscope 에는 에이전트 설치 시 설치되는 *errors.pbd* 라는 PBD(ProbeBuilder 지시문) 파일이 포함되어 있습니다. 이 PBD 의 추적 프로그램은 심각한 오류를 캡처합니다.

ErrorDetector 는 에이전트 설치 시 자동으로 설치됩니다. ErrorDetector 가 설치된 후 Introscope 에서 *errors.pbd* 를 사용하도록 구성하고 ErrorDetector 기능이 사용되도록 설정하십시오. ProbeBuilder 마법사 또는 명령줄 ProbeBuilder 를 사용하는 경우에는 이전에 사용된 PBD 파일 외에도 *errors.pbd* 를 사용하여 응용 프로그램을 다시 계측해야 합니다.

에이전트는 *errors.pbd* 파일에 정의된 대로 오류 정보를 수집합니다.

Workstation 에서는 다음을 볼 수 있습니다.

- Investigator 에서 오류 메트릭 데이터를 볼 수 있습니다.
- "라이브 오류 뷰어"에서 라이브 오류를 볼 수 있습니다.
- 오류 발생 과정에 대한 구성 요소 수준의 정보를 보여 주는 "오류 스냅숏"에서 오류 세부 정보를 볼 수 있습니다.

ErrorDetector 는 트랜잭션 추적 프로그램과 통합되어 있으므로 트랜잭션 경로의 컨텍스트 내에서 심각한 오류가 발생한 원인 및 과정을 정확히 확인할 수 있습니다. 또한 모든 오류 및 트랜잭션이 트랜잭션 이벤트 데이터베이스에 보관되므로 기록 데이터를 분석하여 추세를 파악할 수 있습니다.

Introscope 는 트랜잭션을 서비스에 대한 호출 또는 처리로 정의합니다. 웹 응용 프로그램 컨텍스트에서 트랜잭션은 웹 브라우저에서 보낸 URL 에 대한 호출 및 처리입니다. 웹 서비스 컨텍스트에서 트랜잭션은 SOAP 메시지에 대한 호출 및 처리입니다.

Java Agent 에서 ErrorDetector 사용

Java Agent 가 오류 데이터를 캡처할 수 있도록 하려면 *IntroscopeAgent.profile* 에서 *introscope.agent.errorsnapshots.enable* 속성을 true 로 설정해야 합니다. 기본적으로 Java Agent 는 오류 데이터를 캡처하도록 설정되어 있습니다.

Java Agent 에서 ErrorDetector 데이터 캡처를 사용 또는 사용하지 않도록 설정하려면

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.
2. *introscope.agent.errorsnapshots.enable* 속성이 *true* 로 설정되어 있는지 확인합니다.

   참고: ErrorDetector 를 사용하지 않으려면 이 속성을 *false* 로 설정하십시오.
3. ProbeBuilder 마법사를 사용하는 경우 *errors.pbd* 파일을 *<EM_Home>*/config/custompbd 디렉터리에 복사하고 응용 프로그램을 다시 계측합니다.
4. 에이전트 프로필을 저장합니다.

ErrorDetector 옵션 구성

ErrorDetector 를 구성하여 에이전트가 Enterprise Manager 로 보내는 최대 오류 수를 제한하거나 무시할 오류를 지정할 수 있습니다.

ErrorDetector 를 사용하면 과도한 오버헤드 없이 오류 데이터를 캡처할 수 있습니다. 기본 제공 스로틀은 15 초당 10 개의 오류로 설정되어 있습니다. 이 기간 동안 더 많은 오류를 캡처하기 위해 스로틀을 늘릴 수 있지만 이 경우 오버헤드가 증가할 수 있습니다.

ErrorDetector 스로틀을 변경하려면(선택 사항)

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.
2. *introscope.agent.errorsnapshots.throttle* 속성에 새 값을 입력합니다.
3. 에이전트 프로필을 저장합니다.

   참고: 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

추적하지 않을 오류를 무시하도록 에이전트를 구성할 수 있습니다. 오류를 "태그 지정"하기 위해 지정하는 정보는 정확한 오류 메시지이거나 "와일드카드" 별표 문자와 메시지의 일부일 수 있습니다.

특정 오류를 무시하려면(선택 사항)

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.
2. *introscope.agent.errorsnapshots.ignore* 속성 값으로 원하는 오류 유형을 식별할 수 있는 정보를 정의합니다.

   예를 들어 다음 ignore 속성은 오류 메시지에 "IOException"이라는 용어가 포함된 모든 오류를 무시합니다.

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   ```
3. 추가 오류를 무시하려면 추가 ignore 속성을 순차적으로 추가합니다. 예를 들어 두 가지 오류 유형을 무시하려면 다음과 같이 속성을 설정합니다.

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code *500*
   ```
4. 에이전트 프로필의 변경 사항을 저장합니다.

   참고: 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

고급 오류 데이터 캡처

ErrorDetector 는 기본적으로 일반적인 여러 오류 유형을 캡처하지만 CA Technologies 에서는 필요에 맞게 오류 감지 메커니즘을 사용자 지정하기 위한 옵션을 제공합니다.

다음의 오류 관련 추적 프로그램을 사용하여 오류를 캡처하는 PBD(ProbeBuilder 지시문)를 생성할 수 있습니다.

- **ExceptionErrorReporter** - 표준 예외를 보고합니다.
- **ThisErrorReporter** - 현재 개체를 오류로 보고합니다.
- **HTTPErrorCodeReporter** - HTTP 오류 코드와 관련 오류 메시지를 캡처합니다.
- **MethodCalledErrorReporter** - 특정 메서드가 호출되었는지 여부를 보고합니다.

새로 생성하는 지시문은 이러한 추적 프로그램과 함께 *<Agent_Home>*/wily 디렉터리의 *errors.pbd* 파일에 포함해야 합니다.

새 오류 유형 정의

오류는 PBD 를 사용하여 ErrorDetector 에 대해 정의됩니다. 오류의 존재 여부를 검사하고 오류 메시지를 캡처(구성)하는 여러 특수 추적 프로그램이 있습니다. 오른쪽에 이러한 추적 프로그램을 배치하면 응용 프로그램 또는 그 인프라의 새 오류 유형에 대해 ErrorDetector 를 학습시킬 수 있습니다.

ExceptionErrorReporter

ExceptionErrorReporter 추적 프로그램을 사용하여 계측된 메서드에서 throw 되는 예외를 확인할 수 있습니다. 예외가 throw 되면 이 추적 프로그램은 이를 오류로 처리하고 예외에서 오류 메시지를 가져옵니다. 이 메시지는 오류에 대한 가장 일반적인 정의입니다.

오류 메시지를 캡처하려면 ExceptionErrorReporter 추적 프로그램을 "...WithParameters" 지시문과 함께 사용해야 합니다. 예:

TraceOneMethodWithParametersOfClass: com.bank.CustomerAccount getBalance ExceptionErrorReporter "CustomerAccount:Errors Per Interval"

이 지시문은 *CustomerAccount* 의 *getBalance()* 메서드에서 throw 되는 예외가 오류로 간주되도록 지정합니다.

간격 메트릭별로 오류로 계측하려면 "...WithParameters" 지시문을 사용해야 하지만 임의의 메서드에 대해 "...WithParameters" 지시문을 한 번만 지정하면 해당 메서드에 대한 모든 추적 프로그램이 이 매개 변수를 사용할 수 있게 됩니다. 예를 들어 다음을 지정할 수 있습니다.

TraceOneMethodWithParametersOfClass: com.myClass myMethod BlamePointTracer

이 지시문은 *com.myClass myMethod* 메서드에 대한 매개 변수를 *ExceptionErrorReporter* 추적 프로그램을 포함한 다른 추적 프로그램에서 사용할 수 있도록 합니다.

MethodCalledErrorReporter

MethodCalledErrorReporter 추적 프로그램은 메서드에 사용되며, 이 메서드가 호출되는 것 자체가 오류가 발생했음을 의미합니다. 예:

TraceOneMethodOfClass: com.bank.CheckingAccount cancelCheck MethodCalledErrorReporter "CustomerAccount:Canceled Checks Per Interval"

이 지시문은 cancelCheck() 메서드가 (어떠한 이유로 인해) 호출될 때마다 이것을 오류로 지정합니다. 오류 메시지는 단순히 호출된 클래스 및 메서드가 기술합니다.

예외 또는 오류를 throw 하는 메서드를 모르는 경우 ThisErrorReporter 추적 프로그램을 사용해 보십시오.

ThisErrorReporter

ThisErrorReporter 추적 프로그램은 *MethodCalledErrorReporter* 와 유사하지만 계측된 개체에 대해 *toString()*을 호출하여 오류 메시지를 생성합니다. 이 추적 프로그램은 예외 클래스의 생성자에 사용하는 것이 가장 효과적입니다. 예:

```
TraceOneMethodWithParametersOfClass: ezfids.util.exception.EasyFidsException [set the init variable for your book] ThisErrorReporter "Exceptions|{packageandclassname}:Errors Per Interval"
```

참고: 오류 메시지를 캡처하려면 *ThisErrorReporter* 추적 프로그램을 "...WithParameters" 지시문과 함께 사용해야 합니다.

이 지시문은 *InvalidPINException* 의 생성자("init" 또는 ".ctor")가 호출될 때마다 오류를 발생시킵니다. 오류 메시지는 *InvalidPINException* 에 *toString()*을 호출하여 결정되며, 일반적으로 응용 프로그램 개발자가 지정한 오류 메시지가 반환됩니다.

이 추적 프로그램은 고유한 예외 유형에 기반한 사용자 지정 오류 관리 시스템을 사용하는 경우에 유용합니다.

참고: Introscope 는 *java.** 패키지에 있는 어떠한 코드도 계측할 수 없으므로 *java.lang.Exception* 또는 *java.sql.SQLException* 에 이 추적 프로그램을 배치해도 효과가 없습니다.

HTTPErrorCodeReporter

HTTPErrorCodeTracer 추적 프로그램은 서블릿 및 JSP 로부터 오류 코드를 보고합니다. 이것은 다음의 인시던트를 카운트하는 간격당 카운터입니다.

- 400 이상의 HTTP 응답 코드
- Java 환경에서 코드 400 또는 이상에 대해 *sendError* 또는 *setStatus* 의 *javax.servlet.http.HttpServletResponse* 하위 클래스 호출

사용법 예제는 *errors.pbd* 를 참조하십시오.

오류 추적 프로그램 지시문 사용에 대한 주의

이전 단원에 설명되어 있는 것처럼 추적 프로그램을 사용할 때는 주의를 기울여야 합니다. 가장 좋은 방법은 오류 추적과 관련된 오버헤드를 고려하여 백엔드 시스템에서 발생하는 복구 불가능한 문제와 같이 심각한 문제만 보고하는 것입니다.

기본 *errors.pbd* 는 가능한 한 오버헤드를 최소화하면서 심각한 오류를 보고하도록 설계되었습니다. 모니터링되는 모든 메서드에 ExceptionErrorReporter 를 적용하는 경우처럼 오류 추적을 과도하게 사용하면 오탐지의 양이 크게 증가할 수 있습니다. 예를 들어 사용자가 숫자 필드에 "California"를 입력하면 심각한 문제로 보고되지 않아도 되는 *NumberFormatException* 이 발생할 수 있습니다.

ErrorDetector 사용

ErrorDetector 를 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

제 9 장: 경계 Blame 구성

이 단원에서는 기본 Java Agent Blame 보고 동작과 관련 구성 옵션에 대해 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

경계 Blame 이해

Blame 기술을 사용하면 관리되는 Java 응용 프로그램에서 작업하면서 응용 프로그램의 프런트엔드 및 백엔드에서 메트릭을 확인 수 있습니다. 경계 Blame 이라고 하는 이 기능을 통해 문제를 심사할 수 있습니다.

Introscope 에서 백엔드를 검색하는 방식은 응용 프로그램에 따라 달라집니다. 데이터베이스 동작의 경우 Introscope 는 SQL 에이전트를 사용하여 백엔드를 검색합니다. SQL 에이전트를 사용할 수 없는 경우 클라이언트/서버 데이터베이스, JMS 서버 및 LDAP 서버와 같은 백엔드는 소켓을 통해 액세스되므로 Introscope 는 백엔드 동작을 검색합니다. Oracle 백엔드가 있으며 Introscope SQL 에이전트를 사용하지 않는 경우에는 경계 Blame 및 Oracle 백엔드 (see page 140)를 참조하십시오.

경계 Blame 이 Introscope Investigator 에 표시되는 방식에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

URL 그룹 사용

URL 그룹을 사용하여 경로 접두사가 정의하는 문자열로 시작하는 요청 집합에 대한 브라우저 응답 시간을 모니터링할 수 있습니다. 경로 접두사는 호스트 이름 뒤에 오는 URL 부분입니다. 예를 들어 다음과 같은 URL 을 가정합니다.

http://burger1.com/testWar/burgerServlet?ViewItem&category=11776&item=5550662630&rd=1

여기서 경로 접두사는 다음과 같습니다.

/testWar

URL 의 경로 접두사에서 파생될 수 있는 모든 유용한 요청 범주를 위한 URL 그룹을 정의할 수 있습니다. 예를 들어 응용 프로그램 URL 의 형식에 따라 응용 프로그램이 지원하는 각 고객에 대한 URL 그룹, 각 주요 응용 프로그램에 대한 URL 그룹, 하위 응용 프로그램에 대한 URL 그룹을 정의할 수 있습니다. 이렇게 하면 동의한 서비스 수준의 관점에서 또는 응용 프로그램의 업무상 중요한 부분에 대해 성능을 모니터링할 수 있습니다.

예: URL 그룹 속성이 정의되는 방법

다음 예는 Java Agent 프로필에서 발췌한 내용으로, URL 그룹이 정의되는 방법을 보여 줍니다.

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
introscope.agent.urlgroup.group.alpha.pathprefix=/testWar
introscope.agent.urlgroup.group.alpha.format=foo {host} bar {protocol} baz {port}
quux {query_param:foo} red {path_substring:2:5} yellow {path_delimited:/:0:1} green
{path_delimited:/:1:4} blue {path_substring:0:0}
introscope.agent.urlgroup.group.beta.pathprefix=/nofilterWar
introscope.agent.urlgroup.group.beta.format=nofilter foo {host} bar {protocol} baz
{port} quux {query_param:foo} red {path_substring:2:5} yellow {path_delimited:/:0:1}
green {path_delimited:/:1:4} blue {path_substring:0:0}
introscope.agent.urlgroup.group.gamma.pathprefix=/examplesWebApp
introscope.agent.urlgroup.group.gamma.format=Examples Web App
```

URL 그룹에 대한 키 정의

introscope.agent.urlgroup.keys 속성은 모든 URL 그룹의 ID 또는 키 목록을 정의합니다. URL 그룹의 특성을 선언하는 다른 속성 정의에서 URL 그룹의 키를 참조합니다. 다음 예는 3 개 URL 그룹에 대한 키를 정의합니다.

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
```

URL 그룹을 정의하여 일부 URL 이 여러 그룹에 속하는 경우 속성에서 URL 그룹에 대한 키를 나열하는 순서는 중요합니다. 그룹 구성원이 더 적은 URL 그룹은 그룹 구성원이 더 많은 URL 그룹보다 앞에 와야 합니다. 예를 들어, 키가 **alpha** 인 IP 그룹에 경로 접두사 */examplesWebApp* 가 있고 키가 **delta** 인 URL 그룹에 경로 접두사 /examplesWebApp/cleverones 가 있는 경우, *keys* 매개 변수에서 *delta* 가 *alpha* 앞에 와야 합니다.

각 URL 그룹의 구성원 정의

속성 *introscope.agent.urlgroup.group.***groupKey**.*pathprefix* 는 URL 의 경로 접두사가 일치되는 패턴을 지정하여 어떤 요청이 URL 그룹 내 속하게 되는지 정의합니다.

예: 그룹 키를 URL 경로 접두사에 매핑

이 속성 정의는 URL 의 경로 부분이 */testWar* 로 시작하는 모든 요청을 키가 *alpha* 인 URL 그룹에 할당합니다.

```
introscope.agent.urlgroup.group.alpha.pathprefix=/testWar
```

지정된 *pathprefix* 와 일치하는 요청에는 다음이 포함됩니다.

```
http://burger1.com/testWar/burgerServlet?ViewItem&category=
11776&item=5550662630&rd=1
http://burger1.com/testWar/burgerServlet?Command=Order&item=5550662630
```

예: 응용 프로그램 경로별로 URL 그룹 만들기

콜 센터 서비스를 제공하는 회사가 각 응용 프로그램 기능에 대한 URL 그룹을 설정하여 기능 영역별로 응답 시간을 모니터링할 수 있습니다. 고객이 다음 URL 을 사용하여 서비스에 액세스한다고 가정합니다.
*http://genesystems/us/*application_function/

여기서 *application_function* 은 *OrderEntry*, *AcctService*, *Support* 등의 응용 프로그램을 나타내고, 각 URL 그룹의 *pathprefix* 속성은 적절한 *application_function* 을 지정합니다.

참고: *pathprefix* 속성에서 별표 기호(*)를 와일드카드로 사용할 수 있습니다.

URL 그룹의 이름 정의

속성 *introscope.agent.urlgroup.group.groupKey.format* 은 키가 *groupKey* 인 URL 그룹에 대한 응답 시간 메트릭이 Introscope Workstation 에 표시되는 이름을 결정합니다.

일반적으로 URL 에 대한 이름으로 텍스트 문자열을 할당하는 데 *format* 속성을 사용합니다. 다음 예제는 키가 *alpha* 인 URL 그룹에 대한 메트릭이 *Alpha Group* 이란 이름으로 Workstation 에 표시되도록 합니다.

introscope.agent.urlgroup.group.alpha.format=Alpha Group

URL 그룹에 대한 고급 명명 기술(선택 사항)

서버 포트, 프로토콜과 같은 요청 요소로부터, 또는 요청 URL 의 부분 문자열로부터 URL 그룹 이름을 파생시킬 수 있습니다. 요청을 검사하여 응용 프로그램 모듈이 쉽게 구분되는 경우 이 방법이 유용합니다. 이 단원에서는 *format* 속성의 고급 형식에 대해 설명합니다.

URL 그룹 이름으로 호스트 사용

요청과 관련된 HTTP 서버의 호스트 이름을 반영하는 이름으로 URL 그룹에 대한 메트릭을 구성하려면 다음과 같이 *format* 매개 변수를 정의하십시오.

introscope.agent.urlgroup.group.alpha.format={host}

*format={host}*인 경우, 이러한 요청의 통계는 각각 메트릭 이름 *us.mybank.com* 및 *uk.mybank.com* 으로 표시됩니다.

https://us.mybank.com/mifi/loanApp.....

https://uk.mybank.com/mifi/loanApp.....

URL 그룹 이름으로 프로토콜 사용

요청과 관련된 프로토콜을 반영하는 URL 그룹에 대한 통계를 구성하려면 다음과 같이 *format* 매개 변수를 정의하십시오.

introscope.agent.urlgroup.group.alpha.format={protocol}

*format={protocol}*인 경우, 통계가 요청 URL 의 프로토콜 부분에 해당하는 메트릭 이름으로 Investigator 에 그룹화됩니다. 예를 들어 다음과 같은 요청의 통계는 메트릭 이름 *https* 아래에 나타납니다.

https://us.mybank.com/cgi-bin/mifi/scripts.....

https://uk.mybank.com/cgi-bin/mifi/scripts.....

URL 그룹 이름으로 포트 사용

요청과 관련된 포트를 반영하는 URL 그룹에 대한 통계를 구성하려면 다음과 같이 *format* 매개 변수를 정의하십시오.

introscope.agent.urlgroup.group.alpha.format={port}

*format={port}*인 경우, 통계는 요청 URL 의 포트 부분에 해당하는 이름으로 그룹화됩니다. 예를 들어 다음과 같은 요청의 통계는 이름 *9001* 아래에 나타납니다.

https://us.mybank.com:9001/cgi-bin/mifi/scripts.....

https://uk.mybank.com:9001/cgi-bin/mifi/scripts.....

URL 그룹 이름으로 매개 변수 사용

요청과 관련된 매개 변수의 값을 반영하는 메트릭 이름으로 Investigator 에 URL 그룹에 대한 통계를 구성하려면 다음과 같이 *format* 매개 변수를 정의하십시오.

```
introscope.agent.urlgroup.group.alpha.format={query_param:param}
```

*format={query_param:*param*}*인 경우, 통계가 지정된 매개 변수의 값에 해당하는 메트릭 이름으로 Investigator 에 그룹화됩니다. 매개 변수 없는 요청은 <empty>에 나열됩니다. 예를 들어, 매개 변수 정의가 아래와 같은 경우:

```
introscope.agent.urlgroup.group.alpha.format=
{query_param:category}
```

이러한 요청에 대한 통계는 메트릭 이름 "734"로 표시됩니다.

```
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=3772&tc=photo
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=8574&tc=photo
```

요청 경로의 부분 문자열을 URL 그룹 이름으로 사용

요청 URL 의 경로 부분의 부분 문자열을 반영하는 이름에 URL 그룹에 대한 통계를 구성하려면 다음과 같이 *format* 매개 변수를 정의하십시오.

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:m:n}
```

여기서 *m* 은 첫 번째 문자의 인덱스이고, *n* 은 마지막 문자의 인덱스에 1 을 더한 숫자입니다. 문자열 선택은 *java.lang.String.substring()* 메서드와 같이 동작합니다. 예를 들어 다음과 같이 설정되었다고 가정해 보겠습니다.

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:0:3}
```

다음 요청에 대한 통계는 메트릭 노드 */ht* 아래에 표시됩니다.

```
http://research.com/htmldocu/WebL-12.html
```

요청 경로의 구분된 부분을 URL 그룹 이름으로 사용

문자로 구분된 부분의 요청 URL 경로를 반영하는 이름 아래에 URL 그룹에 대한 통계를 구성하려면 *format* 매개 변수를 다음과 같이 정의하십시오.

```
introscope.agent.urlgroup.group.alpha.format=
{path_delimited:delim_char:m:n}
```

여기서 *delim_char* 는 경로의 세그먼트를 구분하는 문자이고, *m* 은 선택할 첫 번째 세그먼트의 인덱스이며, *n* 은 선택할 마지막 세그먼트의 인덱스보다 1 이 큰 값입니다. 예를 들어 다음과 같이 설정되었다고 가정해 보겠습니다.

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/:2:4}
```

또한 다음 형식의 요청에 대한 통계가 있다고 가정해 보겠습니다.

```
http://www.buyitall.com/userid,sessionid/pageid
```

이 통계는 메트릭 이름 */pageid* 아래에 나타납니다.

다음 사항에 유의하십시오.

- 구분 기호 문자(이 예의 경우 슬래시(/))는 한 세그먼트로 간주됩니다.
- 세그먼트 번호는 0 부터 시작합니다.

이 URL 예제에서 슬래시 문자로 구분된 세그먼트는 다음과 같습니다.

0=*/*, 1=*userid,sessionid*, 2=*/*, and 3=*pageid*

여러 구분 기호를 필요한 대로 지정할 수 있습니다. 예를 들어 다음과 같이 설정되었다고 가정해 보겠습니다.

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/,:3:4}
```

위에 표시된 형식의 요청에 대한 통계는 메트릭 이름 *sessionid* 아래에 나타나며, 이 URL 예제에서 슬래시 및 쉼표로 구분된 세그먼트는 다음과 같습니다.

0=*/*, 1=*userid*, 2=*,* 3=*sessionid*, 4=*/*, and 5=*pageid*

URL 그룹에 대한 다중 명명 방법 사용

아래에 표시된 것처럼 여러 개의 이름 지정 방법을 단일 *format* 문자열에 결합할 수 있습니다.

```
introscope.agent.urlgroup.group.alpha.format=red {host} orange {protocol} yellow
{port} green {query_param:foo} blue {path_substring:2:5} indigo
{path_delimited:/:0:1} violet {path_delimited:/:1:4} ultraviolet
{path_substring:0:0} friend computer
```

URLGrouper 실행

URLGrouper 는 일반적인 형식으로 웹 서버 로그 파일을 분석하는 명령줄 유틸리티입니다. URLGrouper 를 사용하면 사용자 고유의 URL 그룹을 손쉽게 정의할 수 있습니다.

참고: URLGRouper 유틸리티를 사용하여 웹 서버 로그 파일을 분석할 수 있습니다. URLGrouper 는 웹 서버 로그 파일의 내용을 기반으로 잠재적 URL 그룹에 대한 일련의 속성 설정을 출력합니다. URLGrouper 에서는 별표 기호(*)를 와일드카드로 사용할 수 있습니다.

URLGrouper 를 실행하려면

1. 명령 셸을 엽니다.
2. 다음 명령을 입력합니다.
   ```
   java -jar urlgrouper.jar logfile
   ```
   여기서 *logfile* 은 웹 서버 로그 파일의 전체 경로입니다.
3. 일련의 URL 그룹에 대한 속성 정의는 STDOUT 에 출력됩니다.
4. 제안된 URL 그룹을 구성하려면 URLGrouper 에서 생성된 속성 문을 *IntroscopeAgent.profile* 에 복사합니다.

Blame 추적 프로그램 사용

추적 프로그램을 사용하여 응용 프로그램의 프런트엔드 및 백엔드를 명시적으로 표시할 수 있습니다. 자세한 내용은 Blame 추적 프로그램을 사용하여 Blame 지점 표시 (see page 137)를 참조하십시오.

제 10 장: 트랜잭션 추적 옵션 구성

이 단원에서는 기본 트랜잭션 추적 동작과 관련 구성 옵션에 대한 정보를 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

새로운 트랜잭션 추적 모드

CA APM Introscope 9.1 에서는 새로운 에이전트 트랜잭션 아키텍처와 추적 프로그램이 도입되었습니다. 새로운 트랜잭션 추적 모드는 이전 추적 프로그램이 기존 릴리스에 사용하던 트랜잭션 Blame 스택을 대체합니다. 새 모드 구현은 에이전트 성능을 개선하기 위해 계산 및 처리를 최적화합니다.

CA APM Introscope 9.1 을 사용하면 트랜잭션 Blame 스택으로 이전처럼 에이전트를 구성 및 실행할 수 있습니다. 에이전트의 "레거시" 모드가 완전하게 지원되지만 이후 릴리스의 개선된 에이전트 기능은 새 모드에서만 구현될 것입니다. "레거시" 모드로 에이전트를 실행할 경우 다음과 같은 기능을 사용할 수 없습니다.

- 버전 9.1 의 에이전트 CPU 사용 및 응답 시간 최적화
- .NET 에이전트에 대한 동적 계측
- SQL 에이전트 속성

  ```
  introscope.agent.sqlagent.sql.artonly
  ```

  ```
  introscope.agent.sqlagent.sql.turnoffmetrics
  ```

중요! CA APM 에이전트에서 새 모드로 레거시 추적 프로그램을 실행하는 것을 "혼합 모드" 구성이라고 합니다. 메모리가 많이 소모되고 서비스가 중단될 수 있으므로 혼합 모드로 실행하지 **마십시오**.

레거시 모드 추적 프로그램을 실행하는 경우 레거시 모드로 에이전트를 실행하십시오. 레거시 추적 프로그램이 있음을 고객에게 알리기 위해 다음과 같은 내용의 메시지가 에이전트 로그에 기록됩니다.

"An agent tracer using legacy API's has been detected. Running legacy tracers with the agent in new mode is not recommended. Please contact support who can refer you to documentation on how to upgrade your legacy tracers. In the interim, please configure the agent to use legacy mode or use the pre-configured version of the legacy agent package. For example, the legacy package for the Oracle WebLogic agent on UNIX is IntroscopeAgentFiles-Legacy-NoInstallerx.x.x.xweblogic.unix.tar"
(레거시 API 를 사용하는 에이전트 추적 프로그램이 감지되었습니다. 에이전트에서 새 모드로 레거시 추적 프로그램을 실행하지 않는 것이 좋습니다. 레거시 추적 프로그램 업데이트 방법에 대한 설명서를 알려 줄 수 있는 지원 담당자에게 문의하십시오. 그동안에는 레거시 모드를 사용하거나 사전 구성된 버전의 레거시 에이전트 패키지를 사용하도록 에이전트를 구성하십시오. 예를 들어 UNIX 에서 Oracle WebLogic 에이전트용 레거시 패키지는 IntroscopeAgentFiles-Legacy-NoInstallerx.x.x.xweblogic.unix.tar 입니다.)

새 모드 추적 프로그램을 실행할 때까지는 에이전트를 레거시 모드에서 실행되도록 되돌려 놓으십시오.

기본 제공되는 다음 확장을 사용하는 경우 에이전트가 레거시 모드로 실행되도록 구성하십시오.

- CA APM for IBM Websphere Portal
- CA APM for IBM CICS Transaction Gateway
- CA APM for IBM z/OS
- CA APM for CA SiteMinder Web Access Manager
- CA APM Integration for CA LISA

레거시 모드 트랜잭션 추적을 사용하도록 에이전트 구성

레거시 모드 트랜잭션 추적으로 변환하는 기능은 새 모드가 있는 버전의 버전 CA APM 에만 적용됩니다. 9.1 이전 버전의 CA APM 에는 새 모드가 없습니다.

레거시 모드를 사용하도록 에이전트를 구성할 때는 두 가지 옵션을 사용할 수 있습니다.

- 미리 구성된 레거시 에이전트 패키지를 배포합니다. 이러한 패키지는 파일 전용 에이전트 패키지로 제공되며, 설치 관리자가 제공되지 않습니다. 예를 들어 UNIX 에서 Oracle WebLogic 에이전트용 레거시 패키지는 다음과 같습니다.

  IntroscopeAgentFiles-Legacy-NoInstaller<*version_number*>weblogic.unix.tar

- 레거시 모드를 사용하도록 수동으로 구성합니다.

다음 단계를 따르십시오.

1. 모니터링되는 응용 프로그램을 중지합니다.
2. 새 로그를 준비하기 위해 <Agent_Home>/logs 디렉터리에 있는 기존 로그 파일을 아카이브하고 삭제합니다.
3. <Agent_Home>/core/config 디렉터리의 기존 .pbl 및 .pbd 파일 백업합니다.
4. 기존 <Agent_Home>/core/config/IntroscopeAgent.profile 을 백업합니다.
5. <Agent_Home>/examples/legacy 디렉터리에 있는 레거시 .pbl 및 .pbd 파일을 <Agent Home>/core/config 디렉터리에 복사합니다.
6. <Agent_Home>/core/config/IntroscopeAgent.profile 을 열고 다음 변경을 수행합니다.
   - 속성 introscope.agent.configuration.old=true 를 추가합니다.
   - 복사된 적절한 레거시 .pbl 및 .pbd 파일을 가리키도록 에이전트 속성 introscope.autoprobe.directivesFile 을 업데이트합니다. 예를 들어, spm.pbl 을 spm-legacy.pbl 로 대체합니다.
7. 모니터링되는 응용 프로그램을 다시 시작합니다.

표준 설치에서 레거시 모드를 사용하도록 특정 CA APM 확장을 구성하려면 다음 표에 나열되어 있는 .pbl 및 .pbd 파일을 참조하십시오.

확장 이름	레거시 표준 pbl 또는 pbd 파일
CA APM for Oracle Weblogic Server	ppweblogic-legacy.pbd spm-legacy.pbl
CA APM for Oracle Weblogic Portal	powerpackforweblogicportal-legacy.pbl spm-legacy.pbl

확장 이름	레거시 표준 pbl 또는 pbd 파일
CA APM for Oracle Service Bus	OSB-typical-legacy.pbl spm-legacy.pbl
CA APM for IBM Websphere Portal	powerpackforwebsphereportal.pbl spm-legacy.pbl
CA APM for IBM Websphere MQ	webspheremq-legacy.pbl
CA APM for IBM Websphere Process Server/Business Process Management	wps-legacy.pbd wesb-legacy.pbd spm-legacy.pbl
CA APM for TIBCO BusinessWorks	tibcobw-typical-legacy.pbl spm-legacy.pbl
CA APM for Software AG Webmethods Integration Server	webmethods-legacy.pbl spm-legacy.pbl
CA APM for CA SiteMinder Web Access Manager	smwebagentext.pbd
CA APM for IBM CICS Transaction Gateway	PPCTGTranTrace.pbd PPCTGServer-typical.pbd PPCTGClient-typical.pbd
CA APM for IBM z/OS	zos-full.pbl
CA APM for CA LISA	lisa-typical.pbl
CA APM for SYSVIEW	CTG_ECI_Tracer_For_SYSVIEW-legacy.pbd HTTP_Tracer_For_SYSVIEW-legacy.pbd WS_Tracer_For_SYSVIEW-legacy.pbd

자동 트랜잭션 추적 동작 제어

자동 트랜잭션 추적은 트랜잭션 추적을 명시적으로 실행하지 않고도 잠재적으로 문제가 있을 수 있는 트랜잭션 유형의 기록 분석을 수행할 수 있게 해 줍니다. Introscope 는 다음과 같이 두 가지 유형의 자동 트랜잭션 추적을 제공합니다.

- 트랜잭션 추적 샘플링 - URL 그룹화를 기초로 기본적으로 활성화되어 있습니다.
- 구성 가능한 자동 추적 샘플링 - URL 그룹화에 관계없이 추적 정보를 수집합니다.

트랜잭션 추적 구성 요소 클램프

Introscope 에서는 구성 요소 클램프를 설정하여 추적 크기를 제한할 수 있습니다. 기본값은 구성 요소 5,000 개입니다. 이 제한에 도달하면 로그에 경고가 나타나고 추적이 중지됩니다.

예를 들어 서블릿이 개체 상호 작용 및 백엔드 SQL 호출을 수백 개 실행할 경우 예상 구성 요소 수를 초과하는 구성 요소 트랜잭션을 클램프할 수 있습니다. 클램프가 설정되지 않은 경우 트랜잭션 추적 프로그램에서는 이러한 트랜잭션을 무한히 지속되는 하나의 트랜잭션으로 간주합니다. 극단적인 몇몇 경우에는 클램프가 설정되지 않았으면 추적이 완료되기 전에 JVM 에 메모리가 부족할 수 있습니다.

트랜잭션을 클램프하기 위한 속성은 IntroscopeAgent.profile 파일에 있습니다.

introscope.agent.transactiontrace.componentCountClamp=*5000*

클램프된 구성 요소를 생성하는 추적은 별표로 표시됩니다. 이러한 추적을 확인하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

중요! 트랜잭션 추적 구성 요소 클램프 크기가 늘어나면 트랜잭션 추적에 필요한 메모리도 늘어날 수 있습니다. 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

트랜잭션 추적 샘플링

트랜잭션 추적 샘플링은 기본적으로 사용됩니다. 필요한 경우 이 동작을 사용하지 않도록 설정할 수 있습니다. 트랜잭션 추적 속성에 대한 자세한 내용은 트랜잭션 추적 (see page 364)을 참조하십시오.

자동 추적 샘플링을 구성할 때는 지정한 시간 간격 동안 추적할 트랜잭션의 수를 지정하십시오.

참고: 이러한 속성은 Enterprise Manager 속성 파일에 있습니다. *sampling.perinterval* 및 *sampling.interval* 속성의 기본값을 변경하기 전에 샘플링 속도가 높을수록 Enterprise Manager 에서 부하가 증가할 수 있다는 점을 고려해야 합니다. Enterprise Manager 는 이 구성을 연결된 모든 에이전트에 푸시합니다. 에이전트에서 이러한 속성을 구성할 수도 있습니다. 에이전트에서 이러한 속성을 구성할 경우 Enterprise Manager 가 개별 에이전트에 대해 설정한 구성을 덮어쓰게 됩니다.

자동 추적 샘플링을 구성하려면 다음 속성을 수정하십시오.

- *introscope.agent.transactiontracer.sampling.enabled*

  트랜잭션 추적 샘플링을 사용하지 않도록 설정하려면 이 속성을 false 로 설정하십시오. 기본값은 true 입니다.

- *introscope.agent.transactiontracer.sampling.perinterval.count*

  지정한 간격 동안 추적할 트랜잭션의 수를 지정합니다.

  기본 트랜잭션 수는 1 입니다.

- *introscope.agent.transactiontracer.sampling.interval.seconds*

  지정한 트랜잭션 수를 추적할 시간을 지정합니다.

  기본 간격은 2 분입니다.

에이전트 힙 크기 지정

에이전트는 Java 힙 메모리를 사용하여 수집된 데이터를 저장합니다. "GC 힙 개요"에서 에이전트 GC 힙 사용량을 확인할 수 있습니다.

힙 사용량에 적용되는 정보는 다음과 같습니다.

- 힙 사용량이 많은 경우 에이전트를 설치할 때 힙 할당을 늘려야 할 수 있습니다.
- 모니터링되는 응용 프로그램의 트랜잭션이 매우 깊은 수준이거나 장기간 지속되는 경우 트랜잭션 추적 샘플링에 힙 메모리가 추가로 필요할 수 있습니다.

참고: GC 힙 사용량에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오. 고성능의 CA Introscope® 환경을 사용하는 경우 적절한 에이전트 JVM 힙 설정은 CA 전문 서비스에 문의하십시오.

추가 정보:

트랜잭션 추적 구성 요소 클램프 (페이지 201)
트랜잭션 추적 샘플링 (페이지 202)

크로스 프로세스 트랜잭션 추적

CA Introscope?Java Agent 는 WebLogic Server 또는 WebSphere 에서 JVM 경계를 넘는 트랜잭션을 추적할 수 있습니다 시스템 환경은 동일한 공급업체의 호환 가능한 응용 프로그램 서버 버전으로 구성되어야 합니다. 크로스 프로세스 트랜잭션 추적은 동기 트랜잭션(예: EJB 에 대한 서블릿)과 비동기 트랜잭션에 대해 지원됩니다.

중요! 크로스 프로세스 트랜잭션 추적은 CA Introscope® 9.0 이상의 에이전트에서만 사용할 수 있습니다.

추가 정보:

WebLogic Server 에서 크로스 프로세스 추적 사용 (페이지 49)
WebSphere 에서 크로스 프로세스 추적 사용 (페이지 64)

트랜잭션 추적 데이터 수집 확장

서블릿 및 JSP 호출을 위한 사용자 ID 데이터와 HTTP 요청 헤더, 요청 매개 변수, 세션 특성 등의 기타 트랜잭션 추적 데이터를 비롯한 추가 정보를 가져오도록 Introscope Transaction Tracer 를 구성할 수 있습니다. 이 정보를 캡처하려면 IntroscopeAgent.profile 에 조건을 정의해야 합니다.

사용자 ID 데이터 정보

Java Agent 가 서블릿과 JSP 호출을 식별하도록 구성하려면 먼저 관리되는 응용 프로그램이 사용자 ID 를 지정하는 방법에 대한 정보를 얻어야 합니다. 이 정보는 관리되는 응용 프로그램을 개발한 응용 프로그램 구축자가 아마 제공할 수 있을 것입니다.

Introscope Transaction Tracer 는 다음 방법 중 하나로 사용자 ID 를 저장하는 관리되는 응용 프로그램에서 사용자 ID 를 식별할 수 있습니다.

- HttpServletRequest.getRemoteUser()
- HttpServletRequest.getHeader (String key)
- HttpSession.getValue (String key)의 경우, 반환된 개체는 UserID 를 나타내는 String 이거나 toString()가 UserID 로 반환되는 Object 입니다.

관리되는 응용 프로그램이 다음 방법 중 하나를 사용하여 사용자 ID 를 저장하는 경우 추가 트랜잭션 추적 데이터를 수집하도록 에이전트 구성 (see page 205)을 참조하여 사용자 ID 데이터를 수집하도록 Java Agent 설정을 구성하십시오.

서블릿 요청 데이터 정보

Introscope 를 사용하면 사용자가 구성할 수 있는 매개 변수와 일치하는 트랜잭션 추적 데이터를 수집할 수 있습니다. 예를 들어 에이전트가 사용자 에이전트 HTTP 요청 헤더가 포함된 트랜잭션에 대한 트랜잭션 추적 데이터를 수집하도록 지정할 수 있습니다.

Introscope 에서는 다음과 같은 서블릿 요청 정보를 기록할 수 있습니다.

- 요청 헤더
- 요청 매개 변수
- 세션 특성

관리되는 응용 프로그램에 대한 이 서블릿 요청 정보를 기록하려면 추가 트랜잭션 추적 데이터를 수집하도록 에이전트 구성 (see page 205)을 참조하여 이 데이터를 수집하도록 Java Agent 설정을 구성하십시오.

추가 트랜잭션 추적 데이터를 수집하도록 에이전트 구성

사용자 ID, HTTP 요청 헤더, HTTP 요청 매개 변수 또는 HTTP 세션 특성과 같은 추가 트랜잭션 추적 데이터를 수집하도록 Java Agent 를 구성할 수 있습니다.

다음 단계를 따르십시오.

1. 에이전트 프로필 *IntroscopeAgent.profile* 을 엽니다.
2. Transaction Tracer Configuration 이라는 제목 아래에서 트랜잭션 추적 프로그램 속성을 찾습니다.

사용자 ID 데이터 수집

사용자 ID 를 식별하도록 Java Agent 를 구성하려면

1. 관리되는 응용 프로그램에서 사용자 ID 를 저장하는 데 사용하는 메서드에 해당하는 속성을 구성합니다.

   참고: 속성 집합 하나만 주석 처리가 제거되어 있어야 합니다. 그렇지 않으면 잘못된 속성이 사용될 수 있습니다.

2. HttpServletRequest.getRemoteUser()의 경우 다음과 같이 속성의 주석 처리를 제거합니다.

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getRemote
   User
   ```

