CA Application Performance Management

Java エージェント実装ガイド

リリース 9.5

CA Technologies 製品リファレンス

このドキュメントは、以下の CA Technologies 製品および機能に関するものです。

- 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 への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

目次

第 1 章: Java Agent の紹介

このセクションには、以下のトピックが含まれています。

Introscope および Java Agent について

CA Introscope® は、実運用環境にある複雑な Web アプリケーションを常時監視し、顧客に影響を与える前に問題を検出して、すばやく連携的に解決するためのエンタープライズ アプリケーション パフォーマンス管理ソリューションです。 このソリューションのアーキテクチャ主要部分が、オーバーヘッドの低いエージェントです。

エージェントは、Introscope のデータ収集コンポーネントで、トランザクションを実行するときにアプリケーションおよびコンピューティング環境に関する詳細なパフォーマンス情報を収集します。 Java Agent は、Java 仮想マシン（JVM）上で実行されるアプリケーションとリソースからこのような情報を収集し、それを追加処理の目的で Enterprise Manager に送信します。

Java Agent は、アプリケーションが使用する JVM を構成するコンポーネントのバイトコードに*プローブ*を挿入します。 バイトコードへのプローブの挿入は、アプリケーションの監視を有効にする*インスツルメンテーション*プロセスの一部です。

アプリケーションのインスツルメントには、ProbeBuilder ディレクティブ（PBD）ファイルで定義されるトレーサも必要です。 PBD ファイル内の指示またはディレクティブは、監視対象のアプリケーション コンポーネントを識別します。 トレーサは、アプリケーションが JVM 内で実行される際にエージェントが収集するメトリックと収集元のプローブを識別します。 ユーザ環境に合わせて PBD ファイルを変更することで、監視対象を制御できます。

Java エージェントをインストールする場合は、複数のデフォルト PBD ファイルをデプロイして、環境のデフォルトの監視が有効になるようにします。必要とされる可視性とパフォーマンスをバランスよく実現するため、デフォルト監視は変更することができます。 アプリケーションがインスツルメントされた後、Java Agent は対象のデータを収集し、そのデータを Enterprise Manager へレポートします。 Enterprise Manager はデータを受け取ると、リアルタイムおよび履歴レポートのためにデータを処理して保存します。 その後で、Introscope Workstation を使用して、収集したデータを表示および操作して、アラートを作成したり、応答アクションを実行したりできます。

アプリケーション管理における主要アクティビティは以下のとおりです。

- アプリケーション サーバのパフォーマンスと可用性を監視するためのエージェントのデプロイ
- アプリケーション コンポーネントの監視をテスト、調整、および最適化します。
- 必要に応じて、エージェント プロファイルをカスタマイズして、エージェント オペレーションを制御します。
- アプリケーションまたはエージェント固有のメトリック グループ、ダッシュボード、アラート、およびアクションの作成
- アプリケーションに関する問題の調査、切り分け、および診断

Java Agent のデプロイ計画

デプロイを計画する場合、目標はエージェント オーバーヘッドとアプリケーション パフォーマンスの可視性との間で適切なバランスをとることです。 オーバーヘッドの低いエージェントの場合、実運用環境内の全トランザクションをリアルタイム監視することが可能になります。 オーバーヘッドを低く維持することは、クリティカルなアプリケーションおよびサーバ リソースのパフォーマンスおよび可用性に役立ちます。しかし、オーバーヘッドを低くしただけでは、問題が発生した場合にそれを診断するための十分な情報が得られません。 そのため、アプリケーションのライフサイクル全体にわたってエージェント設定をデプロイして調整することが一般的です。 また、アプリケーションの開発中またはテスト中は監視するコンポーネントを増やし、実運用環境へのリリース後はコンポーネントを減らします。

デフォルト機能のインストールおよび評価

エージェントのデプロイの最初の手順は、デフォルトのエージェント設定をインストールおよび評価することです。 デフォルトのエージェント設定では、アプリケーションおよびコンピューティング環境の多くの共通コンポーネントのデータ収集が実行されます。 デフォルトのエージェント設定には、有効にされている複数の機能や、利用頻度の低い、無効にされている機能なども含まれます。

目標は、デフォルトで実現できるデータ収集の深さと広さを評価し、Introscope に関する知識を深め、アプリケーションの監視方法を習得することです。 エージェントによってデフォルトで提供されるパフォーマンス メトリックに慣れてきたら、必要に応じて、データを収集するエージェントをカスタマイズできます。

デフォルトのパフォーマンスを評価する際には、エージェントがより多くのメトリックを収集すると、より多くのシステム リソースが消費されることに注意してください。 エージェントが収集するメトリックが少ないと、潜在的な問題の可視性が低下します。 エージェント設定を調整するときには、データ収集の深度と、許容可能なパフォーマンス レベルの間でバランスをとるようにしてください。 インスツルメンテーションの適切なレベルは、一般的に、エージェントをデプロイする場所に依存します。たとえば、テスト環境内のエージェント監視は、通常、多くのメトリックを収集するように設定されます。 ただし、実運用サーバ上のエージェントは、最重要情報を提供するように通常は設定されます。

設定要件の決定

エージェントをデプロイする前に、データ収集要件を決定します。 この情報は、エージェントのデータ収集動作を調整し、エージェントの代替設定を使用してオーバーヘッドに与える影響の評価に役立ちます。

Introscope は、アプリケーションのライフサイクルの全体にわたって使用できます。 たとえば、開発からテスト、ロードの検証、ステージング、および本番稼働です。 ライフサイクルの各段階では、監視目標、環境の制約、および期待されるサービス レベルが異なることがよくあります。これらの違いを解決するには、監視する環境のタイプに応じて異なる動作をするようにエージェントを設定します。

ユーザの目標は、パフォーマンス詳細の可視性とリソース オーバーヘッドの間のトレードオフを適切に判断することです。 また、監視対象の環境に対し、妥当なコストで可視性の最適なレベルを検討します。

開発などの実運用前のアプリケーション環境では、通常はデータ収集のレベルを上げてアプリケーションのパフォーマンスに関する詳細な可視性を得るようにします。実運用、または大量のトランザクション環境では、通常、レポートされるメトリックを減らして、エージェントのオーバーヘッドを制御します。 また、要件に応じて、特定のエージェントの動作を制御するためにエージェント プロパティを設定できます。 たとえば、収集されたメトリックの最大数や古いメトリックの削除を追跡します。

環境に対して、可視性の適切なレベルと許容可能なパフォーマンス オーバーヘッドを決定し、要件を満たすようにエージェントを設定できます。

適切な設定プロパティを使用したベースライン エージェント プロファイルの定義

監視するアプリケーション環境のタイプを特定した後、エージェント設定の「候補」を作成できます。 ほとんどのエージェント オペレーションはエージェント プロファイル（IntroscopeAgent.profile）内のプロパティを使用して制御します。たとえば、IntroscopeAgent.profile ファイルは、エージェントが使用する ProbeBuilder ディレクティブ ファイルおよび ProbeBuilder リスト ファイルを定義します。 エージェント プロファイルに登録されるファイルは、そのエージェントが収集する特定のメトリックを制御します。IntroscopeAgent.profile ファイルは、特定の機能の有効化または無効化、またはポーリング間隔などのオペレーションを調整するプロパティを提供します。

構成および環境に応じて、エージェント プロファイル内のプロパティを調節して、変更の影響をそれぞれ評価できます。たとえば、ChangeDetector を有効にしていないデフォルトのエージェント プロファイルから開始します。その後、プロファイル内の ChangeDetector プロパティを有効にし、変更の後、その他の機能を追加する前に、エージェントのパフォーマンスを評価することができます。

エージェントのパフォーマンス上のオーバーヘッドの評価

エージェントの構成を評価する際には、収集されるメトリックからアプリケーションのパフォーマンスと可用性に関して十分な可視性が得られるかどうかを検証します。 また、メトリックの量が運用環境に許容できない負荷をかけないかどうかを検証します。 エージェントによるレポートは、パフォーマンスおよび可用性に関する問題の特定および限定に必要とされるメトリックの量を超えないようにする必要があります。

アプリケーションのパフォーマンス特性を理解することにより、エージェントのパフォーマンスを有効に評価できます。 たとえば、デフォルトのモニタリングを実装する前後に、影響を検証するためにアプリケーションの負荷テストを実施できます。

管理された方法でデータ収集を拡張するには、変更の実装前後のエージェントのオペレーションおよびオーバーヘッドを確認します。 たとえば、一度に 1 つのアプリケーションの監視サポートだけを追加し、各アドオンのパフォーマンスを評価します。

エージェントの設定の検証およびデプロイ

候補となるエージェント設定を使用した際に必要な可視性が提供されることを検証してから、その環境に設定をデプロイします。

検証済みの設定をデプロイするには、以下の設定項目をターゲット環境にインストールします。

- IntroscopeAgent.profile ファイル
- 変更済みまたはカスタムの PBD ファイル

Java Agent のデプロイ

エージェントのデプロイには、以下の概略手順が含まれます。

1. ターゲット コンピュータにエージェントをインストールします。
2. アプリケーションをインスツルメントするため、アプリケーション サーバを設定します (P. 43)。

3. エージェントおよびエージェント プロファイルの場所を含めるため、アプリケーション サーバの起動スクリプトを設定する (P. 43) か、または Java コマンドを設定します。
4. Enterprise Manager へのエージェント接続を設定します (P. 71)。
5. アプリケーションをインスツルメントし、データ収集を開始するため、アプリケーション サーバを再起動します。
6. インストール中に設定したプロパティを変更したい場合や、データ収集またはその他のエージェント オペレーションを変更したい場合は、IntroscopeAgent.profile ファイルを設定します (P. 81)。

第 2 章: Java Agent のインストールおよび設定

このセクションには、以下のトピックが含まれています。

エージェントをインストールする前に

エージェントをインストールする前に、以下の手順に従います。

1. Java Agent デプロイ プロセスを確認します。
2. 以下のサポート対象バージョンのアプリケーション サーバがインストールされていることを確認します。

   注: エージェントのインストール先となるサーバには、ローカルで使用できるサポート対象バージョンの JVM がインストールされている必要があります。 アプリケーション サーバおよび JVM の要件については、「*Compatibility Guide*」を参照してください。
3. JVM のサポートされているバージョンがあることを確認します。 サポートされているバージョンを使用できない場合は、旧バージョンを使用してください。
   - Java 1.4.x の場合は 8.x Java エージェントを使用。
   - Java 1.3.x の場合は 7.2 Java エージェントを使用。

4. Introscope Enterprise Manager および Workstation コンポーネントがインストールされていることを確認します。エージェントとの接続に使用する Enterprise Manager のホスト名およびポート番号を確認します。

   エージェントと Enterprise Manager の間の接続状況は ping または telnet を使用して確認できます。

注: Enterprise Manager、Workstation、および WebView コンポーネントのインストールについては、「CA APM インストールおよびアップグレード ガイド」を参照してください。

詳細:

Java Agent のデプロイ計画 (P. 22)
Java Agent のデプロイ (P. 25)

Java エージェントをインストールする方法の選択

以下のいずれかの方法で Java エージェントをインストールすることができます。

- グラフィカル ユーザ インターフェース（GUI）またはテキスト ベースのコンソール インストーラを使用する対話方式
- 対話方式を使用せずに、編集済みの応答ファイルを使用するサイレント方式
- 個々のアプリケーション サーバのファイルを抽出および設定する手動方式

コンピュータにローカルでファイルをインストールする際には、ほとんどの場合対話方式のインストーラを使用します。 対話方式でインストールする場合、表示されるプロンプトは、グラフィカル インターフェースでも、テキストベース インターフェースでも同じです。 ただし、GUI またはテキストベースのインストーラを選択するオプションは、インストーラを実行するオペレーティング システムによって異なります。 たとえば、ほとんどの UNIX 環境では GUI またはテキストベースのコンソール インストーラをサポートしていますが、デフォルトではコンソール インストーラが開始されます。

コンピュータにリモートでファイルをインストールするか、事前に設定されたインストール手順に従ってデプロイする場合は、編集済みの応答ファイルを使用して、サイレント インストーラを実行できます。

対話方式またはサイレント方式のインストーラを使用しない場合は、インストール アーカイブを使用して手動でエージェントのインストールおよび設定を行うことができます。 手動インストールでは、アプリケーション サーバおよびオペレーティング システムに固有のアーカイブからエージェント ファイルの特定のセットを抽出し、展開オプションを手動で設定できます。 CA Technologies では、同じ種類のアプリケーション サーバおよびオペレーティング システムを共有する複数システムへの Java エージェントのデプロイを迅速に行うために、これらのアーカイブを提供しています。

対話方式、サイレント方式、手動でのインストールの詳細については、それぞれのセクションを参照してください。

対話モードでの Java Agent のインストール

お使いのオペレーティング システムに適した Java エージェント インストーラを選択してから、そのインストーラを起動し、プロンプトに応答することによって、Java エージェントを対話モードでインストールできます。GUI インストーラを使用する場合は、ドロップダウン メニューやチェックボックスを使用して、選択を行うことができます。 テキスト ベースのコンソール インストーラでは、テキストを入力することによって選択を行います。

以下の手順に従います。

1. お使いのオペレーティング システムに適したインストール アーカイブを選択します。 例：
   - Microsoft Windows にインストールする場合： IntroscopeAgentInstaller<*バージョン*>windows.zip
   - サポート対象の UNIX または Linux オペレーティング システムにインストールする場合： IntroscopeAgentInstaller<*バージョン*>unix.tar
   - IBM z/OS にインストールする場合： IntroscopeAgentInstaller<*バージョン*>zOS.tar
   - IBM OS/400 （IBM i オペレーティング システム）にインストールする場合： IntroscopeAgentInstaller<*バージョン*>os400.zip

2. オペレーティング システムに適したコマンドを使用して、インストール アーカイブ ファイルを解凍します。たとえば、UNIX または Linux 上では以下のように入力します。
   `tar –xvf IntroscopeAgentInstaller<バージョン>unix.tar`
3. プロンプトに従ってインストールを開始します。
4. アプリケーション サーバのルート ディレクトリの場所を指定します。

   アプリケーション サーバ ルート ディレクトリの場所を指定しない場合は、デフォルトの応答（Windows は C:¥、UNIX は /）をそのまま使用します。
5. 有効なアプリケーション サーバのリストから、アプリケーション サーバのタイプを選択します。

   選択するアプリケーション サーバによって、利用できる追加の監視オプションが決まります。
6. Java Agent のルート インストール ディレクトリを指定します。ほとんどの場合、アプリケーション サーバ ルート ディレクトリを使用します。

   インストーラによって、ルート インストール ディレクトリ内に「wily」ディレクトリが作成されます。このディレクトリが <Agent_Home> ディレクトリになります。
7. エージェント プロファイルを作成するか、または既存のエージェント プロファイルを使用するかを指定します。

   エージェント プロファイルを作成する場合は、以下を選択するよう求められます。

   - 標準またはフル インスツルメンテーション
   - エージェント名およびプロセス名
   - Enterprise Manager のホスト名およびポート番号

   既存のプロファイルを選択する場合は、ファイルの場所を入力するよう求められます。 ファイルの完全修飾パスを指定します。

   指定する設定は必要に応じてインストール後に変更できます。その場合は、IntroscopeAgent.profile ファイルを編集します。
8. アプリケーション サーバを起動するために使用する Java 実行可能ファイルへのパスを指定します。
9. アプリケーションをインスツルメントするための ProbeBuilding を指定します。 ほとんどの場合、JVM AutoProbe を使用します。

10. ChangeDetector エージェント拡張機能を有効にするかどうかを指定します。

    ChangeDetector を有効にする場合は、ChangeDetector エージェントの名前を指定するように求められます。

    ChangeDetector を有効にしない場合、ChangeDetector のファイルは *<Agent_Home>*/examples ディレクトリにインストールされます。 後で ChangeDetector を有効にする場合は、*<Agent_Home>*/core/ext ディレクトリにファイルをコピーし、IntroscopeAgent.profile ファイルを変更します。

    注: ChangeDetector の詳細については、「CA APM ChangeDetector ガイド」を参照してください。

11. インストールする追加の監視オプションを選択します。 たとえば、Web サービスおよびその他の SOA 環境コンポーネントを監視する場合は、CA APM for SOA を選択します。

    インストール中にほかの監視オプションを有効にしない場合、関連ファイルは *<Agent_Home>*/examples ディレクトリにインストールされ、後で有効にすることができます。

12. エージェントと共にインストールするアドオンが含まれる .zip または .tar ファイル用の「ピックアップ フォルダ」の場所を指定します。

    指定された場所に .zip または .tar ファイルがある場合、*<Agent_Home>* ディレクトリに解凍されます。

13. 選択内容を確認し、エージェントのインストールを続行して、インストールを完了します。

注: エージェントのインストールによってアプリケーション サーバが停止または起動することはありません。また、アプリケーション サーバの起動スクリプトが設定されることもありません。 インストール後にこれらのタスクを手動で実行します。具体的な手順は、監視対象のアプリケーション サーバのタイプによって異なります。

サイレント モードでの Java Agent のインストール

サイレント モードでは、コマンド ラインからエージェント インストーラを起動し、インストールの指示が含まれる応答ファイルを指定します。開始後は、インストールはバックグランウンドで実行され、進捗状況などは一切表示されません。 このインストール方法では、操作しなくてもエージェントをインストールできるので、リモート コンピュータにエージェントをインストールしたり、同一の設定内容で複数のエージェントをインストールしたりするのに使用するのが最も一般的です。

過去に Java Agent をインストールしたことがある場合は、自動生成された応答ファイルを使用して追加エージェントをインストールすることもできます。または、インストール用のファイルに含まれるサンプル応答ファイルを手動で編集するか、テキスト エディタを使用して独自の応答ファイルを作成することができます。

自動生成応答ファイル

Java エージェント インストーラを（対話形式またはサイレント方式で）実行するたびに、ユーザが選択したインストール オプションを記録した応答ファイルがインストーラによって作成されます。 この自動的に生成される応答ファイルは、*<Agent_Home>/install* ディレクトリに保存されます。ファイル名は、インストーラが応答ファイルを作成した日付と時間を以下の形式で表します。

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

たとえば、インストール プロセスの完了日時が 2005 年 4 月 30 日午前 7 時 10 分 5 秒の場合、自動生成応答ファイルの名前は以下のようになります。

```
autogenerated.responsefile.2005.4.30.7.10.05
```

この自動生成応答ファイルは、それ以降のサイレント モード インストールにおいて、同じ設定で使用したり、編集して新しい設定で使用したりできます。

サンプル応答ファイル

Java Agent を初めてインストールする場合、または以前のインストールで使用したオプション設定ではなくデフォルトの設定を使用する場合は、Java Agent インストーラに含まれるデフォルトのサンプル応答ファイルを編集できます。デフォルト サンプル応答ファイルは *<Agent_Home>*¥install ディレクトリ内に以下の名前で置かれています。

`SampleResponseFile.Agent.txt`

サンプル応答ファイルでは、ほとんどのプロパティのデフォルト設定が指定されていますが、サイレント インストールで使用するには手動でファイルを編集する必要があります。

応答ファイルのプロパティ設定およびエージェントのインストール

自動生成応答ファイルまたはデフォルト サンプル応答ファイルを使用する場合、またはカスタム応答ファイルを作成する場合のいずれの場合でも、サイレント モードで Java Agent インストーラを呼び出す前にファイル内のプロパティを適切に設定する必要があります。 応答ファイルで設定するプロパティは、インストーラを対話方式で実行するときの選択内容と同じものです。

注: 任意のプロパティの設定方法の詳細については、*<Agent_Home>*/install/SampleResponse.Agent.txt ファイル内のコメントを参照してください。

以下の手順に従います。

1. テキスト エディタで以下の応答ファイルを開きます。
2. 適切なプロパティ値を設定します。設定するプロパティは以下のとおりです。

   USER_INSTALL_DIR=*<ルート インストール ディレクトリ>*

   エージェントをインストールするディレクトリを指定します。 ほとんどの場合、アプリケーション サーバ ルート ディレクトリを使用してください。

   silentInstallChosenFeatures=Agent

   インストールするコンポーネントを指定します。

appServer=Default

監視するアプリケーション サーバのタイプを指定します。 有効な値は、Default、JBoss、Tomcat、WebLogic、WebSphere、Sun、Oracle、または Interstage です。 値は大文字と小文字が区別されます。

この設定は、エージェントと共にインストールする ProbeBuilder ディレクティブ（PBD）と、有効にする追加の監視オプションを制御します。

（オプション）appServerHome=

アプリケーション サーバのホーム ディレクトリを指定します。 USER_INSTALL_DIR プロパティをアプリケーション サーバのルート ディレクトリに設定した場合、このプロパティは必要ありません。

（オプション）appServerJavaExecutable=

アプリケーション サーバを起動するために使用する Java 実行可能ファイルへのパスを指定します。

instrumentationLevel=Typical

フルまたは標準インスツルメンテーションのどちらを使用するかを指定します。 値は大文字と小文字が区別されます。

agentName=Default Agent

Workstation 内に表示するエージェント名を指定します。

processName=Default Process

Workstation 内に表示するプロセス名を指定します。

emHost=localhost

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータのホスト名を指定します。

emPort=5001

エージェントが Enterprise Manager への接続に使用するポート番号を指定します。

（オプション）alternateAgentProfile=

既存のエージェント プロファイルへの絶対パスを指定します。

（オプション）pickupFolder=

エージェントと共にインストールする任意のアドオンの .zip または .tar ファイルを含む「ピックアップ フォルダ」の絶対パスを指定します。

（**オプション**）changeDetectorEnable=false

ChangeDetector エージェント拡張機能を有効にするかどうかを指定します。このプロパティを false に指定すると、ChangeDetector のファイルはインストールされますが、有効にはなりません。 これは後から有効にすることができます。

（**オプション**）changeDetectorAgentID=

ChangeDetector エージェント拡張機能の名前を指定します。ChangeDetector を有効にする場合は、コメントを解除してこのプロパティを設定します。

（**オプション**）shouldEnable*

有効にする追加の CA APM 監視ソリューションを指定します。 オプションの CA APM 監視ソリューションのプロパティはすべてデフォルトで false に設定されます。 有効にすることができるオプションは、アプリケーション サーバのタイプによって異なります。

3. 応答ファイルを保存して、テキスト エディタを閉じます。
4. インストーラの実行可能ファイルのパスと応答ファイルの絶対パスを指定して、インストーラをサイレント モードで起動します。
   `<インストーラへのパス> -f <応答ファイルへの絶対パス>`

   インストーラの起動に使用するコマンドは、コマンドを実行するオペレーティング システムによって異なります。

   たとえば、Windows ではコマンド形式は次のようになります。
   `IntroscopeAgent<バージョン>windows.exe -f C:¥temp¥myResponseFile.txt`

   Linux または UNIX では、コマンド形式は次のようになります。
   `./IntroscopeAgent<バージョン>unix.bin -f /`

   お使いのオペレーティング システムに適したコマンド形式を選択してください。
5. *<Agent_Home>/install/Introscope_Agent_<バージョン>_InstallLog.log* ファイルを参照して、エージェントが正常にインストールされたことを確認します。

インストール アーカイブを使用した手動インストール

Java Agent インストーラを対話モードで実行したり、応答ファイルを設定したりしなくても、エージェント ファイルはシステム上に配置できます。アプリケーション サーバ固有のアーカイブを使用すれば、エージェントをインストールできます。

インストール アーカイブには、エージェント インストーラを実行した場合にインストールされるファイルがすべて含まれています。 アーカイブをコンピュータにコピーしてから展開し、エージェント プロファイルで接続先の Enterprise Manager を指定し、その他のプロパティの設定も行います。 これらのファイルを使用して複数のエージェントをバッチ ジョブでデプロイするか、またはこれらのファイルをエージェント ファイルのデフォルト セットのアーカイブとして保存します。