3. HttpServletRequest.getHeader(문자열 키)의 경우 다음 속성 쌍의 주석 처리를 제거하고 두 번째 속성에 대한 키 문자열을 정의합니다.

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getHeader
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```

4. HttpSession.getValue(문자열 키)의 경우 다음 속성 쌍의 주석 처리를 제거하고 두 번째 속성에 대한 키 문자열을 정의합니다.

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getValue
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```

서블릿 요청 데이터 수집

HTTP 요청 헤더 및 매개 변수와 같은 서블릿 요청 정보를 기록하려면

1. 트랜잭션 추적 데이터를 수집할 HTTP 요청 헤더를 지정하려면 다음 속성의 주석 처리를 제거하고 추적할 HTTP 요청 헤더를 쉼표로 구분된 목록으로 지정합니다.
   ```
   #introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent
   ```
2. 트랜잭션 추적 데이터를 수집할 HTTP 요청 매개 변수를 지정하려면 다음 속성의 주석 처리를 제거하고 추적할 HTTP 요청 매개 변수를 쉼표로 구분된 목록으로 지정합니다.
   ```
   #introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter1,parameter2
   ```
3. 데이터를 추적할 HTTP 세션 특성을 지정하려면 다음 속성의 주석 처리를 제거하고 추적할 HTTP 세션 특성을 쉼표로 구분된 목록으로 지정합니다. 예를 들면 다음과 같습니다.
   ```
   #introscope.agent.transactiontracer.parameter.httpsession.attributes=cartID,deptID
   ```
4. 관리되는 응용 프로그램을 다시 시작합니다.

구성 요소 중단 보고 구성

정의된 시간 동안 프로브 대상 구성 요소에서 응답이 없는 경우 Application Performance Management 가 중단됩니다. 기본적으로 APM Introscope Agent 가 이러한 상태를 감지하고 중단 메트릭을 보고합니다.

에이전트가 중단을 확인할 때마다 메서드 스택 맨 위에 있는 계측 대상 구성 요소가 모두 확인됩니다. 구성 요소가 중단된 경우 중단된 메트릭 및 중단 스냅숏이 생성됩니다. 또한 다운스트림 모니터링을 구독하는 각 구성 요소에 대해 중단 메트릭이 생성됩니다.

메서드 스택 맨 위에 있는 계측 대상 구성 요소에 대한 중단 메트릭이 먼저 생성됩니다. *introscope.agent.stalls.thresholdseconds* 값보다 오래 걸리는 구성 요소에 대해 이러한 메트릭이 생성됩니다.

다운스트림 구독자 구성 요소 중단

다운스트림 모니터링을 구독하는 구성 요소는 중단된 다운스트림 구성 요소를 호출할 때 중단됩니다.

기본적으로 다운스트림 중단 모니터링을 구독하는 구성 요소는 다음과 같습니다.

- FrontendTracer 프로브 대상 구성 요소
- BackendTracer 프로브 대상 구성 요소
- WebServiceBlamepointTracer@Servicelevel 프로브 대상 구성 요소
- WebServiceBlamepointTracer@Opertationlevel 프로브 대상 구성 요소

예

프런트엔드 구성 요소는 중단된 백엔드 구성 요소를 호출하는 Webservice 를 호출합니다.

프런트엔드 --> WebService --> 백엔드

백엔드, Webservice 및 프런트엔드 구성 요소에 대한 중단 메트릭이 생성되는데, 그러한 구성 요소 모두 기본적으로 다운스트림 모니터링을 구독하기 때문입니다.

업스트림 상속 구성 요소 중단

중단 검사기는 먼저 구성 요소 스택의 최상위 구성원을 검사하여 중단되었는지 확인합니다. 최상위 구성 요소가 중단되지 않은 경우 중단 검사기는 중단된 부모 구성 요소가 있는지 검사합니다. 부모가 중단된 경우 스택의 해당 구성 요소 및 각 업스트림 부모에 대한 중단 메트릭이 생성됩니다.

구성 요소 M1()이 여러 구성 요소 M2()를 호출하며 각각 걸리는 시간이 몇 초에 불과할 경우에는 M1()에 대한 중단 메트릭이 보고될 수 있습니다.

예제:

중단 임계값이 15 초로 설정되었습니다. 메서드 M1()이 메서드 M2()를 루프 상태로 100 회 호출합니다. M2()는 매번 응답하는 데 걸리는 시간이 2 초에 불과하므로 중단되지 않습니다. 그러나 결국 M1()은 중단됩니다.

중단을 이벤트로 캡처하지 않도록 설정

기본적으로 Introscope 는 트랜잭션 중단을 트랜잭션 이벤트 데이터베이스에 이벤트로 캡처하고 해당 이벤트로부터 중단 메트릭을 생성합니다. 중단 메트릭은 트랜잭션의 첫 번째 메서드와 마지막 메서드에 대해 생성됩니다. 중단 이벤트 및 관련 메트릭은 Workstation 기록 이벤트 뷰어에서 볼 수 있습니다.

참고: 생성된 중단 메트릭은 항상 사용할 수 있지만 중단 이벤트는 Introscope Error Detector 가 설치되어 있어야만 볼 수 있습니다. 중단은 일반 오류로 저장되며 "오류" TypeView 에서 보거나, "type:errorsnapshot"을 쿼리하여 기록 쿼리 뷰어에서 볼 수 있습니다.

중단을 이벤트로 캡처하지 않도록 설정하거나, 중단 임계값을 변경하거나, 에이전트가 중단을 검사하는 빈도를 변경하려면 다음 속성을 사용하십시오.

- *introscope.agent.stalls.thresholdseconds* - 트랜잭션을 중단된 것으로 판단하는 최소 임계값 응답 시간을 지정합니다.
- *introscope.agent.stalls.resolutionseconds* - 에이전트가 중단을 확인하는 빈도를 지정합니다.

제 11 장: Introscope SQL 에이전트 구성

이 단원에서는 Introscope SQL 에이전트를 구성하는 방법을 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

SQL 에이전트 개요

SQL 에이전트는 Enterprise Manager 에 자세한 데이터베이스 성능 데이터를 보고합니다. SQL 에이전트를 사용하면 관리되는 응용 프로그램과 데이터베이스 간의 상호 작용을 추적하여 응용 프로그램에서 개별 SQL 문의 성능을 파악할 수 있습니다.

SQL 에이전트는 Java Agent 가 응용 프로그램을 모니터링할 때와 동일한 방법으로 SQL 문을 모니터링합니다. SQL 에이전트는 매우 낮은 오버헤드로 응용 프로그램 또는 데이터베이스를 모니터링하므로 다른 프로세스에 방해가 되지 않습니다.

개별 SQL 문 수준까지 의미 있는 성능 측정 데이터를 제공하기 위해 SQL 에이전트는 트랜잭션 관련 데이터를 제외하고 원래 SQL 문을 Introscope 에서 사용되는 정규화된 문으로 변환하는 방법으로 성능 데이터를 요약합니다. 정규화된 문에는 신용 카드 번호와 같은 중요한 정보가 포함되지 않으므로 이 프로세스를 통해 데이터가 보호됩니다.

예를 들어 다음과 같은 SQL 쿼리가 있다고 가정해 보겠습니다.

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

SQL 에이전트는 이 SQL 쿼리를 다음과 같이 정규화된 문으로 변환합니다.

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

마찬가지로 다음과 같은 SQL 업데이트 문이 있다고 가정해 보겠습니다.

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES ('Atwood', 'The Robber Bride')
```

SQL 에이전트는 이 SQL 쿼리를 다음과 같이 정규화된 문으로 변환합니다.

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES (?, ?)
```

참고: 따옴표 내의 텍스트('xyz')만 정규화됩니다.

정규화된 문에 대한 메트릭은 집계되며 Workstation Investigator 의 "JDBC" 노드에서 볼 수 있습니다.

SQL 에이전트 파일

SQL 에이전트는 모든 에이전트 설치에 포함되어 있습니다. SQL 에이전트 기능을 제공하는 파일은 다음과 같습니다.

- wily/core/ext/SQLAgent.jar
- wily/core/config/sqlagent.pbd

참고: 기본적으로 SQLAgent.jar 파일과 같은 에이전트 확장은 wily/core/ext 디렉터리에 설치됩니다. 에이전트 프로필의 introscope.agent.extensions.directory 속성을 사용하여 에이전트 확장 디렉터리의 위치를 변경할 수 있습니다. */ext* 디렉터리의 위치를 변경할 경우에는 */ext* 디렉터리의 내용도 이동해야 합니다.

SQL 문 정규화

일부 응용 프로그램의 경우 엄청나게 많은 수의 고유 SQL 문을 생성할 수 있습니다. EJB 3.0 과 같은 기술을 사용하는 경우 긴 고유 SQL 문이 증가할 수 있습니다. 긴 SQL 문이 있으면 에이전트에서 메트릭이 폭발적으로 증가하여 성능이 저하되고 다른 시스템 문제가 발생할 수 있습니다.

잘못 작성된 SQL 문으로 인한 메트릭 급증

일반적으로 SQL 에이전트 메트릭의 수는 고유 SQL 문의 수에 근접해야 합니다. 응용 프로그램에서 사용하는 SQL 문의 수가 적은데도 SQL 에이전트에서 많은 수의 고유 SQL 메트릭을 보여 주고 있으며 그 수가 계속 증가하고 있다면 SQL 문의 작성 방식에 문제가 있을 수 있습니다.

SQL 문으로 인해 메트릭이 급증할 수 있는 일반적인 경우에는 몇 가지가 있습니다. 예를 들어 주석을 포함하거나 임시 테이블을 생성하거나 값 목록을 삽입하는 SQL 문은 실행될 때마다 뜻하지 않게 고유 메트릭 이름을 생성할 수 있습니다.

예: SQL 문의 주석

메트릭 급증의 일반적인 원인 중 하나는 SQL 문에서 주석이 사용된 방식에 있습니다. 예를 들어 다음과 같은 SQL 문이 있다고 가정합니다.

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

이 경우 SQL 에이전트는 다음과 같이 주석을 메트릭 이름의 일부로 포함하는 메트릭을 생성합니다.

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

SQL 문에 포함된 주석은 데이터베이스 관리자가 누가 어떤 쿼리를 실행하고 있는지 확인하는 데 유용하며, 쿼리를 실행하는 데이터베이스에서는 주석을 무시합니다. 그러나 SQL 에이전트에서는 SQL 문을 캡처할 때 주석 문자열을 구문 분석하지 않습니다. 따라서 SQL 에이전트에서는 각각의 고유한 사용자 ID 마다 고유한 메트릭을 생성하며, 이로 인해 메트릭이 급증하게 될 수 있습니다.

SQL 주석을 작은따옴표로 묶으면 이 문제를 방지할 수 있습니다. 예:

```
"/*' John Doe, user ID=?, txn=? '*/ select * from table..."
```

그러면 SQL 에이전트에서는 다음과 같은 메트릭을 생성하며 이 경우 주석으로 인한 고유 메트릭 이름은 더 이상 생성되지 않습니다.

```
"/* ? */ select * from table..."
```

예: 임시 테이블 또는 자동으로 생성된 테이블 이름

메트릭 급증의 또 다른 가능한 이유는 SQL 문에 자동으로 생성된 이름이 있는 임시 테이블을 참조하는 응용 프로그램에 기인할 수 있습니다. 예를 들어, Investigator 를 열어 Backends(백엔드)|{backendName}|SQL|{sqlType}|sql. 아래의 메트릭을 보는 경우 다음과 같은 메트릭을 확인할 수 있습니다.

```
SELECT * FROM TMP_12398139821038192091 2 WHERE ROW_ID = ?
```

이 SQL 문은 테이블 이름에 추가된 고유 식별자를 갖는 임시 테이블에 액세스합니다. *TMP_* 테이블 이름에 추가된 추가적인 숫자는 이 명령문이 실행될 때마다 고유한 메트릭 이름을 생성하므로 메트릭이 급증하게 됩니다.

예: 값 목록을 생성하거나 값을 삽입하는 문

메트릭 급증의 또 다른 일반적인 원인은 값 목록을 생성하거나 값을 대량으로 수정하는 SQL 문에 있습니다. 예를 들어 메트릭 급증이 발생할 수 있다는 경고를 받은 후 조사 과정에서 다음과 같은 SQL 문을 검토하게 되었다고 가정합니다.

```
#1 INSERT INTO COMMENTS (COMMENT_ID, CARD_ID, CMMT_TYPE_ID,
CMMT_STATUS_ID,CMMT_CATEGORY_ID, LOCATION_ID, CMMT_LIST_ID, COMMENTS_DSC,
USER_ID,LAST_UPDATE_TS) VALUES (?, ?, ?, ?, ?, ?, ?, "CHANGE CITY FROM CARROLTON,TO
CAROLTON, _ ", ?, CURRENT)
```

이 코드를 조사해 보면 *"CHANGE CITY FROM CARROLTON, TO CAROLTON, _ "* 부분에서 도시 배열이 생성됨을 알 수 있습니다.

마찬가지로, 잠재적 메트릭 급증을 조사할 경우 다음과 같은 SQL 문을 검토하게 될 수도 있습니다.

```
CHANGE COUNTRY FROM US TO CA _ CHANGE EMAIL ADDRESS FROM TO BRIGGIN @ COM _ "
```

이 코드를 조사해 보면 CHANGE COUNTRY 로 인해 긴 국가 목록이 생성됨을 알 수 있습니다. 또한 국가에 따옴표를 사용함으로써 사람들의 전자 메일 주소가 SQL 문 내에 삽입되게 되고, 이로 인해 메트릭 급증의 원인이 될 수 있는 고유 메트릭이 생성됩니다.

SQL 문 정규화 옵션

SQL 에이전트에는 긴 SQL 문의 처리를 위해 다음과 같은 노멀라이저가 포함됩니다.

- 기본 SQL 문 노멀라이저 (see page 213)
- 사용자 지정 SQL 문 노멀라이저 (see page 214)
- 정규식 SQL 문 노멀라이저 (see page 216)
- 명령줄 SQL 문 노멀라이저 (see page 220)

기본 SQL 문 노멀라이저

표준 SQL 문 노멀라이저는 기본적으로 SQL 에이전트에 있으며, 작은따옴표 내의 텍스트('xyz')를 정규화합니다. 예를 들어 다음과 같은 SQL 쿼리가 있다고 가정해 보겠습니다.

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

SQL 에이전트는 이 SQL 쿼리를 다음과 같이 정규화된 문으로 변환합니다.

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

정규화된 문에 대한 메트릭은 집계되며 Workstation Investigator 에서 볼 수 있습니다.

사용자 지정 SQL 문 노멀라이저

SQL 에이전트를 사용하면 사용자 지정 정규화를 수행하기 위한 확장을 추가할 수 있습니다. 이렇게 하려면 SQL 에이전트에서 구현된 정규화 체계가 들어 있는 JAR 파일을 생성하십시오.

SQL 문 노멀라이저 확장을 적용하려면

1. 확장 JAR 파일을 만듭니다.

   참고: SQL 문 노멀라이저 확장 파일의 진입점 클래스는 *com.wily.introscope.agent.trace.ISqlNormalizer* 인터페이스를 구현해야 합니다.

   JAR 확장 파일을 만들려면 SQL 노멀라이저 확장에 필요한 특정 키가 들어 있는 매니페스트 파일을 만들어야 합니다. 매니페스트 파일을 만드는 방법은 아래의 2 단계에서 설명합니다. 그러나 확장이 작동하기 위해서는 다른 일반 키도 필요합니다. 이러한 키는 확장 파일을 구성하는 데 사용하는 유형입니다. 생성하는 확장 파일은 예를 들어 *Backends(백엔드)|{backendName}|SQL|{sqlType}|{actualSQLStatement}* 노드 아래의 메트릭과 같은 데이터베이스 SQL 문 텍스트 정규화와 관련됩니다. *{actualSQLStatement}*는 SQL 노멀라이저에 의해 정규화됩니다.

2. 생성된 확장의 매니페스트에 다음 키를 추가합니다.

   - *com-wily-Extension-Plugins-List:testNormalizer1*

     참고: 이 키의 값은 원하는 대로 지정할 수 있습니다. 이 예에서는 *testNormalizer1* 이 사용됩니다. 이 키에 지정한 값을 다음 키에도 사용합니다.

   - *com-wily-Extension-Plugin-testNormalizer1-Type: sqlnormalizer*
   - *com-wily-Extension-Plugin-testNormalizer1-Version: 1*
   - *com-wily-Extension-Plugin-testNormalizer1-Name: normalizer1*

     노멀라이저의 고유 이름(예: normalizer1)을 포함해야 합니다.

   - *com-wily-Extension-Plugin-testNormalizer1-Entry-Point-Class: <Thefully-qualified classname of your implementation of ISQLNormalizer>*

3. 생성한 확장 파일을 *<Agent_Home>/wily/core/ext* 디렉터리에 배치합니다.

4. *IntroscopeAgent.profile* 에서 다음 속성을 찾아 설정합니다.
   `introscope.agent.sqlagent.normalizer.extension`

   생성한 확장의 매니페스트 파일에서 이 속성을 *com-wily-Extension-Plugin-{plugin}-Name* 으로 설정합니다. 이 속성 값은 대/소문자를 구분하지 않습니다. 예:
   `introscope.agent.sqlagent.normalizer.extension=normalizer1`

   중요! 이 속성은 핫 속성입니다. 확장 이름을 변경하면 확장이 다시 등록됩니다.

5. 필요한 경우 *IntroscopeAgent.profile* 에서 다음 속성을 추가하여 오류 스로틀 수를 설정할 수 있습니다.
   `introscope.agent.sqlagent.normalizer.extension.errorCount`

   오류 및 예외에 대한 자세한 내용은 예외 (see page 215)를 참조하십시오.

   참고: 사용자 지정 노멀라이저 확장에서 throw 된 오류가 오류 스로틀 수를 초과할 경우 확장이 사용하지 않도록 설정됩니다.

6. *IntroscopeAgent.profile* 을 저장합니다.
7. 응용 프로그램을 다시 시작합니다.

예외

사용자가 만든 확장이 한 쿼리에 대해 예외를 throw 하는 경우 기본 SQL 문 노멀라이저는 해당 쿼리에 대해 기본 정규화 체계를 사용합니다. 이 경우 로그에 확장에서 예외가 throw 되었다는 오류 메시지가 기록되고 스택 추적 정보를 포함하는 디버그 메시지가 기록됩니다. 하지만 이와 같은 예외가 5 번 throw 되면 기본 SQL 문 노멀라이저가 사용자가 만든 확장이 사용되지 않도록 설정하고 노멀라이저가 변경될 때까지 이후의 쿼리에서 해당 확장을 사용하려는 시도를 중단시킵니다.

Null 또는 비어 있는 문자열

사용자가 만든 확장이 쿼리에 대해 null 문자열이나 비어 있는 문자열을 반환한 경우 StatementNormalizer 는 해당 쿼리에 대해 기본 정규화 체계를 사용하고 확장이 null 값을 반환했음을 나타내는 정보 메시지를 로그에 기록합니다. 하지만 그와 같은 null 또는 비어 있는 문자열이 5 번 반환된 후에는 StatementNormalizer 가 메시지 기록을 중지하고 확장을 계속 사용하려고 시도합니다.

정규식 SQL 문 노멀라이저

SQL 에이전트는 구성 가능한 정규식(regex)을 기반으로 SQL 문을 정규화하는 확장과 함께 제공됩니다. 이 *RegexNormalizerExtension.jar* 파일은 *<Agent_Home>/wily/core/ext* 디렉터리에 있습니다.

정규식 SQL 문 노멀라이저를 사용하는 방법에 대한 예제는 정규식 SQL 문 노멀라이저 예제 (see page 218)를 참조하십시오.

정규식 확장을 적용하려면

1. *IntroscopeAgent.profile* 을 엽니다.
2. 다음 속성을 찾아 설정합니다.
   - *introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer*

     미리 구성된 정규화 체계를 재정의하는 데 사용할 SQL 노멀라이저 확장의 이름을 지정합니다. 정규식 확장을 사용하도록 설정할 때 이 속성을 RegexSqlNormalizer 로 설정하십시오.
   - *introscope.agent.sqlagent.normalizer.regex.keys=key1*

     이 속성은 regex 그룹 키를 지정하며, 지정된 키는 나열된 순서대로 평가됩니다. 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요합니다. 기본값은 없습니다.
   - *introscope.agent.sqlagent.normalizer.regex.key1.pattern=A*

     이 속성은 SQL 문과 일치시키는 데 사용되는 regex 패턴을 지정합니다. *java.util.Regex package* 클래스에서 허용되는 유효한 모든 정규식을 여기에 사용할 수 있습니다. 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요합니다. 기본값은 없습니다.
   - *introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=B*

     이 속성은 대체 문자열 형식을 지정합니다. *java.util.Regex package* 클래스에서 허용되는 유효한 모든 regex 를 여기에 사용할 수 있습니다. 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요합니다. 기본값은 없습니다.

- *introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false*

  이 속성이 true 로 설정되어 있으면 SQL 문자열이 모든 regex 키 그룹과 비교하여 평가됩니다. 구현은 체인으로 연결됩니다. 따라서 SQL 문자열이 여러 키 그룹과 일치하는 경우 group1 의 정규화된 SQL 출력이 group2 에 대한 입력으로 제공되는 방식으로 처리됩니다.

  이 속성이 false 로 설정되어 있으면 키 그룹이 SQL 문자열과 일치하는 즉시 해당 그룹의 정규화된 SQL 출력이 반환됩니다. MatchFallThrough 속성으로는 확장을 사용하거나 사용하지 않도록 설정할 수 없습니다.

  예를 들어 Select * from A where B 와 같은 SQL 문자열이 있는 경우 다음 속성을 설정합니다.

  ```
  introscope.agent.sqlagent.normalizer.regex.keys=key1,key2
  introscope.agent.sqlagent.normalizer.regex.key1.pattern=A
  introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=X
  introscope.agent.sqlagent.normalizer.regex.key2.pattern=B
  introscope.agent.sqlagent.normalizer.regex.key2.replaceFormat=Y
  ```

introscope.agent.sqlagent.normalizer.regex.matchFallThrough 가 *false* 이면 SQL 은 key1 regex 에 대해 정규화됩니다. 해당 regex 의 출력은 *Select * from X where B* 가 됩니다. 이 SQL 이 반환됩니다.

introscope.agent.sqlagent.normalizer.regex.matchFallThrough 가 *true* 이면 SQL 은 먼저 key1 regex 에 대해 정규화됩니다. 해당 regex 의 출력은 *Select * from X where B* 입니다. 그런 다음 이 출력이 *key2 regex* 에 제공됩니다. *key2 regex* 의 출력은 *Select * from X where Y* 입니다. 이 SQL 이 반환됩니다.

참고: 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요하지 않습니다.

- *introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false*

  이 속성은 패턴 일치 시 대/소문자 구분 여부를 지정합니다. 기본값은 false 입니다. 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요하지 않습니다.

- *introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false*

  이 속성이 false 로 설정되어 있으면 SQL 에서 첫 번째로 나타나는 일치하는 패턴이 대체 문자열로 바뀝니다. 이 속성이 true 로 설정되어 있으면 SQL 에서 나타나는 일치하는 패턴이 모두 대체 문자열로 바뀝니다.

  예를 들어 Select * from A where A like Z 와 같은 SQL 문자열이 있는 경우 다음과 같이 속성을 설정합니다.

  ```
  introscope.agent.sqlagent.normalizer.regex.key1.pattern=A
  introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=X
  ```

introscope.agent.sqlagent.normalizer.regex.key1.replaceAl 이 *false* 이면 정규화된 SQL 문 *Select * from X where A like Z* 가 반환됩니다.

introscope.agent.sqlagent.normalizer.regex.key1.replaceAl 이 *true* 이면 정규화된 SQL 문 *Select * from X where* **X** *like Z* 가 반환됩니다.

기본값은 false 입니다. 이 속성은 정규식 확장을 사용하도록 설정하는 데 필요하지 않습니다.

참고: 정규식 패턴이 입력 SQL 과 일치하지 않으면 RegexNormalizer 는 Null 문자열을 반환합니다. 그러면 문 노멀라이저는 기본 정규화 체계를 사용합니다.

3. *IntroscopeAgent.profile* 을 저장합니다.

중요! 위에 나열된 모든 속성은 핫 속성이므로, *IntroscopeAgent.profile* 을 저장하면 이러한 속성의 변경 사항이 적용됩니다.

정규식 SQL 문 노멀라이저 예제

아래의 3 개 예제는 정규식 SQL 문 노멀라이저를 구현하는 방법을 이해하는 데 도움을 줍니다.

예제 1

다음은 정규식 SQL 문 정규화 이전의 SQL 쿼리입니다.

```
INSERT INTO COMMENTS (COMMENT_ID, CARD_ID, CMMT_TYPE_ID,CMMT_STATUS_ID,
CMMT_CATEGORY_ID, LOCATION_ID, CMMT_LIST_ID,COMMENTS_DSC, USER_ID, LAST_UPDATE_TS)
VALUES(?, ?, ?, ?, ?, ?,?, ‘’CHANGE CITY FROM CARROLTON, TO CAROLTON, _ ", ?, CURRENT)
```

다음은 원하는 정규화된 SQL 문입니다.

```
INSERT INTO COMMENTS (COMMENT_ID, ...) VALUES (?, ?, ?, ?, ?, ?,?, CHANGE CITY FROM
( )
```

다음은 위에 표시된 정규화된 SQL 문이 반환되도록 하기 위해 *IntroscopeAgent.profile* 파일에 필요한 구성입니다.

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=key1,key2
introscope.agent.sqlagent.normalizer.regex.key1.pattern=(INSERT INTO
COMMENTS \\(COMMENT_ID,)(.*)(VALUES.*)''(CHANGE CITY FROM \\().*(\\))
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1 ...) $3$4 $5
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
introscope.agent.sqlagent.normalizer.regex.key2.pattern='[a-zA-Z1-9]+'
introscope.agent.sqlagent.normalizer.regex.key2.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.key2.replaceFormat=?
introscope.agent.sqlagent.normalizer.regex.key2.caseSensitive=false
```

예제 2

다음은 정규식 SQL 문 정규화 이전의 SQL 쿼리입니다.

```
SELECT * FROM TMP_12398139821038192O912 WHERE ROW_ID =
```

다음은 원하는 정규화된 SQL 문입니다.

```
SELECT * FROM TMP_ WHERE ROW_ID =
```

다음은 위에 표시된 정규화된 SQL 문이 반환되도록 하기 위해 *IntroscopeAgent.profile* 파일에 필요한 구성입니다.

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=key1
introscope.agent.sqlagent.normalizer.regex.key1.pattern=(TMP_)[1-9]*
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
```

예제 3

다음과 같은 SQL 문을 정규화하려면: *Select .... ResID1, CustID1* 여기서 *ResID1=.. OR ResID2=.. n times OR CustID1=.. OR n times*, 다음과 같이 속성을 설정할 수 있습니다:

```
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=default,def
introscope.agent.sqlagent.normalizer.regex.default.pattern=(ResID)[1-9]
introscope.agent.sqlagent.normalizer.regex.default.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.default.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.default.caseSensitive=true
introscope.agent.sqlagent.normalizer.regex.def.pattern=(CustID)[1-9]
introscope.agent.sqlagent.normalizer.regex.def.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.def.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.def.caseSensitive=true
```

명령줄 SQL 문 노멀라이저

정규식 SQL 노멀라이저를 사용하지 않는 경우 WHERE 절의 값을 큰따옴표(" ")로 묶은 SQL 문이 있으면 다음 명령줄 명령을 사용하여 SQL 문을 정규화하십시오.

```
-DSQLAgentNormalizeDoubleQuoteString=true
```

중요! 이 명령 대신 정규식 SQL 노멀라이저를 사용하여 큰따옴표로 묶은 SQL 문을 정규화할 수 있습니다. 자세한 내용은 정규식 SQL 문 노멀라이저 (see page 216)를 참조하십시오.