アーカイブから Java Agent を手動でインストールする場合は、インストール先のコンピュータに 35 MB の空きディスク領域があることを確認します。

以下の手順に従います。

1. アプリケーション サーバとオペレーティング システムに適したインストール アーカイブを選択します。
2. お使いのオペレーティング システムに適したコマンドを使用して、アーカイブの中身を JVM がアクセス可能な場所に解凍します。たとえば、UNIX または Linux 上では以下のように入力します。
   `tar -xvf IntroscopeAgentFilesOnly-NoInstaller<バージョン><app_server>.<os>.tar`
3. *<Agent_Home>*/core/config/IntroscopeAgent.profile ファイルをテキスト エディタで開き、Enterprise Manager への接続を設定します。

   Enterprise Manager との通信を有効にするプロパティの設定方法の詳細については、「Enterprise Manager への接続設定 (P. 71)」を参照してください。
4. 任意の追加プロパティを設定し、IntroscopeAgent.profile ファイルを保存して閉じます。
5. Java Agent 起動ファイルおよびエージェント プロファイルの場所を使って、アプリケーション サーバを設定します。
6. アプリケーション サーバを再起動します。

Java Agent のディレクトリ構造について

エージェントのインストール時には、以下のディレクトリ構造がルート インストール ディレクトリに作成されます。

wily

このディレクトリは *<Agent_Home>* ディレクトリであり、エージェントの起動に使用する Agent.jar が含まれます。

<Agent_Home> ディレクトリのサブディレクトリでは、Java エージェントのさまざまな機能を有効にするライブラリと拡張ファイルが提供されています。

core

- config

  ここには、IntroscopeAgent.profile、ProbeBuilder ディレクティブ ファイル（.pbd）、および ProbeBuilder リスト ファイル（.pbl）が含まれます。これらのファイルにより、エージェントのオペレーション、メトリック データの収集、およびインスツルメンテーション処理が制御されます。 IntroscopeAgent.profile 内でどのプロパティが定義されているか、またどの .pbd および .pbl ファイルがデプロイされエージェント プロファイルで参照されているかは、インストール時の選択内容に応じて異なります。

  config ディレクトリ内の hotdeploy サブディレクトリを使用すると、IntroscopeAgent.profile の編集やアプリケーションの再起動を行わなくても、新しいディレクティブをデプロイできます。 hotdeploy ディレクトリ内のファイルに無効な構文が含まれていたり、メトリックが過剰に含まれていると、インスツルメンテーションの失敗やアプリケーション パフォーマンスの低下につながることがあります。

- ext

  有効化されたエージェント拡張機能やその他の機能のファイルを含んでいます。たとえば、このディレクトリには、アプリケーション問題切り分けマップや LeakHunter のファイルが含まれます。

connectors

特定環境でのインスツルメンテーションを可能にする AutoProbe コネクタの設定に使用する CreateAutoProbeConnector.jar ユーティリティが含まれます。

common

拡張機能用の設定ファイルとプロパティ ファイルが含まれます。

examples

CA APM for SOA などのオプションのエージェント拡張機能のフォルダとファイルが含まれます。 インストール時に拡張機能を有効にしなかった場合、後でこのディレクトリ内のファイルを使用して、拡張機能を設定できます。

install

インストール プロセスを記録したログ ファイルと、サイレント インストールで使用できるファイルが含まれます。 たとえば、生成される応答ファイルや SampleResponseFile.Agent.txt ファイルは、このディレクトリに置かれます。

インストール アーカイブから手動で Java Agent をインストールする場合、このディレクトリは作成されません。

logs

エージェントのログ ファイルが格納されます。

tools

以下のコンテンツを保持します。

- Web サーバのログ ファイルを分析する URLGrouper.jar コマンド ライン ユーティリティ。
- 拡張機能のファイルのリスト。 たとえば、configurePMI.bat、CreateIU.jar、listServers.bat、MergeUtility.jar、NetInterface.jar、setPmiModules.jacl ファイルは、このディレクトリにあります。
- TagScript ツール ファイル： TagScript.jar、TagScript.bat、および TagScript.sh コマンドライン スクリプト。

UninstallerData

エージェントと関連リソースのアンインストールに使用する実行可能ファイルと関連リソースが入ったサブディレクトリが含まれます。

インストール アーカイブから手動で Java Agent をインストールする場合、このディレクトリは作成されません。

version

オプションの CA APM 監視ソリューションのバージョン情報が含まれます。 この情報は、インストール時に何を有効にするように選択したかに関わりなくインストールされます。

合成トランザクションの検出の設定

合成トランザクションの監視設定は、introscope.agent.synthetic.header.names パラメータを使用して行います。

introscope.agent.synthetic.header.names パラメータの値には、監視対象の HTTP 要求が合成トランザクションの一部かどうかを判断するために使用する HTTP ヘッダ パラメータをリストします。個々のパラメータ名はカンマで区切ります。 このパラメータが未定義、または値が空の場合、合成トランザクションは検出されません。複数の HTTP ヘッダ パラメータ名が定義されている場合、指定された順に検査されます。値を持つ最初の HTTP パラメータは、合成トランザクションを定義するために使用されます。

合成トランザクションがレポートされるノードは、以下のように、各トランザクションの検出に使用される特定の HTTP ヘッダ パラメータに応じて異なります。

- パラメータ値が *lisaframeid* または *x-wtg-info* 以外の場合は、HTTP パラメータの値自体がノード名として使用されます。有効なノード名が使用されるよう、適切な変更を行います。
- パラメータ値が *lisaframeid* の場合、合成ノード名は CA LISA になります。
- パラメータ値が *x-wtg-info* の場合、HTTP ヘッダ パラメータの値には名前と値の組み合わせが使用されます。 ペアは、アンパサンド記号で区切られます。 各ペアの属性名と値は、等号で区切られます。 合成トランザクション ノード名は、*group*、*name*、*ipaddress*、*request_id* の値とノード区切り記号（|）で構成されます。

たとえば、以下のパラメータについて考えてみます。

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

以下の *x-wtg-info* ヘッダが、［SampleGroup|sample|192.168.193.1|start］ノードの下のメトリックにレポートされます。

```
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 ヘッダ パラメータ値で定義されていない属性のデフォルト値は、以下のとおりです。

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

introscope.agent.synthetic.header.names が定義されていない場合、以下の設定パラメータは無視されます。

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

トランザクションが合成と認識されたノードにレポートされます。このノードは *Frontends|Apps|<WebAppName>* の下にあります。ここで *<WebAppName>* は Web アプリケーション名です。 この値のデフォルトは、Synthetic Users です。

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

トランザクションが合成と認識されていないノードにレポートされます。このノードは *Frontends|Apps|<WebAppName>* の下にあります。ここで *<WebAppName>* は Web アプリケーション名です。 定義されていない場合、*<WebAppName>* の下に追加のノードは作成されません。

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

値が合成ユーザ名として使用されている HTTP ヘッダ パラメータの名前です。 合成ユーザ名は、異なる合成トランザクションを区切るために使用されます。 各合成ユーザ名のノードは、*Synthetic User* ノードの下に作成されます。この設定パラメータが定義され、この名前の HTTP ヘッダ パラメータが存在する場合、合成トランザクション メトリックがレポートされます。 トランザクションがレポートされるノードは、*<Synthetic Users>|<Synthetic User>* です。

- *<Synthetic Users>* ノード名は、*introscope.agent.synthetic.node.name* 設定パラメータによって決定されます。
- *<Synthetic User>* ノード名は、HTTP ヘッダ パラメータ値によって決定されます。

注: これらのプロパティへの変更はただちに有効になります。管理対象アプリケーションを再起動する必要はありません。

TagScript ユーティリティの使用

CA TagScript ユーティリティは、合成ユーザ情報の抽出を指定するために HP Vugen と共に使用できます。

TagScript ユーティリティを使用する方法

1. TagScript ユーティリティを開きます。

   Windows の場合

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

   UNIX の場合

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

   どの環境のスクリプトを変更するかを確認する画面が表示されます。

2. 以下のいずれかのオプションを選択します。
   - Performance Testing - HP Loadrunner スクリプト
   - Production - HP Business Process Monitor または Sitescope スクリプト
   - Un-tag - タグ付け処理を元に戻します
3. HP Vugen スクリプトがあるディレクトリに移動します。各 .c スクリプトをダブルクリックして開きます。

   HP Vugen .c スクリプト ファイルがすべてバックアップされ、変更されたバージョンで置き換えられます。

4. HP Vugen が開いていて、ユーティリティが実行されている場合、変更したスクリプトを再ロードするように求められます。プロンプトが表示されたら、［Yes to All］をクリックします。
5. TagScript ユーティリティを終了するか、またはファイル選択ダイアログ ボックスで［cancel］ボタンをクリックすることができます。
   TagScript ユーティリティを終了することは、必須ではありません。HP Vugen を使用している間、多くのユーザはこのユーティリティを終了していません。スクリプトが変更されているか、新しいスクリプトが作成されている場合、ユーティリティを終了しないことで処理を簡略化できます。
6. スクリプトの以下の場所にタグ付けされたことを確認します。
   - HP Vugen コードの新しいパラグラフは、各スクリプトの先頭に挿入されます。
   - タグは、すべての lr_start_transaction、lr_end_transaction の前、およびスクリプトの末尾に挿入されます。
7. （オプション） Blame スタックの個別のセットを使用して、HP Loadrunner のパフォーマンス テストで各仮想ユーザを追跡できます。各ユーザを追跡するには、スクリプトの先頭の宣言部にある以下の行のコメント化を解除します。

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

   注: Production タグの付いたスクリプトでこのオプションのコメント化が解除されている場合は、存在する各ポイントまたは合成ジェネレータで Blame スタックの個別のセットが作成されます。

Java 7 Autoprobe

Java 1.7 でコンパイルおよび実行されるアプリケーションを使用する場合は、-XX:UseSplitVerifier を追加します。追加しない場合、エージェントは以下のようなバイトコード検証エラーを引き起こす可能性があります。

```
java.lang.VerifyError: StackMapTable error: bad offset,
```

または

```
java.lang.ClassFormatError: Illegal local variable table
```

javaagent または Xbootclasspath によって JVM AutoProbe を使用している場合に、この手順が適用されます。

アプリケーションをインスツルメントする方法

エージェントをインストールしたら、アプリケーションをインスツルメントするためにアプリケーション サーバを設定する必要があります。 アプリケーションをインスツルメントするための最も一般的な方法は、JVM AutoProbe および *-javaagent* コマンド ライン オプションを使用することです。 JVM AutoProbe は、実行時のアプリケーションを動的にインスツルメントします。 これは、ファイル システムからロードされるすべてのバイトコードを Introscope が参照できるようにするために、ブートストラップまたはアプリケーション サーバ クラス ローダ内にフックを提供するすべての J2EE アプリケーション サーバに適しています。

ほとんどの JVM プロバイダは *-javaagent* オプションをサポートしています。 このオプションのサポートを提供しない JVM を使用している場合は、代替のインスツルメンテーション方法を使用する必要があります。

ProbeBuilder を手動で実行する必要があるのは、アプリケーションの起動前にバイトコードを静的にインスツルメントしなければならない場合のみです。 ProbeBuilder を手動で実行するには、ProbeBuilder ウィザードを使用するか、コマンドライン プロンプトから開始します。 実行により、バイトコードがインスツルメントされ、新しい名前の付いた、インスツルメント済みの jar ファイルまたは class ファイルが出力されます。この新しくインスツルメントされたバイトコードは、アプリケーションの起動前に、そのクラスパスの前方に配置されます（または所定の場所で名前変更されます）。

JVM AutoProbe の代替方法の詳細については、付録 B「インスツルメンテーションの代替方法」を参照してください。

Java Agent を起動するためのアプリケーション サーバの設定

エージェントのプライマリ *.jar* ファイルおよびエージェント プロファイルのパスを追加するには、インスツルメント対象のアプリケーション サーバを設定する必要があります。 この作業はほとんどの場合、アプリケーション サーバの起動スクリプトを編集し、アプリケーション サーバを再起動することで完了します。 アプリケーション サーバが再起動するとき、Java Agent は、JVM およびアプリケーション環境のデフォルト コンポーネントについて検出されたクラスをインスツルメントします。 実行される具体的な手順は、アプリケーション サーバによって異なります。

Java Agent を使用するための Apache Tomcat の設定

Java Agent を使用するために Apache Tomcat を設定するには、Tomcat 起動スクリプトを編集する必要があります。 デフォルトでは、Tomcat 起動スクリプトは *$CATALINA_HOME/bin* ディレクトリの *catalina.sh* または *catalina.bat* です。 Web サーバの要件に応じて、別の起動スクリプトまたは起動スクリプトの別の場所を使用する場合があります。

以下の手順に従います。

1. Tomcat 起動スクリプトがあるディレクトリに移動します。 例：
   `cd /apache-tomcat-6.0.18/bin`
2. Tomcat 起動スクリプトをテキスト エディタで開きます。 例：
   `vi catalina.sh`
3. Java オプションを設定するためのコマンド ラインを探し、エージェントの起動 *.jar* ファイルおよびエージェント プロファイルのパスを指定するための以下のコマンドライン オプションを追加します。
   `-javaagent:<`*Agent.jar のパス*`> -Dcom.wily.introscope.agentProfile=<`*エージェント プロファイルのパス*`>`

   たとえば *Agent.jar* を使用する場合は、サーバを起動するコマンドの前に以下のようなコードを追加します。
   ```
   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. 起動スクリプトを保存します。
5. （オプション） JMX メトリックを収集するようにエージェント プロファイルを設定することで、Apache Tomcat JMX メトリックのレポートを有効にします。 IntroscopeAgent.profile を開いて、以下のプロパティを設定します。