문 메트릭 해제

일부 응용 프로그램에서는 많은 수의 고유한 SQL 문을 생성할 수 있으며, 이로 인해 SQL 에이전트에 메트릭이 급증할 수 있습니다. SQL 에이전트에서 SQL 문 메트릭을 해제할 수 있습니다.

참고: 문 메트릭을 해제해도 백엔드 또는 최상위 JDBC 메트릭은 손실되지 않습니다.

문 메트릭을 해제하려면

1. sqlagent.pbd 파일을 열고 SQL 문을 찾습니다. 예:
   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements
   executeQuery(Ljava/lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends(백엔드)|{database}|SQL|{commandtype}|Query|{sql}"
   ```
2. 해제할 추적 지시문에서 *{sql}*을 제거합니다. 예:
   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements executeQuery(Ljava/
   lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends(백엔드)|{database}|SQL|{commandtype}|Query"
   ```
3. *sqlagent.pbd* 파일을 저장합니다.

   SQL 에이전트의 SQL 문 메트릭이 해제되었습니다.

SQL 메트릭

SQL 에이전트 메트릭은 Introscope Workstation Investigator 의 백엔드 노드 아래에 표시됩니다. SQL 문 메트릭은 Backends(백엔드)|<backendName>|SQL 노드 아래에 있습니다.

참고: *평균 응답 시간(ms)*은 데이터 판독기를 반환하는 쿼리(예: *ExecuteReader()* 메서드를 통해 실행된 쿼리)만 표시합니다. 이 메트릭은 데이터 판독기의 *Close()* 메서드에서 소비된 평균 시간을 나타냅니다.

SQL 데이터 관련 메트릭 유형을 다음을 포함합니다:

- 연결 수 - 메모리에서 라이브 연결 개체의 수입니다.

  연결은 드라이버의 *connect()* 메서드가 호출될 때 열리고 *close()* 메서드를 통해 연결 호출이 종료될 때 닫힙니다. SQL 에이전트는 하나의 집합으로 연결에 대한 약한 참조를 유지합니다. 연결 개체가 가비지 수집되면 이 변경이 수에 반영됩니다.

- 평균 결과 처리 시간(ms) - 쿼리의 평균 처리 시간입니다.

  이 메트릭은 *executeQuery()* 호출의 끝부터 ResultSet 의 *close()* 메서드 호출까지의 ResultSet 처리에 걸린 평균 시간을 나타냅니다.

참고: 계측된 XADataSources 는 커밋 또는 롤백 메트릭을 보고하지 않을 수 있습니다. 다른 계측된 데이터 원본은 이러한 메트릭에 데이터가 포함되어 있지 않은 한 커밋 또는 롤백 메트릭을 보고하지 않을 수 있습니다.

제 12 장: JMX 보고를 사용하도록 설정

이 단원에는 Java Agent 가 JMX 데이터를 보고할 수 있도록 설정하는 방법에 대한 정보가 포함되어 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

Java Agent JMX 지원

CA Introscope 는223 응용 프로그램 서버나 Java 응용 프로그램이 JMX 호환 MBean 으로 제공하는 관리 데이터를 수집할 수 있습니다. CA Introscope 는223 Investigator 메트릭 트리에 JMX 데이터를 표시합니다. CA Introscope 는223 다음 응용 프로그램 서버에서 JMX 정보의 수집을 지원합니다.

- Glassfish - Glassfish 가 JMX 메트릭을 보고하기 위한 추가 구성은 필요하지 않습니다.
- JBoss - JMX 메트릭을 보고하도록 JBoss 를 구성하는 지침은 Java Agent 를 사용하도록 JBoss 구성 (see page 44)을 참조하십시오.
- Tomcat - JMX 메트릭을 보고하도록 Tomcat 을 구성하는 지침은 Java Agent 를 사용하도록 Apache Tomcat 구성 (see page 42)을 참조하십시오.
- WebLogic - JMX 보고를 사용하도록 WebSphere 및 WebLogic 을 구성 (see page 228)할 수 있습니다.
- WebSphere - JMX 보고를 사용하도록 WebSphere 및 WebLogic 을 구성 (see page 228)할 수 있습니다.

참고: CA Introscope®는 Sun JMX 사양에 맞게 빌드된 모든 MBean 을 지원합니다. Sun JMX 사양에 대한 자세한 내용은 Java Management Extensions를 참조하십시오.

CA Introscope 는224 이 JMX 데이터를 CA Introscope 메트릭 형식으로 변환하여 Investigator 의 다음 노드 아래에 표시합니다.

<*도메인*>|<*호스트*>|<*프로세스*>|<*에이전트*>|JMX|

WebLogic JMX 메트릭에 대한 Introscope 지원

WebLogic 은 다음과 같은 MBean 을 JMX 메트릭의 원본으로 제공합니다.

- RuntimeServiceMBean: 서버별 런타임 메트릭(활성 유효 구성 포함)
- DomainRuntimeServiceMBean: 도메인 차원 런타임 메트릭
- EditServiceMBean: 영구 구성을 사용자가 편집할 수 있도록 함

CA Introscope 는224 다음과 같은 이유로 RuntimeServiceMBean 만 폴링합니다.

- RuntimeServiceMBean 은 로컬 액세스(효율성 측면)를 지원합니다.
- RuntimeServiceMBean 은 관련된 것으로 예상되는 대부분의 데이터를 포함합니다.

기본 JMX 메트릭 변환 프로세스

CA Introscope 는224 기본적으로 JMX 메트릭을 Investigator 에 표시할 수 있도록 변환합니다. 다음과 같은 경우 CA Introscope 는224 MBean 을 변환합니다.

- WebLogic 을 사용하는 경우
- JMX 메트릭의 효율성 향상을 위해 기본 키 변환을 구성 (see page 225)하지 않은 경우

참고: 일치하는 MBean 이 없는 기본 키를 지정하는 경우 CA Introscope®는 기본 변환 방법을 사용합니다.

기본 변환 방법에서 CA Introscope 는225 특성의 이름과 값을 모두 표시하고, 메트릭 트리에서 사전순으로 이 쌍을 나열합니다.

<Domain>|<Host>|<Process>|AgentName|JMX|*<domain name>|<key1>=<value1>|<key2>=<value2>:<metric>*

예를 들어, 다음과 같은 WebLogic MBean 의 경우:

- 도메인 이름: WebLogic
- 키/값 쌍: category=server, type=jdbc
- 메트릭 이름: connections

기본 키가 IntroscopeAgent.profile 속성 introscope.agent.jmx.name.primarykeys 에 지정되지 않은 경우 MBean 특성은 다음과 같은 CA Introscope 메트릭으로 변환됩니다.

<Domain>|<Host>|<Process>|AgentName|JMX|Weblogic|*category=server|type=jdbc:connections*

키/값 쌍은 CA Introscope 메트릭에서 사전순으로 표시됩니다.

기본 키 변환을 사용하여 JMX 메트릭의 효율성 향상

필요한 경우 메트릭이 JMX 노드에 표시되는 순서를 구성할 수 있습니다. 에이전트 프로필에서 기본 키를 정의할 수 있습니다. 기본 키를 통해 MBean 의 ObjectName 을 식별할 수 있습니다.

기본 키 변환을 구성하지 않을 경우 CA Introscope 가225 JMX 데이터를 변환 (see page 224)합니다. 기본 변환을 사용할 때 메트릭은 Investigator 의 JMX 노드 아래에 사전순으로 나열됩니다. JMX 데이터를 메트릭으로 변환하는 이 방법을 통해 메트릭 이름의 효율을 높일 수 있습니다. 생성된 메트릭에서 키/값 쌍 정보의 순서를 제어할 수 있습니다.

동작은 에이전트 프로필의 introscope.agent.jmx.name.primarykeys 속성에 구성됩니다. primarykeys 속성의 값은 MBeans JMX ObjectName 에서 MBean 을 고유하게 식별하는 부분을 지정합니다. 예를 들어 WebLogic MBean 의 ObjectName 에는 다음과 같은 키가 있습니다.

- MBean 종류를 지정하는 Type 키
- MBean 이 나타내는 리소스의 이름을 지정하는 Name 키

ObjectName 의 키/값 쌍은 MBean 의 다른 유형마다 차이가 있을 수 있습니다.

CA Introscope 는226 다음 규칙에 따라 introscope.agent.jmx.name.primarykeys 속성 값으로 식별되는 MBean 을 변환하고 제공합니다.

- 키 값 정보만 표시되고 키 이름은 표시되지 않습니다.
- 값은 primarykeys 속성에 정의된 순서에 따라 순서가 지정됩니다.
- 값은 대소문자를 구분합니다.

예를 들어, 다음과 같은 WebLogic MBean 의 경우:

- 도메인 이름: WebLogic
- MBean ObjectName 키/값 쌍: category=server, type=jdbc
- 메트릭 이름: connections

다음과 같이 구성하면:

introscope.agent.jmx.name.primarykeys=type,category 연결 특성이 다음 구조로 Investigator 트리에 표시됩니다.

<IntroscopeDomain>|*<Host>*|*<Process>*|*AgentName*|JMX|Weblogic|jdbc|server:connections

참고: WebLogic 9.0 에는 전역에서 사용 가능한 기본 키가 없습니다. 기본 변환 방법에서 검색되는 키/값 쌍 메트릭 명명 규칙을 사용하십시오. 따라서, WebLogic 9.0 의 JMX 메트릭 트리는 다른 구조를 갖습니다.

추가 정보:

기본 JMX 메트릭 변환 프로세스 (페이지 224)

JMX 필터를 사용하여 메트릭 볼륨 관리

JMX 필터를 정의하면 CA Introscope®에서 수집 및 표시되는 JMX MBean 정보가 결정됩니다. 필터가 설정되어 있지 않으면 에이전트에서 *모든* JMX MBean 정보를 Enterprise Manager 에 보고하므로 시스템 오버헤드가 증가합니다.

필터는 에이전트 프로필 IntroscopeAgent.profile 의 introscope.agent.jmx.name.filter 속성에서 설정됩니다. 필터는 키워드로서, 이 속성에서 쉼표로 구분된 문자열로 입력됩니다. CA Introscope®는 별표(*) 및 물음표(?) 와일드카드 문자가 포함된 필터 문자열을 지원합니다.

CA Introscope®는 필터 문자열을 JMX 생성 메트릭과 비교합니다. 일치 항목이 검색되는 경우 일치하는 메트릭은 CA Introscope®에 보고됩니다.

반환되는 메트릭의 볼륨을 제한하려면 필터 문자열을 가능한 한 좁은 범위로 정의하십시오. 예를 들어 특정 MBean 특성과 일치하고 여러 MBean 에 대해 존재하는 필터 문자열을 정의할 수 있습니다. 그러한 *각* MBean 의 메트릭이 보고됩니다. 선택한 MBean 에 대한 특성만 필요한 경우에는 필터 문자열에 MBean 이름을 사용하여 특성 이름을 한정할 수 있습니다.

예를 들어 JMSDestinationRuntime MBean 의 MessagesCurrentCount 특성 값을 캡처하려고 한다고 가정합니다.

MessagesCurrentCount 의 정규화된 메트릭 이름이 다음과 같은 경우

```
*SuperDomain*|host-name|Process|Agent-name|JMX|comp-1|
JMSDestinationRuntime|comp-2:MessagesCurrentCount
```

IntroscopeAgent.profile 에서 introscope.agent.jmx.name.filter 를 다음과 같이 정의하십시오.

```
JMX|comp-1|JMSDestinationRuntime|comp-2:MessagesCurrentCount
```

WebLogic 용 JMX 필터

WebLogic 에 대한 *IntroscopeAgent.profile* 파일에서 다음 키워드가 이미 정의되어 있습니다.

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

JMX 보고를 사용하도록 WebSphere 및 WebLogic 구성

JMX 를 지원하도록 CA Introscope®를 구성하는 방법은 사용하는 응용 프로그램 서버에 따라 달라집니다. 여기에서는 WebLogic Server 및 WebSphere 에서 JMX 데이터를 수집하여 표시하도록 CA Introscope®를 구성하는 방법을 설명합니다.

다음 단계를 따르십시오.

1. 관리되는 응용 프로그램을 종료합니다.
2. WebSphere 에이전트의 경우에만 IntroscopeAgent.profile 에서 introscope.agent.jmx.enable 을 true 로 설정합니다.

   참고: WebSphere 에이전트 프로필에서 기본값은 false 입니다.
3. IntroscopeAgent.profile 에서 기본 키를 구성합니다.
   - WebLogic 9.0 의 경우 다음을 주석 처리합니다.
     `introscope.agent.jmx.name.primarykeys`
   - 다른 WebLogic 버전의 경우 다음의 주석 처리를 제거합니다.
     `introscope.agent.jmx.name.primarykeys`

   참고: 속성 값을 수정하는 경우에는 대/소문자를 구분하고 여러 키를 쉼표로 구분해야 합니다.

4. IntroscopeAgent.profile 에서 introscope.agent.jmx.name.filter 속성의 주석 처리가 제거되었는지 확인하십시오.

   참고: 필터에는 버전 번호 같은 MBean 특성이 포함되어야 합니다. 예를 들어 전체 메트릭 이름이 다음과 같다고 가정합니다.

   ```
   *SuperDomain*|MyServer01|WebSphere|LODVMAPM032Node02Cell/server1|JMX|WebSphere|cell=LODVMAPM032Node02Cell|mbeanIdentifier=default_host/hello|name=hello-2_1_1_2_war#hello-2.1.1.2.war|node=LODVMAPM032Node02|platform=dynamicproxy|process=server1|spec=1.0|type=SessionManager|version=6.1.0.0|StatsImpl|LiveCount:HighWaterMark
   ```

   이 경우 버전 번호 특성을 포함하는 필터링된 버전은 다음과 같습니다.

   ```
   PlantsByWebSphere|name=PlantsByWebSphere#PlantsByWebSphere.war|node=LODVMAPM033Node01|platform=dynamicproxy|process=server1|spec=1.0|type=SessionManager|version=6.1.0.0
   ```

   또는 간단히 다음과 같습니다.

   ```
   Plant*re|*version*
   ```

5. 속성에 원하는 문자열을 입력하고 쉼표로 구분합니다.

   CA Introscope®가 필터링된 문자열을 제대로 일치시키도록 하려면 문자열을 정확한 철자로 대/소문자를 구분하여 입력해야 합니다.

6. 변경 사항을 저장합니다.
7. 관리되는 응용 프로그램을 시작합니다.
8. WebLogic Server 또는 WebSphere 의 사용자 지정 서비스에서 CA Introscope® 시작 클래스를 구성하여 JMX 데이터를 사용하도록 설정합니다.

추가 정보:

WebLogic 용 시작 클래스 생성 (페이지 48)

WAS 에서 JSR-77 데이터를 사용하도록 설정하고 JMX 메트릭 보기

WebSphere 에서 JSR-77 JMX MBean 개체에 대한 메트릭을 수집, 유지 및 보고하도록 Introscope 를 구성할 수 있습니다. J2EE 관리 사양 JSR-77 은 J2EE 아키텍처의 관리 가능한 부분을 요약하고 관리 정보에 액세스하기 위한 인터페이스를 정의합니다.

참고: JSR-77 을 지원하려면 JVM 버전 1.4 이상이 필요합니다. JVM 버전이 1.4 인 경우에는 응용 프로그램 서버에서도 JSR-77 을 지원해야 합니다.

다음 단계를 따르십시오.

1. 관리되는 응용 프로그램을 종료합니다.
2. WebSphere 에서 사용자 지정 서비스를 구성 (see page 59)합니다.
3. IntroscopeAgent.profile 에서 다음 속성이 true 인지 확인합니다.

   ```
   introscope.agent.jmx.enable=true
   ```

4. IntroscopeAgent.profile 에서 다음 속성을 설정하여 JSR-77 을 사용하도록 설정합니다.

   ```
   introscope.agent.jmx.name.jsr77.disable=false
   ```

5. IntroscopeAgent.profile 에서 다음 속성의 주석 처리를 제거하여 메트릭 변환의 기본 키 방법을 구성합니다.

   ```
   introscope.agent.jmx.name.primaryKeys=J2EEServer,Application,
   j2eeType,JDBCProvider,name,mbeanIdentifier
   ```

   참고: WebSphere 용으로 Introscope 와 함께 제공되는 IntroscopeAgent.profile 에만 이 속성 정의가 포함되어 있습니다.

6. 다음 속성의 주석 처리를 제거하고 보고할 JSR-77 메트릭을 식별하도록 설정합니다.

   ```
   introscope.agent.jmx.name.filter
   ```

   필터링이 반드시 필요하지는 않지만 사용하는 것이 좋습니다.

7. 다음 속성의 주석 처리를 제거하고 원하는 대로 업데이트하여 JSR-77 메트릭에서 제외할 Mbean 특성을 지정합니다.

   ```
   introscope.agent.jmx.ignore.attributes=server
   ```

참고: 자세한 내용은 http://ca.com/support 에서 KB 문서 TEC534087: WebSphere Performance Viewer Data vs. Introscope PMI Data(TEC534087: WebSphere Performance Viewer 데이터 대 Introscope PMI 데이터)를 참조하십시오.

추가 정보:

기본 키 변환을 사용하여 JMX 메트릭의 효율성 향상 (페이지 225)
JMX 필터를 사용하여 메트릭 볼륨 관리 (페이지 227)

제 13 장: 플랫폼 모니터링 구성

이 단원에서는 Introscope 플랫폼 모니터를 구성하는 방법을 설명합니다.

이 섹션은 다음 항목을 포함하고 있습니다.

플랫폼 모니터

Java Agent 는 플랫폼 모니터를 사용하여 CPU 통계를 비롯한 시스템 메트릭을 Enterprise Manager 에 보고할 수 있습니다. 플랫폼 모니터는 Introscope Agent 설치 관리자와 함께 포함되어 있습니다.

다음 정보를 고려하십시오.

- Windows Server 2003 및 AIX 를 제외한 모든 운영 체제에서는 Java Agent 설치 시 플랫폼 모니터가 자동으로 사용하도록 설정됩니다. Windows Server 2003 및 AIX 플랫폼 모니터는 최소 구성 요건을 충족해야 작동합니다.
- 플랫폼 모니터 바이너리는 응용 프로그램 서버 및 운영 체제 비트 모드에 종속되지 않습니다. 또한 플랫폼 모니터 바이너리는 JVM 아키텍처에 완전히 종속됩니다.
- HP-UX 용 Java Agent 플랫폼 모니터는 프로세스가 둘 이상인 경우 CPU 사용률을 보고합니다. 예를 들어 프로세스가 네 개인 경우 최대 CPU 사용률은 400 %입니다(프로세스 네 개가 각각 100 %의 CPU 를 사용). 한 프로세스에서 110 %를 사용하는 경우 이는 CPU 를 1.1 개 사용하고 있음을 의미합니다.

참고: 시스템 요구 사항에 대해서는 *Compatibility Guide*(호환성 안내서)를 참조하십시오.

Java Agent 는 다음 플랫폼 메트릭을 생성합니다.

- **ProcessID(프로세스 ID)**
- **Processor Count**(프로세스 수) - CPU 의 수를 나타냅니다.
- **Utilization % (process)**(사용률 % (프로세스)) - Java Agent 프로세스에 대해 이 프로세스가 사용 중인 모든 프로세서의 총 수용 능력 비율(%)을 나타냅니다. 프로세서 수에 관계없이 이 메트릭은 숫자 하나만 생성합니다.
- **Utilization % (aggregate)**(사용률 % (집계)) - 이 프로세서에 대해 시스템의 모든 프로세스가 사용하는 총 사용률(%)을 나타냅니다. 각 프로세서는 Investigator 트리에 "리소스"로 표시됩니다.

Windows Server 2003 에서 플랫폼 모니터 사용

Windows Server 2003 에서 플랫폼 모니터를 실행하려면 *관리자* 권한이 있어야 합니다.

Windows 에서 플랫폼 모니터링이 작동하기 위해서는 시스템 개체가 사용하도록 설정되어 있어야 합니다.

시스템 개체가 사용하도록 설정되어 있는지 여부를 확인하려면

1. "시작" > "실행"으로 이동합니다.
2. *perfmon* 을 입력하고 "실행'을 클릭합니다.
3. 대화 상자에서 "추가"를 클릭합니다.
4. "추가" 대화 상자의 드롭다운 목록 상자에 "Process" 및 "Processor" 성능 개체가 있으면 시스템 개체가 사용하도록 설정되어 있는 것입니다.

시스템 개체를 사용하도록 설정하려면

1. "시작" > "보조프로그램"으로 이동하고 명령 프롬프트를 마우스 오른쪽 단추로 클릭한 후 "다음 계정으로 실행" > "관리자"를 클릭합니다.
2. *lodctr /r* 명령을 실행합니다.

   "Process" 및 "Processor" 개체가 사용하도록 설정됩니다.

AIX 에서 플랫폼 모니터 사용

AIX 에서 플랫폼 모니터를 사용하도록 설정할 수 있습니다.

다음 단계를 따르십시오.

1. Java Agent 를 설치한 후 다음 파일이 *wily/core/ext* 디렉터리에 설치되어 있는지 확인합니다.
   - introscopeAIXPSeries32Stats.jar
   - introscopeAIXPSeries64Stats.jar
   - libIntroscopeAIXPSeries32Stats.so
   - libIntroscopeAIXPSeries64Stats.so
2. Perfstat Library 를 설치합니다. IBM FTP 사이트에서 다음 패키지를 설치합니다.
   - bos.perf.libperfstat
   - bos.perf.perfstat
3. 컴퓨터를 다시 시작합니다.

   패치가 적용되고 플랫폼 모니터가 사용하도록 설정됩니다.

플랫폼 모니터 사용 안 함

해당 .jar 파일을 다른 디렉터리로 이동하여 플랫폼 모니터를 사용하지 않도록 설정할 수 있습니다.

다음 단계를 따르십시오.

1. /wily/core/ext 디렉터리로 이동합니다.
2. 예를 들어 introscopeSolarisAmd32Stats.jar 같은 사용 중인 플랫폼에 해당하는 introscope<*platform*>.jar 파일을 선택합니다.
3. .jar 파일을 /wily/core/ext 디렉터리에서 다른 디렉터리로 이동합니다.

HP-UX 에서 플랫폼 모니터에 액세스할 수 있도록 사용 권한 구성

.zip 또는 .tar 설치 관리자를 사용하여 HP-UX 에 플랫폼 모니터를 설치하려면 관리자가 다음 파일에 대한 읽기-쓰기 권한을 777 로 변경해야 합니다.

```
wily/ext/introscopeHpuxItaniumStats.jar
wily/ext/introscopeHpuxItaniumStats.so
wily/ext/introscopeHpuxParisc32Stats.jar
wily/ext/introscopeHpuxParisc32Stats.so
```

예(루트 사용자):

```
chmod 777 /<Agent_Home>/wily/ext/introscopeHpuxItaniumStats.jar
```

이러한 변경 작업은 에이전트를 설치한 후 에이전트를 시작하기 전에 수행합니다.

위에서 처음 두 개의 파일을 사용하는 경우에는 에이전트를 시작하기 전에 gcc 도 설치해야 합니다. HP 지원 웹 사이트를 검색하여 C 및 C++용 컴파일러인 gcc 를 다운로드할 수 있습니다.

플랫폼 모니터링 문제 해결

대부분의 경우 플랫폼 모니터는 운영 체제를 감지하여 해당 운영 체제가 지원되면 실행됩니다. 드물지만 감지되지 않을 경우에는 Java Agent 프로필에 운영 체제를 명시적으로 지정하여 플랫폼 모니터가 실행되도록 할 수 있습니다.

다음 단계를 따르십시오.

1. IntroscopeAgent.profile 을 엽니다.
2. Platform Monitor Configuration 제목 아래에서 사용 중인 운영 체제와 정확하게 일치하는 값을 찾아서 해당 속성의 주석 처리를 제거합니다. 예를 들어 Sun SPARC 하드웨어의 Sun Solaris 64 비트 운영 체제인 경우 다음 값을 찾아 주석 처리를 제거합니다.

   ```
   #introscope.agent.platform.monitor.system=SolarisSparc64
   ```

3. 관리되는 응용 프로그램을 다시 시작합니다.

Windows 의 플랫폼 모니터링 문제 해결

Windows 플랫폼에서는 Java Agent 에 대해 다음과 같은 오류가 나타날 수 있습니다.

```
11/28/06 08:29:55 AM PST [ERROR] [IntroscopeAgent] An error occurred polling for platform data
```

이 오류가 드물게 발생하는 경우 Windows 자체에서 발생하는 일시적인 오류로 인한 것일 수 있으며 해를 끼치지 않습니다. 이 오류가 Windows 이외의 플랫폼에서 발생하거나 항상 발생하는 경우에는 매우 심각한 상태임을 나타내므로 Introscope 의 CA Support 에 보고해야 합니다.

SAP Netweaver

SAP 사용자 계정에 대한 사용 권한이 충분하지 않으면 플랫폼 모니터가 SAP Netweaver 에서 올바르게 작동하지 않을 수 있습니다.

기본적으로 SAP 사용자 계정은 "컴퓨터 관리" > "시스템 도구" > "로컬 사용자 및 그룹" > "그룹" > "Performance Monitor Users" 아래에 등록되어 있지 않습니다. "Performance Monitor Users" 아래에 등록된 사용자는 Perfmon 관련 데이터(HKEY_PERFORMANCE_DATA)에 액세스할 수 있습니다.

SAP Netweaver 및 Introscope 성능 모니터링에 문제가 발생할 경우 "Performance Monitor Users" 그룹에 SAP 사용자 계정을 추가하고 Windows 컴퓨터를 다시 시작하여 이 문제를 해결할 수 있습니다.

제 14 장: CA APM 과 CA LISA 통합

이 섹션은 다음 항목을 포함하고 있습니다.

CA APM 과 CA LISA 통합

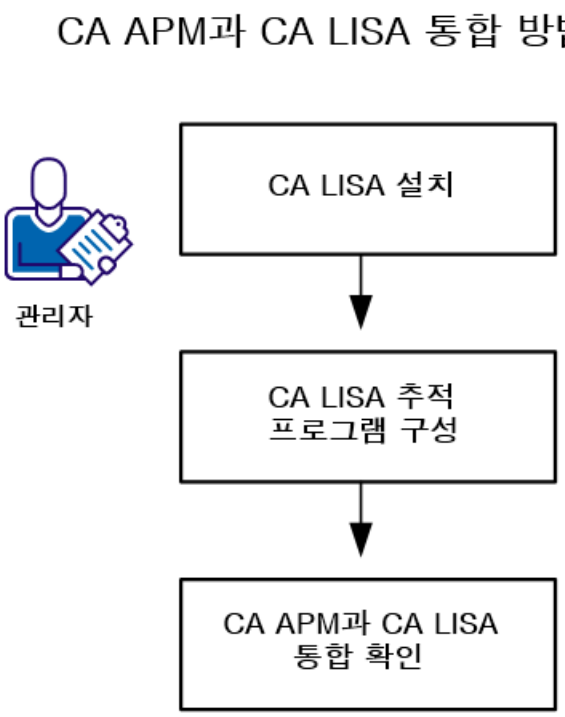
CA LISA 와 CA APM 을 통합함으로써 프로덕션 전 환경에서 CA LISA 의 부하 및 회귀 테스트로부터 생성된 가상 트랜잭션을 사용하여 응용 프로그램 성능 문제를 모니터, 감지, 심사 및 진단할 수 있습니다.

가상 트랜잭션은 다음 용도로 사용할 수 있는 실제 트랜잭션 성능을 나타냅니다.

- 응용 프로그램을 프로덕션 환경에 릴리스하기 전에 테스트 환경에서 데이터를 모니터합니다. 가상 데이터를 모니터링하면 응용 프로그램이 정의된 제한 내에서 성능을 발휘하며 기대한 서비스 수준 계약, 리소스 소비량 및 기준 성능 특성을 보증하는지 확인할 수 있습니다.
- 고객이 프로덕션 환경에 더 높은 품질의 응용 프로그램을 제공할 수 있도록 응용 프로그램 성능 문제 또는 병목 현상의 근본 원인을 소프트웨어 개발 수명 주기의 초기에 감지하고 식별합니다.
- 프로덕션 용량 계획을 위한 지침으로 사용하기 위해 프로덕션 전 환경에서 응용 프로그램 성능 및 리소스 사용량 수준에 대한 보고서를 생성합니다.

- 수집된 메트릭을 통해, 모니터링되는 프로덕션 응용 프로그램의 주요 측면에 대한 가시성이 제공되도록 합니다.

다음 그림에서는 CA LISA 를 CA APM 과 통합하기 위해 관리자가 수행해야 하는 작업을 보여 줍니다.



1. CA LISA 설치 (see page 242)
2. CA LISA 추적 프로그램을 구성합니다 (see page 243).
3. CA APM 과 CA LISA 의 통합을 확인합니다 (see page 243).

CA LISA 설치

다음과 같은 방법을 사용하여 CA LISA 를 설치할 수 있습니다.

- 수동으로 CA LISA 파일 다운로드 및 추출 (see page 239)
- 대화식 설치 관리자를 사용하여 CA LISA 설치 (see page 242)

수동으로 CA LISA 파일 다운로드 및 추출

CA APM 과 통합할 CA LISA 파일을 다운로드하고 추출합니다.

중요! CA LISA 통합을 설치하기 전에 먼저 JRE 가 설치되었는지 확인합니다.

다음 단계를 따르십시오.

1. CA Support의 CA APM 소프트웨어 다운로드 영역에서 CALISAIntegrationNoInstaller<*VersionNumber*>.windows.zip 또는 CALISAIntegrationNoInstaller<*VersionNumber*>.unix.tar 를 다운로드합니다.
2. 다른 에이전트 배포가 없는 디렉터리에 배포 파일 내용을 추출합니다.

   파일이 지정된 디렉터리에 추출됩니다.

CA APM 을 계측하여 CA LISA 의 테스트 이벤트 메트릭 및 데이터 보기

CA APM 을 계측하여 다음 CA LISA 프로세스에 대해 CA LISA 의 테스트 이벤트 메트릭과 데이터를 확인합니다.

- CA LISA Coordinator
- CA LISA Test Runner
- CA LISA Workstation

추가 CA LISA 프로세스를 계측할 수 있습니다. 그러나 CPU, 메모리 사용량, 에이전트 및 JVM ID 와 관련된 최소한의 메트릭만 보고됩니다.

다음 단계를 따르십시오.

1. Windows 또는 Linux 프로세스에 대한 사용 가능한 실행 파일 또는 셸 스크립트 파일과 동일한 이름의 .vmoptions 파일을 만들어 *<LISA_Home>*\bin 디렉터리에 저장합니다. 예를 들어 CA LISA Workstation 을 계측하려면 LISAWorkstation.vmoptions 파일을 *<LISA_Home>*\bin 디렉터리에 생성하십시오.

   다음 목록에서는 사용 가능한 실행 파일 또는 셸 스크립트 이름과 계측 가능한 프로세스를 보여 줍니다.

   - Coordinator 프로세스 - .vmoptions 확장명을 사용하여 CoordinatorServer.exe 또는 CoordinatorServer.sh 에 해당하는 파일을 생성합니다.
   - Registry 프로세스 - .vmoptions 확장명을 사용하여 Registry.exe, Registry.sh, TestRegistry.exe 또는 TestRegistry.sh 파일에 해당하는 파일을 생성합니다.
   - Simulator 프로세스 - .vmoptions 확장명을 사용하여 Detached_Simulator.exe, Detached_Simulator.sh, Simulator.exe 또는 Simulator.sh 파일에 해당하는 파일을 생성합니다.
   - TestRunner 프로세스 - .vmoptions 확장명을 사용하여 TestRunner.exe 또는 TestRunner.sh 파일에 해당하는 파일을 생성합니다.
   - Virtual Service Environment 프로세스 - .vmoptions 확장명을 사용하여 VirtualServiceEnvironmentService.exe 또는 VirtualServiceEnvironmentService.sh 파일에 해당하는 파일을 생성합니다.
   - Workstation 프로세스 - .vmoptions 확장명을 사용하여 LISAWorkstation.exe 또는 LISAWorkstation.sh 파일에 해당하는 파일을 생성합니다.

2. (선택 사항) Windows 서비스에 대해 사용 가능한 실행 파일 또는 셸 스크립트 파일과 동일한 이름의 .vmoptions 파일을 만들어 *<LISA_Home>*\bin 디렉터리에 저장합니다.

   다음 목록에서는 실행 파일 이름과 계측 가능한 서비스를 보여 줍니다.

   - Coordinator 서비스 - .vmoptions 확장명을 사용하여 CoordinatorService.exe 에 해당하는 파일을 생성합니다.
   - Simulator 서비스 - .vmoptions 확장명을 사용하여 SimulatorService.exe 파일에 해당하는 파일을 생성합니다.
   - Registry 서비스 - .vmoptions 확장명을 사용하여 TestRegistryService.exe 파일에 해당하는 파일을 생성합니다.
   - Virtual Service Environment 서비스 - .vmoptions 를 사용하여 extensionVirtualServiceEnvironmentService.exe 파일에 해당하는 파일을 생성합니다.

3. Java com.wily.introscope.agent.agentName 시스템 속성을 정의하는 행을 추가하여 에이전트 노드 이름을 변경합니다. 예를 들어 노드를 CA LISA Workstation Agent 라는 이름으로 지정하려면 다음 행을 추가합니다.
   ```
   -Dcom.wily.introscope.agent.agentName=CA LISA Workstation Agent
   ```

4. .vmoptions 파일에 다음 행을 추가합니다.
   ```
   -javaagent:<Agent_HOME>/Agent.jar
   -Dcom.wily.introscope.agentProfile=<Agent_HOME>/core/config/IntroscopeAgent.profile
   ```

   여기서 *<Agent_Home>*은 CA LISA 특정 에이전트 설치의 경로입니다. 일반적으로 *<Agent_Home>* 경로는 절대 경로지만 CA LISA 프로세스가 실행되는 현재 디렉터리에 상대적인 경로일 수 있습니다.

5. (선택 사항) 추가할 각 JVM 명령줄 옵션 또는 시스템 속성마다 별도의 행으로 입력합니다.

6. 계측되는 CA LISA 프로세스가 Java 1.7 에서 실행되는 경우 -XX:-UseSplitVerifier 를 추가합니다.

7. 응용 프로그램 서버를 다시 시작합니다.