   introscope.agent.jmx.enable=true

   注: JMX リモート管理サーバをプラットフォーム固有の MBeanServer と共に使用することによって、コンソールで Apache Tomcat からの JMX メトリックを表示する場合は、IntroscopeAgent.profile に com.wily.use.platform.mbeanserver=true を追加します。 この設定は、プラットフォーム固有の MBeanServer の使用がコマンド ラインで設定されていた Introscope の以前のバージョンから変更されています。
6. IntroscopeAgent.profile ファイルを保存して閉じます。
7. Tomcat サーバを再起動します。

データ収集のカスタマイズ

Apache Tomcat サーバに Java エージェントをインストールした後は、データ収集をカスタマイズするために PBD を変更できます。

以下の手順に従います。

1. <*Agent_Home*>¥wily¥core¥config ディレクトリに移動します。
2. テキスト エディタで tomcat.pbd ファイルを開きます。
3. カスタマイズするセクションを変更します。
4. ファイルを保存して閉じます。

Java エージェントを使用するための JBoss の設定

Java エージェントを使用するために JBoss を設定するには、使用する JBoss バージョンに対応する JBoss 起動スクリプトを編集します。

- JBoss 7 以降のバージョン用の JBoss 起動スクリプト

  スタンドアロン モードのデフォルトの JBoss 起動スクリプトは、$JBOSS_HOME/bin ディレクトリの standalone.sh または standalone.bat です。ドメイン モードのデフォルトは、$JBOSS_HOME/bin ディレクトリの domain.sh または domain.bat です。

- JBoss 6 およびそれ以前のバージョン用の JBoss 起動スクリプト

  デフォルトでは、JBoss 起動スクリプトは $JBOSS_HOME/bin ディレクトリの run.sh または run.bat です。

Web サーバの要件に応じて、別の起動スクリプトまたは起動スクリプトの別の場所を使用できます。

以下の手順に従います。

1. JBoss 起動スクリプトがあるディレクトリに移動します。以下に例を示します。

   cd /jboss.GA/bin

2. JBoss 起動スクリプトをテキスト エディタで開きます。以下に例を示します。

   vi run.sh / vi standalone.sh

3. Java オプションを設定するコマンド ラインを入力します。エージェントの起動 .jar ファイルおよびエージェント プロファイルへのパスを指定するための以下のコマンド ライン オプションを追加します。

   - JBoss 7 以降の場合

     -Djboss.modules.system.pkgs=com.wily,com.wily.*
     -javaagent:<Agent.jar のパス> -Dcom.wily.introscope.agentProfile=<*エージェント プロファイルのパス*>

   - JBoss 6 およびそれ以前のバージョンの場合

     -javaagent:<Agent.jar のパス> -Dcom.wily.introscope.agentProfile=<*エージェント プロファイルのパス*>

   たとえば Agent.jar を使用する場合は、サーバを起動するコマンドの前に以下の例のようなコードを追加します。

   - JBoss 7 以降の場合

     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 およびそれ以前のバージョンの場合

     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 または standalone.bat ファイルを保存します。
5. （オプション） JMX メトリックを収集するようにエージェント プロファイルを設定することで、JBoss JMX メトリックのレポートを有効にします。

   a. テキスト エディタで IntroscopeAgent.profile を開き、introscope.agent.jmx.enable=true を設定します。

      注: JMX リモート管理サーバをプラットフォーム固有の MBeanServer と共に使用することによって、JConsole で JBoss からの JMX メトリックを表示する場合は、IntroscopeAgent.profile に com.wily.use.platform.mbeanserver=true を追加します。この設定は、プラットフォーム固有の MBeanServer の使用がコマンド ラインで設定されていた Introscope の以前のバージョンから変更されています。

b. introscopeAgent.profile ファイルを保存して閉じます。

c. JBoss 7 以降の場合、JBoss インストールの <Agent_Home>/wily/common ディレクトリに移動し、WebAppSupport.jar ファイルを <Agent_Home>¥wily¥core¥ext ディレクトリに移動します。

d. JBoss 6 およびそれ以前のバージョンの場合

- <Agent_Home>/wily/common ディレクトリに移動し、WebAppSupport.jar ファイルを /server/<server_configuration>/lib ディレクトリに移動します。
- <Agent_Home>/wily/deploy ディレクトリに移動し、introscope-jboss-service.xml ファイルを /server/<server_configuration>/deploy ディレクトリに移動します。

注: これらのパスはデフォルト設定を使用していることを想定しています。使用していない場合は、ファイルを対応する JBoss インストール ディレクトリへ移動します。

JBoss アプリケーション サーバ ログの考慮事項

JBoss アプリケーション サーバを使用する場合、以下の点を考慮する必要があります。

症状:

以下の動作は、アプリケーション サーバでログ記録を実行するエージェントに共通です。

- JBoss 6 システムの場合、アプリケーション サーバは boot.log ファイルを作成しません。
- JBoss 7 システムの場合、エージェントが起動しないか、または、以下のエラーが表示されます。

```
The LogManager was not properly installed
```

解決方法:

これらの問題を解決するには、以下のいずれかの方法を使用します。

- エージェント プロファイルでエージェントのログを無効にし、サーバを起動します。 サーバが起動したら、ログを有効にします。

- JBoss 起動スクリプトをテキスト エディタで開き、以下のように更新します。

```
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 および PBL

JBoss アプリケーション サーバに Java Agent をインストールすると、データ収集をカスタマイズするための以下の JBoss 固有の PBD および PBL ファイルを利用できます。

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

自動名前付けのための JBoss 7 の設定

webappsupport.jar ファイルをアプリケーション サーバのデプロイ フォルダにコピーした場合のみ、JBoss 7 で自動名前付け機能が動作します。

例

スタンドアロン サーバの場合は、webappsupport.jar ファイルを JBOSS7_HOME/standalone/deployments/ フォルダにコピーします。

Java エージェントを使用するための Oracle WebLogic の設定

Java エージェントを使用するために Oracle WebLogic を設定するには、WebLogic 起動スクリプトを編集します。 要件に応じて、使用する起動スクリプトは WebLogic ドメインに固有である場合があります。 デフォルトでは、WebLogic 起動スクリプトは、*$WEBLOGIC_HOME*/samples/domain/<*ドメイン名*>/bin ディレクトリの startWebLogic.sh または startWebLogic.cmd です。 別の起動スクリプトを使用することもできます。たとえば、アプリケーション固有の起動スクリプトまたは起動スクリプトの別の場所を使用できます。

以下の手順に従います。

1. 変更する WebLogic 起動スクリプトがあるディレクトリに移動します。
   例：
   `cd $WL_HOME/samples/domain/wl_server`
2. WebLogic 起動スクリプトをテキスト エディタで開きます。 例：
   `vi startWebLogic.sh`
3. Java オプションを設定するためのコマンド ラインまたは Java コマンド ラインを探して、以下のコマンドライン オプションを追加します。
   例： -javaagent:<*PathToAgentJar*>
   -Dcom.wily.introscope.agentProfile=<*PathToAgentProfile*>
4. WebLogic 起動スクリプトまたはアプリケーション固有の起動スクリプトを保存して閉じます。

WebLogic 用の起動クラスの作成

アプリケーション サーバまたはクラスタ用の WebLogic 起動クラスを作成すると、Java Agent はアプリケーション サーバから追加情報を収集できるようになります。 起動クラスを設定する場合、Java Agent は自動的にその名前を判別できます。 また、起動クラスにより Java Agent は、Workstation 内でアプリケーションの稼働状況を判断するのに使用する JMX メトリックをレポートできます。

以下の手順に従います。

1. WebLogic Administrative Console を開きます。
2. 左側のペインで、［Environments］フォルダを展開します。
3. ［Startup & Shutdown］フォルダをクリックします。
   ［Startup and Shutdown］ページが表示されます。

4. ［Configure a New Startup Class］をクリックします。

   ［Configuration］タブが表示されます。

5. ［Name］フィールドに、以下を入力します。

   ```
   Introscope Startup Class
   ```

6. ［ClassName］フィールドに、以下を入力します。

   ```
   com.wily.introscope.api.weblogic.IntroscopeStartupClass
   ```

7. ［Create］をクリックします。

   ［Target and Deploy］タブが表示されます。

8. この起動クラスを利用可能にするサーバのボックスをオンにします。
9. ［Apply］をクリックし、［Run before application deployments］オプションを選択します。
10. WebAppSupport.jar の場所を、<*サーバまたはアプリケーション サーバ*> 起動クラスパスに追加します。
11. アプリケーション サーバを再起動します。

WebLogic Server でのプロセスにまたがる追跡の有効化

トランザクション追跡セッションでは、互換性のある JVM バージョンを備えたコンピュータ上の JVM 境界にまたがるトランザクションを含めて、トランザクションで発生するすべてのオペレーションを追跡できます。

プロセスにまたがるトランザクション追跡は、サーブレットから EJB などの同期トランザクション、および非同期トランザクションでサポートされています。

WebLogic でプロセスにまたがるトランザクション追跡を有効にする方法

1. 「WebLogic 用の起動クラスの作成 (P. 49)」の説明に従って、起動クラスを作成します。
2. 「-Dweblogic.TracingEnabled=true」 を WebLogic Server を起動する Java コマンド ラインに追加します。
3. IntroscopeAgent.profile ファイルをテキスト エディタで開きます。
4. introscope.agent.weblogic.crossjvm プロパティを探し、true に設定します。 例：

   ```
   introscope.agent.weblogic.crossjvm=true
   ```

5. IntroscopeAgent.profile ファイルを保存して閉じます。

JMX メトリックの Java エージェント サポート

CA Introscope® は、アプリケーション サーバまたは Java アプリケーションが JMX 互換の MBean として公開する管理データを収集したり、Investigator メトリック ツリーに JMX データを表示したりできます。WebLogic は、以下の JMX メトリックのソースを提供します。

- RuntimeServiceMBean : サーバごとの実行時メトリック。アクティブで有効な設定を含みます。
- DomainRuntimeServiceMBean： ドメイン全体の実行時メトリック。
- EditServiceMBean： ユーザによる永続コレクションの編集を可能にします。

CA Introscope® は、メトリック用に RuntimeServiceMBean だけをポーリングします。 RuntimeServiceMBean は、効率の問題からローカル アクセスをサポートしており、また、妥当と考えられるデータのほとんどが含まれています。 ただし、CA Introscope® は、Sun JMX 仕様に組み込まれた MBean をサポートできます。

注: Sun JMX 仕様の詳細については、http://java.sun.com/products/JavaManagement/ （英語）を参照してください。

JMX メトリックの収集をサポートするため、WebLogic の IntroscopeAgent.profile ファイルには以下のキーワードが定義されており、デフォルトで有効になっています。

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

CA Introscope® は、JMX メトリックを、Investigator ツリーの以下のノード下に表示します。

<Domain>|<Host>|<Process>|AgentName|JMX|

詳細:

JMX レポートの有効化 (P. 231)

Weblogic でリソース メトリックを設定する方法

さまざまなエージェントが CA Introscope® Workstation のリソース メトリック カテゴリをレポートできます。 Java Agent がリソース メトリックをレポートできるように、Oracle WebLogic サーバを設定できます。

Oracle WebLogic のリソース メトリックを設定するには、以下の手順に従います。

1. 「CA APM for Oracle WebLogic Server ガイド」の手順に従って、CA APM for Oracle WebLogic をインストールします。
2. 「Oracle データベース用 CA APM ガイド」の手順に従って、Oracle データベース用 CA APM オプションを有効にします。
3. 「CA APM 設定および管理ガイド」に記載されている、リソース メトリックを設定するための手順に従います。

WebLogic Diagnostic Framework (WLDF)について

WebLogic Diagnostic Framework (WLDF) は、モニタリングおよび診断フレームワークです。これは、WebLogic Server プロセス内で動作し、標準のサーバ ライフ サイクルに含まれる一連のサービスを定義および実装します。WLDF を使用すると、実行中のサーバおよびそのコンテナに導入されたアプリケーションによって生成される診断データの作成、収集、解析、およびアーカイブ化を実行でき、データにアクセスできるようになります。このデータによって、サーバおよびアプリケーションの実行時の実際の性能がわかります。また、障害が発生した場合には、その特定および診断にデータを利用できます。

WLDF では、標準インターフェースからサーバ データに動的にアクセスできます。また、指定時刻にアクセスされるデータの量も、サーバをシャットダウンして再起動しなくても変更できます。

WLDF データが Introscope メトリックに変換される仕組み

WLDF では、情報はそれぞれに複数の列を含む一連の Data Accessor に整理されます。 Introscope は、この WLDF 情報を Introscope 固有のメトリック形式に変換して、Investigator 内の以下のノード下に表示します。

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

Data Accessor 内の情報は、ドメイン名と 1 つ以上のキーと値のペアで定義されます。 Introscope は、Data Accessor 列を、Introscope 生成メトリック内でアルファベット順に表示される、キーと値の情報に変換します。 使用される構文の例を以下に示します。

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

たとえば、以下の表は、HTTPAccessLog Data Accessor の BYTECOUNT 列の情報を示しています。

ドメイン名	キーと値のペア	メトリック名
WebLogic	Name=HTTPAccessLog、 Type=WLDFDataAccessRuntime=Accessor、 WLDFRuntime=WLDFRuntime	BYTECOUNT

上の表の Data Accessor 情報は、以下の Introscope メトリックに変換されます。

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

キーと値のペアは、Introscope メトリックではアルファベット順に表示されることに注意してください。

WLDF レポートの有効化

デフォルトでは、WLDF レポートは、Introscope では有効ではありません。WLDF メトリックのレポートを有効にするために、エージェント プロファイルを変更することができます。

WLDF レポートを有効にする方法

1. 管理対象アプリケーションが実行されている場合は終了します。
2. WebLogic 起動クラスを設定します。
3. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
4. 以下のプロパティを探して設定します。
   ```
   introscope.agent.wldf.enable=true
   ```
5. *IntroscopeAgent.profile* ファイルを保存して閉じます。

Java エージェントを使用するための、JRockit JVM と組み合わせた WebLogic の設定

対象: JRockit JVM 1.5 以降と組み合わせた WebLogic 9.0 以降

注: WebLogic のサポートされているバージョンについては、「*Compatibility Guide*」を参照してください。古いバージョンの WebLogic または JRockit JVM を使用している場合は、AutoProbe コネクタ ファイル (P. 410)を作成および実行して、アプリケーションをインスツルメントできます。

WebLogic を JRockit JVM と組み合わせて使用している場合は、以下のコマンドライン オプションを使用して JVM を起動します。

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

ソケット メトリックを表示するための、JRockit JVM と組み合わせた WebLogic の設定

対象: JRockit JVM 1.5.0 と組み合わせた WebLogic 9.0

症状:

サードパーティ製品との互換性の問題のため、ソケット クライアントのメトリックを表示する際に問題が発生する場合があります。

解決方法:

以下の Managed Socket オプションを使用して、ソケット クライアント用メトリックをオンにします。

以下の手順に従います。

1. toggles_typical.pbl または toggles_full.pbl ファイルをテキスト エディタで開きます。
2. ソケットのメトリックを追跡する Managed Socket オプションをオンにします。 例：