   계측 설정이 적용됩니다.

설치 관리자를 사용하여 CA LISA 설치

설치 관리자를 사용하여 CA LISA 와 CA APM 을 통합할 수 있습니다.

다음 단계를 따르십시오.

1. 운영 체제에 적합한 설치 관리자를 선택합니다.
   - CALISAIntegrationInstaller<*VersionNumber*>.unix.tar
   - CALISAIntegrationInstaller<*VersionNumber*>.windows.zip
2. 선택한 설치 관리자를 CA LISA 서버에서 실행합니다.
   a. 설치 중에 CA LISA 설치의 홈 디렉터리 및 설치에 사용할 디렉터리를 지정합니다.
   b. 계측할 CA LISA 프로세스를 선택합니다.
   c. EM 및 포트를 지정합니다.

      설치 관리자는 계측 CA APM 을 계측하여 CA LISA 의 테스트 이벤트 메트릭 및 데이터 보기 (see page 240)에 설명된 대로 <*LISA_Home*>\bin 디렉터리에 필요한 vmoptions 파일을 생성합니다.

      참고: 계측되는 CA LISA 응용 프로그램이 Java 1.7 에서 실행되는 경우 설치 후 수동으로 vmoptions 파일을 편집하여 -XX:-UseSplitVerifier 를 추가해야 합니다.
3. EM 설치 중에 모니터링 옵션으로 CA APM Integration for CA LISA 를 선택합니다.

   관리 모듈 및 Typeview 에 CA LISA 데이터가 표시됩니다.

   참고: 설치 관리자를 실행할 때 CA APM Integration for CA LISA 모니터링 옵션을 선택하지 않은 경우 EM 설치 디렉터리에 있는 <EM_Home>\examples\CAAPMIntegrationForCALISA 폴더에서 파일을 복사하여 수동으로 이 단계를 수행할 수 있습니다.

CA LISA 추적 프로그램 구성

CA LISA 추적 프로그램을 구성하여 모니터할 CA LISA 시스템 구성 요소를 사용자 지정합니다. CA LISA 추적 프로그램을 사용하면 다음 노드에 메트릭이 보고되는 가장 낮은 수준을 제어하여 CA APM 에 보고되는 메트릭 수를 제한할 수 있습니다.

- Test Case(테스트 사례)
- Simulator(시뮬레이터)
- Test Steps(테스트 단계)

메트릭이 보고되는 최소 수준은 Test Case(테스트 사례) 노드에 있습니다.

다음 단계를 따르십시오.

1. *<LISA_Home>*\core\config 로 이동하고 lisa.pbd 구성 파일을 열어 환경의 필요에 맞게 구성한 다음 파일을 저장합니다.
2. *<LISA_Home>*\core\config 로 이동하고 IntroscopeAgent.profile 구성 파일을 열어 환경의 필요에 맞게 구성한 다음 파일을 저장합니다.
3. 응용 프로그램 서버를 다시 시작합니다.

   구성 설정이 적용됩니다.

CA APM 과 CA LISA 의 통합 확인

CA LISA 프로세스를 호출하고 Investigator 트리에 계측된 노드가 있는지 확인하여 CA APM 과 CA LISA 통합을 확인할 수 있습니다.

참고: CA LISA 테스트를 사용해 호출되어 Investigator 트리의 <CA LISA> | Test Case 노드 아래에 나타나는 메트릭은 CA LISA 테스트가 CA LISA Coordinator, CA LISA Test Runner 또는 CA LISA Workstation 에서 실행된 후에만 Enterprise Manager 에 보고됩니다.

다음 단계를 따르십시오.

1. CA LISA Workstation 을 사용하여 테스트를 실행합니다. CA LISA 사용에 대한 자세한 내용은 CA LISA 설명서 집합을 참조하십시오.
2. Enterprise Manager 가 실행되고 있지 않으면 시작합니다.
3. CA LISA 프로세스가 실행되고 있지 않으면 시작합니다.

   CA LISA 프로세스를 다시 시작하면 에이전트가 시작됩니다.
4. Workstation 을 시작하고 Investigator 를 엽니다.
5. 다음 노드 아래에서 데이터를 찾습니다.

   SuperDomain | *<Host_Name>* | CA LISA | *<CA LISA Workstation>* | CA *LISA* | Test Case | *<Test Case Name>*

   - <Host_Name>은 CA LISA 에이전트가 설치된 컴퓨터가 속한 노드입니다.
   - <CA LISA Workstation>은 LISAWorkstation.vmoptions 파일에 에이전트 이름이 지정된 노드입니다. 이 노드의 구성에 대한 자세한 내용은 CA APM 을 계측하여 CA LISA 의 테스트 이벤트 메트릭 및 데이터 보기 (see page 240)를 참조하십시오.
   - <Test Case Name>은 유효성 검사의 첫 번째 단계에서 실행되는 테스트 사례의 이름입니다.
6. 폴더를 확장합니다.

   구성된 노드 아래에 활성 메트릭이 표시됩니다.

제 15 장: CA APM 과의 CA APM Cloud Monitor 통합

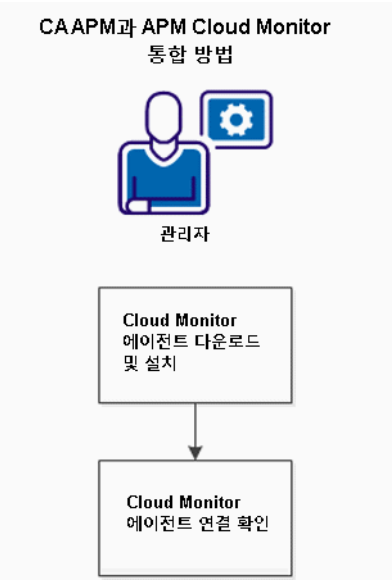
CA APM Cloud Monitor 를 사용하면 다음 작업을 수행할 수 있습니다.

- 40 개 이상의 국가에 있는 60 개 이상의 모니터링 스테이션을 통해 전체 사용자 환경을 이해합니다.
- 실제 브라우저를 모니터링하여 사용자 경험을 정확하게 측정합니다.
- SaaS 공급업체와 MSP 에서 제공하는 응용 프로그램을 모니터링하여 SLA 에 적절한 상태로 유지합니다.
- 가상 트랜잭션을 통해 방화벽 외부로부터의 응용 프로그램 응답 시간을 테스트하여 글로벌 최종 사용자 경험을 이해하고 심지어 실제 사용자 트래픽이 없을 때도 성능을 모니터링할 수 있습니다.
- 실제 사용자 트랜잭션을 복제하고 이를 통해 응용 프로그램 인프라 전체의 성능을 모니터링하여 문제를 빠르게 식별, 진단 및 해결합니다.

CA APM 배포에서 CA APM Cloud Monitor 를 통합하는 방법

APM 관리자는 CA APM 배포에서 CA APM Cloud Monitor 를 통합하여 모니터링 및 심사 기능을 더 추가할 수 있습니다.

다음 순서도에서는 온 프레미스 CA APM 배포에서 CA APM Cloud Monitor 를 통합하는 방법을 보여 줍니다.



1. CA APM Cloud Monitor 에이전트를 다운로드하여 설치합니다. (see page 246)
2. 에이전트 연결의 유효성을 검사합니다. (see page 247)

CA APM Cloud Monitor 에이전트 다운로드 및 설치

이 작업에서는 CA APM Cloud Monitor 와 CA APM 을 통합하는 데 필요한 CA APM Cloud Monitor 에이전트를 다운로드하고 설치합니다.

참고: 네트워크의 원하는 컴퓨터에 CA APM Cloud Monitor 에이전트를 설치할 수 있습니다. 이 에이전트의 기능이 인터넷을 통해 CA APM Cloud Monitor 데이터를 수신하여 Enterprise Manager 에 전달하는 것이므로 모든 컴퓨터에 이 에이전트를 설치할 수 있지만, 서버 수준 CPU/RAM 이 있는 컴퓨터를 선택해야 합니다.

CA APM Cloud Monitor 에이전트를 다운로드하여 설치하려면

1. CA APM Cloud Monitor 에이전트 아카이브 파일을 다운로드합니다. 사용 중인 컴퓨터에 파일을 저장합니다.

   Windows, Linux 및 Solaris 용 설치 관리자는 CA APM 소프트웨어 다운로드 웹 사이트에 있습니다.

2. 에이전트를 설치합니다.

   a. CA APM Cloud Monitor 설치 관리자 파일을 시작합니다.

   b. 라이선스 계약의 조건에 동의합니다.

   c. *<Agent_Home>* 디렉터리를 지정합니다.

   d. 에이전트가 연결될 Enterprise Manager 호스트와 포트를 지정합니다.

   e. "Cloud Monitor Account Settings"(클라우드 모니터 계정 설정) 화면에 CA APM Cloud Monitor 사용자 ID 와 암호를 입력합니다. 이러한 자격 증명은 CA APM Cloud Monitor 웹 사이트에 로그인하는 데 사용하는 것과 동일합니다.

CA APM Cloud Monitor 에이전트 설치가 완료됩니다.

참고: 에이전트를 시작하고 연결의 유효성을 검사하는 방법에 대한 자세한 내용은 CA APM Cloud Monitor 에이전트 연결의 유효성 검사 (see page 247)를 참조하십시오.

CA APM Cloud Monitor 에이전트 연결의 유효성 검사

유효성 검사를 통해 CA APM Cloud Monitor 에이전트가 올바로 설치되었는지 확인하려면 에이전트를 시작하고 CA APM WebView 또는 Workstation 을 사용하여 데이터를 확인하십시오. CA APM Cloud Monitor 에서 들어오는 데이터가 최신인지 확인하여 연결의 유효성을 검사하십시오.

참고: WebView 또는 Workstation 에서 CA APM Cloud Monitor 데이터를 보려면 먼저 CA APM Cloud Monitor 에서 폴더와 모니터를 설정해야 합니다. 이 단계를 수행하지 않았으면 *CA APM Workstation 사용자 안내서*의 CA APM Cloud Monitor 사용 방법 단원을 참조하십시오.

다음 단계를 따르십시오.

1. CA APM Cloud Monitor 에이전트가 설치되어 있는 디렉터리로 이동한 후 다음 파일 중 하나를 실행하여 에이전트를 시작합니다.
   - Windows 에서는 APMCloudMonitor.bat 를 두 번 클릭합니다.
   - Linux 또는 Solaris 에서는 APMCloudMonitor.sh 를 실행합니다.

   콘솔 창에서 에이전트가 시작됩니다.
2. CA APM WebView 또는 Workstation 을 시작합니다.
3. CA APM Cloud Monitor 에서 데이터를 찾습니다.
   a. Investigator 창이 열려 있지 않은 경우 "파일" > "새 Investigator"를 선택합니다.
   b. "메트릭 브라우저" 탭을 찾습니다.
   c. 다음 노드를 확장합니다.

      SuperDomain | *<Host_Name>* | APMCloudMonitor | APMCloudMonitorAgent | APM Cloud Monitor

      *<Host_Name>*은 일반적으로 CA APM Cloud Monitor 에이전트를 설치한 컴퓨터이지만 메트릭 브라우저 트리에 표시되는 내용은 APMCloudMonitor.properties 파일에 있는 apmcm.agent.hostName 속성의 값입니다.
   d. 폴더를 확장하여 개별 메트릭을 보고 메트릭이 최신 상태인지 확인합니다.

문제 해결

Enterprise Manager 설치 중에 "클라우드 모니터"를 모니터링 옵션으로 선택하지 못한 경우 WebView/Workstation 은 CA APM Cloud Monitor 데이터의 일부 표시를 허용하지 않습니다.

이 문제를 해결하려면

- *<EM_Home>*\examples\CAAPMIntegrationForCloudMonitor\ 디렉터리의 내용을 <EM_Home>/config/modules/ 및 <EM_Home>/ext/xmltv 에 복사합니다.

  그러면 클라우드 모니터 메트릭 데이터를 제대로 표시하는 데 필요한 관리 모듈과 기타 파일을 사용할 수 있게 됩니다.

데이터를 제한하는 방법

성능을 향상시키기 위해 CA APM Cloud Monitor 에이전트가 Enterprise Manager 에 보내는 데이터 양을 제한할 수 있습니다.

CA APM Cloud Monitor 속성을 구성하여 데이터 제한

APMCloudMonitor.properties 파일의 속성을 사용하여 CA APM Cloud Monitor 에이전트가 Enterprise Manager 에 보내는 데이터를 필터링할 수 있습니다.

CA APM Cloud Monitor 에이전트 속성을 구성하여 데이터를 필터링하려면 *<CloudMonitor_Agent_Home>*/CloudMon/conf/APMCloudMonitor.properties 에서 메트릭 필터 섹션을 편집합니다.

이러한 설정에 대한 자세한 내용은 APMCloudMonitor.properties 에 대한 속성 참조를 참조하십시오.

검사점을 제거하여 데이터 제한

CA APM Cloud Monitor 는 다섯 대륙에서 60 개 이상의 검사점 스테이션에 액세스할 수 있습니다. 클라우드 모니터는 이러한 스테이션 중에서 무작위로 선택해 해당 스테이션으로부터 사용자의 웹 사이트나 응용 프로그램에 대해 가용성과 성능을 검사합니다. 지속적으로 사용하도록 설정된 모든 스테이션에서 이 검사를 수행하여 60 개 이상의 각 사이트에서 데이터가 로깅됩니다.

사용 가능한 검사점 스테이션 중 일부를 제거하여 CA APM Cloud Monitor 가 CA APM 으로 전송하는 데이터의 양을 제한할 수 있습니다.

다음 단계를 따르십시오.

1. CA APM Cloud Monitor 웹 사이트 cloudmonitor.ca.com 에 로그인합니다.
2. "Subscriptions"(구독) > "Preferences"(기본 설정)를 선택합니다.

   모든 스테이션은 기본적으로 선택되어 있습니다.

3. 다음처럼 기본 선택을 변경합니다.
   - 개별 스테이션에 대한 확인란을 선택 취소합니다. *또는*
   - "지우기"를 클릭하여 모두 선택 취소하고 목록 맨 위의 그룹 중에서 선택합니다.

     예를 들어 북아메리카에 있는 스테이션만을 선택하려는 경우에는

     a. "Clear"(지우기)를 클릭합니다.

     b. "North America"(북아메리카)를 클릭합니다.
4. 페이지 아래쪽에서 "Change"(변경)를 클릭합니다.

일정을 조정하여 데이터 제한

기본적으로 모니터는 가용성과 성능을 매일 매시간마다 정기적으로 검사합니다. 시간이 경과함에 따라 CA APM 이 반환하는 데이터 양은 사용자가 원하는 양보다 많아질 수 있습니다.

다음 단계를 따르십시오.

1. 클라우드 모니터 웹 사이트 cloudmonitor.ca.com 에 로그인합니다.
2. "Settings"(설정) > "Monitors"(모니터)를 선택합니다.
3. 개별 모니터를 선택하고 "More Options"(추가 옵션)를 선택합니다.
4. 다음 설정을 다시 설정합니다.
   - Delay between checks(검사 간 지연 시간)
   - Check period(검사 기간)
   - Check on these days only(최근 날짜에 대해서만 검사)
   - Maintenance schedule(유지 관리 일정)

부록 A: Java Agent 속성

IntroscopeAgent.profile 위치 구성

에이전트는 IntroscopeAgent.profile 에 있는 속성을 참조하여 기본 연결 및 이름 지정 속성을 확인합니다. 에이전트를 설치하면 *<Agent_Home>*/wily/core/config 디렉터리에 에이전트 프로필이 설치됩니다.

Introscope 는 다음 위치에서 순서대로 에이전트 프로필을 찾습니다.

- 시스템 속성 com.wily.introscope.agentProfile 에 정의된 위치
- com.wily.introscope.agentResource 에 정의된 위치
- *<working directory>*/wily/core/config 디렉터리

참고: Windows 컴퓨터에 있는 경로를 추가할 경우 백슬래시는 C:\\Introscope\\lib\\Agent.jar 와 같이 다른 백슬래시로 이스케이프하십시오.

IntroscopeAgent.profile 의 위치를 변경하려면

1. 다음 방법 중 하나를 사용하여 새 위치를 정의합니다.
   - 다음과 같이 Java 명령줄에서 **-D** 옵션으로 시스템 속성을 정의하여 IntroscopeAgent.profile 파일의 전체 경로를 지정합니다.
   - -D com.wily.introscope.agentProfile.
   - 클래스 경로의 리소스에서 IntroscopeAgent.profile 을 사용할 수 있도록 설정합니다. *com.wily.introscope.agentResource* 를 설정하여 에이전트 프로필이 들어 있는 리소스의 경로를 지정합니다.

     참고: IntroscopeAgent.profile 의 위치를 변경할 경우에는 AutoProbe 로그 위치도 변경해야 합니다. 자세한 내용은 ProbeBuilder 로그 관리 (see page 166)를 참조하십시오.
2. ProbeBuilder 지시문(PBD 및 PBL 파일)을 에이전트 프로필과 동일한 위치(프로필 위치에 상대적인 위치)로 이동합니다.

Sun ONE 을 사용하는 경우에는 Sun ONE *server.xml* 파일에 에이전트 프로필의 새 위치를 추가하십시오.

Sun ONE 용 IntroscopeAgent.profile 의 위치를 변경하려면

1. Sun ONE 7.0 용 시작 스크립트에 Introscope 정보를 추가하려면 관리자 또는 루트로 로그인합니다.
2. <*SunOne_Home*>/domains/domain1/server1/config/에 있는 server.xml 파일을 엽니다.
3. server.xml 의 jvm-options 스탠자에 행을 추가합니다.

```
<jvm-options>
-Dcom.wily.introscope.agentProfile=SunOneHome/wily/core/config/IntroscopeAgent.profile
</jvm-options>
```

명령줄 속성 재정의

Introscope 에서 명령줄을 사용하여 Enterprise Manager, 에이전트, Workstation 및 WebView 의 특정 속성을 재정의할 수 있습니다. Java Agent 와 관련하여 이 기능은 여러 에이전트 복사본이 공유되는 클러스터 환경에서 모니터링되는 각 응용 프로그램에 대한 에이전트 설정 중 일부를 수정하려는 경우에 유용합니다.

다음 단계에서는 모니터링할 응용 프로그램 서버에 에이전트를 설치하고 구성했으며 에이전트가 Enterprise Manager 에 올바르게 연결되어 있다고 가정합니다.

명령줄을 사용하여 에이전트 속성을 재정의하려면

1. 에이전트를 시작하도록 Java 명령을 수정한 파일을 엽니다.

   이 파일의 위치는 사용자 환경에서 사용하는 응용 프로그램 서버에 따라 다릅니다.

2. *-D* 명령을 추가하여 속성을 재정의합니다. 예를 들어 다음 명령을 추가하여 에이전트가 *weblogic-full.pbl* 파일도 사용하도록 설정할 수 있습니다.

   ```
   -Dintroscope.autoprobe.directivesFile=weblogic-full.pbl
   ```

   열려 있는 파일에서 다른 *-D* 명령 옆에 이 명령을 배치합니다.

   참고: 이 명령을 사용하여 핫 배포 가능 속성을 재정의할 경우 해당 속성은 더 이상 핫 배포 가능 속성이 아닙니다. 또한 나중에 구성 파일에서 이 속성을 수정하면 재정의된 속성을 수정했으므로 변경 사항이 적용되지 않는다는 경고 메시지가 Workstation 에 표시됩니다. 이 문제를 방지하려면 구성 파일의 속성을 수정하기 전에 재정의 명령을 제거하십시오.

3. 파일을 저장합니다.

4. 에이전트를 다시 시작합니다.

   위에 사용된 예제의 경우 Workstation 의 에이전트 노드에 추가 WebLogic 메트릭이 표시됩니다.

   중요! 시스템 속성은 Introscope 속성의 속성 공간에 포함되므로 IndexedProperties 를 사용하는 모든 항목에서 *java.io.tmpdir* 같은 속성을 볼 수 있습니다.

에이전트 장애 조치

에이전트 장애 조치 속성은 Java Agent 와 기본 Enterprise Manager 의 연결이 끊어질 경우 에이전트가 장애 조치될 Enterprise Manager 와 기본 Enterprise Manager 에 재연결을 시도할 빈도를 지정합니다.

introscope.agent.enterprisemanager.connectionorder

에이전트와 기본 Enterprise Manager 간의 연결이 끊어질 경우 에이전트에서 사용하는 백업 Enterprise Manager 의 연결 순서를 지정합니다.

속성 설정

에이전트가 연결할 수 있는 다른 Enterprise Manager 의 이름

기본값

DEFAULT 호스트 이름, 포트 번호 및 소켓 팩터리 속성으로 정의된 Enterprise Manager

예

```
introscope.agent.enterprisemanager.connectionorder=DEFAULT
```

참고

- 쉼표로 구분된 목록을 사용하십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds

거부된 에이전트가 다음 Enterprise Manager 에 다시 연결하려고 하는 각 시도 사이의 시간 간격(초)을 지정합니다.

- 에이전트 프로필 introscope.agent.enterprisemanager.connectionorder 속성 값에 구성된 순서를 기반으로 하는 Enterprise Manager
- loadbalancing.xml 구성을 기반으로 하는 허용되는 모든 Enterprise Manager

에이전트는 Enterprise Manager 에 연결할 수 없는 경우 다음과 같은 방식으로 연결을 처리합니다.

- 허용되는 다음 Enterprise Manager 에 연결을 시도합니다.
- 허용되지 않는 Enterprise Manager 에는 연결하지 않습니다.

참고: loadbalancing.xml 구성 및 에이전트와 Enterprise Manager 간의 연결 구성에 대한 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

기본값

기본 간격은 120 초입니다.

예

```
introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120
```

참고

이 속성의 변경 사항이 적용되도록 관리되는 응용 프로그램을 다시 시작하십시오.

이 속성은 기본적으로 주석으로 처리되어 있습니다.

이 속성은 에이전트가 다음 CA APM 구성 요소를 통과하여 연결할 수 있는 환경에 유용합니다.

- 클러스터
- 수집기 및 독립 실행형 Enterprise Manager
- 클러스터, 수집기 및 독립 실행형 Enterprise Manager 의 조합

다음 예에서는 에이전트가 다른 클러스터의 Enterprise Manager 에 연결할 수 있고 이 속성이 설정되어 있지 않은 경우에 발생하는 결과를 보여 줍니다.

1. 클러스터의 Enterprise Manager 에 연결된 에이전트가 연결이 끊어집니다.
2. 에이전트는 클러스터 2 의 Enterprise Manager 에 허용되지 않는 모드로 연결합니다.
3. 에이전트는 클러스터 1 의 허용되는 Enterprise Manager 를 사용할 수 있게 되는 시기를 알 수 없습니다.

이 속성은 Enterprise Manager 에 연결할 수 있게 될 때까지 에이전트가 허용되는 Enterprise Manager 에 계속 연결을 시도하도록 합니다.

에이전트 HTTP 터널링

터널링 기술을 사용하여 정보를 보내도록 에이전트를 구성하면 에이전트를 Enterprise Manager 에 원격으로 연결할 수 있습니다. 이렇게 하려면 HTTP 터널링 웹 서비스가 호스팅되는 Enterprise Manager 의 포함된 웹 서버에 연결하도록 에이전트를 구성해야 합니다.

IntroscopeAgent.profile 에서 HTTP 터널링 통신을 새 에이전트 연결로 구성하려면 다음을 지정하십시오.

- Enterprise Manager 가 실행되는 컴퓨터의 호스트 이름. 자세한 내용은 introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 258)를 참조하십시오.
- Enterprise Manager 웹 서버에 대한 연결 포트. 이는 에이전트가 연결할 Enterprise Manager 의 *IntroscopeEnterpriseManager.properties* 에 지정된 *introscope.enterprisemanager.webserver.port* 속성 값입니다.
- HTTP 터널링 소켓 팩터리. 다음과 같은 클라이언트 소켓 팩터리를 지정합니다.
  `com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory`

프록시 서버에 대한 에이전트 HTTP 터널링

다음 속성은 HTTP 를 통해 터널링하도록 구성된 에이전트에만 적용되며 프록시 서버를 사용하여 Enterprise Manager 에 연결해야 합니다.

- introscope.agent.enterprisemanager.transport.http.proxy.host (see page 257)
- introscope.agent.enterprisemanager.transport.http.proxy.port (see page 257)
- introscope.agent.enterprisemanager.transport.http.proxy.username (see page 257)
- introscope.agent.enterprisemanager.transport.http.proxy.password (see page 258)

자세한 내용은 HTTP 터널링을 위한 프록시 서버 구성 (see page 70)을 참조하십시오.

introscope.agent.enterprisemanager.transport.http.proxy.host

프록시 서버 호스트 이름을 지정합니다.

기본값

주석 처리됨. 지정되지 않음

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.http.proxy.port

프록시 서버 포트 번호를 지정합니다.

기본값

주석 처리됨. 지정되지 않음

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.http.proxy.username

프록시 서버는 에이전트가 인증해야 하며, 인증을 위한 사용자 이름을 지정합니다.

기본값

주석 처리됨. 지정되지 않음

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.http.proxy.password

프록시 서버는 에이전트가 인증해야 하며, 인증을 위한 암호를 지정합니다.

기본값

주석 처리됨. 지정되지 않음

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

에이전트 HTTPS 터널링

HTTPS 를 사용하여 정보를 보내도록 에이전트를 구성하면 에이전트를 Enterprise Manager 에 원격으로 연결할 수 있습니다.

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 258)
- introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT (see page 259)
- introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT (see page 259)

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

에이전트가 기본적으로 연결되는 Enterprise Manager 를 실행하는 컴퓨터의 호스트 이름을 지정합니다.

기본값

localhost

예

```
introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=localhost
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

Enterprise Manager 를 호스트하는 컴퓨터의 포트 번호를 지정합니다. HTTPS 터널링을 사용하는 경우 에이전트에서 연결을 수신하는 기본 포트는 8444 입니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다.

기본값

8444

예

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

HTTPS 을 사용할 때 에이전트에서 Enterprise Manager 로의 연결에 사용할 클라이언트 소켓 팩터리를 지정합니다.

기본값

com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory

예

```
introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

에이전트 메모리 오버헤드

상당한 에이전트 메모리 오버헤드는 극단적인 몇몇 경우에만 발생합니다. 일반적으로 메모리 소비를 줄이면 대신 응답 시간이 늘어날 수 있습니다. 그러나 각 응용 프로그램은 고유하며 메모리 사용량과 응답 시간 사이의 균형도 응용 프로그램 자체에 따라 달라질 수 있습니다.

introscope.agent.reduceAgentMemoryOverhead

이 속성은 사용할 에이전트 구성을 지정합니다. 에이전트 메모리 오버헤드를 줄이려면 주석 처리를 제거하십시오. 이 속성은 기본적으로 true 로 설정되고 주석 처리되어 있습니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.reduceAgentMemoryOverhead=true
```

참고

이 속성의 변경 내용이 적용되도록 관리되는 응용 프로그램을 다시 시작하십시오.

에이전트 메트릭 만료 처리

에이전트 메트릭 만료 처리는 사용되지 않는 메트릭을 주기적으로 에이전트 메모리 캐시에서 제거합니다. 사용되지 않는 메트릭은 구성된 시간 동안 새 데이터를 보고하지 않은 메트릭입니다. 오래된 메트릭을 제거하면 에이전트 성능을 향상하고 발생할 수 있는 메트릭 급증을 방지할 수 있습니다.

참고: 시스템이 처리할 수 있는 것보다 많은 메트릭을 보고하도록 부주의하게 에이전트를 설정하는 경우 메트릭이 급증합니다. 너무 많은 메트릭이 보고되면 응용 프로그램 서버의 성능에 영향을 미칠 수 있고 심한 경우 서버가 전혀 작동하지 않게 될 수도 있습니다.

그룹에 속한 메트릭은 그룹의 모든 메트릭이 제거 후보에 해당하는 경우에만 제거됩니다. 현재 *BlamePointTracer* 및 *MetricRecordingAdministrator* 메트릭만 그룹으로 제거됩니다. 다른 메트릭은 개별적으로 제거됩니다.

MetricRecordingAdministrator 에는 메트릭 그룹을 생성, 검색 또는 제거하기 위한 다음과 같은 인터페이스가 있습니다.

- *getAgent().IAgent_getMetricRecordingAdministrator.addMetricGroup*

  문자열 구성 요소, 수집 메트릭. 구성 요소 이름은 메트릭 그룹의 메트릭 리소스 이름입니다. 동일한 메트릭 노드 아래에 매트릭이 있어야만 그룹으로 간주됩니다. 메트릭은 *com.wily.introscope.spec.metric.AgentMetric* 데이터 구조의 모음입니다. 이 모음에는 *AgentMetric* 데이터 구조만 추가할 수 있습니다.

- *getAgent().IAgent_getMetricRecordingAdministrator.getMetricGroup*

  문자열 구성 요소. 메트릭 리소스 이름인 구성 요소 이름을 기준으로 메트릭의 모음을 가져올 수 있습니다.

- *getAgent().IAgent_getMetricRecordingAdministrator.removeMetricGroup*

  문자열 구성 요소. 메트릭 그룹은 메트릭 리소스 이름인 구성 요소를 기반으로 제거됩니다.

- *getAgent().IAgent_getDataAccumulatorFactory.isRemoved*

  메트릭이 제거되었는지 확인합니다. 확장에 있는 누산기의 인스턴스를 유지하려는 경우 이 인터페이스를 사용합니다. 메트릭 만료 처리로 인해 누산기가 제거되면 이 인터페이스를 통해 사용되지 않는 참조를 유지하지 않도록 합니다.

중요! 다른 CA Technologies 제품에서 사용하기 위한 용도 등으로 *MetricRecordingAdministrator* 인터페이스를 사용하는 확장을 생성하는 경우 사용자 고유의 누산기 인스턴스를 제거해야 합니다. 호출되지 않았다는 이유로 메트릭이 만료 처리되면 나중에 해당 메트릭의 데이터를 사용할 수 있게 되어도 이전 누산기 인스턴스가 새 메트릭 데이터 포인트를 생성하지 않습니다. 이러한 상황을 방지하려면 사용자 고유의 누산기 인스턴스를 삭제하지 말고 대신 *getDataAccumulatorFactory* 인스턴스를 사용하십시오.

에이전트 메트릭 만료 처리 구성

에이전트 메트릭 만료 처리는 기본적으로 사용됩니다. 이 기능을 해제하려면 introscope.agent.metricAging.turnOn (see page 262) 속성을 사용합니다. 이 속성을 *IntroscopeAgent.profile* 에서 제거하면 에이전트 메트릭 만료 처리가 기본적으로 해제됩니다.

에이전트 메트릭 만료 처리는 에이전트의 하트비트에 실행됩니다. 하트비트는 introscope.agent.metricAging.heartbeatInterval (see page 263) 속성을 사용하여 구성됩니다. 하트비트 빈도를 낮게 유지해야 합니다. 하트비트가 높으면 에이전트와 CA Introscope®의 성능에 영향을 미칩니다.

각 하트비트 동안 특정 메트릭 집합이 확인됩니다. 이 집합은 introscope.agent.metricAging.dataChunk (see page 263) 속성을 사용하여 구성할 수 있습니다. 이 값도 낮게 유지해야 합니다. 이 값이 높으면 성능에 영향을 줍니다. 기본값은 하트비트당 메트릭을 500 개 확인합니다. 메트릭 500 개 각각을 확인하여 제거 후보에 해당하는지 확인합니다. 예를 들어 이 속성을 하트비트당 메트릭 500 개 단위의 청크를 확인하도록 설정하는 경우 에이전트 메모리의 메트릭이 총 10,000 개 있으면 메트릭 10,000 개를 모두 확인하느라 성능이 저하되고 시간이 오래 걸립니다. 그러나 이 속성을 더 높은 값으로 설정하면 모든 10,000 개 메트릭을 더 빨리 확인하지만 오버헤드가 높아집니다.

특정 기간 후 메트릭에 새 데이터가 수신되지 않은 경우 메트릭은 제거 후보입니다. 이 기간은 introscope.agent.metricAging.numberTimeslices (see page 264) 속성을 사용하여 구성할 수 있습니다. 이 속성은 기본적으로 3000 으로 설정되어 있습니다. 메트릭이 제거 조건을 충족하면 해당 그룹의 모든 메트릭이 메트릭 제거 후보인지 확인합니다. 이 요구 사항도 충족하면 메트릭이 제거됩니다.

introscope.agent.metricAging.turnOn

에이전트 메트릭 만료 처리를 설정하거나 해제합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.metricAging.turnOn=true
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.metricAging.heartbeatInterval

제거할 메트릭을 확인하는 시간 간격을 초 단위로 지정합니다.

기본값

1800

예

```
introscope.agent.metricAging.heartbeatInterval=1800
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.metricAging.dataChunk

각 간격에서 확인할 메트릭의 수를 지정합니다.

기본값

500

예

```
introscope.agent.metricAging.dataChunk=500
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.metricAging.numberTimeslices

이 속성은 데이터를 제거 대상으로 만들기 전에 새 데이터가 없는지 확인하는 간격의 수를 지정합니다.

기본값

3000

예

```
introscope.agent.metricAging.numberTimeslices=3000
```

참고

이 속성에 대한 변경 내용은 즉시 적용되므로 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.metricAging.metricExclude.ignore.0

지정한 메트릭을 제거하지 않도록 제외합니다. 만료 처리에서 하나 이상의 메트릭을 제외하려면 목록에 메트릭 이름이나 메트릭 필터를 추가합니다.

속성 설정

메트릭을 쉼표로 구분한 목록입니다. 메트릭 이름에서 별표(*)를 와일드카드로 사용할 수 있습니다.

기본값

기본값은 Threads 로 시작하는 메트릭 이름(*Threads**)입니다.

예

```
introscope.agent.metricAging.metricExclude.ignore.0=Threads*
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

에이전트 메트릭 클램프

에이전트가 Enterprise Manager 로 보내지는 메트릭의 수를 대략적으로 클램프하도록 구성할 수 있습니다. 생성된 메트릭 수가 속성 값을 초과하면 에이전트는 새 메트릭의 수집 및 전송을 중지합니다.

introscope.agent.metricClamp

에이전트가 Enterprise Manager 로 보내지는 메트릭의 수를 대략적으로 클램프하도록 구성합니다.

기본값

5000

예

```
introscope.agent.metricClamp=5000
```

참고

- 이 속성이 설정되어 있지 않으면 메트릭 클램프가 적용되지 않습니다. 즉, 오래된 메트릭도 여전히 값을 보고합니다.
- 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.
- 이 클램프 속성은 apm-events-thresholds-config.xml 파일에 있는 introscope.enterprisemanager.agent.metrics.limit 속성과 함께 작동합니다.