```
#####################
# 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. 変更を保存します。

   Managed Socket オプションがオンになります。

Java エージェントを使用するための IBM WebSphere の設定

Java エージェントを使用するために IBM WebSphere を設定するには、WebSphere 起動スクリプトを編集します。 要件に応じて、使用する起動スクリプトはアプリケーション サーバ ノードに固有である場合があります。

以下の手順に従います。

1. $WEBSPHERE_HOME/profiles/<*アプリケーション サーバ名*>/config/cells/<*セル名*>/nodes/<*ノード名*>/servers/server1 ディレクトリに移動します。
2. server.xml ファイルを編集します。 このファイルは、WebSphere のデフォルトの起動スクリプトおよび場所です。

   注: アプリケーション サーバの要件に応じて、別の起動スクリプトを使用できます。 たとえば、アプリケーション固有の起動スクリプトまたは起動スクリプトの別の場所を使用できます。 また、別のオペレーティング システム上での WebSphere の異なる組み合わせや、別の JVM ベンダーまたは JVM バージョンの使用によって、特別な要件が存在する可能性があります。
3. ファイルを保存して閉じます。

注: 詳細については、使用する WebSphere Application Server 環境に関して最も詳しいセクションを参照してください。

UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 での WebSphere Application Server 6.1 の設定

対象: UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 上の WebSphere Application Server 6.1

動的インスツルメンテーション機能では、クラスの再定義をサポートする必要があります。 IBM JDK バージョン 5 を実行している場合、クラスの再定義を使用するとパフォーマンスに著しい影響を与える場合があります。CA Introscope® および IBM JDK バージョン 5 のユーザが動的インスツルメンテーションを使用する場合、このパフォーマンス オーバーヘッドについて注意する必要があります。 この設定を使用する場合、QA 環境のみで動的インスツルメンテーション機能を使用することをお勧めします。

注: このパフォーマンス オーバーヘッドの詳細については、IBM の「*Java Diagnostics Guide*」を参照してください。

IBM JVM 1.5 を使用して WebSphere 6.1 を実行している場合は、別のバージョンの Java エージェント .jar ファイルおよび Java エージェント プロファイルを使用します。 その AgentNoRedef.jar および IntroscopeAgent.NoRedef.profile という名前のファイルは、*<Agent_Home>*/core/config ディレクトリにあります。

注: AllAppServer エージェントの配布を使用している場合、代替プロファイルは IntroscopeAgent.websphere.NoRedef.profile という名前です。

前述のファイルおよび構文を使用すると、以下の内容のメトリックがレポートされなくなります。

- システム クラス
- NIO（ソケットとデータグラム）
- SSL

以下の機能にも影響があります。

- ソケット インスツルメンテーションは、9.0 より前の CA Introscope® の ManagedSocket のスタイルを使用します。
- リモートの動的インスツルメンテーションが無効になります。
- PBD ファイルを変更する場合は、変更を適用するために、インスツルメントされた JVM を再起動する必要があります。
- 複雑な継承と階層をサポートするインスツルメンテーションが無効になります。

前述のファイルおよび構文を使用すると、エージェントは以下のアクションによってクラス再定義のステータスをレポートします。

- *Agent Class Redefinition Enabled* という名前のメトリックを Investigator のエージェント ノード下に追加する。 メトリックの値は true または false です。
- ログ メッセージをエージェントのログ ファイルに書き込む。
  - クラスの再定義が有効である場合、ログ メッセージは WARN レベルで書き込まれ、以下のようになります。

    ```
    Introscope Agent Class Redefinition is enabled. Enabling class redefinition
    on IBM 1.5 JVMs is known to incur significant overhead.
    ```

  - クラスの再定義が無効である場合、ログ メッセージは INFO レベルで書き込まれ、以下のようになります。

    ```
    Introscope Agent Class Redefinition is disabled.
    ```

- メッセージを標準エラーに書き込む（クラスの再定義が有効である場合のみ）。内容は以下のとおりです。

  ```
  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.
  ```

IBM 以外の JVM または 1.5 以外のバージョンの IBM JVM を使用する場合、前述のメトリックとメッセージは出力されません。

エージェントを使用するために WebSphere Application Server を設定します。

以下の手順に従います。

1. WebSphere Administrator Console を開きます。
2. ［Application Servers］-［<サーバ名>］-［Server Infrastructure］-［Java and Process Management］-［Process Definition］-［Java Virtual Machine］をクリックします。
3. 以下のように［Generic JVM Argument］フィールドを設定します。

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

   インスツルメントされたアプリケーションとインスツルメントされていないアプリケーションが両方とも同じコンピュータ上に存在する場合は、［Generic JVM Argument］に -Xshareclasses:none の設定を含めます。 この設定は、AIX でのエラーを回避します。

   注: WebSphere の複数のバージョンが同じエージェント ディレクトリを使用している場合は、一意のディレクトリが必要です。

UNIX、Windows、Sun、HP、その他の JVM 1.5 での WebSphere Application Server 6.1 の設定

対象: UNIX、Windows、Sun、HP、その他のすべての JVM 1.5 上の WebSphere Application Server 6.1

注: 1.5 以外に IBM JVM 以外のバージョンまたは IBM JVM バージョンを使用する場合、一部のメトリックおよびメッセージは出力されません。

エージェントを使用するために WebSphere Application Server を設定できます。

以下の例では、WebSphere 6.1 に Java エージェントをインストールする場合に、特定の JVM で使用する Java の引数と .jar ファイルを示します。

以下の手順に従います。

1. WebSphere Administrator Console を開きます。
2. ［Application Servers］-［<サーバ名>］-［Server Infrastructure］-［Java and Process Management］-［Process Definition］-［Java Virtual Machine］をクリックします。
3. 以下のように［Generic JVM Argument］フィールドを設定します。

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

4. WebSphere アプリケーション サーバを再起動します。

UNIX、Windows、OS/400、JVM 1.5 での WebSphere Application Server 7.0 の設定

対象: WebSphere Application Server 7.0 - UNIX、Windows、OS/400、JVM 1.5 以降

エージェントを使用するために WebSphere Application Server 7.0 を設定できます。 WebSphere 7.0 を監視する場合は、Java エージェントを起動するためのサーバ設定を行う前に、ビルド 7.0.0.8 以降がインストールされていることを確認します。

以下の手順に従います。

1. WebSphere Administrator Console を開きます。
2. ［Application Servers］-［<サーバ名>］-［Server Infrastructure］-［Java and Process Management］-［Process Definition］-［Java Virtual Machine］をクリックします。

3. 以下のように［Generic JVM Argument］フィールドを設定します。

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

4. WebSphere アプリケーション サーバを再起動します。

z/OS - JVM 1.5 での WebSphere Application Server 7.0 の設定

対象: z/OS - JVM 1.5 以降での WebSphere Application Server 7.0

エージェントを使用するために WebSphere Application Server 7.0 を設定できます。Java エージェントを起動するためのサーバ設定を行う前に、ビルド 7.0.0.8 以降がインストールされていることを確認します。

以下の手順に従います。

1. WebSphere Administrator Console を開きます。
2. ［Application Servers］-［<サーバ名>］-［Server Infrastructure］-［Java and Process Management］-［Process Definition］-［Java Virtual Machine］をクリックします。
3. 以下のように［Generic JVM Argument］フィールドを設定します。

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

4. WebSphere アプリケーション サーバを再起動します。

Java2 セキュリティ ポリシーの変更

Java2 のセキュリティを有効にした WebSphere 環境で AutoProbe を正しく実行するには、Java2 セキュリティ ポリシーへのアクセス権の追加が必要になる場合があります。

Java2 セキュリティ ポリシーに権限を追加する方法

1. $*WebSphere home/properties/server.policy* ファイルをテキスト エディタで開きます。
2. ファイルに以下の権限を追加します。

```
// 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";
};
```

   注: 読みやすくするために改行を入れてありますが、*server.policy* ファイルへアクセス権を追加する際には、必要ありません。

3. ファイルを保存して閉じます。

WebSphere でのカスタム サービスの設定

WebSphere Application Server でカスタム サービスを作成または変更することができます。 カスタム サービスを使用すると、Java エージェントでアプリケーション サーバから追加の情報を収集することができます。 カスタム サービスを設定した場合、Java エージェントは自動的にその名前を判別できます。 また、カスタム サービスを使用すると、Java エージェントで JMX および Performance Monitoring Infrastructure（PMI）メトリックをレポートすることも可能になります。 Introscope Workstation は、［アプリケーションの概要］タブで、これらのメトリックを使用してアプリケーション稼働状況を判定します。 カスタム サービスは、CA Introscope® の JMX メトリックにアクセスするために使用するユーザ クレデンシャルを暗号化できます。

注: SIBus 関連のメトリックまたは WebSphere Application Server に追加された新しい PMI モジュールを取得するには、既存のカスタム サービスを無効にしてからカスタム サービスを作成してください。

以下の手順に従います。

1. WebSphere Administrator Console を開きます。
2. 設定するサーバを選択し、［Server Infrastructure］-［Administration］-［Custom Services］をクリックします。
3. 目的のカスタム サービスを変更するか、または新たに 1 つ作成します。
4. Configuration ページで以下のフィールドを入力し［OK］をクリックします。

   Enable service at server startup

   サーバの起動時にサービスを開始するように指定します。

   External Configuration URL

   設定プロパティ ファイルの場所を指定します。 JMX メトリック設定については、jmxconfig.properties ファイルを使用します（例: <Agent_Home>/wily/common/jmxconfig.properties）。

   Classname

   カスタム サービス クラスの名前を指定します。以下に例を示します。

   com.wily.introscope.api.websphere.IntroscopeCustomService

   com.wily.powerpack.websphere.agent.PPCustomService

   Display Name

   CA Introscope® で表示する名前を指定します（例: Introscope Custom Service）。

   Classpath

   プロパティ ファイルの完全修飾パス名を指定します。以下に例を示します。

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

5. JMX メトリックにアクセスするためのユーザ クレデンシャルを設定します。
   a. <Agent_Home>/wily/common に移動し、テキスト エディタで jmxconfig.properties ファイルを開きます。

b. コメント化を解除し、プロパティの説明に従って値を設定します。

c. ファイルを保存して閉じます。

6. アプリケーション サーバを再起動します。

サーバが起動すると、カスタム サービスが開始され、ユーザ クレデンシャルが暗号化されます。 その後、サーバが起動するときは常に、暗号化されたパスワードが使用されます。

例: パスワード暗号化用の jmxconfig.properties File の設定

以下のテキストは、パスワード暗号化用に設定された jmxconfig.properties の例を示しています。

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

詳細:

JMX レポートを使用するための WebSphere および WebLogic の設定 (P. 236)

WebSphere PMI メトリックの収集の有効化

CA Introscope® は、WebSphere で提供される PMI インターフェースを使用して取得される WebSphere PMI（Performance Monitoring Infrastructure）メトリックを抽出することによって、WebSphere パフォーマンス データを提供することができます。

CA Introscope® でデータを利用できるようにするには、まず、WebSphere で PMI データ収集を有効にします。 WebSphere では、すべてのパフォーマンス モニタ設定がデフォルトでオフになっています。

以下の手順に従います。

1. WebSphere で CA Introscope® 用のカスタム サービスを設定します。
2. WebSphere で PMI データ収集を有効にします。
3. IntroscopeAgent.profile ファイル内で PMI データのレポートを有効にします。

WebSphere で PMI データ収集を有効にした後は、PMI データを CA Introscope® メトリックとして表示できます。必要に応じて、CA Introscope® に取り込むメトリック カテゴリをフィルタできます。

注: z/OS 上の WebSphere の場合、CA APM for WebSphere z/OS を使用することをお勧めします。 CA APM for WebSphere z/OS では、WebSphere 固有の PBD およびメトリックが提供されています。 これらの PBD およびメトリックは、WebSphere 固有のメトリックを取得するためのオーバーヘッドの低いトレーサ技術を使用します。 CA APM for WebSphere z/OS を使用する場合、WebSphere for z/OS 内で PMI を有効にする必要がありません。 また、WebSphere for z/OS の PMI レポートを有効にすることもできます。ただし、この方法は多くのシステム リソースを消費します。

Introscope での WebSphere PMI メトリック レポートの設定

WebSphere でパフォーマンス モニタリング設定をオンにした後で、Introscope で PMI データ収集を有効にし、さらにレポートを表示するメトリック カテゴリを有効にする必要があります。

Introscope で PMI メトリック レポートを設定する方法

1. 管理対象アプリケーションをシャットダウンします。
2. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
3. ［WebSphere PMI Configuration］セクションで *introscope.agent.pmi.enable* プロパティを探し、*true* に設定されていることを確認します。

4. 以下に示す、高レベルの PMI メトリック カテゴリのプロパティを探します。

```
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
```

   次の 4 つのカテゴリ、*threadPool*、*servletSessions*、*connectionPool*、および *j2c* は、デフォルトで *true* に設定されています。 追加のメトリック カテゴリを有効にするには、該当するプロパティを *true* に設定します。また、レポートされるメトリックの量を減らすには、カテゴリをコメント化します。

5. 変更を保存してファイルを閉じます。
6. 管理対象アプリケーションを再起動します。

Introscope で PMI の収集を有効にすると、利用可能な PMI メトリックが、Investigator ツリーの WebSpherePMI ノードに表示されます。

WebSphere 用リソース メトリック マップ データのレポートの設定

さまざまなエージェントが CA Introscope® Workstation のリソース メトリック カテゴリをレポートできます。 WebSphere 内で PMI データ収集を有効にした後、リソース メトリック マップ データがレポートされるようにアプリケーション サーバを設定できます。

以下の手順に従います。

1. WebSphere コンソールにログインします。
2. ［Monitoring and Tuning］、［Performance Monitoring Infrastructure］を選択します。
3. レポート用に使用するサーバを選択します（server1 など）。
4. ［Configuration］タブをクリックし、［Custom］オプションを選択します。
5. ［Custom］リンクをクリックします。

   オプションの設定ツリーが表示されます。
6. ツリー上の［JDBC Connection Pools］を選択し、右ペイン上の［WaitingThreadCount］を有効にします。
7. ツリー上の［ThreadPools］を選択し、右ペイン上の［ActiveCount］を有効にします。
8. 設定を保存し、アプリケーション サーバを再起動します。

   リソース メトリックをレポートするようにアプリケーション サーバが設定されます。

注: リソース メトリック マップ データの詳細については、「CA APM 設定および管理ガイド」を参照してください。

WebSphere でのプロセスにまたがる追跡の有効化

トランザクション追跡セッションでは、互換性のある JVM バージョンを備えたコンピュータ上の JVM 境界にまたがるトランザクションを含めて、トランザクションで発生するすべてのオペレーションを追跡できます。 プロセスにまたがるトランザクション追跡は、サーブレットから EJB などの同期トランザクション、および非同期トランザクションでサポートされています。

WebSphere でプロセスにまたがる追跡を有効にする方法

1. 「WebSphere 6.1 でのカスタム サービスの設定 (P. 61)」の説明に従って、カスタム サービスを作成します。
2. 作業領域のサービスを有効にします。

   管理ページで、［Servers］-［Application servers］を選択し、［server1］-［Business Process Services］-［Work Area Service］をクリックします。次に、［Enable service at server startup］ボックスをオンにします。
3. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
4. *introscope.agent.websphere.crossjvm* プロパティを探し、*true* に設定します。 例：

   ```
   introscope.agent.websphere.crossjvm=true
   ```
5. *IntroscopeAgent.profile* ファイルを保存して閉じます。

WebSphere for z/OS のログに関する考慮事項

WebSphere z/OS 環境でログを記録する場合には、いくつかの考慮事項があります。Java Agent のログ記録オプションの詳細については、「Java Agent のモニタリングおよびログ記録 (P. 161)」を参照してください。

ログ出力を EBCDIC とするタグの付加

WebSphere for z/OS では、デフォルト エンコーディングが EBCDIC CP1047 から ASCII ISO8859-1 に変わりました。 z/OS は、通常は EBCDIC マシンであるため、Java Agent または AutoProbe によって書き込まれたログ データには、最終的な出力ストリームとして、ASCII の代わりに EBCDIC を使用するようにタグを付ける必要があります。

データに ASCII の代わりに EBCDIC のタグを付加する方法

1. *<Agent_Home>¥wily¥core¥config* ディレクトリの *IntroscopeAgent.profile* ファイルを開きます。
2. 以下のプロパティを、*IntroscopeAgent.profile* に追加します。

   ```
   log4j.appender.console.encoding=IBM-1047
   log4j.appender.logfile.encoding=IBM-1047
   ```
3. *IntroscopeAgent.profile* を保存します。

ログ機能での起動タイミングの問題の解消

WebSphere for z/OS では、プロパティを使用して、CA Introscope® のログ機能で発生する可能性がある起動タイミングの問題が発生しないようにできます。

以下の手順に従います。

1. *<Agent_Home>*¥wily¥core¥config ディレクトリの IntroscopeAgent.profile ファイルを開きます。
2. 以下のプロパティを IntroscopeAgent.profile に追加します。
   ```
   introscope.agent.logger.delay=100000
   ```
   値はミリ秒単位なので、この例でのデフォルトの遅延は 100 秒です。
3. IntroscopeAgent.profile を保存します。

Java Agent を使用するための Oracle Application Server の設定

Oracle Application Server（OAS）では、すべてのアプリケーションの管理に 1 つの環境設定ファイルを使用します。そのため、Oracle Console によって管理されるすべての JVM はこのファイルを使用します。 通常、このファイルは *opmn.xml* と呼ばれます。

Java Agent を使用するために Oracle Application Server を変更する方法

1. アプリケーション サーバをシャットダウンし、*opmn.xml* のバックアップを作成します。
2. *opmn.xml* で、インスツルメントするアプリケーションセクションを探します。 多くの場合、この段階では複数のアプリケーションをインスツルメントしたいと考えます。
3. 選択したアプリケーションの *<category id="start-parameters">* の下で、*<data id="java-options"* という名前のセクションを探します。