  참고: introscope.enterprisemanager.agent.metrics.limit 속성에 대한 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

  introscope.enterprisemanager.agent.metrics.limit 클램프 값이 introscope.agent.metricClamp 값보다 먼저 트리거되면 Enterprise Manager 가 에이전트 메트릭을 읽기는 하지만 Investigator 메트릭 브라우저 트리에서 해당 메트릭을 보고하지는 않습니다.

  introscope.agent.metricClamp 클램프 값이 introscope.enterprisemanager.agent.metrics.limit 클램프 값보다 먼저 트리거되면 에이전트에서 Enterprise Manager 로의 메트릭 전송이 중지됩니다.

에이전트 이름 지정

응용 프로그램 서버에 대한 Java Agent 이름 등을 가져오도록 속성을 구성할 수 있습니다.

추가 정보:

Java Agent 이름 이해 (페이지 143)

introscope.agent.agentAutoNamingEnabled

지원되는 응용 프로그램 서버에 대한 Java Agent 이름을 가져오는 데 에이전트 자동 이름 지정을 사용할지 여부를 지정합니다.

속성 설정

True 또는 False

기본값

응용 프로그램 서버에 따라 다릅니다. 아래의 참고를 참조하십시오.

예

```
introscope.agent.agentAutoNamingEnabled=false
```

참고

- WebLogic, WebSphere 및 JBoss 응용 프로그램 서버에서는 에이전트 자동 이름 지정이 **사용하도록 설정**되어 있습니다.
- 이 속성을 사용하려면 WebLogic 의 경우 시작 클래스를 지정하고 WebSphere 의 경우 사용자 지정 서비스를 지정해야 합니다.
- Oracle 응용 프로그램 서버, Interstage, Sun ONE 및 Tomcat 응용 프로그램 서버에서는 에이전트 자동 이름 지정이 **사용하지 않도록 설정**되어 있습니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

중요! WebLogic, WebSphere 및 JBoss 의 경우 *introscope.agent.agentAutoNamingEnabled* 속성은 기본적으로 **TRUE** 로 설정되어 있습니다.

introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds

Enterprise Manager 에 연결하기 전에 에이전트가 이름 지정 정보를 기다리는 시간(초)을 지정합니다.

기본값

120

예

```
introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds=120
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.agentAutoRenamingIntervalInMinutes

에이전트가 이름이 변경되었는지 확인하는 시간 간격(분)을 지정합니다.

기본값

10

예

```
introscope.agent.agentAutoRenamingIntervalInMinutes=10
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.agentName

다른 에이전트 이름 지정 방법이 실패하는 경우 기본 에이전트 이름을 제공하도록 이 속성의 주석 처리를 제거하십시오.

속성 설정

모든 설치에서 이 속성의 값이 잘못되었거나 이 속성이 프로필에서 삭제된 경우 에이전트 이름은 *UnnamedAgent* 가 됩니다.

예

```
#introscope.agent.agentName=AgentName
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 응용 프로그램 서버 관련 에이전트 설치 관리자와 함께 제공된 에이전트 프로필에서 기본값은 응용 프로그램 서버(예: *WebLogic Agent*)를 반영합니다.
- 기본 에이전트 설치 관리자와 함께 제공된 에이전트 프로필에서 속성 값은 *AgentName* 이고 이 줄은 주석 처리가 제거되어 있습니다.

introscope.agent.agentNameSystemPropertyKey

Java 시스템 속성의 값을 사용하여 에이전트 이름을 지정하려면 이 속성을 사용하십시오.

기본값

지정되지 않았습니다.

예

```
introscope.agent.agentNameSystemPropertyKey
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.disableLogFileAutoNaming

자동 이름 지정 옵션을 사용할 때 에이전트 로그 파일의 자동 이름 지정을 사용하지 않도록 설정할지 여부를 지정합니다.

이 속성을 **true** 로 설정하면 에이전트 이름 또는 타임스탬프를 포함하여 에이전트, AutoProbe 및 LeakHunter 에 대한 로그 파일의 자동 이름 지정이 사용하지 않도록 설정됩니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.disableLogFileAutoNaming=false
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 로그 파일 자동 이름 지정은 Java 시스템 속성 또는 응용 프로그램 서버 사용자 지정 서비스를 사용하여 에이전트 이름을 확인할 수 있는 경우에만 적용됩니다.

introscope.agent.clonedAgent

동일한 컴퓨터에서 응용 프로그램의 동일한 복사본을 실행할 수 있도록 합니다. 동일한 컴퓨터에서 응용 프로그램의 동일한 복사본이 실행 중인 경우에는 이 속성을 true 로 설정하십시오.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.clonedAgent=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.customProcessName

Introscope Enterprise Manager 및 Workstation 에 표시되어야 하는 프로세스 이름을 지정하십시오.

기본값

응용 프로그램 서버별로 차이가 있습니다.

예

```
introscope.agent.customProcessName=CustomProcessName
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 응용 프로그램 서버 관련 에이전트 설치 관리자와 함께 제공된 에이전트 프로필에서 기본값은 응용 프로그램 서버(예: "WebLogic")를 반영합니다.
- 기본 에이전트 설치 관리자와 함께 제공된 에이전트 프로필에서 이 속성은 주석 처리가 제거되어 있습니다.

introscope.agent.defaultProcessName

사용자 지정 프로세스 이름이 제공되지 않고 에이전트가 기본 응용 프로그램 클래스의 이름을 파악할 수 없는 경우, 이 값이 프로세스 이름에 사용됩니다.

기본값

UnknownProcess

예

```
introscope.agent.defaultProcessName=UnknownProcess
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.display.hostName.as.fqdn

이 속성은 에이전트 이름이 FQDN(정규화된 도메인 이름)으로 표시되는지 여부를 지정합니다. FQDN 을 사용하도록 설정하려면 이 속성 값을 'true'로 설정하십시오. 기본적으로 에이전트는 호스트 이름을 표시합니다.

참고: Catalyst 가 통합된 경우 이 속성을 'true'로 설정하십시오.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.display.hostName.as.fqdn=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

에이전트 기록(비즈니스 기록)

에이전트에서 비즈니스 트랜잭션 기록을 처리하는 방식을 제어할 수 있습니다.

참고: 에이전트 비즈니스 기록에 대한 자세한 내용은 *CA APM 트랜잭션 정의 안내서*를 참조하십시오.

introscope.agent.bizRecording.enabled

에이전트의 비즈니스 트랜잭션 기록을 사용하거나 사용하지 않도록 설정합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.bizRecording.enabled=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

에이전트의 비즈니스 트랜잭션 기록을 보다 세부적으로 구성하려면 응용 프로그램 심사 맵에 대한 추가 속성을 참조하십시오.

추가 정보:

응용 프로그램 심사 맵 (페이지 276)
응용 프로그램 심사 맵 비즈니스 트랜잭션 POST 매개 변수 (페이지 281)
응용 프로그램 심사 맵에서 관리되는 소켓 구성 (페이지 283)

에이전트 스레드 우선 순위

다음 속성은 에이전트 스레드의 우선 순위를 제어합니다.

- introscope.agent.thread.all.priority (see page 273)

introscope.agent.thread.all.priority

에이전트 스레드의 우선 순위를 제어합니다.

속성 설정

이 속성은 1(낮음) - 10(높음) 사이에서 설정할 수 있습니다.

기본값

주석 처리됨. 5

예

```
introscope.agent.thread.all.priority=5
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

에이전트와 Enterprise Manager 의 연결

에이전트가 Enterprise Manager 에 연결하는 방법을 제어할 수 있습니다.

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

에이전트가 기본적으로 연결되는 Enterprise Manager 를 실행하는 컴퓨터의 호스트 이름을 지정합니다.

기본값

localhost

예

```
introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=localhost
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

Enterprise Manager 를 호스팅하는 컴퓨터에서 에이전트로부터의 연결을 수신 대기하는 포트 번호를 지정합니다.

기본값

기본 포트는 구성하려는 통신 채널의 유형에 따라 달라집니다. 에이전트와 Enterprise Manager 간의 직접 통신에 사용되는 기본 포트는 5001 입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- HTTPS(HTTP over SSL)를 사용하여 Enterprise Manager 에 연결하려는 경우 기본 포트는 8444 입니다. SSL 을 사용하여 Enterprise Manager 에 연결하려는 경우 기본 포트는 5443 입니다. 그러나 이러한 기본 설정은 기본적으로 주석으로 처리되어 있습니다.

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

에이전트에서 Enterprise Manager 에 연결할 때 사용할 기본 클라이언트 소켓 팩터리를 지정합니다.

기본값

기본 소켓 팩터리는 구성하려는 통신 채널의 유형에 따라 달라집니다. 에이전트와 Enterprise Manager 간의 직접 통신에 사용되는 기본 소켓 팩터리는 다음과 같습니다.

com.wily.isengard.postofficehub.link.net.DefaultSocketFactory

예

```
introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.DefaultSocketFactory
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT

에이전트가 기본적으로 연결되는 Enterprise Manager 를 실행하는 컴퓨터의 IP 주소를 지정합니다.

기본값

이 속성은 기본적으로 정의되어 있지 않습니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT=<address>
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT

에이전트가 기본적으로 연결되는 Enterprise Manager 를 실행하는 컴퓨터의 로컬 포트를 지정합니다.

기본값

이 속성은 기본적으로 정의되어 있지 않습니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT=CA Portal
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

응용 프로그램 심사 맵

응용 프로그램 심사 맵 데이터를 구성할 수 있습니다.

참고: 응용 프로그램 심사 맵을 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

introscope.agent.appmap.enabled

응용 프로그램 심사 맵에 대한 모니터링된 코드의 추적을 활성화 또는 비활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.appmap.enabled=true
```

참고

기본적으로 사용하도록 설정됩니다.

introscope.agent.appmap.metrics.enabled

응용 프로그램 심사 맵 노드에 대한 메트릭의 추적을 활성화 또는 비활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.appmap.metrics.enabled=false
```

참고

이 속성은 기본적으로 주석으로 처리되어 있습니다.

introscope.agent.appmap.queue.size

응용 프로그램 심사 맵에 대한 버퍼 크기를 설정합니다.

속성 설정

양의 정수

기본값

1000

예

```
introscope.agent.appmap.queue.size=1000
```

참고

- 값은 양의 정수여야 합니다.
- 값을 0 으로 설정하면 버퍼에 제한이 없습니다.
- 이 속성은 기본적으로 주석으로 처리되어 있습니다.

introscope.agent.appmap.queue.period

응용 프로그램 심사 맵 데이터를 Enterprise Manager 로 보내는 빈도(밀리초)를 설정합니다.

속성 설정

양의 정수

기본값

1000

예

```
introscope.agent.appmap.queue.period=1000
```

참고

- 반드시 양의 정수여야 합니다.
- 값이 0 으로 설정되면 기본값이 사용됩니다.
- 이 속성은 기본적으로 주석으로 처리되어 있습니다.

introscope.agent.appmap.intermediateNodes.enabled

응용 프로그램의 프런트엔드 노드와 백엔드 노드 사이에 중간 노드를 포함하는 기능을 사용하거나 사용하지 않도록 설정합니다.

속성 설정

True 또는 False

기본값

False

예

```
#introscope.agent.appmap.intermediateNodes.enabled=true
```

참고

- 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.
- 이 속성을 true 로 설정하면 에이전트 성능이 저하될 수 있습니다.
- 이 속성은 기본적으로 주석으로 처리되어 있습니다.

응용 프로그램 심사 맵 및 Catalyst 통합

Catalyst 통합에 맞게 응용 프로그램 심사 맵을 구성할 수 있습니다.

참고: 응용 프로그램 심사 맵을 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

정보 전송 기능 구성

이 속성은 Catalyst 와의 통합을 위해 에이전트에서 추가 정보를 보내는 기능을 사용하거나 사용하지 않도록 설정합니다.

다음 단계를 따르십시오.

1. 기본 IntroscopeAgent.profile 파일을 텍스트 편집기에서 엽니다.

introscope.agent.appmap.catalystIntegration.enabled=<false|true> 행을 찾아 값을 다음과 같이 설정합니다.

true

Catalyst 와의 통합을 위해 에이전트에서 추가 정보를 보내는 기능을 사용하도록 설정합니다.

false

이 구성을 사용하지 않도록 설정합니다.

다음 예에서는 형식을 보여 줍니다.

```
introscope.agent.appmap.catalystIntegration.enabled=false
```

참고: 이 속성은 기본적으로 주석으로 처리되어 있습니다.

2. 파일을 저장하고 닫습니다.

에이전트가 이 구성을 사용하도록 설정됩니다.

사용 가능한 네트워크 목록 구성

introscope.agent.primary.net.interface.name 속성은 Catalyst 통합을 위해 에이전트에서 사용하는 호스트 컴퓨터의 기본 네트워크 인터페이스 이름을 지정합니다. 이 속성의 구성을 변경할 수 있으며 변경 사항은 자동으로 적용됩니다.

참고: 에이전트 로깅 수준이 DEBUG 로 설정된 경우 구성에 사용할 수 있는 네트워크 인터페이스 이름에 대한 정보가 로그 파일에 나타납니다. 또는 이 속성에 대해 기본 네트워크 인터페이스 이름을 결정하기 위해 네트워크 인터페이스 유틸리티를 사용할 수 있습니다.

다음 단계를 따르십시오.

1. 기본 IntroscopeAgent.profile 파일을 텍스트 편집기에서 엽니다.
2. introscope.agent.primary.net.interface.name=<false|true> 줄을 찾고 이름 값을 지정합니다.

   다음 예제는 이름 형식을 나타냅니다.

   ```
   introscope.agent.primary.net.interface.name=eth4
   ```

   참고: 기본 값은 정의되어 있지 않습니다. 이 속성이 설정되지 않은 경우 에이전트는 첫 번째 사용 가능한 네트워크 인터페이스를 기본 인터페이스로 지정합니다. 이 속성의 이름 값을 결정하기 위해 네트워크 인터페이스 유틸리티를 사용할 수 있습니다.

3. (선택 사항) 하위 인터페이스 번호(0 부터 시작)를 지정하여 여러 네트워크 주소를 사용할 수 있습니다.

   다음 예제는 하위 인터페이스 번호 형식을 나타냅니다.
   ```
   introscope.agent.primary.net.interface.name=eth4.1
   ```
4. 파일을 저장하고 닫습니다.

   프로필이 구성을 사용하도록 설정됩니다.

추가 정보:

네트워크 인터페이스 유틸리티 사용 (페이지 409)

응용 프로그램 심사 맵 비즈니스 트랜잭션 POST 매개 변수

POST 매개 변수와 일치시켜 보다 복잡한 모니터링을 수행할 수 있도록 Local Product Shorts 를 구성할 수 있습니다.

introscope.agent.bizdef.matchPost

이 속성은 POST 매개 변수와 일치되는 시기를 결정합니다.

속성 설정

이 속성의 유효한 설정은 *never*, *before* 또는 *after* 입니다.

- 에이전트 기능을 모두 사용하고 성능을 향상시키려는 경우에는 이 속성을 **never** 로 설정하십시오. 이 설정을 사용하면 응용 프로그램에서 URL, 쿠키 또는 헤더 매개 변수를 사용하여 모든 비즈니스 트랜잭션을 식별할 수 있지만, POST 매개 변수를 통해서만 식별되는 비즈니스 트랜잭션은 일치시킬 수 없습니다.
- 최대의 에이전트 성능을 얻으려면 이 속성을 **before** 로 설정하십시오. 이 설정을 사용하면 응용 프로그램에서 POST 매개 변수를 사용하여 일부 또는 모든 비즈니스 트랜잭션을 식별할 수 있지만, HTTP 양식 요청을 위해 서블릿 스트림에 직접 액세스할 수는 없습니다. 이 속성이 *before* 로 설정되어 있으면 새로 배포되는 응용 프로그램에서도 표준 API 를 따라야 합니다.

  중요! 이 속성을 *before* 로 설정하면 응용 프로그램에 악영향을 줄 수도 있습니다. 구현 전에 CA Technologies 담당자와 함께 이 속성 설정을 검토하십시오.

- 비즈니스 트랜잭션을 POST 매개 변수와 안전하게 일치시키되 에이전트 기능을 제한하려면 이 속성을 **after** 로 설정하십시오. 이 속성이 *after* 로 설정되어 있으면 에이전트는 프로세스에서 POST 매개 변수로 식별된 비즈니스 트랜잭션을 매핑하거나 이러한 비즈니스 트랜잭션에 대한 전체 메트릭 집합을 생성할 수 없습니다. 또한 이 설정은 다른 옵션에 비해 CPU 시간을 좀 더 소비하지만, POST 매개 변수 기능이 필요한 경우에는 가장 안전한 설정으로 간주됩니다. 이 설정을 사용하면 응용 프로그램에서 POST 매개 변수를 사용하여 일부 또는 모든 비즈니스 트랜잭션을 식별할 수 있지만, 서블릿 스트림에 대한 직접 액세스를 항상 금지할 수는 없습니다.

예

```
introscope.agent.bizdef.matchPost=after
```

참고

- *never* - POST 매개 변수를 일치시키지 않습니다. 이는 가장 빠른 옵션이지만 비즈니스 트랜잭션 구성 요소 일치가 부정확하게 될 수 있습니다.
- *before* - 서블릿이 실행되기 전에 POST 매개 변수를 일치시킵니다.
- *after* - 서블릿이 실행된 후에 POST 매개 변수 패턴을 일치시킵니다. 크로스 프로세스 매핑과 일부 메트릭은 사용할 수 없습니다. 이 매개 변수의 기본 설정입니다.

알려진 제한 사항

에이전트 기록을 사용하여 정의된 메트릭은 Investigator 의 응용 프로그램 심사 맵에 표시됩니다. 에이전트 기록을 구성할 경우 정규식 사용과 관련하여 몇 가지 알려진 제한 사항이 있습니다. 대부분의 제한 사항은 POST 매개 변수와 관련이 있습니다.

알려진 제한 사항은 다음과 같습니다.

- POST 매개 변수 값에는 행 마침 표시(.)를 사용할 수 없습니다.
- POST 매개 변수 정의가 비즈니스 트랜잭션 정의에 종속된 경우 비즈니스 트랜잭션 구성 요소에 세 가지 메트릭만 제공됩니다. 해당 메트릭은 다음과 같습니다.
  - 평균 응답 시간
  - 간격당 응답 수
  - 간격당 오류 수

- POST 매개 변수 정의가 비즈니스 트랜잭션 정의에 종속된 경우 트랜잭션 추적 구성 요소의 비즈니스 구성 요소 이름은 비즈니스 서비스, 비즈니스 트랜잭션 및 비즈니스 트랜잭션 구성 요소의 특정 이름이 아니라 일반 이름입니다. 이 내용은 일치하지 않는 POST 매개 변수 정의에 종속된 비즈니스 트랜잭션 정의에도 적용됩니다.
- JBoss 및 Tomcat 의 일부 버전에서는 헤더 키를 소문자 값으로 저장하여 *caseSensitiveName* 특성이 *HEADER_TYPE* 에 대해 올바로 작동하지 않게 될 수 있습니다.

참고: 에이전트 기록에 대한 자세한 내용은 *CA APM 트랜잭션 정의 안내서*를 참조하십시오.

응용 프로그램 심사 맵에서 관리되는 소켓 구성

다음 속성을 사용하여 응용 프로그램 심사 맵에 소켓 메트릭을 표시하거나 표시하지 않도록 설정할 수 있습니다.

- introscope.agent.sockets.managed.reportToAppmap (see page 284)
- introscope.agent.sockets.managed.reportClassAppEdge (see page 284)
- introscope.agent.sockets.managed.reportMethodAppEdge (see page 285)
- introscope.agent.sockets.managed.reportClassBTEdge (see page 285)
- introscope.agent.sockets.managed.reportMethodBTEdge (see page 286)

응용 프로그램 심사 맵을 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

introscope.agent.sockets.managed.reportToAppmap

응용 프로그램 심사 맵에 관리되는 소켓이 표시되도록 합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.sockets.managed.reportToAppmap=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.sockets.managed.reportClassAppEdge

관리되는 소켓이 클래스 수준 응용 프로그램 가장자리를 응용 프로그램 심사 맵에 보고하도록 합니다.

속성 설정

True 또는 False

기본값

False

예

introscope.agent.sockets.managed.reportClassAppEdge=false

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.sockets.managed.reportMethodAppEdge

관리되는 소켓이 메서드 수준 응용 프로그램 가장자리를 응용 프로그램 심사 맵에 보고하도록 합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.sockets.managed.reportMethodAppEdge=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.sockets.managed.reportClassBTEdge

관리되는 소켓이 응용 프로그램 심사 맵에 클래스 수준 비즈니스 트랜잭션 가장자리를 보고하도록 합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.sockets.managed.reportClassBTEdge=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.sockets.managed.reportMethodBTEdge

관리되는 소켓이 응용 프로그램 심사 맵에 메서드 수준 비즈니스 트랜잭션 가장자리를 보고하도록 합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.sockets.managed.reportMethodBTEdge=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

AutoProbe

다음 속성은 AutoProbe 를 구성합니다.

- introscope.autoprobe.directivesFile (see page 287)
- introscope.autoprobe.enable (see page 287)

introscope.autoprobe.directivesFile

AutoProbe 용 ProbeBuilder 지시문 파일을 지정합니다.

기본값

설치 관리자에 따라 다릅니다.

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 이 속성에 디렉터리가 하나 이상 포함되어 있고 동적 계측이 사용하도록 설정되어 있는 경우, 응용 프로그램을 다시 시작하지 않아도 에이전트가 지정된 디렉터리에서 지시문 파일을 로드합니다.

introscope.autoprobe.enable

이 속성 설정은 프로브를 바이트 코드에 자동으로 삽입하는 기능을 사용하도록 설정하거나 사용하지 않도록 설정합니다.

속성 설정

true 또는 false

기본값

true

예

```
introscope.autoprobe.enable=true
```

참고: 이 속성을 false 로 설정하면 프로브를 응용 프로그램 바이트 코드에 자동으로 삽입하는 기능만 비활성화되고 에이전트 또는 에이전트 보고 기능은 비활성화되지 않습니다. 변경 내용을 적용하려면 JVM 을 다시 시작해야 합니다.

introscope.autoprobe.logfile

Introscope AutoProbe 는 항상 변경한 사항을 기록하려고 합니다. 로그 파일의 위치를 기본값 이외의 다른 위치로 이동하려면 이 속성을 설정하십시오.

속성 설정

절대 파일 경로 또는 절대 경로가 아닌 경로. 절대 이름이 아닌 이름은 이 속성 파일의 위치를 기준으로 확인됩니다.

기본값

../../logs/AutoProbe.log

예

```
introscope.autoprobe.logfile=../../logs/AutoProbe.log
```

로깅을 사용하지 않도록 설정하려면 로그 파일에서 다음과 같이 주석 처리하십시오.

```
introscope.autoprobe.logfile=logs/AutoProbe.log
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

Bootstrap Classes Instrumentation Manager

다음 속성은 Bootstrap Classes Instrumentation Manager 를 구성합니다.

- introscope.bootstrapClassesManager.enabled (see page 289)
- introscope.bootstrapClassesManager.waitAtStartup (see page 289)

Bootstrap Classes Instrumentation Manager 는 에이전트 부트스트랩 이후에 일련의 클래스를 계측하여 Java NIO 및 SSL(Secure Sockets Layer)을 위한 간편한 추적 프로그램 구현 및 개선된 에이전트 성능을 가능하게 합니다. *IntroscopeAgent.profile* 에서 주석 처리를 제거하여 이 속성을 비활성화할 수 있습니다.

introscope.bootstrapClassesManager.enabled

Bootstrap Manager 를 활성화 또는 비활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.bootstrapClassesManager.enabled=true
```

참고

- 이 속성은 Java 1.5 이상을 실행하는 JVM 에서만 동작합니다.
- false 로 설정되면 시스템 클래스가 계측되지 않습니다.
- 이 속성이 설정되지 않으면 기본값은 false 입니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.bootstrapClassesManager.waitAtStartup

시작 후 에이전트가 부트스트랩 클래스를 계측할 때까지 기다리는 시간(초)을 설정합니다.

속성 설정

시간(초)

기본값

- HP-UX, Interstage, WebLogic, WebSphere Application Server 에서 사용하는 경우 240 초입니다.
- JBoss, Oracle, Sun, Tomcat 에서 사용하는 경우 5 초입니다.

예

```
introscope.bootstrapClassesManager.waitAtStartup=5
```

참고

- 이 속성은 Java 1.5 이상을 실행하는 JVM 에서만 동작합니다.
- 이 속성이 활성화되면 건너뛰도록 지정된 클래스에 대해 이 속성이 우선합니다. 건너뛴 클래스가 계속 계측되는 경우 CA Technologies 담당자나 CA Support 에 문의하십시오.

CA CEM 에이전트 프로필 속성

CA CEM 관련 IntroscopeAgent.profile 속성을 구성할 수 있습니다. CA Introscope 에이전트 프로필 파일은 *<Agent_Home>*\wily 디렉터리에 있습니다.

모든 CA CEM 관련 속성은 CA CEM 및 CA Introscope 와의290 통합이 작동하는 데 필요한 옵션으로 미리 구성되어 있습니다.

introscope.autoprobe.directivesFile

directivesFile 속성 구성을 통해 ServletHeaderDecorator / HTTPHeaderDecorator 및 CEMTracer 를 사용하도록 설정해야 합니다.

지시문 파일 속성은 AutoProbe 용 지시문 파일(PBD) 또는 지시문 목록(PBL)을 찾을 위치를 지정합니다.

AutoProbe 는 지시문을 사용하여 응용 프로그램을 사용하도록 설정하고 에이전트가 Enterprise Manager 에 보고할 메트릭을 결정합니다.

설정

설치된 에이전트 응용 프로그램 서버에 따라 다르며, 형식은 *<appserver>*-full.pbl 또는 *<appserver>*-typical.pbl 입니다.

기본값

```
default-typical.pbl
```

예

```
introscope.autoprobe.directivesFile=weblogic-typical.pbl
```

참고

단순히 이 속성 목록의 끝에 "ServletHeaderDecorator.pbd" 또는 "httpheaderdecorator.pbd"를 추가해도 되지만 다음 단계를 따르는 것이 더 좋습니다.

1. 속성에 지정된 PBL 파일(위 예의 경우 weblogic-typical.pbl)을 찾습니다.
2. 텍스트 편집기에서 PBL 파일을 엽니다.
3. Java Agent 의 경우 ServletHeaderDecorator.pbd 행의 주석 처리를 제거하여 이 행을 활성화합니다.
4. .NET 에이전트의 경우 httpheaderdecorator.pbd 행의 주석 처리를 제거하여 이 행을 활성화합니다.
5. PBL 파일의 변경 내용을 저장합니다.

introscope.agent.remoteagentconfiguration.allowedFiles

이 속성은 임의의 컴퓨터에서 에이전트 디렉터리로 원격 복사가 허용되는 파일을 식별합니다.

Enterprise Manager 는 이 속성의 파일 이름을 사용하여 에이전트로 보낼 유효한 CA CEM 도메인 구성 파일을 식별합니다. 도메인 구성 파일은 CA CEM 비즈니스 서비스와 트랜잭션 정의를 포함하고 있습니다.

설정

올바른 파일 이름을 사용하십시오.

기본값

domainconfig.xml

예

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

참고

이 속성은 또한 Introscope CLW(Command-Line Workstation) 구성 파일 보내기 명령에도 적용됩니다.

자세한 내용은 Command-Line Workstation 사용을 참조하십시오.

이 속성은 CA CEM 릴리스에 적용됩니다.

introscope.agent.remoteagentconfiguration.enabled

이 부울 값이 true 로 설정되면 다른 컴퓨터에서 에이전트로 원격 파일 복사를 허용할 수 있습니다.

Enterprise Manager 가 CA CEM 도메인 구성 파일을 에이전트에 보내려면 이 속성을 true 로 설정해야 합니다. 도메인 구성 파일은 CA CEM 비즈니스 서비스와 트랜잭션 정의를 포함하고 있습니다.

설정

true 또는 false

기본값

- Java Agent 의 경우 true
- .NET 에이전트의 경우 false

예

```
introscope.agent.remoteagentconfiguration.enabled=true
```

참고

원격 사용자는 또한 CA Introscope 명령줄 Workstation(CLW) 구성 파일 보내기 명령을 사용하여 introscope.agent.remoteagentconfiguration.allowedFiles 속성에 지정된 파일을 에이전트 디렉터리에 복사할 수 있습니다.

이 속성은 CA CEM 4.0 및 4.1 릴리스에 대해 유효합니다. 이 속성은 또한 "Introscope 설정" 페이지에서 "CEMTracer 4.0 / 4.1 지원" 옵션이 선택된 경우 CA CEM 4.2 / 4.5 에서만 유효합니다.

"CEMTracer 4.0 / 4.1 지원" 옵션을 사용하면 시간이 지남에 따라 4.0 또는 4.1 에서 4.2 / 4.5 로 에이전트 마이그레이션을 스태거할 수 있습니다. 이 옵션은 필요한 경우에만 사용하십시오.

호환되지 않는 에이전트, 즉 지원되지 않는 .NET 에이전트, EPA 에이전트 또는 기타 Java 가 아닌 에이전트의 경우에는 introscope.agent.remoteagentconfiguration.enabled 속성을 false 로 설정하십시오.

introscope.agent.decorator.enabled

이 부울 값이 true 로 설정되어 있으면 에이전트가 HTTP 응답 헤더에 추가적인 성능 모니터링 정보를 추가하도록 구성됩니다. ServletHeaderDecorator / HTTPHeaderDecorator 는 각 트랜잭션에 GUID 를 연결하고 이 GUID 를 HTTP 헤더 x-apm-info 에 삽입합니다.

그러면 CA CEM 과 CA Introscope 간의 트랜잭션 상관 관계가 설정됩니다.

설정

true 또는 false

기본값

- Java Agent 의 경우 false
- .NET 에이전트의 경우 true

예

```
introscope.agent.decorator.enabled=false
```

introscope.agent.decorator.security

이 속성에 따라 CA CEM 으로 전송되는 데코레이트된 HTTP 응답 헤더의 형식이 결정됩니다.

설정

- clear - 일반 텍스트 인코딩
- encrypted - 헤더 데이터가 암호화됨

기본값

clear

예

```
introscope.agent.decorator.security=clear
```

참고

기본 설정 clear 는 초기 테스트에 적절합니다. 그러나 이 설정은 방화벽 바깥으로 노출하고 싶지 않은 트랜잭션 헤더의 정보를 노출할 수 있습니다. 보다 안전한 프로덕션 환경을 위해 이 속성을 encrypted 로 설정하십시오.

이 속성을 encrypted 로 설정하려면 지원되는 JVM 을 사용하십시오.

참고: JVM 지원 정보는 *Compatibility Guide*(호환성 안내서)를 참조하십시오.

introscope.agent.cemtracer.domainconfigfile

CA CEM 비즈니스 서비스 및 트랜잭션 계층을 지정하는 CA CEM 도메인 구성 파일의 이름입니다. CEMTracer 는 설치 디렉터리에서 이 이름의 파일을 찾습니다.

CA CEM 관리자가 CA CEM 에서 "모든 모니터 동기화"를 클릭할 때마다 도메인 구성 파일이 Enterprise Manager 로 푸시된 다음 연결된 각 에이전트로 푸시됩니다.

CA CEM 릴리스 관련 정보는 아래의 **참고**를 보십시오.

설정

임의의 유효한 파일 이름을 사용할 수 있습니다.

기본값

domainconfig.xml

예

introscope.agent.cemtracer.domainconfigfile=domainconfig.xml

참고

이 속성은 CA CEM 4.0 및 4.1 릴리스에 대해 유효합니다. 이 속성은 또한 "Introscope 설정" 페이지에서 "CEMTracer 4.0 / 4.1 지원" 옵션이 선택된 경우에만 CA CEM 4.2 / 4.5 에서 유효합니다.

"CEMTracer 4.0 / 4.1 지원" 옵션을 사용하면 시간이 지남에 따라 4.0 또는 4.1 에서 4.2 / 4.5 로 에이전트 마이그레이션을 스태거할 수 있습니다. 이 옵션은 필요한 경우에만 사용하십시오.

- 에이전트가 Enterprise Manager 에 연결되지 않은 경우 도메인 구성 파일이 전송되지 않습니다.
- 에이전트 디렉터리가 읽기 전용인 경우 도메인 구성 파일을 기록할 수 없습니다.
- CEMTracer 4.0 / 4.1 이 에이전트에서 활성화되지 않은 경우 전달된 이후에는 도메인 구성 파일에서 어떤 동작도 수행되지 않습니다.

introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes

에이전트가 도메인 구성 파일을 다시 로드하는 빈도(분)입니다. (4.0 / 4.1 에이전트는 Enterprise Manager 가 전달할 때마다 도메인 구성 파일을 자동으로 다시 로드하지 않습니다. 변경되지 않은 경우 에이전트는 이 파일을 다시 로드하지 않습니다.)

CA CEM 릴리스 관련 정보는 아래의 **참고**를 보십시오.

설정

숫자 값

기본값

1

예

```
introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes=1
```

참고

이 속성은 CA CEM 4.0 및 4.1 릴리스에 대해 유효합니다. 이 속성은 또한 "Introscope 설정" 페이지에서 "CEMTracer 4.0 / 4.1 지원" 옵션이 선택된 경우에만 CA CEM 4.2 / 4.5 에서 유효합니다.

"CEMTracer 4.0 / 4.1 지원" 옵션을 사용하면 시간이 지남에 따라 4.0 또는 4.1 에서 4.2 로 에이전트 마이그레이션을 스태거할 수 있습니다. 이 옵션은 필요한 경우에만 사용하십시오.

introscope.agent.distribution.statistics.components.pattern

BlamePointTracer 로부터 응답 시간 분포 정보를 수집하려면 이 속성의 주석 처리를 제거하고 속성을 편집하십시오. 이 응답 시간 정보는 평균 응답 시간 메트릭을 생성하는 데 사용할 수 있습니다.

추가 정보:

분포 통계 메트릭을 수집하도록 에이전트를 구성하는 방법 (페이지 83)

세션 ID 수집 구성

introscope.agent.transactiontracer.parameter.capture.sessionid 속성은 트랜잭션 추적 프로그램 데이터에서 세션 ID 수집을 사용하거나 사용하지 않도록 설정합니다. 기본적으로 이 속성은 트랜잭션 추적 프로그램 데이터에서 사용하도록 설정되고 기록됩니다. 이 속성을 사용하지 않을 경우 필터에 데이터를 사용할 수 없습니다.

다음 단계를 따르십시오.

1. IntroscopeAgent.profile 파일을 텍스트 편집기에서 엽니다.

다음 행을 찾습니다.