4. ご使用の環境に適したパスを使用して、*java-options* 行の末尾の *"/>* の前に、以下の内容を挿入します。

   -javaagent:<Agent.jar のパス> -Dcom.wily.introscope.agentProfile=<エージェント プロファイルのパス>

   たとえば、1 つのアプリケーションのエントリ全体を 1 行で記述します。

```
<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 エージェントを使用するための GlassFish の設定

Java エージェントを使用するように、Sun GlassFish オープン ソース アプリケーション サーバ プロジェクトを設定できます。

注: Sun GlassFish のサポートされているバージョンについては、「*Compatibility Guide*」を参照してください。

以下の手順に従います。

1. GlassFish で、以下の場所に移動します。

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

2. domain.xml ファイルをテキスト エディタで開きます。
3. Agent.jar ファイルと IntroscopeAgent.profile ファイルの完全パスを、JVM オプション として domain.xml ファイルの java-config エレメントに追加します。 例：

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

4. テキスト エディタで構成プロパティ ファイルを開き、wily プロパティを追加します。

   たとえば Glassfish 3.2 の場合は、プロパティ ファイルは <glassfish_home>/glassfish/config/osgi.properties です。

編集するプロパティは以下のとおりです。

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

5. WebAppSupport.jar ファイルを <Agent_Home>/wily ルート ディレクトリから <Agent_Home>/wily/ext ディレクトリにコピーします。
6. アプリケーション サーバを起動します。

Java エージェントを使用するための SAP Netweaver の設定

Java エージェントを使用するために、SAP Netweaver 用の JVM AutoProbe を設定します。

注: SAP NetWeaver のサポートされているバージョンについては、「*Compatibility Guide*」を参照してください。

以下の手順に従います。

1. SAP Configtool（configtool.bat）を起動します。
2. 変更するインスタンスを選択します。
3. 右側のペインで、［VM Parameters］ - ［System］を選択します。
4. パラメータを作成します（-D を指定しません）。 例：
   名前： `com.wily.introscope.agentProfile`
   値： `<Agent_Home>/core/config/IntroscopeAgent.profile`
5. ［Additional］タブをクリックして、パラメータを作成します。たとえば、以下のように作成します。
   名前： `-javaagent:<Agent_Home>¥Agent.jar`
   値： `<leave empty>`
6. 変更を保存します。
7. SAP サーバを再起動します。
8. Configtool で行った変更が完了したことを確認するには、以下のファイルを開きます。
   `<ドライブ>:¥usr¥sap¥...¥j2ee¥cluster¥instance.properties`
9. *ID<サーバID>.JavaParameters* で始まる行を探し、入力した情報が含まれていることを確認します。

Enterprise Manager への接続の設定

メトリックをレポートするには、エージェントを Enterprise Manager に接続する必要があります。デフォルトの通信設定では、エージェントはポート 5001 を使用して、ローカルの Enterprise Manager に接続できます。 ただし、通常はエージェントと Enterprise Manager は同じシステム上には配置されていません。デフォルト設定の変更は、エージェントのインストール時に行うことができます。または、エージェントのインストール後に、*IntroscopeAgent.profile* ファイルを変更して行うこともできます。

要件に応じて、エージェントと Enterprise Manager の間で使用する通信を設定できます。

- 直接ソケット接続
- HTTP トンネル接続またはプロキシ サーバ経由の HTTP トンネル
- HTTP over Secure Sockets Layer 接続（HTTPS）
- Secure Socket Layer （SSL）接続

直接ソケット接続を使用した Enterprise Manager への接続

エージェントから Enterprise Manager に接続するための最も一般的な方法は、直接ソケット接続です。 可能な限り、Enterprise Manager への直接ソケット接続を使用することをお勧めします。

エージェントから Enterprise Manager への直接ソケット接続を設定する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* プロパティを探し、エージェントがデフォルトで接続する Enterprise Manager のホスト名または IP アドレスを指定します。 例：
   `introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=sfcollect01`
   複数の Enterprise Manager が存在するクラスタを使用する場合は、必ずコレクタ Enterprise Manager を指定します。
3. introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT プロパティを Enterprise Manager のリスニング ポートに設定します。 例：
   `introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001`

4. introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT プロパティを Enterprise Manager への接続に使用されるソケット ファクトリに設定します。 例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.DefaultSocketFactory
   ```
5. （オプション）プライマリ Enterprise Manager への接続が失われた場合に備えて、エージェントが接続できる 1 つ以上のバックアップ Enterprise Manager を指定します。
6. *IntroscopeAgent.profile* ファイルを保存して閉じます。

HTTP トンネルを使用した Enterprise Manager との接続

Enterprise Manager への直接ソケット接続が実現可能でない場合は、HTTP 経由で Enterprise Manager に接続するようにエージェントを設定できます。この設定によって、HTTP トラフィックのみが許可されているファイアウォールをパススルーして通信できるようになります。

注: HTTP トンネルを使用すると、アプリケーション サーバおよび Enterprise Manager にかかる CPU とメモリのオーバーヘッドが、直接ソケット接続の場合より大きくなります。

エージェント用に HTTP トンネルを設定する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* プロパティを、エージェントがデフォルトで接続する Enterprise Manager のホスト名または IP アドレスに設定します。 例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=webhost
   ```
3. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* プロパティを、Enterprise Manager の埋め込み Web サーバの HTTP リスニング ポートに設定します。 例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8081
   ```
   このプロパティは、Enterprise Manager 上の *<EM_Home>/config/IntroscopeEnterpriseManager.properties* ファイルで指定された *introscope.enterprisemanager.webserver.port* プロパティの値と一致する必要があります。デフォルトでは、このポートは 8081 です。

4. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* プロパティを、HTTP トンネル ソケット ファクトリに設定します。例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory
   ```
5. *IntroscopeAgent.profile* ファイルを保存して閉じます。

HTTP トンネルのためのプロキシ サーバの設定

HTTP トンネルで接続しているエージェントが、プロキシ サーバ経由で Enterprise Manager に接続するように設定できます。 これは、フォワード プロキシ サーバ構成で必要です。この構成では、エージェントはプロキシ サーバ経由の送信 HTTP トラフィックのみが許可されているファイアウォールの外部で実行されます。

プロキシ サーバの設定プロパティは、エージェントで HTTP トンネリングを行うように設定されている場合にのみ適用されます。 プロキシ サーバの設定は、単一の接続ではなく、エージェントに設定された HTTP トンネル接続すべてに適用されます。この設定は、それぞれの Enterprise Manager への接続に HTTP を使用している複数の Enterprise Manager 間のフェールオーバを設定する際に、特に重要になります。

重要: エージェントの HTTP トンネルを有効にするには、HTTP/1.1 が必要です。また、プロキシ サーバは HTTP Post もサポートする必要があります。

重要: プロキシがアクセス可能でない場合、エージェントはプロキシを経由せず、Enterprise Manager に直接接続します。 プロキシがアクセス可能で認証が失敗する場合、エージェントはプロキシ経由で Enterprise Manager への接続を再試行します。

HTTP トンネル用にプロキシ サーバを設定する方法

1. IntroscopeAgent.profile ファイルをテキスト エディタで開きます。
2. *introscope.agent.enterprisemanager.transport.http.proxy.host* プロパティを、プロキシ サーバのホスト名または IP アドレスに設定します。
3. *introscope.agent.enterprisemanager.transport.http.proxy.port* プロパティを、プロキシ サーバのポート番号に設定します。
4. （オプション）プロキシ サーバで認証用のユーザ クレデンシャルが必要な場合は、以下のプロパティを設定します。
   `introscope.agent.enterprisemanager.transport.http.proxy.username=<`*ユーザ名*`>`
   `introscope.agent.enterprisemanager.transport.http.proxy.password=<`*ユーザ パスワード*`>`
5. IntroscopeAgent.profile ファイルを保存して閉じます。

HTTPS トンネルを使用した Enterprise Manager との接続

HTTP over Secure Sockets Layer （SSL）を使用してエージェントから Enterprise Manager へ接続できるようにするには、*IntroscopeAgent.profile* ファイル内のプロパティを設定します。

HTTPS 経由での接続を設定する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* プロパティを、ターゲット Enterprise Manager のホスト名または IP アドレスに設定します。
3. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* プロパティを、Enterprise Manager の埋め込み Web サーバの HTTPS リスニング ポートに設定します。 例：
   `introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444`
4. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* プロパティを、HTTPS トンネル ソケット ファクトリに設定します。例:
   `introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory`
5. *IntroscopeAgent.profile* ファイルを保存して閉じます。

SSL 経由での Enterprise Manager との接続

Secure Socket Layer （SSL）の直接接続を使用する場合は、HTTP トンネルなしの SSL を使用して、Enterprise Manager とのエージェント通信を設定できます。

SSL を使用して接続するようにエージェントを設定する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. SSL ソケット ファクトリを使用して、エージェントを Enterprise Manager の SSL リスニング ポートに接続するよう設定します。
3. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* プロパティを、ターゲット Enterprise Manager のホスト名または IP アドレスに設定します。
4. *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* プロパティを、Enterprise Manager の SSL リスニング ポートに設定します。
5. *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* プロパティを、SSL ソケット ファクトリに設定します。 たとえば、以下のようにプロパティを設定します。
   ```
   com.wily.isengard.postofficehub.link.net.SSLSocketFactory
   ```
6. エージェントが Enterprise Manager を認証する必要がある場合は、*introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT* プロパティのコメント化を解除し、Enterprise Manager の証明書が含まれるトラストストアの場所に設定します。 トラストストアを指定しないと、エージェントはすべての証明書を信頼します。エージェント プロファイルの絶対パスまたは相対パスを指定できます。 例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/certs
   ```
7. 必要に応じて、*introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT* プロパティを設定して、トラストストア パスワードを指定します。
8. Enterprise Manager がクライアント認証を要求する場合は、*introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT* プロパティを、エージェントの証明書が含まれるキーストアの場所に設定します。エージェント プロファイルの絶対パスまたは相対パスを指定できます。 Windows では、円記号をエスケープ処理する必要があります。 例：
   ```
   introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=C:¥¥keystore
   ```

9. 必要に応じて、*introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT* プロパティを、キーストア パスワードに設定します。
10. *introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT* プロパティを、許可された暗号スイートのカンマ区切りリストに設定します。

    このプロパティの値を指定しない場合、デフォルトの暗号スイートが使用されます。
11. *IntroscopeAgent.profile* ファイルを保存して閉じます。

エージェント負荷分散の設定

エージェントによってレポートされるメトリックが主な作業負荷であるようなクラスタでは、MOM エージェント負荷分散を設定することで、全体的なクラスタ キャパシティを最適化できます。

注: MOM エージェント負荷分散の詳細については、「CA APM 設定および管理ガイド」を参照してください。

複数のエージェント タイプのアップグレード

環境によっては何千ものエージェントがさまざまな異なるアプリケーション サーバに配布されている場合があります。 たとえば、ある環境では 8,000 のエージェントがあり、WebLogic に 3,000 のエージェント、WebSphere に 2,000 のエージェント、JBoss に 3,000 のエージェントがそれぞれあるとします。 複数のアプリケーション サーバにまたがったエージェント アップグレードをより簡単に行うには、エージェント スーパーセット パッケージを使用できます。 エージェント スーパーセット パッケージはオペレーティング システム固有のパッケージであり、SAP NetWeaver を除く、サポート対象となるすべてのアプリケーション サーバのファイルが含まれます。

以下のエージェント スーパーセット パッケージが利用可能です。

- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.windows.zip
- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.unix.tar
- IntroscopeAgentFiles-NoInstaller<*version*>allappserver.zOS.tar
- IntroscopeAgentFiles-NoInstaller<バージョン>allappserver.os400.zip

注: これらのファイル名で、<*バージョン*> は、Java Agent の現在のリリース番号を示しています。

各パッケージには以下のファイルが含まれます。

- アプリケーション サーバ固有のすべての PBD および PBL
- アプリケーション サーバのすべてのエージェント プロファイル。該当するアプリケーション サーバ名がファイル名の一部に使用されています。 例 :
  - IntroscopeAgent.weblogic.profile
  - IntroscopeAgent.websphere.profile

  注: デフォルトの IntroscopeAgent.profile は含まれていません。詳細については、手順 3 を参照してください。
- オペレーティング システムのタイプに適したすべてのエージェント JAR ファイルおよびプラットフォーム モニタ

以下の手順に従います。

1. ターゲットのオペレーティング システムに適したスーパーセット パッケージを選択します。
2. 選択したエージェント パッケージをアプリケーション サーバのホーム ディレクトリに解凍します。

   注: 使用しないアプリケーション サーバを参照する <*Agent_Home*>/core/config ディレクトリ内の余分な PBD および PBL ファイルは、無視できます。
3. IntroscopeAgent.profile をまだ設定して*いない* 場合は、以下の手順に従います。
   a. 適切な IntroscopeAgent.<*アプリケーション サーバ名*>.profile を選択します。
   b. その名前を IntroscopeAgent.profile に変更します。
   c. 環境での使用に合わせてファイルを設定します。

4. IntroscopeAgent.profile をすでに設定している場合は、以下の手順に従います。
   a. 対応する IntroscopeAgent.<*a アプリケーション サーバ名*>.profile をエディタで開きます。
   b. 使用する新しいプロパティを探します。
   c. これらのプロパティを既存の IntroscopeAgent.profile に転送します。

詳細:

インストール アーカイブを使用した手動インストール (P. 36)

Java Agent のアンインストール

Java Agent をアンインストールするには、監視対象のアプリケーションごとに、Java Agent がインストールされている場所を把握しておく必要があります。

Java Agent インストーラを使用して Java Agent をインストールした場合は、アンインストール プログラムを使用して、インストールされているエージェント ファイルを削除できます。 アンインストーラを起動し、画面の指示に従います。