```
# Uncomment the following property to disable sessionid capture in
TransactionTracer data.
# By default, it is enabled and recorded in the TT Data.

# introscope.agent.transactiontracer.parameter.capture.sessionid=true
```

2. 지시에 따라 행을 주석으로 처리하거나 주석 처리를 제거하여 속성을 사용하거나 사용하지 않도록 설정합니다.

```
# introscope.agent.transactiontracer.parameter.capture.sessionid=true
```

3. 파일을 저장하고 닫은 다음 에이전트를 다시 시작합니다.

   에이전트 구성이 세션 ID 수집에 대해 지정된 값을 사용하도록 설정됩니다.

ChangeDetector 구성

로컬 에이전트가 ChangeDetector 와 함께 작동하는 방식을 제어할 수 있습니다.

참고: ChangeDetector 사용에 대한 자세한 내용은 *CA APM ChangeDetector 사용자 안내서*를 참조하십시오.

introscope.changeDetector.enable

ChangeDetector 사용 여부를 지정합니다. ChangeDetector 가 사용되도록 설정하려면 이 속성을 true 로 설정합니다. 주석 처리되어 기본적으로 false 로 설정되어 있습니다. ChangeDetector 가 사용되도록 설정한 경우 ChangeDetector 관련 추가 속성도 설정해야 합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.changeDetector.enable=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.changeDetector.agentID

로컬 에이전트를 식별하기 위해 ChangeDetector 가 사용하는 텍스트 문자열을 지정합니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다. ChangeDetector 를 활성화하는 경우 이 속성의 주석 처리를 제거하고 적절한 값으로 설정해야 합니다.

기본값

기본값은 *SampleApplicationName* 입니다.

예

```
introscope.changeDetector.agentID=SampleApplicationName
```

introscope.changeDetector.rootDir

ChangeDetector 파일의 루트 디렉터리를 지정합니다. 루트 디렉터리는 ChangeDetector 가 해당 로컬 캐시 파일을 생성하는 폴더입니다.

속성 설정

ChangeDetector 파일의 루트 디렉터리에 대한 전체 경로를 나타내는 텍스트 문자열입니다.

기본값

기본 경로는 *c://sw//AppServer//wily//change_detector* 입니다.

예

```
introscope.changeDetector.rootDir=c://sw//AppServer//wily//change_detector
```

참고

예제에서처럼 백슬래시를 사용하여 백슬래시 문자를 이스케이프하십시오.

introscope.changeDetector.isengardStartupWaitTimeInSec

에이전트가 시작된 후 ChangeDetector 가 Enterprise Manager 연결을 시도하기 전에 대기하는 기간을 초 단위로 지정합니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다.

기본값

기본값은 15 초입니다.

예

```
introscope.changeDetector.isengardStartupWaitTimeInSec=15
```

introscope.changeDetector.waitTimeBetweenReconnectInSec

Enterprise Manager 연결을 다시 시도하기 전에 ChangeDetector 가 대기하는 기간을 초 단위로 지정합니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다.

기본값

기본값은 10 초입니다.

예

```
introscope.changeDetector.waitTimeBetweenReconnectInSec=10
```

introscope.changeDetector.profile

ChangeDetector 데이터 원본 구성 파일의 절대 또는 상대 경로를 지정합니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다.

기본값

기본값은 ChangeDetector-config.xml 입니다.

예

```
introscope.changeDetector.profile=CDConfig\\ChangeDetector-config.xml
```

참고

예제에서처럼 백슬래시를 사용하여 백슬래시 문자를 이스케이프하십시오.

introscope.changeDetector.profileDir

데이터 원본 구성 파일이 들어 있는 디렉터리에 대한 절대 또는 상대 경로를 지정합니다. 이 속성이 설정된 경우 *introscope.changeDetector.profile* 속성으로 지정된 모든 파일과 함께 이 디렉터리의 모든 데이터 원본 구성 파일이 사용됩니다. 이 속성은 기본적으로 주석으로 처리되어 있습니다.

기본값

기본값은 changeDetector_profiles 입니다.

예

```
introscope.changeDetector.profileDir=c:\\CDconfig\\changeDetector_profiles
```

참고

백슬래시를 사용하여 백슬래시 문자를 이스케이프 처리하십시오.

introscope.changeDetector.compressEntries.enable

ChangeDetector 데이터 버퍼에 압축을 사용할 수 있는지 여부를 지정합니다. 시작 시 메모리 소비 문제가 발생할 경우 성능 향상을 위해 이 속성을 true 로 설정할 수 있습니다.

속성 설정

True 또는 False

기본값

에이전트 프로필에서 이 속성이 설정되어 있지 않거나 주석으로 처리된 경우 기본값은 false 입니다.

예

```
introscope.changeDetector.compressEntries.enable=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.changeDetector.compressEntries.batchSize

이 속성은 위의 introscope.changeDetector.compressEntries.enable 에 설정된 압축 작업의 배치 크기를 정의합니다.

기본값

1000

예

```
introscope.changeDetector.compressEntries.batchSize=1000
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

WebLogic Server 에서 크로스 프로세스 추적

다음 속성은 WebLogic Server 에서 크로스 프로세스 추적을 구성합니다.

- introscope.agent.weblogic.crossjvm (see page 302)

introscope.agent.weblogic.crossjvm

속성 설정

True 또는 False

기본값

주석 처리됨. True

예

```
introscope.agent.weblogic.crossjvm=true
```

프로세스 간 트랜잭션 추적

테일 필터로 인해 발생하는 다운스트림 추적을 자동으로 수집하도록 설정하려면 다음 속성의 주석 처리를 제거하십시오.

- introscope.agent.transactiontracer.tailfilterPropagate.enable (see page 303)

이 속성을 사용하도록 설정한 상태에서 테일 필터가 포함된 트랜잭션 추적 세션을 오랫동안 실행하면 원치 않는 추적이 Enterprise Manager 에 필요 이상으로 전송됩니다.

introscope.agent.transactiontracer.tailfilterPropagate.enable

테일 필터가 있을 경우 다운스트림 에이전트에서 자동 추적 수집이 트리거되는지 여부를 제어합니다. 이 속성은 헤드 필터 전달로 인한 자동 다운스트림 추적의 수집에는 영향을 주지 않습니다.

속성 설정

True 또는 False

기본값

False(주석 처리됨)

예

```
introscope.agent.transactiontracer.tailfilterPropagate.enable=false
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

동적 계측

사용자 지정 PBD 를 작성하거나 응용 프로그램 서버를 다시 시작하거나 에이전트를 다시 시작하지 않고도 클래스 및 메서드를 동적으로 계측할 수 있도록 설정할 수 있습니다.

참고: Introscope Workstation 에서 트랜잭션 추적 및 동적 계측을 사용하는 방법에 대한 자세한 내용은 *CA APM Workstation 사용자 안내서*를 참조하십시오.

introscope.autoprobe.dynamicinstrument.enabled

JDK 1.5 에서 실행되며 AutoProbe 를 사용하는 에이전트에 해당

이 속성은 에이전트에 동적 ProbeBuilding 을 사용하도록 설정합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.autoprobe.dynamicinstrument.enabled=false
```

추가 정보:

동적 ProbeBuilding (페이지 89)

autoprobe.dynamicinstrument.pollIntervalMinutes

JDK 1.5 에서 실행되며 AutoProbe 및 동적 ProbeBuilding 을 사용하는 에이전트에 해당

이 속성에 따라 에이전트가 새 PBD 및 변경된 PBD 를 폴링하는 빈도가 결정됩니다.

기본값

1

예

```
autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs

일부 클래스 로더 구현에서는 대량의 클래스 파일을 반환하는 것으로 확인되었습니다. 이것은 메모리 오류를 방지하기 위함입니다.

기본값

1

예

```
introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo

한 번에 너무 많은 클래스를 다시 정의하면 CPU 가 무리하게 사용될 수 있습니다. 이 속성을 사용하면 PBD 의 변경 사항으로 인해 많은 수의 클래스를 다시 정의해야 하는 경우 프로세스가 안정적인 속도로 일괄 처리됩니다.

기본값

10

예

```
introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10
```

introscope.agent.remoteagentdynamicinstrumentation.enabled

동적 계측의 원격 관리를 사용하거나 사용하지 않도록 설정합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.remoteagentdynamicinstrumentation.enabled=true
```

참고

- 이 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 동적 계측은 CPU 를 많이 사용하는 작업입니다. 계측되는 클래스를 최소화하는 구성을 사용하십시오.

introscope.autoprobe.dynamicinstrument.pollIntervalMinutes

PBD 변경 사항을 폴링할 폴링 간격(분)을 정의합니다.

기본값

1

예

```
introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

ErrorDetector

에이전트가 ErrorDetector 와 상호 작용하는 방식을 제어할 수 있습니다.

introscope.agent.errorsnapshots.enable

에이전트가 심각한 오류에 대한 트랜잭션 세부 정보를 캡처할 수 있도록 합니다. Introscope ErrorDetector 는 에이전트와 함께 기본적으로 설치됩니다. 오류 스냅숏을 볼 수 있도록 하려면 이 속성을 true 로 설정해야 합니다.

속성 설정

True 또는 False

기본값

True

참고

이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.errorsnapshots.throttle

에이전트가 15 초 동안 보낼 수 있는 최대 오류 스냅숏 수를 지정합니다.

기본값

10

예

```
introscope.agent.errorsnapshots.throttle=10
```

참고

이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.errorsnapshots.ignore.<index>

오류 메시지 필터를 하나 이상 지정합니다. 속성 이름에 인덱스 식별자(예: .0, .1, .2 ...)를 추가하여 필터를 필요한 수만큼 지정할 수 있습니다. 와일드카드(*)를 사용하여 지정한 조건과 일치하는 오류 메시지가 무시되도록 할 수도 있습니다. 정의한 필터와 일치하는 오류에 대해서는 오류 스냅숏이 생성되지 않으며, Enterprise Manager 로 해당 오류 이벤트가 전송되지도 않습니다.

중요! 이 속성은 SOAP 오류 메시지를 필터링하는 데 사용할 수 없습니다.

기본값

다음과 같은 예제 정의가 주석 처리되어 제공됩니다.

예

```
introscope.agent.errorsnapshots.ignore.0=*com.company.HarmlessException*
introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code: 404*
```

참고

이 속성은 동적 속성입니다. 런타임 중에 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

확장

에이전트 확장의 위치를 구성할 수 있습니다.

introscope.agent.extensions.directory

에이전트가 로드할 모든 확장의 위치를 지정합니다. 디렉터리의 절대 또는 상대 경로를 지정할 수 있습니다. 절대 경로를 지정하지 않을 경우, 지정한 값은 *IntroscopeAgent.profiles* 파일의 위치를 기준으로 확인됩니다.

기본값

기본 위치는 *<Agent_Home>*/ext 디렉터리에 있는 *ext* 디렉터리입니다.

예

```
introscope.agent.extensions.directory=../ext
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.common.directory

에이전트 확장 관련 파일의 위치를 구성합니다.

기본값

기본 위치는 <Agent_Home>/common 디렉터리에 있는 common 폴더입니다.

예

```
introscope.agent.common.directory=../../common
```

GC 모니터

"GC Monitor"(GC 모니터) 노드 아래의 메트릭은 가비지 수집기 및 메모리 풀에 대한 정보를 보고합니다. 이러한 메트릭은 성능에 부정적인 영향을 주는 메모리 관련 문제를 감지하는 데 유용합니다. 에이전트 프로필에서 직접 이러한 메트릭의 수집을 사용하도록 설정해야 합니다.

introscope.agent.gcmonitor.enable

이 속성은 가비지 수집기 및 메모리 풀에 대한 메트릭을 사용하거나 사용하지 않도록 설정합니다.

속성 설정

True 또는 False

기본값

기본값은 true 입니다.

예

```
introscope.agent.gcmonitor.enable=true
```

참고

이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

Sun 또는 IBM JVM 을 모니터링하는 에이전트에 대해서만 "GC Monitor"(GC 모니터) 메트릭을 보고할 수 있습니다.

Java NIO

Java Agent 는 Java New I/O(Java NIO 또는 NIO) 기능을 지원합니다. Java NIO 는 최신 운영 체제의 낮은 수준 I/O 작업에 액세스할 수 있도록 설계된 API 의 모음입니다. Java NIO 메트릭은 계측된 응용 프로그램에서 Java NIO 를 사용하는 방식에 대한 정보를 캡처합니다.

참고: CA Introscope® Java NIO 메트릭 및 Java NIO 정보 메트릭 수집 기능은 Java 1.5 JVM 이상에서만 사용할 수 있습니다.

CA Introscope?Java Agent 는 NIO 채널에 대한 메트릭을 수집합니다.

특정 NIO 메트릭의 생성을 제한할 수 있습니다.

Java NIO 추적 프로그램 그룹은 기본적으로 사용하도록 설정되어 있습니다. 이러한 추적 프로그램 그룹을 해제하여 메트릭 생성을 추가적으로 제한할 수 있습니다.

추가 정보:

채널 (페이지 311)
Java NIO 메트릭 제한 (페이지 312)
기본 추적 프로그램 그룹 및 toggles 파일 (페이지 106)

채널

Java NIO 채널을 사용하면 NIO 버퍼뿐만 아니라 외부 시스템과도 대량 데이터 전송이 가능합니다. Java NIO 채널은 표준 Java I/O 내의 성능 및 확장성 문제를 해결하기 위해 특별하게 설계된 낮은 수준의 데이터 전송 메커니즘입니다.

채널은 버퍼와 외부 시스템 간에 바이트를 이동하기 위한 메커니즘을 제공합니다. Introscope 채널 메트릭의 특징은 채널을 통한 데이터 흐름 속도에 있습니다. 수집된 NIO 채널 메트릭은 표준 Java I/O 기술을 사용하여 파일 및 소켓 I/O 에 대해 현재 생성된 메트릭에 해당합니다. Workstation Investigator 에서는 다음 채널 유형에 대한 메트릭이 개별적으로 수집되어 표시됩니다.

- 데이터그램 채널
- 소켓 채널

NIODatagramTracing metrics(I/O 메트릭)

UDP 가 연결 없는 프로토콜이긴 하지만 Java Agent 가 데이터그램 메트릭을 수집하는 방법을 기술하기 위해 NIODatagramTracing 의 설명에 "연결"이란 표현이 사용되었습니다. Java Agent 는 '전송' 또는 '수신'된 각 원격 끝점 데이터그램에 대해 별도의 메트릭을 수집합니다. 각 '대상' 및 '원본' 끝점에서 관찰된 첫 번째 데이터그램의 방향에 따라 클라이언트 또는 서버로 분류됩니다.

예를 들어, 첫 번째 데이터그램이 ' 원본' 끝점인 경우 Java Agent 는 해당 끝점에서 주고받는 모든 이후 데이터그램을 *NIO|Channels|Datagrams|Server|Port {PORT}* 아래에 분류합니다. 여기서 *{PORT}*는 로컬 포트입니다.

첫 번째 데이터그램이 ‘ 대상’ 끝점인 경우 해당 끝점에서 주고받는 모든 이후 데이터그램은 *NIO|Channels|Datagrams|Client|{HOST}|Port {PORT}* 아래에 분류됩니다. 여기서 *{HOST}* 및 *{PORT}*는 원격 끝점입니다.

'수신' 메서드가 읽는 데이터그램의 경우를 제외하고, Java Agent 는 또한 클라이언트 "연결"에 대한 백엔드 메트릭을 생성합니다. DatagramChannel 연결 메서드를 사용하여 생성된 데이터그램 채널은 관찰된 첫 번째 데이터그램의 방향과 관계없이 클라이언트 연결로 간주됩니다.

참고: UDP 연결을 사용하여(연결 메서드 사용) 생성된 데이터그램 채널은 클라이언트 연결로 간주됩니다.

Java NIO 메트릭 제한

속성을 구성하여 Java NIO 계측이 작동하는 방식을 제어할 수 있습니다. 데이터그램 및 소켓 메트릭의 생성을 제한할 수도 있습니다. 이러한 속성은 NIOSocketTracing 및 NIODatagramTracing 추적 프로그램 그룹에서 생성된 세부 메트릭에만 적용됩니다.

참고: 추적 프로그램 그룹에 대한 자세한 내용은 기본 추적 프로그램 그룹 및 toggles 파일 (see page 106)을 참조하십시오.

introscope.agent.nio.datagram.client.hosts

메트릭 보고를 지정된 호스트가 있는 '클라이언트' UDP "연결"로 제한합니다.

속성 설정

쉼표로 구분된 호스트 목록입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.datagram.client.hosts=hostA,hostB
```

참고

- 목록이 비어 있는 경우 호스트 제한이 적용되지 않습니다.
- 호스트는 이름이나 IP 주소(IPv4 또는 IPv6 형식)의 텍스트 표현으로 지정할 수 있습니다.
- 잘못된 호스트 이름은 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.
- 중복 호스트 이름은 삭제됩니다. 여러 개의 호스트 이름이 단일 IP 에 매핑되는 경우에는 하나의 이름만 유지됩니다. 그러나 이 속성은 클라이언트 연결을 동의어 이름 집합의 임의 항목과 일치시킵니다.

introscope.agent.nio.datagram.client.ports

NIO 메트릭을 보고하는 포트를 나열합니다. 지정된 포트에 대한 '클라이언트' 데이터그램 메트릭만 생성됩니다.

속성 설정

쉼표로 구분된 *포트*번호의 목록입니다. *포트*는 데이터그램을 주고받는 원격 포트입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.datagram.client.ports=123,456,789
```

참고

- 목록이 비어 있는 경우 포트 제한이 적용되지 않습니다.
- 잘못된 포트 번호는 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.
- 중복 포트는 삭제됩니다. 여러 개의 포트가 단일 IP 에 매핑되는 경우에는 하나의 포트만 유지됩니다.

introscope.agent.nio.datagram.server.ports

NIO 메트릭을 보고하는 포트를 나열합니다. 지정된 포트에 대한 '서버' 데이터그램 메트릭만 생성됩니다.

속성 설정

쉼표로 구분된 *포트* 번호의 목록입니다. *포트*는 데이터그램을 주고받는 로컬 포트입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.datagram.server.ports=123,456,789
```

참고

- 목록이 비어 있는 경우 포트 제한이 적용되지 않습니다.
- 잘못된 포트 번호는 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.nio.socket.client.hosts

메트릭 보고를 지정된 호스트가 있는 '클라이언트' TCP "연결"로 제한합니다.

속성 설정

쉼표로 구분된 호스트 목록입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.socket.client.hosts=hostA, hostB
```

참고

- 목록이 비어 있는 경우 호스트 제한이 적용되지 않습니다.
- 호스트는 이름이나 IP 주소(IPv4 또는 IPv6 형식)의 텍스트 표현으로 지정할 수 있습니다.
- 잘못된 호스트 이름은 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.nio.socket.client.ports

NIO 메트릭을 보고하는 포트를 나열합니다. 지정된 포트에 대한 '클라이언트' 소켓 메트릭만 생성됩니다.

속성 설정

쉼표로 구분된 *포트*번호의 목록입니다. *포트*는 데이터그램을 주고받는 원격 포트입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.socket.client.ports=123,456,789
```

참고

- 목록이 비어 있는 경우 포트 제한이 적용되지 않습니다.
- 잘못된 포트 번호는 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.nio.socket.server.ports

NIO 메트릭을 보고하는 포트를 나열합니다. 지정된 포트에 대한 '서버' 소켓 메트릭만 생성됩니다.

속성 설정

쉼표로 구분된 *포트* 번호의 목록입니다. *포트*는 데이터그램을 주고받는 로컬 포트입니다.

기본값

정의되지 않음(값 없음)

예

```
introscope.agent.nio.socket.client.ports=123,456,789
```

참고

- 목록이 비어 있는 경우 포트 제한이 적용되지 않습니다.
- 잘못된 포트 번호는 에이전트 로그에 보고되고 무시됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

Java NIO 메트릭은 Workstation Investigator 에서 에이전트의 최상위 수준 노드 아래에 있는 NIO 노드 아래에 표시됩니다. 모든 '클라이언트' 연결에 대한 추가 NIO 메트릭은 "백엔드" 노드 아래에 표시됩니다.

개별 NIO 메트릭은 표시하지 않을 메트릭에 대한 *TraceOneMethodIfFlagged* 또는 *TraceOneMethodWithParametersIfFlagged* 지시문의 주석 처리를 제거하여 표시하지 않을 수 있습니다. 하지만 이름이 *BackendTracer* 또는 *MappingTracer* 로 끝나는 추적 프로그램은 주석 처리를 제거하지 마십시오.

예를 들어, 데이터그램에 대한 "동시 판독기" 메트릭을 표시하지 않으려면 다음의 주석 처리를 제거하십시오.

```
TraceOneMethodWithParametersIfFlagged: NIODatagramTracing read
NIODatagramConcurrentInvocationCounter "Concurrent Readers"
```

JMX

다음 속성은 JMX 메트릭을 구성합니다.

- introscope.agent.jmx.enable (see page 319)
- introscope.agent.jmx.ignore.attributes (see page 320)
- introscope.agent.jmx.name.filter (see page 321)
- introscope.agent.jmx.name.jsr77.disable (see page 322)
- introscope.agent.jmx.name.primarykeys (see page 322)
- introscope.agent.jmx.excludeStringMetrics (see page 324)

introscope.agent.jmx.enable

JMX 메트릭의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

에이전트 버전별로 차이가 있습니다.

예

```
introscope.agent.jmx.enable=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.jmx.ignore.attributes

무시될 JMX MBean 특성(있는 경우)을 제어합니다.

속성 설정

쉼표로 구분된 키워드의 목록입니다.

기본값

주석 처리됨. *서버*

예

```
introscope.agent.jmx.ignore.attributes=server
```

참고

- MBean 특성 이름이 목록에 있는 한 이름과 일치하면 해당 특성이 무시됩니다.
- 모든 MBean 특성을 포함하려면 목록을 비워두십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.jmx.name.filter

Introscope 에서 수집하고 표시하는 JMX 데이터를 결정하기 위한 필터 문자열을 쉼표로 구분된 목록으로 지정합니다.

Introscope 에서는 필터 문자열과 일치하는 JMX 생성 메트릭을 보고합니다. 필터 문자열에는 별표(*) 및 물음표(?) 와일드카드 문자를 포함할 수 있습니다.

- *는 0 개 이상의 문자와 일치합니다.
- ?는 단일 문자열과 일치합니다.

리터럴 * 또는 ?를 일치시키려면 문자를 \\로 이스케이프하십시오.

예제:

- *ab*c* 는 *ab*c* 를 포함하는 메트릭 이름과 일치합니다.
- *ab*c* 는 *abc, abxc, abxxc* 등을 포함하는 메트릭 이름과 일치합니다.
- *ab?c* 는 *abxc* 를 포함하는 메트릭 이름과 일치합니다.
- *ab\\?c* 는 *ab?c* 를 포함하는 메트릭 이름과 일치합니다.

기본값

주석 처리됨

WebLogic 의 경우:

```
ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType
```

예

```
#introscope.agent.jmx.name.filter=ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType
```

참고

- 시스템에서 사용할 수 있는 모든 MBean 데이터를 포함하려면 이 속성을 비워 두십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.jmx.name.jsr77.disable

이 속성은 Introscope 가 복잡한 JMX 데이터를 포함한 전체 JSR77 데이터를 수집하고 보고할지 여부를 제어합니다.

이 속성은 WebLogic 및 WebSphere *IntroscopeAgent.profile* 파일에서만 사용할 수 있습니다.

속성 설정

True 또는 False

기본값

True

참고

- 이 속성을 적용하려면 응용 프로그램 서버에서 JSR-77 관리 지원을 제공해야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.jmx.name.primarykeys

MBean 정보의 사용자 정의 순서로, 이름 변환을 간단하게 해 줍니다.

속성 설정

특정 MBean 을 고유하게 식별할 키의 쉼표로 구분되고 순서가 지정된 목록입니다.

기본값

기본 *IntroscopeAgent.profile* 파일에서는 주석으로 처리되어 있습니다.

예

```
introscope.agent.jmx.name.primarykeys=J2EEServer
```

참고

- WebLogic 의 경우 속성 설정은 다음과 같습니다.
  - 유형
  - 이름
- WebLogic Server 9.0 을 사용하는 경우에는 이 속성을 주석으로 처리하십시오.
- WebSphere 의 경우 속성 설정은 다음과 같습니다.
  - J2EEServer
  - 응용 프로그램
  - j2eeType
  - JDBCProvider
  - 이름
  - mbeanIdentifier
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.jmx.excludeStringMetrics

문자열 값 메트릭을 포함할지 여부를 제어합니다. 문자열 값 메트릭을 사용하도록 설정하려면 이 속성 값을 false 로 설정하십시오.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.jmx.excludeStringMetrics=true
```

참고

- 문자열 값 메트릭을 제외하면 전체 메트릭 수가 줄어들어 에이전트 및 EM 성능이 향상됩니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

LeakHunter

다음 속성은 LeakHunter 와 에이전트의 상호 작용을 구성합니다.

- introscope.agent.leakhunter.collectAllocationStackTraces (see page 325)
- introscope.agent.leakhunter.enable (see page 326)
- introscope.agent.leakhunter.leakSensitivity (see page 327)
- introscope.agent.leakhunter.logfile.append (see page 328)
- introscope.agent.leakhunter.logfile.location (see page 328)
- introscope.agent.leakhunter.timeoutInMinutes (see page 329)

introscope.agent.leakhunter.collectAllocationStackTraces

LeakHunter 가 잠재 누수에 대한 할당 스택 추적을 생성하는지 여부를 제어합니다. 이 속성을 *true* 로 설정하면 잠재 누수의 할당에 대한 보다 정확한 데이터를 얻을 수 있지만 필요한 메모리 및 CPU 오버헤드가 늘어납니다. 따라서 기본값은 *false* 입니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.leakhunter.collectAllocationStackTraces=
false
```

참고

- 이 속성을 *true* 로 설정하면 CPU 사용량 및 메모리와 관련하여 시스템 오버헤드가 높아질 수 있습니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.leakhunter.enable

LeakHunter 기능이 사용되는지 여부를 제어합니다. LeakHunter 를 사용하려면 값을 true 로 설정하십시오.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.leakhunter.enable=false
```

참고

- 이 옵션을 설정하면 CPU 및 메모리 사용량이 높아질 수 있습니다. 다른 메트릭을 통해 메모리 누수가 있음을 파악한 경우에만 이 기능을 사용하십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.leakhunter.leakSensitivity

LeakHunter 누수 감지 알고리즘의 민감도 수준을 제어합니다. 민감도 설정이 높으면 보고되는 잠재 누수가 많아지고, 민감도가 낮으면 보고되는 잠재 누수가 적어집니다.

속성 설정

누수 민감도 값은 1(낮음)에서 10(높음) 사이의 양의 정수여야 합니다.

기본값

5

예

```
introscope.agent.leakhunter.leakSensitivity=5
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.leakhunter.logfile.append

응용 프로그램을 다시 시작할 때 로그 파일을 대체하거나 기존 로그 파일에 정보를 추가할지 여부를 지정합니다.

속성 설정

True 또는 False

- False 는 로그 파일을 대체합니다.
- True 는 기존 로그 파일에 정보를 추가합니다.

기본값

False

예

```
introscope.agent.leakhunter.logfile.append=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.leakhunter.logfile.location

LeakHunter.log 파일의 위치를 제어합니다. 파일 이름의 절대 또는 상대 경로를 지정할 수 있습니다. 상대 경로는 *<Agent_Home>* 디렉터리를 기준으로 확인됩니다. LeakHunter 가 로그 파일에 데이터를 기록하지 않도록 하려면 이 값을 비워 두거나 주석 처리하십시오.

기본값

기본 경로는 *../../logs/LeakHunter.log* 입니다.이 경우 로그 파일이 *<Agent_Home>logs* 디렉터리에 배치됩니다.

예

```
introscope.agent.leakhunter.logfile.location=../../logs/LeakHunter.log
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.leakhunter.timeoutInMinutes

LeakHunter 에서 새 잠재 누수를 찾는 데 걸리는 시간(분)을 제어합니다. 지정된 시간이 지나면 LeakHunter 는 새 잠재 누수 찾기를 중지하지만 이전에 식별한 잠재 누수는 계속 추적합니다.

속성 설정

음수가 아닌 양의 정수여야 합니다.

기본값

기본값은 120 분입니다.

예

```
introscope.agent.leakhunter.timeoutInMinutes=120
```

참고

- LeakHunter 가 항상 새 잠재 누수를 찾도록 하려면 이 값을 0 으로 설정하십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.leakhunter.ignore.<number>

지정된 패턴과 일치하는 클래스를 무시하려면 이 속성을 사용하십시오. 기본적으로 10 개의 클래스가 지정되어 있으며, 사용할 클래스는 주석으로 처리하십시오.

속성 설정

패턴과 일치하는 클래스의 쉼표로 구분된 목록입니다.

기본값

없음

예

```
introscope.agent.leakhunter.ignore.4=java.util.SubList
introscope.agent.leakhunter.ignore.5=com.sun.faces.context.BaseContextMap$EntrySe
t
introscope.agent.leakhunter.ignore.6=com.sun.faces.context.BaseContextMap$Key
```

참고

- 일부 컬렉션은 LeakHunter 와 함께 사용할 수 없습니다. LeakHunter 에서 컬렉션을 안전하게 사용하려면 언제 어떤 스레드에서든지 *size()*를 안전하게 호출할 수 있어야 합니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.
- "*" 와일드카드를 사용할 수 있습니다.

Logging(로깅)

다음 속성은 에이전트 로깅 옵션을 구성합니다.

- log4j.logger.IntroscopeAgent (see page 331)
- log4j.appender.logfile.File (see page 332)
- log4j.logger.IntroscopeAgent.inheritance (see page 332)
- log4j.appender.pbdlog.File (see page 333)
- log4j.appender.pbdlog (see page 333)
- log4j.appender.pbdlog.layout (see page 334)
- log4j.appender.pbdlog.layout.ConversionPattern (see page 334)
- log4j.additivity.IntroscopeAgent.inheritance (see page 335)

log4j.logger.IntroscopeAgent

이 속성은 로그 정보에 대한 로깅 수준과 출력 위치를 제어합니다.

속성 설정

세부화 수준 값은 다음일 수 있습니다.

- *INFO*
- *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel*

대상 값은 다음일 수 있습니다.

- *console*
- *logfile*
- *console* 및 *logfile* 모두

기본값

INFO, console, logfile

예

```
log4j.logger.IntroscopeAgent=INFO,console,logfile
```

에이전트 로깅을 사용하지 않도록 설정하려면 다음과 같이 이 속성에서 옵션을 제거합니다.

이전:

```
log4j.logger.IntroscopeAgent=INFO, console, logfile
```

이후:

```
log4j.logger.IntroscopeAgent=
```

참고

- 이 속성에 대한 변경 사항은 즉시 적용되므로 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

log4j.appender.logfile.File

log4j.logger.IntroscopeAgent 에 로그 파일이 지정된 경우 IntroscopeAgent.log 파일의 이름 및 위치를 지정합니다. 파일 이름은 에이전트 프로필이 포함된 디렉터리를 기준으로 합니다.

기본값

IntroscopeAgent.log

예

```
log4j.appender.logfile.File=../../logs/IntroscopeAgent.log
```

참고

시스템 속성(Java 명령줄 -D 옵션)은 파일 이름의 일부로 확장됩니다. 예를 들어 Java 가 *-Dmy.property=Server1* 로 시작되는 경우 *log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* 는 *log4j.appender.logfile.File=../../logs/Introscope-Server1.log* 로 확장됩니다.

log4j.logger.IntroscopeAgent.inheritance

계측이 필요한 클래스에 대한 로그 메시지의 로그 수준 및 대상을 제어합니다.

속성 설정

계측되지 않은 클래스는 슈퍼 형식 또는 인터페이스를 확장하므로 이러한 클래스의 로깅을 구성하려면 이 속성을 *INFO, pbdlog* 로 설정하십시오.

상속 클래스 로깅에 대한 자세한 내용은 지시문 로깅 제어 (see page 97)를 참조하십시오.

기본값

없음

예

```
log4j.logger.IntroscopeAgent.inheritance=INFO,pbdlog
```

log4j.appender.pbdlog.File

계측이 필요한 클래스에 대한 메시지를 기록하는 로그 파일을 식별합니다.

속성 설정

계측되지 않은 클래스는 슈퍼 형식 또는 인터페이스를 확장하므로 이러한 클래스의 로깅을 구성하려면 *pbdupdate.log* 로 설정하십시오.

기본값

없음

예

```
log4j.appender.pbdlog.File=../../pbdupdate.log
```

log4j.appender.pbdlog

계측이 필요한 클래스에 대한 메시지 로깅의 패키지를 지정합니다.

속성 설정

계측되지 않은 클래스는 슈퍼 형식 또는 인터페이스를 확장하므로 이러한 클래스의 로깅을 구성하려면 이 속성을 *com.wily.introscope.agent.AutoNamingRollingFileAppender* 로 설정하십시오.

기본값

없음

예

```
log4j.appender.pbdlog=com.wily.introscope.agent.AutoNamingRollingFileAppender
```

log4j.appender.pbdlog.layout

계측이 필요한 클래스에 대한 메시지 로깅의 규칙을 지정합니다.

속성 설정

계측되지 않은 클래스는 슈퍼 형식 또는 인터페이스를 확장하므로 이러한 클래스의 로깅을 구성하려면 이 속성을 *com.wily.org.apache.log4j.PatternLayout* 으로 설정하십시오.

기본값

없음

예

```
log4j.appender.pbdlog.layout=com.wily.org.apache.log4j.PatternLayout
```

log4j.appender.pbdlog.layout.ConversionPattern

계측이 필요한 클래스에 대한 메시지 로깅의 규칙을 지정합니다.

속성 설정

계측되지 않은 클래스는 슈퍼 형식 또는 인터페이스를 확장하므로 이러한 클래스의 로깅을 구성하려면 이 속성을 다음과 같이 설정하십시오.

```
%d{M/dd/yy hh:mm:ss a z} [%-3p] [%c] %m%n
```

기본값

없음

예

```
log4j.appender.pbdlog.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z} [%-3p]
[%c] %m%n
```

log4j.additivity.IntroscopeAgent.inheritance

다중 수준 상속에 대한 지시문이 *pbdupdate.log* 파일에만 로깅되도록 합니다.

속성 설정

True 또는 False

기본값

True

예

```
log4j.additivity.IntroscopeAgent.inheritance=true
```

참고

pbdupdate.log 에서만 다중 수준 상속 지시문의 로깅을 구성하려면 이 속성을 에이전트 프로필에 추가하고 *false* 로 설정하십시오.

메트릭 수

다음 속성은 Investigator 에서 *메트릭 수* 메트릭이 표시되는 위치에 영향을 줍니다.

- introscope.ext.agent.metric.count (see page 336)

introscope.ext.agent.metric.count

Investigator 에서 *메트릭 수* 메트릭이 표시되는 위치를 제어합니다. 기본적으로 *메트릭 수*는 *사용자 지정 메트릭 에이전트* 노드 아래에 표시됩니다. *에이전트 상태* 노드 아래에 *메트릭 수* 메트릭이 표시되도록 하려면 이 속성을 *IntroscopeAgent.profile* 에 추가하십시오.

속성 설정

True 또는 False

기본값

IntroscopeAgent.profile 에는 없음. False

예

```
introscope.ext.agent.metric.count=true
```

참고

'메트릭 수' 메트릭이 '에이전트 상태' 노드 아래에 표시되도록 하려면 이 속성을 *IntroscopeAgent.profile* 에 추가하고 true 로 설정하십시오.

다중 상속

인터페이스나 상위 클래스에 기반한 지시문의 경우, 에이전트는 다중 상속을 감지할 수 없으므로 이러한 클래스는 계측되지 않습니다. 응용 프로그램 서버 또는 에이전트 프로세스가 시작된 이후에 이러한 사례를 파악하려면 다음 속성을 활성화하십시오. 이러한 속성은 계측되어야 하는데 아직 계측되지 않았고 변경 사항을 반영하기 위해 동적 계측에 의존하는 클래스를 로깅합니다.

- introscope.autoprobe.hierarchysupport.enabled (see page 337)
- introscope.autoprobe.hierarchysupport.runOnceOnly (see page 338)
- introscope.autoprobe.hierarchysupport.pollIntervalMinutes (see page 338)
- introscope.autoprobe.hierarchysupport.executionCount (see page 339)
- introscope.autoprobe.hierarchysupport.disableLogging (see page 340)
- introscope.autoprobe.hierarchysupport.disableDirectivesChange (see page 340)

introscope.autoprobe.hierarchysupport.enabled

AutoProbe 및 동적 계측을 사용하여 JDK 1.5 에서 실행되는 에이전트의 경우 이 속성을 사용하여 슈퍼 형식 또는 인터페이스를 확장하는 클래스의 계측을 활성화할 수 있습니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.autoprobe.hierarchysupport.enabled=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.hierarchysupport.runOnceOnly

슈퍼 클래스 또는 인터페이스를 확장하는 클래스의 계측을 활성화한 경우, 이 속성을 사용하여 이 기능을 활성화하는 유틸리티가 한 번만 실행될지 또는 지정된 간격으로 실행될지 여부를 제어할 수 있습니다.

주기적 감지가 필요한 경우에만 이 속성을 **true** 로 변경하십시오.

속성 설정

True 또는 False

기본값

False

예

```
introscope.autoprobe.hierarchysupport.enabled=false
```

참고

- 동적 계측과 관련된 로깅 속성은 Logging 에 정의되어 있습니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.hierarchysupport.pollIntervalMinutes

다중 상속으로 인해 계측되지 못한 클래스를 확인할 폴링 간격입니다. 대부분의 경우 폴링은 한 번만 발생하지만 응용 프로그램 서버 초기화를 고려하여 보수적인 값이 권장됩니다.

기본값

5

예

```
introscope.autoprobe.hierarchysupport.pollIntervalMinutes=5
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.hierarchysupport.executionCount

단 한 번 또는 주기적으로 실행하는 대신 폴링이 지정된 횟수만큼 실행되도록 하려면 이 속성을 사용하여 폴링 간격이 실행되는 정확한 횟수를 지정하십시오. 실행해야 하는 정확한 횟수를 지정하려면 항상 이 속성을 사용하십시오.

이 속성을 사용하면 단 한 번 실행 설정이 무시됩니다.

속성 설정

양의 정수

기본값

3

예

```
introscope.autoprobe.hierarchysupport.executionCount=3
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.hierarchysupport.disableLogging

감지하는 클래스를 로깅할 필요가 없으면 이 속성의 주석 처리를 제거하십시오. 동적 계측이 활성화된 경우에만 이 속성의 주석 처리를 제거하십시오.

속성 설정

True 또는 False

기본값

True

예

```
#introscope.autoprobe.hierarchysupport.disableLogging=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.autoprobe.hierarchysupport.disableDirectivesChange

변경 사항만 로깅하고 동적 계측의 트리거를 비활성화하려면 이 속성의 주석 처리를 제거하십시오.

속성 설정

True 또는 False

기본값

True

예

```
introscope.autoprobe.hierarchysupport.disableDirectivesChange=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

플랫폼 모니터링

다음 속성은 플랫폼 모니터링 메트릭을 구성합니다.

- introscope.agent.platform.monitor.system (see page 341)

introscope.agent.platform.monitor.system

플랫폼 모니터를 로드할 운영 체제의 이름입니다.

속성 설정

이 속성의 옵션에 대한 자세한 내용은 플랫폼 모니터 문제 해결 (see page 234)을 참조하십시오.

기본값

주석 처리됨. 플랫폼마다 다양함

예

```
introscope.agent.platform.monitor.system=Solaris
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

원격 구성

다음 속성은 Java Agent 의 원격 구성을 가능하게 합니다.

- introscope.agent.remoteagentconfiguration.enabled (see page 342)
- introscope.agent.remoteagentconfiguration.allowedFiles (see page 342)

introscope.agent.remoteagentconfiguration.enabled

이 속성은 에이전트의 원격 구성을 활성화 또는 비활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.remoteagentconfiguration.enabled=true
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.remoteagentconfiguration.allowedFiles

이 속성은 이 에이전트로 원격 전송이 허용된 파일의 정확한 목록을 나열합니다.

속성 설정

domainconfig.xml

기본값

domainconfig.xml

예

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

보안

다음 속성은 CA CEM 으로 보내는 HTTP 헤더의 보안을 구성합니다.

- introscope.agent.decorator.security (see page 343)

introscope.agent.decorator.security

이 속성은 CA CEM 으로 보내는 데코레이트된 HTTP 응답 헤더의 형식을 결정합니다.

속성 설정

Clear: 일반 텍스트 인코딩

Encrypted: 헤더 데이터가 암호화됨

기본값

Clear

예

```
introscope.agent.decorator.security=clear
```

Servlet 헤더 데코레이터

다음 속성은 CA CEM 과 Introscope 간의 트랜잭션 상관 관계를 설정합니다.

- introscope.agent.decorator.enabled (see page 344)

introscope.agent.decorator.enabled

이 부울 값이 true 로 설정되어 있으면 에이전트가 HTTP 응답 헤더에 추가적인 성능 모니터링 정보를 추가하도록 구성됩니다. ServletHeaderDecorator 는 각 트랜잭션에 GUID 를 연결하고 이 GUID 를 HTTP 헤더(예: x-apm-info)에 삽입합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.decorator.enabled=false
```

소켓 메트릭

I/O 소켓의 생성은 다음 매개 변수에 의해 제한될 수 있습니다.

- introscope.agent.sockets.reportRateMetrics (see page 345)
- introscope.agent.io.socket.client.hosts (see page 345)
- introscope.agent.io.socket.client.ports (see page 346)
- introscope.agent.io.socket.server.ports (see page 346)

introscope.agent.sockets.reportRateMetrics

개별 소켓의 I/O(입력/출력) 대역폭 비율 메트릭을 보고하도록 설정합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.sockets.reportRateMetrics=false
```

참고

- 이 속성은 ManagedSocketTracing 이 사용하도록 설정되고 SocketTracing 은 사용하지 않도록 설정된 경우에만 작동합니다. 자세한 내용은 이전 버전과의 호환성 (see page 160)을 참조하십시오.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.io.socket.client.hosts

계측된 소켓 클라이언트 연결을 지정된 원격 호스트가 있는 연결로 제한합니다.

속성 설정

쉼표로 구분된 값의 목록입니다.

예

```
introscope.agent.io.socket.client.hosts=
```

참고

- 개별 값이 잘못된 경우 해당 값은 무시됩니다.
- 어떠한 매개 변수도 정의되지 않은 경우 또는 잘못된 값을 제외한 후 목록이 비어 있으면 해당 매개 변수에 제한이 적용되지 않습니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.io.socket.client.ports

계측된 소켓 클라이언트 연결을 지정된 원격 포트가 있는 연결로 제한합니다.

속성 설정

쉼표로 구분된 값의 목록입니다.

예

```
introscope.agent.io.socket.client.ports=
```

참고

- 개별 값이 잘못된 경우 해당 값은 무시됩니다.
- 어떠한 매개 변수도 정의되지 않은 경우 또는 잘못된 값을 제외한 후 목록이 비어 있으면 해당 매개 변수에 제한이 적용되지 않습니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.io.socket.server.ports

계측된 소켓 클라이언트 연결을 지정된 로컬 포트를 사용하는 연결로 제한합니다.

속성 설정

쉼표로 구분된 값의 목록입니다.

예

```
introscope.agent.io.socket.server.ports=
```

참고

- 개별 값이 잘못된 경우 해당 값은 무시됩니다.
- 어떠한 매개 변수도 정의되지 않은 경우 또는 잘못된 값을 제외한 후 목록이 비어 있으면 해당 매개 변수에 제한이 적용되지 않습니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

SQL 에이전트

SQL 에이전트의 요소를 구성할 수 있습니다.

추가 정보:

introscope.agent.sqlagent.normalizer.extension

이 속성은 미리 구성된 정규화 체계를 재정의하는 데 사용되는 SQL 노멀라이저 확장의 이름을 지정합니다.

사용자 지정 정규화 확장 작업을 수행하려면 해당 매니페스트 특성 *com-wily-Extension-Plugin-{pluginName}-Name* 의 값이 이 속성에 지정된 값과 일치해야 합니다.

이름을 쉼표로 구분된 목록으로 지정할 경우 에이전트는 기본 노멀라이저 확장을 사용합니다.

예를 들어 다음과 같은 설정을 사용하면 정규화에 RegexSqlNormalizer 가 사용됩니다.

```
introscope.agent.sqlagent.normalizer.extension=ext1, ext2
```

이 속성은 SQL 에이전트 메트릭에 대해 Investigator 트리에 나타나는 SQL 문의 크기(바이트)를 제한합니다.

속성 설정

미리 구성된 정규화 체계를 재정의하는 데 사용되는 SQL 노멀라이저 확장의 이름입니다.

기본값

RegexSqlNormalizer

예

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
```

참고

기본 설정을 사용하는 경우에는 다음과 같이 정규식 SQL 문의 노멀라이저 속성도 구성해야 합니다.

- introscope.agent.sqlagent.normalizer.regex.matchFallThrough (see page 348)
- introscope.agent.sqlagent.normalizer.regex.keys (see page 349)
- introscope.agent.sqlagent.normalizer.regex.key1.pattern (see page 350)
- introscope.agent.sqlagent.normalizer.regex.key1.replaceAll (see page 351)
- introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat (see page 352)
- introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive (see page 353)

이 속성의 변경 내용은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.matchFallThrough

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성이 true 로 설정되어 있으면 SQL 문자열이 모든 regex 키 그룹과 비교하여 평가됩니다.

구현은 체인으로 연결됩니다. 예를 들어 SQL 이 여러 키 그룹과 일치하는 경우 group1 의 정규화된 SQL 출력이 group2 에 대한 입력으로 제공되는 방식으로 처리됩니다.

이 속성이 false 로 설정되어 있으면 키 그룹이 일치하는 즉시 해당 그룹의 정규화된 SQL 출력이 반환됩니다.

속성 설정

True 또는 False

기본값

false

예

```
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.keys

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성은 regex 그룹 키를 지정합니다. regex 그룹 키는 순서대로 평가됩니다.

기본값

key1

예

```
introscope.agent.sqlagent.normalizer.regex.keys=key1
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.key1.pattern

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성은 SQL 과 일치시키는 데 사용되는 regex 패턴을 지정합니다.

속성 설정

java.util.Regex 패키지에서 허용되는 유효한 모든 regex 항목을 여기에 사용할 수 있습니다.

기본값

.*call(.*\)\.FOO(.*\)

예

```
introscope.agent.sqlagent.normalizer.regex.key1.pattern=.*call(.*\)\.FOO(.*\)
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.key1.replaceAll

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성이 false 로 설정되어 있으면 SQL 쿼리에서 첫 번째로 나타나는 일치하는 패턴이 대체 문자열로 바뀝니다. 이 속성이 true 로 설정되어 있으면 SQL 쿼리에서 나타나는 일치하는 패턴이 모두 대체 문자열로 바뀝니다.

속성 설정

True 또는 False

기본값

false

예

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성은 대체 문자열 형식을 지정합니다.

속성 설정

java.util.Regex 패키지와 *java.util.regex.Matcher* 클래스에서 허용되는 유효한 모든 regex 항목을 여기에 사용할 수 있습니다.

기본값

$1

예

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive

정규식 SQL 문 노멀라이저를 설정하려면 이 속성을 introscope.agent.sqlagent.normalizer.extension (see page 347)과 함께 사용하십시오. 이 속성은 패턴 일치 시 대/소문자 구분 여부를 지정합니다.

속성 설정

true 또는 false

기본값

false

예

```
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.sqlagent.sql.artonly

introscope.agent.sqlagent.sql.artonly 속성은 "평균 응답 시간" 메트릭만 생성하고 보내도록 에이전트를 구성하는 데 사용할 수 있습니다. 백엔드 아래의 모든 SQL 에이전트 메트릭에 영향을 미칩니다. 이 속성 값이 true 이면 SQL 메트릭 및 트랜잭션 추적에 대한 에이전트 성능이 향상될 수 있습니다.

참고: introscope.agent.sqlagent.sql.turnoffmetrics (see page 354)를 true 로 설정하면 이 속성이 재정의됩니다.

중요! 이 속성 설정이 작동하려면 다음 추적 프로그램 매개 변수를 설정해야 합니다.

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

이 속성은 기본적으로 다음과 같이 해제되어 있습니다.

```
introscope.agent.sqlagent.sql.artonly=false
```

이 속성의 변경 사항은 즉시 적용되며 관리 인터페이스를 사용하여 변경할 수 있습니다.

참고: 이 속성은 연결 수와 같은 사용자 지정 메트릭은 제어하지 않습니다.

introscope.agent.sqlagent.sql.rawsql

introscope.agent.sqlagent.sql.rawsql 속성은 "트랜잭션 추적"에서 *정규화되지 않은* SQL 을 SQL 구성 요소에 대한 매개 변수로 추가하도록 에이전트를 구성합니다. 이 속성 값이 true 이면 SQL 메트릭 및 트랜잭션 추적에 대한 에이전트 성능이 향상될 수 있습니다.

이 속성은 기본적으로 다음과 같이 해제되어 있습니다.

introscope.agent.sqlagent.sql.rawsql=false

이 속성의 변경 사항은 관리되는 응용 프로그램을 다시 시작한 후에 적용됩니다.

중요! 이 속성을 사용하도록 설정하면 암호 및 중요한 정보가 "트랜잭션 추적"에 표시될 수 있습니다.

introscope.agent.sqlagent.sql.turnoffmetrics

introscope.agent.sqlagent.sql.turnoffmetrics 속성을 사용하여 SQL 문 메트릭을 해제하면 에이전트에서 Enterprise Manager 로 전송되는 메트릭의 수를 줄일 수 있습니다. 이 속성 값이 true 이면 SQL 메트릭 및 트랜잭션 추적에 대한 에이전트 성능이 향상될 수 있습니다.

중요! 이 속성 설정이 작동하려면 다음 추적 프로그램 매개 변수를 설정해야 합니다.

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

이 속성은 기본적으로 다음과 같이 해제되어 있습니다.

```
introscope.agent.sqlagent.sql.turnoffmetrics=false
```

이 속성은 introscope.agent.sqlagent.sql.artonly 속성을 재정의합니다.

이 속성의 변경 사항은 즉시 적용되며 관리 사용자 인터페이스를 사용하여 변경할 수 있습니다.

introscope.agent.sqlagent.sql.turnofftrace

introscope.agent.sqlagent.sql.turnofftrace 속성은 에이전트가 백엔드 아래의 SQL 문에 대해 트랜잭션 추적 구성 요소를 생성하여 Enterprise Manager 로 전송할지 여부를 제어합니다. 이 속성 값이 true 이면 SQL 메트릭 및 트랜잭션 추적에 대한 에이전트 성능이 향상될 수 있습니다.

중요! 이 속성 설정이 작동하려면 다음 추적 프로그램 매개 변수를 설정해야 합니다.

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

이 속성은 기본적으로 다음과 같이 해제되어 있습니다.

```
introscope.agent.sqlagent.sql.turnofftrace=false
```

이 속성의 변경 사항은 즉시 적용되며 관리 사용자 인터페이스를 사용하여 변경할 수 있습니다.

SSL 통신

에이전트는 SSL 을 통해 Enterprise Manager 에 연결할 수 있습니다. 이 통신을 구성하려면 다음 속성을 사용하십시오.

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 258)
- introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

에이전트가 기본적으로 연결되는 Enterprise Manager 를 실행하는 컴퓨터의 호스트 이름을 지정합니다.

기본값

localhost

예

```
introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=localhost
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

Enterprise Manager 를 호스팅하는 컴퓨터에서 에이전트로부터의 연결을 수신 대기하는 포트 번호를 지정합니다. SSL(Secure Socket Layer) 프로토콜을 사용하는 경우 에이전트로부터 연결을 수신하는 기본 포트는 5443 입니다.

기본값

5443

예

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5443
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

SSL 을 사용할 때 에이전트에서 Enterprise Manager 로의 연결에 사용할 클라이언트 소켓 팩터리를 지정합니다.

기본값

com.wily.isengard.postofficehub.link.net.SSLSocketFactory

예

```
ntroscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.SSLSocketFactory
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT

트러스트된 Enterprise Manager 인증서가 포함된 신뢰 저장소의 위치입니다. 신뢰 저장소가 지정되지 않으면 에이전트는 모든 인증서를 신뢰합니다.

속성 설정

에이전트의 작업 디렉터리에 대한 절대 경로 또는 상대 경로입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/var/trustedcerts
```

참고

Windows 에서는 백슬래시를 이스케이프해야 합니다. 예: C:\\keystore

introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT

신뢰 저장소의 암호입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT=
```

introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT

에이전트의 인증서를 포함하는 키 저장소의 위치입니다. Enterprise Manager 가 클라이언트 인증을 요구하는 경우 키 저장소가 필요합니다.

속성 설정

에이전트의 작업 디렉터리에 대한 절대 경로 또는 상대 경로입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=c:\\keystore
```

참고

Windows 에서는 백슬래시를 이스케이프해야 합니다. 예: C:\\keystore

introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT

키 저장소의 암호입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT=MyPassword76
8
```

introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT

활성화된 암호 그룹을 설정합니다.

속성 설정

쉼표로 구분된 암호 그룹의 목록입니다.

예

```
introscope.agent.enterprisemanager.transport.tcp.
ciphersuites.DEFAULT=SSL_DH_anon_WITH_RC4_128_MD5
```

참고

지정되지 않으면 기본 활성화된 암호 그룹을 사용합니다.

중단 메트릭

다음 속성은 중단 메트릭에 대한 속성입니다.

- introscope.agent.stalls.thresholdseconds (see page 359)
- introscope.agent.stalls.resolutionseconds (see page 359)

중단 메트릭 속성에 대한 자세한 내용은 중단을 이벤트로 캡처하지 않도록 설정 (see page 208)을 참조하십시오.

introscope.agent.stalls.thresholdseconds

이 속성은 실행 중인 프로세스가 중단된 프로세스로 간주되기 전까지 허용되는 시간(초)을 지정합니다. "중단 수" 메트릭의 정확성을 보장하려면 중단 임계값을 15 초 이상으로 설정해야 합니다. 이 설정을 통해 Enterprise Manager 가 하베스트 주기를 완료하기까지 허용되는 시간을 지정할 수 있습니다.

기본값

기본값은 30 초입니다.

예

```
introscope.agent.stalls.thresholdseconds=30
```

참고

이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.stalls.resolutionseconds

이 속성은 에이전트가 중단 발생 여부를 검사하는 빈도를 지정합니다. "중단 수" 메트릭의 정확성을 보장하려면 중단 레졸루션을 10 초 이상으로 설정해야 합니다 이 설정을 통해 Enterprise Manager 가 하베스트 주기를 완료하기까지 허용되는 시간을 지정할 수 있습니다.

기본값

기본값은 매 10 초입니다.

예

```
introscope.agent.stalls.resolutionseconds=10
```

참고

이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

스레드 덤프

다음 속성은 CA Introscope 스레드 덤프 기능의 에이전트 요소를 사용하도록 설정하고 구성합니다.

- introscope.agent.threaddump.enable (see page 361)
- introscope.agent.threaddump.deadlockpoller.enable (see page 362)
- introscope.agent.threaddump.deadlockpollerinterval (see page 362)
- introscope.agent.threaddump.MaxStackElements (see page 363)

참고: 스레드 덤프를 구성하는 방법에 대한 자세한 내용은 스레드 덤프를 사용하도록 설정하고 구성하는 방법 (see page 80)을 참조하십시오.

introscope.agent.threaddump.enable

에이전트 JVM 에서 스레드 덤프를 수집하도록 설정하고 사용자가 "스레드 덤프" 탭을 볼 수 있도록 합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.threaddump.enable=true
```

참고

- 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.
- 이 속성은 true 로 설정된 경우 Enterprise Manager 스레드 덤프 기능을 사용하도록 설정하는 *IntroscopeEnterpriseManager.properties* 파일의 introscope.enterprisemanager.threaddump.enable 속성과 함께 작동합니다.

introscope.agent.threaddump.deadlockpoller.enable

메트릭 브라우저 트리에서 "Deadlock Count"(교착 상태 수) 메트릭을 사용하도록 설정하여 에이전트 JVM 의 현재 교착 상태 수를 표시합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.threaddump.deadlockpoller.enable=true
```

참고

- 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.threaddump.deadlockpollerinterval

CA Introscope 가362 교착 상태가 된 스레드에 대해 에이전트 JVM 을 폴링하는 빈도(밀리초)입니다.

속성 설정

0 보다 큰 정수

기본값

15000(밀리초)

예

```
introscope.agent.threaddump.deadlockpollerinterval=15000
```

참고

- 이 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작하십시오.

introscope.agent.threaddump.MaxStackElements

스레드 스택 추적의 총 행 수에 따라 CA Introscope 스레드 덤프의 크기가 결정됩니다. 이 속성은 스레드 스택에서 허용되는 행 수를 설정합니다.

속성 설정

0 보다 크고 25,000 보다 작거나 같은 정수

기본값

12,000

예

```
introscope.agent.threaddump.MaxStackElements=12000
```

참고

이 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작하십시오.

트랜잭션 추적

다음은 속성은 트랜잭션 추적과 샘플링에 대한 속성입니다.

- introscope.agent.bizdef.turnOff.nonIdentifying.txn (see page 364)
- introscope.agent.transactiontracer.parameter.httprequest.headers (see page 365)
- introscope.agent.transactiontracer.parameter.httprequest.parameters (see page 366)
- introscope.agent.transactiontracer.parameter.httpsession.attributes (see page 366)
- introscope.agent.transactiontracer.userid.key (see page 367)
- introscope.agent.transactiontracer.userid.method (see page 367)
- introscope.agent.transactiontrace.componentCountClamp (see page 368)
- introscope.agent.crossprocess.compression (see page 369)
- introscope.agent.crossprocess.compression.minlimit (see page 370)
- introscope.agent.crossprocess.correlationid.maxlimit (see page 371)
- introscope.agent.transactiontracer.sampling.enabled (see page 372)
- introscope.agent.transactiontracer.sampling.perinterval.count (see page 372)
- introscope.agent.transactiontracer.sampling.interval.seconds (see page 373)
- introscope.agent.transactiontrace.headFilterClamp (see page 373)

자세한 내용은 트랜잭션 추적 옵션 구성 (see page 197)을 참조하십시오.

introscope.agent.bizdef.turnOff.nonIdentifying.txn

비식별 트랜잭션의 추적을 사용하거나 사용하지 않도록 설정합니다.

이 속성을 *introscopeAgent.profile* 에서 FALSE 로 설정하면 비식별 트랜잭션에 대한 추적이 생성됩니다.

CEM UI 에서 이 기능을 사용하도록 설정해도 비식별 트랜잭션에 대한 추적은 기본적으로 생성되지 않습니다.

속성 설정

TRUE 또는 FALSE

기본값

TRUE

예

```
introscope.agent.bizdef.turnOff.nonIdentifying.txn=FALSE
```

introscope.agent.transactiontracer.parameter.httprequest.headers

캡처할 HTTP 요청 헤더 데이터를 쉼표로 구분된 목록으로 지정합니다. 쉼표로 구분된 목록을 사용하십시오.

기본값

주석 처리됨. *User-Agent*

예

```
introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent
```

참고

IntroscopeAgent.profile 에는 이 속성의 값을 Null 값으로 설정하는 주석 처리된 문이 포함되어 있습니다. 필요한 경우 문의 주석 처리를 제거하고 원하는 헤더 이름을 지정할 수 있습니다.

introscope.agent.transactiontracer.parameter.httprequest.parameters

캡처할 HTTP 요청 매개 변수 데이터를 쉼표로 구분된 목록으로 지정합니다.

기본값

주석 처리됨. 일반 매개 변수

예

```
introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter1,parameter2
```

참고

IntroscopeAgent.profile 에는 이 속성의 값을 Null 값으로 설정하는 주석 처리된 문이 포함되어 있습니다. 필요한 경우 문의 주석 처리를 제거하고 원하는 매개 변수 이름을 지정할 수 있습니다.

introscope.agent.transactiontracer.parameter.httpsession.attributes

캡처할 HTTP 세션 특성 데이터를 쉼표로 구분된 목록으로 지정합니다.

기본값

주석 처리됨. 일반 매개 변수

예

```
introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

참고

IntroscopeAgent.profile 에는 이 속성의 값을 Null 값으로 설정하는 주석 처리된 문이 포함되어 있습니다. 필요한 경우 문의 주석 처리를 제거하고 원하는 매개 변수 이름을 지정할 수 있습니다.

introscope.agent.transactiontracer.userid.key

사용자 정의된 키 문자열입니다.

기본값

주석 처리됨. 일반 매개 변수

예

```
#introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

참고

IntroscopeAgent.profile 에는 이 속성의 값을 Null 값으로 설정하는 주석 처리된 문이 포함되어 있습니다. *HttpServletRequest.getHeader* 또는 *HttpServletRequest.getValue* 를 사용하여 사용자 ID 를 액세스하는 경우 사용자는 필요하면 이 명령문의 주석 처리를 제거하고 올바른 값을 제공할 수 있습니다.

자세한 내용은 introscope.agent.transactiontracer.userid.method (see page 367)를 참조하십시오.

introscope.agent.transactiontracer.userid.method

사용자 ID 를 반환하는 메서드를 지정합니다. 에이전트 프로필에는 세 개의 허용 가능한 각 값에 대한 주석 처리된 속성 정의가 포함되어 있습니다.

getRemoteUser, getHeader 또는 getValue 중 어떤 메서드로 사용자 ID 에 액세스할지에 따라 적절한 문의 주석 처리를 제거하십시오.

속성 설정

허용 가능한 값은 다음과 같습니다.

- *HttpServletRequest.getRemoteUser*
- *HttpServletRequest.getHeader*
- *HttpServletRequest.getValue*

기본값

주석 처리됨. 위의 옵션 참조

예

IntroscopeAgent.profile 에는 세 개의 허용 가능한 각 값에 대한 주석 처리된 속성 정의가 포함되어 있습니다. 사용할 속성의 주석 처리를 제거할 수 있습니다.

```
introscope.agent.transactiontracer.userid.method=HttpServletRequest.getRemoteUser
#introscope.agent.transactiontracer.userid.method=HttpServletRequest.getHeader
#introscope.agent.transactiontracer.userid.method=HttpSession.getValue
```

introscope.agent.transactiontrace.componentCountClamp

트랜잭션 추적에서 허용되는 구성 요소 수를 제한합니다.

기본값

5000

중요! 클램프 크기가 늘어나면 메모리 요구 사항도 높아집니다. 극단적인 경우 JVM 의 최대 힙 크기를 조정해야 할 수 있습니다. 그러지 않으면 관리되는 응용 프로그램에 메모리가 부족할 수 있습니다.

예

```
introscope.agent.transactiontrace.componentCountClamp=5000
```

참고

- 클램프를 초과하는 트랜잭션 추적은 에이전트에서 삭제되고 에이전트 로그 파일에 경고 메시지가 기록됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.
- 설정된 제한에 도달하면 로그에 경고가 나타나고 추적이 중지됩니다.
- 0 은 올바른 값이 **아닙니다**. introscope.agent.transactiontrace.componentCountClamp=0 을 설정하지 **마십시오**.

introscope.agent.crossprocess.compression

크로스 프로세스 트랜잭션 추적 데이터의 크기를 줄이려면 이 속성을 사용하십시오.

속성 설정

lzma, gzip, none

기본값

lzma

예

```
introscope.agent.crossprocess.compression=lzma
```

참고

- 이 옵션을 사용하면 에이전트 CPU 오버헤드가 늘어나지만 프로세스 간 헤더의 크기는 줄어듭니다.
- *lzma* 압축이 *gzip* 보다 효율적이지만 CPU 를 더 많이 사용할 수 있습니다.
- .NET 에이전트는 *gzip* 옵션을 지원하지 않으므로 상호 운용성이 필요한 경우에는 *gzip* 을 사용하면 안 됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.crossprocess.compression.minlimit

압축을 적용할 크로스 프로세스 매개 변수 데이터 길이에 대한 최소 길이를 설정하려면 이 속성을 사용하십시오.

속성 설정

0 에서 총 최대 제한의 두 배까지 introscope.agent.crossprocess.correlationid.maxlimit (see page 371)에 설정할 수 있습니다.

기본값 1500 보다 작게 설정되면 압축이 보다 빈번히 실행되므로 더 많은 CPU 오버헤드를 발생시킵니다. 기본 설정 값인 1500 은 일반적으로 정상적인 조건에서 CPU 에 오버헤드를 발생시키지 않습니다.

기본값

1500

예

```
introscope.agent.crossprocess.compression.minlimit=1500
```

참고

- 위의 introscope.agent.crossprocess.compression 속성에서만 사용됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.crossprocess.correlationid.maxlimit

크로스 프로세스 매개 변수 데이터의 최대 허용 크기입니다.

크로스 프로세스 매개 변수 데이터의 총 크기가 이 제한보다 큰 경우에는 압축을 적용한 후에도 일부 데이터가 드롭되고 일부 크로스 프로세스 상관 관계 기능이 제대로 작동하지 않습니다.

그러나 이 설정을 사용하면 너무 큰 헤더 크기로 인해 사용자 트랜잭션이 네트워크 전송 시 실패하지 않도록 보호됩니다.

기본값

4096

예

```
introscope.agent.crossprocess.correlationid.maxlimit=4096
```

참고

- 위의 *introscope.agent.crossprocess.compression* 및 *introscope.agent.crossprocess.compression.minlimit* 속성과 함께 사용됩니다.
- 이 속성은 동적 속성입니다. 런타임 동안 이 속성의 구성을 변경할 수 있으며, 변경 사항은 자동으로 선택됩니다.

introscope.agent.transactiontracer.sampling.enabled

트랜잭션 추적 프로그램 샘플링을 사용하지 않으려면 다음 속성의 주석 처리를 제거하십시오.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.transactiontracer.sampling.enabled=false
```

참고

이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.

introscope.agent.transactiontracer.sampling.perinterval.count

이 속성은 일반적으로 Enterprise Manager 에서 구성됩니다. 이 속성을 에이전트에서 구성할 경우 Enterprise Manager 의 구성은 사용되지 않습니다. 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

기본값

1

예

```
introscope.agent.transactiontracer.sampling.perinterval.count=1
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.transactiontracer.sampling.interval.seconds

이 속성은 일반적으로 Enterprise Manager 에서 구성됩니다. 이 속성을 에이전트에서 구성할 경우 Enterprise Manager 의 구성은 사용되지 않습니다.

참고: 자세한 내용은 *CA APM 구성 및 관리 안내서*를 참조하십시오.

기본값

120

예

```
introscope.agent.transactiontracer.sampling.interval.seconds=120
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.transactiontrace.headFilterClamp

헤드 필터링에 허용되는 최대 구성 요소 수준을 지정합니다. 헤드 필터링은 전체 트랜잭션을 수집할 목적으로 트랜잭션의 시작을 검사하는 프로세스입니다. 헤드 필터링 기능은 첫 번째 Blame 관련 구성 요소가 나타날 때까지 각 구성 요소를 검사합니다. 호출 스택 수준이 매우 깊은 트랜잭션의 경우 클램프가 적용되지 않으면 문제가 될 수 있습니다. 클램프 값은 에이전트가 고정된 수준까지만 조회하도록 하여 이 동작이 메모리 및 CPU 사용률에 미치는 영향을 제한합니다.

기본값

30

경고! 클램프 크기가 늘어나면 메모리 요구 사항도 높아집니다. 이 경우 가비지 수집 동작이 영향을 받으므로 응용 프로그램 전체 성능에 영향을 줍니다.

예

```
introscope.agent.transactiontrace.headFilterClamp=30
```

참고

- 이 속성의 변경 사항은 즉시 적용되며 관리되는 응용 프로그램을 다시 시작할 필요가 없습니다.
- 샘플링이나 사용자가 시작한 트랜잭션 추적과 같은 일부 다른 메커니즘이 수집할 트랜잭션을 선택하도록 활성화되어 있지 *않은 경우*에는 가능한 수집에 대해 클램프를 초과하는 수준의 트랜잭션 추적이 더 이상 검사되지 않습니다.

introscope.agent.ttClamp

이 속성은 보고 주기마다 에이전트에 의해 보고되는 트랜잭션 수를 제한합니다.

속성 설정

정수

기본값

50

예

```
introscope.agent.ttClamp=50
```

참고

- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.
- 이 속성이 설정되지 않으면(비어 있음) 기본값 200 이 사용됩니다.

URL 그룹화

다음은 속성은 프런트엔드 메트릭을 위한 URL 그룹을 구성하는 데 사용되는 속성입니다.

- introscope.agent.urlgroup.keys (see page 375)
- introscope.agent.urlgroup.group.default.pathprefix (see page 376)
- introscope.agent.urlgroup.group.default.format (see page 376)

자세한 내용은 URL Workflow 사용 (see page 190)을 참조하십시오.

introscope.agent.urlgroup.keys

프런트엔드 이름 지정에 대한 구성 설정입니다.

기본값

기본값

예

```
introscope.agent.urlgroup.keys=default
```

참고

URL 주소가 두 개의 URL 그룹에 속하는 경우 이 속성에서 URL 그룹에 대한 키를 나열하는 순서는 중요하지 않습니다. 더 좁은 패턴에 의해 정의된 URL 그룹은 더 넓은 패턴에 의해 정의된 URL 그룹에 선행해야 합니다.

예를 들어, 키가 alpha 인 URL 그룹에 하나의 주소가 포함되어 있고 키가 beta 인 URL 그룹에 첫 번째 URL 그룹의 주소를 포함하는 네트워크 세그먼트에 있는 모든 주소를 포함하는 경우, 키 매개 변수에서 alpha 가 beta 에 선행해야 합니다.

introscope.agent.urlgroup.group.default.pathprefix

프런트엔드 이름 지정에 대한 구성 설정입니다.

기본값

*

예

```
introscope.agent.urlgroup.group.default.pathprefix=*
```

introscope.agent.urlgroup.group.default.format

프런트엔드 이름 지정에 대한 구성 설정입니다.

기본값

기본값

예

```
introscope.agent.urlgroup.group.default.format=default
```

WebSphere PMI

다음 속성은 WebSphere PMI 메트릭을 구성합니다.

- introscope.agent.pmi.enable (see page 378)
- introscope.agent.pmi.enable.alarmManagerModule (see page 378)
- introscope.agent.pmi.enable.beanModule (see page 379)
- introscope.agent.pmi.enable.cacheModule (see page 379)
- introscope.agent.pmi.enable.connectionPoolModule (see page 380)
- introscope.agent.pmi.enable.hamanagerModule (see page 380)

- introscope.agent.pmi.enable.j2cModule (see page 381)
- introscope.agent.pmi.enable.jvmpiModule (see page 381)
- introscope.agent.pmi.enable.jvmRuntimeModule (see page 382)
- introscope.agent.pmi.enable.objectPoolModule (see page 382)
- introscope.agent.pmi.enable.orbPerfModule (see page 383)
- introscope.agent.pmi.enable.schedulerModule (see page 383)
- introscope.agent.pmi.enable.servletSessionsModule (see page 384)
- introscope.agent.pmi.enable.systemModule (see page 384)
- introscope.agent.pmi.enable.threadPoolModule (see page 385)
- introscope.agent.pmi.enable.transactionModule (see page 385)
- introscope.agent.pmi.enable.webAppModule (see page 386)
- introscope.agent.pmi.enable.webServicesModule (see page 386)
- introscope.agent.pmi.enable.wlmModule (see page 387)
- introscope.agent.pmi.enable.wsgwModule (see page 387)
- introscope.agent.pmi.filter.objrefModule (see page 388)

이러한 속성은 *IntroscopeAgent.websphere.profile* 파일 또는 WebSphere 설치의 기본 에이전트 프로필에서 찾을 수 있습니다.

introscope.agent.pmi.enable

WebSphere PMI 에서 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.pmi.enable=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.alarmManagerModule

true 로 설정된 경우 PMI 경고 관리자 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.alarmManagerModule=false
```

참고

- 경고 관리자 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.beanModule

PMI bean 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.beanModule=false
```

introscope.agent.pmi.enable.cacheModule

true 로 설정된 경우 PMI 캐시 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.cacheModule=false
```

참고

- 캐시 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.connectionPoolModule

PMI connectionPool 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.pmi.enable.connectionPoolModule=true
```

introscope.agent.pmi.enable.hamanagerModule

true 로 설정된 경우 PMI 관리자 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.hamanagerModule=false
```

참고

- 관리자 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.j2cModule

true 로 설정된 경우 PMI J2C 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.pmi.enable.j2cModule=true
```

참고

- J2C 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.jvmpiModule

PMI JVM PI 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.jvmpiModule=false
```

참고

이 모듈에 데이터가 제공되도록 하려면 JVMPI 가 WebSphere 에서 켜져 있어야 합니다.

introscope.agent.pmi.enable.jvmRuntimeModule

PMI JVM 런타임 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.jvmRuntimeModule=false
```

참고

- 이 모듈에 데이터가 제공되도록 하려면 JVMPI 가 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.objectPoolModule

true 로 설정된 경우 PMI 개체 풀 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.objectPoolModule=false
```

참고

- 개체 풀 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.orbPerfModule

true 로 설정된 경우 PMI orbPerf 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.orbPerfModule=false
```

참고

- orbPerf 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.schedulerModule

true 로 설정된 경우 PMI 스케줄러 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.schedulerModule=false
```

참고

- 스케줄러 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.servletSessionsModule

PMI servletSessions 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.pmi.enable.servletSessionsModule=true
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.systemModule

true 로 설정된 경우 PMI 시스템 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.systemModule=false
```

참고

- 시스템 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.threadPoolModule

true 로 설정된 경우 PMI 스레드 풀 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

True

예

```
introscope.agent.pmi.enable.threadPoolModule=true
```

참고

- 스레드 풀 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.transactionModule

PMI 트랜잭션 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.transactionModule=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.webAppModule

PMI webApp 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.webAppModule=false
```

참고

이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.webServicesModule

true 로 설정된 경우 PMI 웹 서비스 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.webServicesModule=false
```

참고

- 웹 서비스 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.wlmModule

true 로 설정된 경우 PMI WLM 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.wlmModule=false
```

참고

- WLM 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.enable.wsgwModule

true 로 설정된 경우 PMI WSGW 데이터의 수집을 활성화합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.enable.wsgwModule=false
```

참고

- WSGW 데이터 범주는 Introscope 데이터로서 표시되도록 WebSphere 에서 켜져 있어야 합니다.
- 이 속성의 변경 내용을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

introscope.agent.pmi.filter.objrefModule

하드 코딩된 필터에 대한 제어입니다.

objref 필터는 "@xxxxx"로 끝나는 이름을 필터링합니다. 여기서 "xxxxx"는 숫자입니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.pmi.filter.objrefModule=false
```

참고

이 속성의 변경 사항을 적용하려면 관리되는 응용 프로그램을 다시 시작해야 합니다.

WLDF metrics(I/O 메트릭)

다음 속성은 WLDF 메트릭을 구성합니다.

- introscope.agent.wldf.enable (see page 389)

introscope.agent.wldf.enable

이 속성은 WLDF 메트릭을 수집하도록 설정합니다.

속성 설정

True 또는 False

기본값

False

예

```
introscope.agent.wldf.enable=false
```

부록 B: 대체 계측 방법

이 단원에서는 JVM AutoProbe 를 사용할 수 없는 경우 응용 프로그램을 계측하기 위한 대체 방법에 대해 설명합니다. 대부분의 경우 이 단원에 설명된 대체 방법보다는 JVM AutoProbe 를 사용하는 것이 좋습니다. 그러나 특정 응용 프로그램 서버에 JVM AutoProbe 를 사용할 수 없는 경우에는 이 단원의 지침을 따라 응용 프로그램을 계측할 수 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

다른 응용 프로그램 서버에 Java Agent 배포

JVM AutoProbe 는 응용 프로그램을 계측하는 데 일반적으로 사용되는 방법입니다. JVM AutoProbe 를 사용하여 응용 프로그램을 계측하는 것이 가장 좋습니다.

그러나 다음 응용 프로그램 서버에서 JVM 1.4 이하 버전이 실행되고 있는 경우에는 Application Server AutoProbe 를 사용할 수 있습니다.

- Sun ONE 7.0

  Application Server AutoProbe 는 Sun ONE 버전 7 응용 프로그램 서버에서만 지원됩니다.

- Oracle 10g 10.0.3

  Application Server AutoProbe 는 Oracle 버전 10g 10.0.3 응용 프로그램 서버에서만 지원됩니다.

- WebSphere 또는 WebLogic

중요! 다음 응용 프로그램 서버에서는 Application Server AutoProbe 가 지원되지 않습니다.

- JVM 1.5 이상
- OS/400

중요! 응용 프로그램을 계측하는 데는 한 가지 방법만 사용하십시오. 이미 JVM AutoProbe 를 사용하고 있는 경우 Application Server AutoProbe 를 사용하면 안 됩니다.

응용 프로그램 서버를 시작할 때 클래스 이름에 대한 식별자로 하이픈(-) 문자를 사용하지 않는 것이 좋습니다. CA Introscope®는 이 문자를 구문 분석하지 않으므로 이 문자를 사용할 경우 에이전트 로그에 클래스 로드 오류가 기록될 수 있습니다.

추가 정보:

AutoProbe 를 사용하도록 Sun ONE 구성 (페이지 392)
AutoProbe 를 사용하도록 Oracle 구성 (페이지 394)
Java Agent 를 사용하도록 IBM WebSphere 구성 (페이지 53)
Java Agent 를 사용하도록 Oracle WebLogic 구성 (페이지 47)

AutoProbe 를 사용하도록 Sun ONE 구성

Sun ONE 7.0 에 해당

AutoProbe 를 사용하여 응용 프로그램을 계측하도록 Sun ONE 설치를 구성할 수 있습니다.

다음 단계를 따르십시오.

참고: 아래 .xml 예제에서 "..."는 해당 정보가 .xml 코드에 표시되지 않음을 나타냅니다. 이 정보는 예제와 관련이 없습니다.

1. 관리자 또는 루트로 로그인합니다.
2. 다음 위치로 이동하고 server.xml 파일을 엽니다.
   <Sun ONE install dir>/domains/*domain1/server1/config/*

   참고: 항목 구분 기호는 콜론(:)입니다.

3. server.xml 파일에서 java-config 요소의 "server-classpath" 속성에 wily/Agent.jar 의 전체 경로를 추가합니다. 예:
   ```
   <java-config ... server-classpath="/sw/sun/sunone7/wily/Agent.jar:..." ...>
   ```
4. 다음 java-config 요소로 이동합니다.
   - bytecode-preprocessors 속성을 추가하고 com.wily.introscope.api.sun.appserver.SunONEAutoProbe 값으로 설정합니다.

     예:
     ```
     <java-config ...
     bytecode-preprocessors="com.wily.introscope.api.sun.appserver.SunONEAutoP
     robe">
     ```
   - jvm-options 요소를 추가하고 에이전트 프로필의 위치를 정의합니다. com.wily.introscope.agentProfile 또는 com.wily.introscope.agentResource 를 정의합니다.

     com.wily.introscope.agentProfile 의 예는 다음과 같습니다.
     ```
     <java-config ...>
     ...
     <jvm-options>-Dcom.wily.introscope.agentProfile=/sw/sun/sunone7/wily/core
     /config/IntroscopeAgent.profile </jvm-options>
     </java-config>
     ```

     com.wily.introscope.agentResource 의 예는 다음과 같습니다.
     ```
     <java-config ...>
     ...
     <jvm-options>-Dcom.wily.introscope.agentResource=<virtual path
     to>/IntroscopeAgent.profile</jvm-options>
     </java-config>
     ```
   - (선택 사항) com.wily.introscope.agentResource 를 구성한 경우 서버 클래스 경로에 해당 리소스 파일을 추가합니다.
5. 추적 프로그램 그룹을 구성하여 서블릿 데이터를 수집합니다.

AutoProbe 를 사용하도록 Oracle 구성

Oracle 10g 10.0.3 에 해당

AutoProbe 를 사용하여 응용 프로그램을 계측하도록 Oracle 설치를 구성할 수 있습니다.

다음 단계를 따르십시오.

1. 응용 프로그램 서버 클래스 경로에 Agent.jar 를 추가합니다.
2. 시스템 속성 oracle.classpreprocessor.classes 을 com.wily.introscope.api.oracle.OracleAutoProbe 값으로 설정합니다.
3. 시스템 속성 oracle.j2ee.class.preprocessing 을 true 값으로 설정합니다.
4. 명령줄에서 다음 명령을 실행하십시오.

   -Dcom.wily.introscope.probebuilder.oracle.enable=true
5. 다음 명령을 사용하여 Oracle Application Server 10g 를 다시 시작합니다.

   ```
   java
   -Doracle.classpreprocessor.classes=com.wily.introscope.api.oracle.OracleAutoP
   robe -Doracle.j2ee.class.preprocessing=true
   -Dcom.wily.introscope.probebuilder.oracle.enable=true -classpath
   oc4j.jar:<path to wily install dir>/wily/Agent.jar
   com.evermind.server.OC4JServer -config <path to oracle install
   dir>/config/server.xml
   ```

   중요! Windows 호스트에서 Sun JDK 1.42 를 사용하여 Oracle 10g 릴리스 2 를 실행하는 사용자는 ^(캐럿) 문자를 사용하여 슬래시를 이스케이프해야 합니다. 예:

   ```
   -Xbootclasspath^/p:<IntroscopeAgent.jar path>
   ```
6. 추적 프로그램 그룹을 구성하여 서블릿 데이터를 수집합니다.

추가 정보:

HTTP 서블릿 추적 구성 (페이지 395)

WebLogic Server 구성

AutoProbe 를 사용하여 응용 프로그램을 계측하도록 WebLogic Server 를 구성할 수 있습니다.

다음 단계를 따르십시오.

1. wily/Agent.jar 파일을 포함하도록 응용 프로그램 시작 스크립트(예: *startMedRecServer.cmd*)의 클래스 경로를 편집합니다.
2. Java 명령줄에서 -D 옵션으로 응용 프로그램 시작 스크립트의 다음 속성을 설정하여 Introscope AutoProbe 를 활성화합니다.
   ```
   -Dweblogic.classloader.preprocessor=
   com.wily.introscope.api.weblogic.PreProcessor
   ```
3. HTTP 서블릿 추적을 구성 (see page 395)하여 서블릿 데이터를 수집하도록 추적 프로그램 그룹을 구성합니다.

HTTP 서블릿 추적 구성

응용 프로그램 서버에서 AutoProbe 를 사용하여 응용 프로그램을 계측하려면 먼저 *toggles-full.pbd* 및 *toggles-typical.pbd* 파일에서 추적 프로그램 그룹을 구성해야 합니다. 이렇게 하면 서블릿 데이터가 수집됩니다.

추적 프로그램 그룹 중 하나는 해제하고 다른 하나는 설정합니다.

HTTP 서블릿 추적을 구성하려면

1. *<your-application-server-home>/wily/core/config/toggles-full.pbd* 파일을 찾아 엽니다.
2. PBD 의 *HTTP Servlets Configuration* 섹션으로 이동합니다.
3. 행 맨 앞에 파운드 기호를 추가하여 *HTTPServletTracing* 추적 프로그램 그룹을 **해제**합니다. 예:
   ```
   #TurnOn: HTTPServletTracing
   ```
4. 행 맨 앞에서 파운드 기호를 제거하여 *HTTPAppServerAutoProbeServletTracing* 추적 프로그램 그룹을 **설정**합니다. 예:
   ```
   TurnOn: HTTPAppServerAutoProbeServletTracing
   ```
5. *<your-app-server-home>/wily/core/config/toggles-typical.pbd* 파일에 대해 2~4 단계를 반복합니다.

AutoProbe 커넥터 파일 생성

더 이상 사용되지 않는 JVM AutoProbe 방법을 올바르게 사용하려면 커넥터 .jar 파일이 필요합니다. AutoProbe 커넥터를 생성하려면 다음 절차를 따르십시오. JVM 1.5 의 경우 JVM AutoProbe 의 지침을 따르십시오.

다음 단계를 따르십시오.

1. 작업 디렉터리를 설치 디렉터리 아래의 wily/connectors 로 변경합니다.
2. 다음 명령 중 하나를 사용하여 Create AutoProbe Connector(AutoProbe 커넥터 생성) 도구를 실행합니다.
   - 이 도구를 실행하는 JVM 을 사용하여 JVM 을 지정하려면 다음 명령을 사용하십시오.

     ```
     java -jar CreateAutoProbeConnector.jar -current
     ```
   - 명령줄에서 JVM 디렉터리를 전달하여 JVM 을 지정하려면 다음 명령을 사용하십시오.

     ```
     java -jar CreateAutoProbeConnector.jar -jvm <directory>
     ```

   출력은 wily/connectors/AutoProbeConnector.jar 형식의 파일입니다.
3. (선택 사항) 생성된 .jar 파일을 보다 관리하기 쉽고 일반적인 이름으로 변경합니다. 예를 들면 다음과 같이 지정합니다.
   - wily/connectors/AutoProbeConnector131_02_Sun.jar
   - wily/connectors/AutoProbeConnector130_IBM.jar

추가 정보:

HTTP 서블릿 추적 구성 (페이지 395)

JVM 용 AutoProbe 커넥터 실행

Sun 또는 IBM JVM 용 AutoProbe 커넥터를 생성한 후에는 생성된 파일을 실행하여 응용 프로그램을 계측합니다. 커넥터를 실행하는 방법은 사용하는 응용 프로그램 서버에 따라 달라집니다. 자세한 내용은 사용 중인 응용 프로그램 서버에 해당하는 단원을 참조하십시오.

SAP J2EE 6.20 용 AutoProbe 커넥터를 실행하려면

1. 다음 파일을 엽니다.
   <drive>:\usr\sap\<J2EE_ENGINE_ID>\j2ee\j2ee_<INSTANCE>\cluster\server\cmdline.properties
2. **JavaParameters** 섹션에 다음 명령을 추가합니다.
   ```
   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>
   -Dcom.wily.introscope.agent.agentName=<yourAgentName>
   ```
   예:
   ```
   Xbootclasspath/p:C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/connectors/AutoProbeConnector.jar;C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/Agent.jar
   -Dcom.wily.introscope.agentProfile=C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/core/config/IntroscopeAgent.profile
   ```
3. SAP 서버를 다시 시작합니다.

NetWeaver 04/SAP J2EE 6.40 용 AutoProbe 커넥터를 실행하려면

1. **SAP J2EE Configtool** 을 실행합니다.
2. 수정할 서버를 선택합니다.
3. **Java Parameters**(Java 매개 변수) 필드에 다음과 같은 새 java 매개 변수를 추가합니다.
   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>
   예:
   ```
   Xbootclasspath/p:D:/usr/sap/ccms/wily/connectors/AutoProbeConnector.jar;D:/usr/sap/ccms/wily/Agent.jar
   -Dcom.wily.introscope.agentProfile=D:/usr/sap/ccms/wily/core/config/IntroscopeAgent.profile
   ```
   참고: Windows 에서 실행되는 NetWeaver 6.40 의 경우 이러한 java 매개 변수의 슬래시는 정방향 슬래시여야 합니다.
4. **Disk**(디스크)를 클릭하여 저장합니다.
5. 각 서버에 대해 2~4 단계를 반복합니다.
6. SAP 서버를 다시 시작합니다.
7. Configtool 변경 사항이 적용되었는지 확인하려면 다음 파일을 엽니다.
   <drive>:\usr\sap\ccms\P66\JC00\j2ee\cluster\instance.properties
8. *ID<server_id>.JavaParameters* 로 시작하는 행을 찾아 입력한 행이 포함되어 있는지 확인합니다.

Sun ONE 용 AutoProbe 커넥터를 실행하려면

1. 관리자 또는 루트로 로그인합니다.

   Sun ONE 7.0 용 시작 스크립트에 Introscope 정보를 추가하려면 관리자 또는 루트 권한으로 로그인해야 합니다.

2. 다음 위치에 있는 server.xml 파일을 엽니다.
   <SunONE install dir>/domains/domain1/server1/config/

3. *server.xml* 파일에 다음 행을 추가합니다.

```
<jvm-options>
-Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar
</jvm-options>
```

항목 구분 기호는 콜론(:)입니다. 예:

```
<jvm-options>
-Xbootclasspath/p:/sw/sun/sunone7/wily/connectors/AutoProbeConnector.jar:/sw/sun/sunone7/wily/Agent.jar
</jvm-options>
```

Oracle 10g 용 AutoProbe 커넥터를 실행하려면

- AutoProbe 커넥터를 실행하려면 다음과 같이 부트스트랩 클래스 경로를 수정합니다.

```
-Xbootclasspath/p:wily/connectors/AutoProbeConnectorJar:PathToAgentJar
```

Windows 호스트에서 Sun JDK 1.42 를 사용하여 Oracle 10g 릴리스 2 를 실행하는 사용자는 ^(캐럿) 문자를 사용하여 슬래시를 이스케이프해야 합니다. 예:

```
-Xbootclasspath^/p:<IntroscopeAgent.jar path>
```

다른 버전의 WebLogic 에서는 다른 버전의 Java 를 사용하여 실행합니다. Java 1.4 이하 버전을 사용하는 경우에는 다음 단계에 따라 AutoProbe 커넥터를 실행하십시오. Java 1.5 이상을 사용하는 경우 자세한 내용은 JVM AutoProbe 를 참조하십시오.

WebLogic 용 AutoProbe 커넥터를 실행하려면

1. 다음 명령을 사용하여 생성한 *AutoProbeConnector.jar* 를 포함하도록 응용 프로그램 시작 스크립트(예: *startMedRecServer.cmd*)의 부트스트랩 클래스 경로를 편집합니다.

```
-Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar
```

스크립트 끝에서 *JAVA_VM* 및 *JAVA_OPTIONS* 다음에 있는 마지막 시작 명령에 -X 스위치를 추가합니다. 아래 예제에서는 스위치를 삽입할 올바른 위치를 보여 줍니다.

```
"$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}
-Xbootclasspath/p:${WL_HOME}/wily/connectors/AutoProbeConnector.jar:${WL_HOME
}/wily/Agent.jar
-Dweblogic.Name=${SERVER_NAME}
-Dweblogic.management.username=${WLS_USER}
-Dweblogic.management.password=${WLS_PW}
-Dweblogic.ProductionModeEnabled=${PRODUCTION_MODE}
-Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy"
weblogic.Server
```

2. 기본 부트스트랩 클래스 경로 이외의 부트스트랩 클래스 경로를 사용하는 경우 사용자 지정한 부트스트랩 클래스 경로의 시작 부분에 *Agent.jar* 및 *AutoProbeConnector.jar* 파일을 추가합니다.

JRockit JVM 이 포함된 WebLogic 용 AutoProbe 커넥터를 실행하려면

- JVM 을 시작할 때 다음 명령줄 옵션을 추가합니다.

```
-Xbootclasspath/a:<PathToAgentJar>
-Xmanagement:class=com.wily.introscope.api.jrockit.AutoProbeLoader
```

기타 응용 프로그램 서버용 AutoProbe 커넥터를 실행하려면

- AutoProbe 커넥터를 실행하려면 다음 명령을 사용하여 응용 프로그램 서버 부트스트랩 클래스 경로에 *Agent.jar* 및 AutoProbe 커넥터를 추가합니다.

```
-Xbootclasspath/p:wily/connectors/AutoProbeConnector.jar:PathToAgentJar
```

예제: Xbootclasspath 를 사용하여 WAS 계측

이 예제에서는 – Xbootclasspath 옵션을 사용하여 WebSphere Application Server 를 계측하는 방법을 보여 줍니다. 이 옵션을 사용하면 JVM 에 의해 기본적으로 부트스트랩 시간에 로드되는 엔터티(클래스, jar, 디렉터리, zip)를 무시할 수 있습니다. agent.jar 파일에 Xbootclasspath 를 직접 사용할 수 없기 때문에 계측하고자 하는 JVM 에 대한 AutoprobeConnector.jar 파일을 생성합니다.

다음 단계를 따르십시오.

1. WebSphere Application Server 가 사용하는 Java 실행 파일을 예를 들어 AppServer/java/jre/bin 아래와 같은 위치에서 찾습니다.
2. 명령 프롬프트를 열고 다음 명령을 입력합니다.

   ```
   cd <agent_install_dir>/wily/connectors
   <path_to_WAS>/AppServer/java/jre/bin/java –jar CreateAutoProbeConnector.jar
   –current
   ```

   AutoprobeConnector.jar 가 생성됩니다.
3. 다음 명령을 입력합니다.

   ```
   -Xbootclasspath/p:<path_to_created_Autoprobe_jar>/AutoprobeConnector.jar:<pat
   h_to_agent>/Agent[NoRedef].jar
   –Dcom.wily.introscope.agentProfile=<path_to_agent>/IntroscopeAgent[NoRedef].p
   rofile
   ```

   -Xbootclasspath/p:

   기본 부트스트랩 클래스 경로 앞에 추가할 쉼표로 구분된 디렉터리 경로, JAR 아카이브 및 ZIP 아카이브를 지정합니다. JVM 이 기본적으로 부트스트랩 시간에 로드하는 엔터티를 무시합니다.

   참고: UNIX 시스템에서는 Xbootclasspath 에 콜론(:)을 사용하고 Windows 시스템에서는 세미콜론(;)을 사용합니다.

수동 ProbeBuilder 실행 정보

ProbeBuilder 를 수동으로 실행하는 방법은 동적이 아닌 응용 프로그램 계측 방법입니다. ProbeBuilder 를 수동으로 실행하면 ProbeBuilder 는 응용 프로그램 서버가 실행되기 전에 디스크의 클래스를 계측합니다. 사용 중인 환경에서 AutoProbe 가 지원되지 않거나 AutoProbe 를 사용하지 않으려는 경우에 수동 ProbeBuilding 을 사용합니다.

수동 ProbeBuilding 은 다른 계측 방법과 함께 사용하면 **안 되며**, *최후의 수단으로 사용해야 합니다.*

수동 ProbeBuilding 에 대한 지침에서는 다음 설치 및 구성 작업을 이미 수행했다고 가정합니다.

1. Java Agent 설치. 자세한 내용은 Java Agent 설치를 참조하십시오.
2. Java Agent 연결 속성 구성. 자세한 내용은 Enterprise Manager 에 연결 (see page 68)을 참조하십시오.
3. Java Agent 이름 구성. 자세한 내용은 Java Agent 이름 지정 (see page 143)을 참조하십시오.
4. ProbeBuilder 의 옵션 구성. 자세한 내용은 AutoProbe 및 ProbeBuilding 옵션 (see page 87)을 참조하십시오.

AutoProbe for WebSphere for z/OS 구성

WebSphere 6.1 및 7.0 for z/OS 에 해당

z/OS 설치에서 AutoProbe 를 사용하여 응용 프로그램을 계측하도록 WebSphere 를 구성할 수 있습니다. AutoProbe 에 대한 자세한 내용은 AutoProbe 및 ProbeBuilding 옵션 (see page 87)을 참조하십시오.

참고: 다음 절차에 따라 WebSphere 7.0 for z/OS 를 계측할 경우 z/OS 의 WAS 7 을 위한 JVM 1.5 AutoProbe 방법을 사용할 때처럼 세부적인 메트릭을 얻을 수 없습니다 예를 들어 스레드 메트릭 수준은 계측되지 않습니다.

중요! Java Agent 9.0 이상을 사용하여 z/OS 에서 실행되는 WebSphere 7.0 을 모니터링하는 경우 응용 프로그램 서버 프로세스가 반복적으로 다시 시작될 수 있습니다. 이 문제를 방지하려면 WAS 7.0 빌드 수준 7.0.0.8 이상으로 업그레이드하십시오.

다음 단계를 따르십시오.

1. WebSphere 에서 "Administrator Console"(관리자 콘솔)을 시작합니다.
2. "Application Servers"(응용 프로그램 서버) > <*your server*> > "Process Definition"(프로세스 정의)을 선택합니다.

   "Control"(컨트롤) 및 "Servant"(보조) 항목이 나열됩니다.
3. "Servant"(보조)를 클릭한 다음 "JavaVirtualMachine"을 클릭합니다.

4. "Generic JVM Argument"(일반 JVM 인수) 필드를 설정하여 클래스 로더 플러그 인과 IntroscopeAgent.profile 파일의 위치를 지정합니다. 다음 중 하나를 설정합니다.

   ```
   com.wily.introscope.agentProfile
   ```

   또는

   ```
   com.wily.introscope.agentResource
   ```

   인수는 다음과 같은 값을 가집니다. 인수 하나에 여러 속성이 설정됩니다.

   ```
   -Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
   -Dcom.wily.introscope.agentProfile=<path to IntroscopeAgent.profile>
   ```

   또는

   ```
   -Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
   -Dcom.wily.introscope.agentResource=<path to Resource containing IntroscopeAgent.profile>
   ```

5. <*WebSphere Instance dir*>/lib/ext 디렉터리에 Agent.jar 파일을 배치합니다.

   참고: Agent.jar 파일을 WebSphere 설치 디렉터리에 배치하지 마십시오.

   아래에서는 잘못된 디렉터리와 올바른 디렉터리의 예를 보여 줍니다.

   ```
   잘못된 디렉터리: /usr/lpp/zWebSphere/V5R0M0/lib/ext
   올바른 디렉터리: /WebSphere/V5R0M0/AppServer/lib/ext
   ```

6. ./wily 디렉터리 내에 새로 생성된 모든 CA Introscope® 파일 및 디렉터리에 대한 읽기 권한이 WebSphere 프로세스에 있는지 확인합니다.

7. 모든 *.log 파일에 WebSphere 프로세스에 대한 쓰기 권한이 있는지 확인합니다. Java Agent 및 ProbeBuilder 는 이러한 파일을 ./wily 폴더에 씁니다. 다음과 같은 파일이 포함됩니다.

   - 모든 CA Introscope® 파일 및 디렉터리
   - <*WAS instance dir*>/lib/ext 내의 CA Introscope® 파일

8. WebSphere Application Server 를 다시 시작합니다.

9. WebSphere 에서 "Open for e-business"(e-비즈니스용으로 열기)라는 메시지가 표시되면 "Administrator Console"(관리자 콘솔)을 엽니다.

   메트릭의 보고가 시작됩니다.

10. Java2 보안이 사용되는 WebSphere 환경에서 AutoProbe 가 올바르게 실행되도록 하려면 Java2 보안 정책에 대한 사용 권한을 추가 (see page 59)하십시오.

11. HTTP 서블릿 추적을 구성 (see page 395)하여 서블릿 데이터를 수집합니다.

추가 정보:

AutoProbe 및 ProbeBuilding 옵션 (페이지 87)

부록 C: PBD Generator 사용

PBD Generator 도구를 사용하여 에이전트에서 사용할 사용자 지정 Java 클래스 파일을 계측할 수 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

CA PBD Generator 정보

PBD Generator 유틸리티는 Java 코드를 주석 처리하는 데 사용된 Javadoc 태그에서 PBD 파일을 생성하여 Java Agent 에서 사용할 사용자 지정 Java 클래스 파일의 계측을 용이하게 합니다.

이 생성기는 일련의 Java 소스 파일을 검사하고 Javadoc 태그 *@instrument* 가 포함된 클래스의 메서드를 계측합니다.

PBD Generator 도구를 사용하여 다음을 수행할 수 있습니다.

- PBD 파일을 자동으로 빌드하여 PBD 파일을 수동으로 생성할 때 발생할 수 있는 오류를 방지할 수 있습니다.
- PBD 생성을 빌드 시스템과 통합하여 PBD 파일을 자동으로 생성 및 업데이트하고 Java 소스의 변경 사항을 통합할 수 있습니다.

PBDGenerator.jar 파일을 사용하여 PBD Generator 를 Apache Ant 대상에 통합한 다음 Ant Javadoc 작업으로 실행하여 PBD Generator 를 구성합니다.

PBD Generator 구성

이 도구는 Ant 대상의 Javadoc 작업으로 Ant 기반 빌드 시스템에 통합할 수 있습니다.

이 샘플 Javadoc 작업에서는 Ant 에서 이 도구를 사용하는 방법을 보여 줍니다.

```
<javadoc sourcepath="/src/engineering/products/introscope/source"
    destdir="/src/engineering/products/introscope/source/generatedpbd"
    maxmemory="512m"
    packagenames="com.wily.introscope.console.thornhill.ui.util"
    verbose="false"
    private="true">
  <doclet name="com.wily.util.build.javadoc.PBDInstrumentDoclet"
    path="/Wily/tools/WilyPBDGenerator.jar">
    <param name="-d"
    value="/src/engineering/products/introscope/source/generatedpbd"/>
  </doclet>
</javadoc>
```

PBD Generator 매개 변수가 필요합니다.

다음과 같은 주요 PBD Generator 매개 변수가 필요합니다.

sourcepath

Java 소스 트리의 루트 디렉터리

destdir

도구에서 출력되는 PBD 파일의 디렉터리 경로

packagenames

계측에 대해 검사될 Java 패키지의 쉼표로 구분된 목록

doclet path

이 도구를 포함하고 있는 PBD Generator jar 파일을 찾을 경로

param name="-d"

여기에는 *destdir* 와 같은 값이 포함되어 있어야 합니다.

PBD Generator 사용

PBD Generator 를 사용하기 전에 특수 Javadoc 태그를 계측할 Java 소스 파일에 삽입하십시오.

JavaDoc 태그의 구문은 다음과 같습니다.

```
@instrument <valid metric prefix> <optional tracer name>
```

다음은 각 요소에 대한 설명입니다.

<valid metric prefix> - 유효한 Introscope 메트릭 접두사로, 콜론(:) 문자를 포함하지 않는 문자열입니다. 파이프(|) 문자는 사용할 수 있습니다.

<optional tracer name> - BlamePointTracer, FrontendMarker 또는 BackendMarker 일 수 있습니다. 추적 프로그램 이름이 누락된 경우 기본값은 BlamePointTracer 입니다.

부록 D: 네트워크 인터페이스 유틸리티 사용

네트워크 인터페이스 유틸리티를 사용하여 Catalyst 통합을 위해 에이전트에서 사용되는 호스트 컴퓨터의 네트워크 인터페이스 이름 값을 확인할 수 있습니다.

이 섹션은 다음 항목을 포함하고 있습니다.

네트워크 인터페이스 이름 확인 (페이지 409)

네트워크 인터페이스 이름 확인

네트워크 인터페이스 유틸리티는 introscope.agent.primary.net.interface.name 속성에 대한 이름 및 하위 인터페이스 값을 제공합니다. 에이전트에서 사용되는 것과 동일한 JVM 및 응용 프로그램 서버에서 이 유틸리티를 실행하십시오.

다음 단계를 따르십시오.

1. 명령줄에서 다음 디렉터리로 이동합니다.

   <Agent_Home>/wily/tools

2. 다음 명령을 실행하여 유틸리티를 호출합니다.

   ```
   java -jar NetInterface.jar
   ```

   브라우저의 "네트워크 인터페이스" 탭에 Java 에서 지원되는 네트워크 인터페이스 이름의 목록이 표시됩니다.

추가 정보:

사용 가능한 네트워크 목록 구성 (페이지 280)