サポート対象の JVM から Java Agent をアンインストールする方法

1. アプリケーション サーバを停止します。

   注: アンインストール プログラムを実行する前に、監視対象の JVM をシャットダウンする必要があります。

2. アプリケーション サーバの起動スクリプトを開き、Java コマンド ラインから Java Agent スイッチを削除します。 アプリケーション サーバのタイプに応じて、起動オプションには以下のような記述が含まれます。
   - *-Xbootclasspath*
   - *-javaagent:<Agent.jar のパス>*
   - *-Dcom.wily.introscope.agentProfile=<エージェント プロファイルのパス>*

3. アプリケーション サーバを再起動します。
4. *<Agent_Home>*/UnstallerData ディレクトリ内にある *Uninstall_Introscope_Agent* プログラムを実行します。
5. *<Agent_Home>* ディレクトリを手動で削除します。

z/OS からの Java Agent のアンインストール

z/OS から Java Agent をアンインストールするための推奨方法は、*<Agent_Home>* ディレクトリを *rm -rf* コマンドを使用して削除する方法です。 これは、サードパーティのバグにより、実行形式のアンインストーラが z/OS 上で正しく動作しないために必要になります。

重要: 監視対象の JVM をシャットダウンした後にのみ、ファイルを削除します。

z/OS 上で Introscope のインストールをアクティブにしておくには、*UninstallerData* フォルダをそのままにしておくことが重要です。 *UninstallerData* フォルダを削除すると、今後 Introscope のバージョンをアップグレードできなくなってしまいます。 インスタンス全体をアンインストールするよう決定した場合を除き、*UninstallerData* フォルダは削除しないでください。

第 3 章: エージェント プロパティの設定

ほとんどのエージェント オペレーションは IntroscopeAgent.profile ファイル内のプロパティを使用して設定します。 このセクションでは設定の際に最もよく使用されるエージェント プロパティについて説明します。 使用している環境によっては追加のプロパティが必要な場合があります。 エージェントのバージョンが異なると、設定に使用できるプロパティが異なっていたり、デフォルト値が異なっていたりする場合があります。

このセクションには、以下のトピックが含まれています。

Enterprise Manager との通信を変更する方法

メトリックをレポートするには、エージェントを Enterprise Manager に接続する必要があります。 エージェントのインストール後、使用する通信チャネルを変更するには、*IntroscopeAgent.profile* ファイルを変更します。

要件に応じて、エージェントと Enterprise Manager の間で使用する通信を設定できます。

- 直接ソケット接続
- HTTP トンネル接続またはプロキシ サーバ経由の HTTP トンネル
- HTTP over Secure Sockets Layer 接続（HTTPS）
- Secure Socket Layer （SSL）接続

エージェントから Enterprise Manager に接続するための最も一般的な方法は、直接ソケット接続です。 可能な限り、Enterprise Manager への直接ソケット接続を使用することをお勧めします。 別のタイプの接続を使用する場合は、該当のセクションを参照してください。

バックアップの Enterprise Manager およびフェールオーバ プロパティを設定する方法

エージェントのインストール時には、エージェントがデフォルトで接続する Enterprise Manager のホスト名とポート番号を指定します。 オプションで、1 つ以上のバックアップ Enterprise Manager を指定することもできます。 エージェントとプライマリ Enterprise Manager の接続が切断された場合、エージェントはバックアップ Enterprise Manager への接続を試行できます。

エージェントがバックアップ Enterprise Manager に接続できるようにするには、エージェント プロファイルで Enterprise Manager の通信プロパティを指定します。 プライマリ Enterprise Manager が利用可能でない場合、エージェントは、許可されている接続のリスト上で次に利用できる Enterprise Manager に接続を試みます。 リスト内の最初のバックアップ Enterprise Manager に接続できない場合は、その次の Enterprise Manager を試します。 プロセスは、接続が成功するまで、リストに記載された順に各 Enterprise Manager への接続を試行します。 どの Enterprise Manager にも接続できない場合は、再試行する前に 10 秒間待機します。

以下の手順に従います。

1. *Introscope.Agent.profile* ファイルをテキスト エディタで開きます。
2. 各バックアップ Enterprise Manager のエージェント プロファイルに以下のプロパティを追加して、1 つ以上の代替 Enterprise Manager 通信チャネルを指定します。

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

   NAME を新しい Enterprise Manager チャネルの識別子と置き換えます。チャネルを作成するときには、名前に「DEFAULT」または既存のチャネルの名前を使用しないでください。 たとえば、2 つのバックアップ Enterprise Manager を作成するには、以下のように指定します。

   ```
   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* プロパティを探し、プライマリおよびバックアップ Enterprise Manager の識別子のカンマ区切りリストを設定します。 識別子を登録する順序によって、それらが接続される順序が決まります。 例：
   `introscope.agent.enterprisemanager.connectionorder=DEFAULT,BackupEM1,BackupEM2`
4. *introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds* プロパティを探し、エージェントがプライマリ Enterprise Manager に接続を試みる頻度を指定します。デフォルトの間隔は 120 秒（2 分）です。例：
   `introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120`
5. 変更を保存し、*IntroscopeAgent.profile* ファイルを閉じます。
6. アプリケーションを再起動します。

追加の GC メトリックを有効にして使用する方法

ガベージ コレクションおよびメモリ管理は、アプリケーションのパフォーマンスに重要な効果をもたらします。基本的な GC ヒープ メトリックはデフォルトで利用できます。 ガベージ コレクション処理およびメモリ プール使用量に関する詳細を追加で取得できるようにするためのオプションのメトリックもあります。 このような追加メトリックは、有効にすると、Investigator の［GC 監視］ノードに表示されます。GC 監視メトリックによって、メモリ プール割り当ておよびガベージ コレクション処理を最適化するのに役立つ情報がレポートされます。 そのため、アプリケーションを開発またはテストを行うときや、アプリケーション パフォーマンスの問題を調査するときは、通常、これらのメトリックが有効にされます。 ほとんどの場合、これらのメトリックは実運用環境におけるリアルタイムのアプリケーション管理には使用されず、デフォルトで無効になっています。

GC 監視メトリックを有効にする場合は、テキスト エディタでエージェント プロファイルを開き、*introscope.agent.gcmonitor.enable* プロパティを true に設定します。 プロパティを有効にすると、監視している JVM でのガベージ コレクションおよびメモリ プールに関する詳細を表示できます。メトリックの詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

スレッド ダンプを有効にして設定する方法

スレッド ダンプからは、エージェント JVM 内で発生していることに関する有用な詳細情報を得ることができます。 スレッド ダンプの機能は、メトリック ブラウザ ツリー内の各エージェント ノードに関連付けられている［スレッド ダンプ］タブで提供されます。

スレッド ダンプの収集および分析については、「CA APM Workstation ユーザ ガイド」を参照してください。 Thread_Dump 権限を設定すると、［スレッド ダンプ］タブが表示され、そのすべての機能を使用できるようになります。 詳細については、「CA APM セキュリティ ガイド」を参照してください。

IntroscopeAgent.profile プロパティと *IntroscopeEnterpriseManager.properties* プロパティはどちらも、スレッド ダンプを有効にするために必要です。デフォルトでは、［スレッド ダンプ］タブおよびその機能は有効です。 ただし、スレッド ダンプ プロパティのいずれか一方または両方のプロパティを false に設定した場合、［スレッド ダンプ］タブは表示されません。

MOM 上でスレッド ダンプを有効または無効にした場合、その設定はクラスタ内のすべてのコレクタに適用されます。 そのため、MOM 上でスレッド ダンプを無効にすると、すべてのコレクタ上でも無効になります。

スレッド ダンプを有効にする方法

1. <*Agent_Home*>/wily/core/config ディレクトリの *IntroscopeAgent.profile* ファイルを開き、以下のプロパティを設定します。
   ```
   introscope.agent.threaddump.enable=true
   ```
2. *IntroscopeAgent.profile* ファイルを保存して閉じます。
3. <*EM_Home*>/config ディレクトリの *IntroscopeEnterpriseManager.properties* ファイルを開き、以下のプロパティを設定します。
   ```
   introscope.enterprisemanager.threaddump.enable=true
   ```
4. *IntroscopeEnterpriseManager.properties* ファイルを保存して閉じます。

CA Introscope® ユーザがデッドロック数メトリックを表示するためには、*IntroscopeAgent.profile* を設定します。追加の設定を行うと、エージェントの［スレッド］ノードのメトリックを表示できます。

デッドロック数メトリック コレクションを有効にする方法

1. *<Agent_Home>*/wily/core/config ディレクトリの *IntroscopeAgent.profile* ファイルを開きます。
2. デッドロック数メトリック コレクションを有効にするには、以下のプロパティを true に設定します。
   `introscope.agent.threaddump.deadlockpoller.enable=true`
3. （オプション） エージェントの［スレッド］ノードにメトリックを表示するには、フル バージョンの PBL を設定します。
   - PBL ファイルの名前を、このプロパティで指定します : introscope.autoprobe.directivesFile。

     たとえば、WebLogic Server のフル バージョンの標準 PBL を使用するには、以下のようにプロパティを設定します。
     `introscope.autoprobe.directivesFile=weblogic-full.pbl`

     ［*<エージェント>*］ | ［スレッド］の下に、アクティブ スレッド数やスレッド グループなどのメトリックを表示できます。
4. *IntroscopeAgent.profile* ファイルを保存して閉じます。

IntroscopeAgent.profile プロパティと *IntroscopeEnterpriseManager.properties* プロパティの両方を使用して、スレッド ダンプを設定します。

スレッド ダンプを設定する方法

1. *<EM_Home>*/config ディレクトリにある *IntroscopeEnterpriseManager.properties* ファイルを開きます。
2. （オプション）Enterprise Manager 上の特定のディレクトリにスレッド ダンプ ファイルを保存するには、以下のプロパティを設定します。たとえば、TestThreadDumps となります。
   `introscope.enterprisemanager.threaddump.storage.dir=TestThreadDumps`
3. （オプション）指定した日数より古いスレッド ダンプ ファイルをパージするには、以下のプロパティを設定します 30 日の場合は以下のように指定します。
   `introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days=30`

4. （オプション）指定した日数が経過したらスレッド ダンプ ファイルをパージするには、以下のプロパティを設定します 2 日おきの場合は以下のように指定します。
   `introscope.enterprisemanager.threaddump.storage.clean.disk.freq.days=2`
5. （オプション） Enterprise Manager 上に保存できるスレッド ダンプ ファイルの最大数を制限するには、以下のプロパティを設定します。5,000 ファイルの場合は以下のように指定します。
   `introscope.enterprisemanager.threaddump.storage.max.disk.usage=5000`

   注: 以下の条件があります。

   * 保存されるスレッド ダンプ ファイルの数が、introscope.enterprisemanager.threaddump.storage.max.disk.usage プロパティで設定された制限を超えている

   および

   * introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days プロパティで設定された日数より古いファイルがない

   これらの条件が当てはまる場合、Enterprise Manager はスレッド ダンプ ファイルを保存しません。
6. *IntroscopeEnterpriseManager.properties* ファイルを保存して閉じます。
7. Enterprise Manager を再起動します。

Enterprise Manager がダウンした場合は、スレッド ダンプ ファイルを別の Enterprise Manager にコピーすれば、スレッド ダンプ データを表示できます。

重要: スレッド ダンプ ディレクトリでファイルの追加または削除を行ったら、Enterprise Manager を再起動します。 Enterprise Manager 間でスレッド ダンプ ファイルを移動することは推奨されません。

Enterprise Manager 間でスレッド ダンプ ファイルをコピーする方法

1. スレッド ダンプ ファイルが含まれる Enterprise Manager （EM1）上の <*EM_Home*>/threaddumps ディレクトリに移動します。
2. スレッド ダンプ ファイルをコピーします。

3. スレッド ダンプの表示先となる Enterprise Manager （EM2）の <*EM_Home*>/threaddumps ディレクトリにファイルを貼り付けます。
4. Enterprise Manager EM1 および EM2 の両方を再起動します。
5. 必要に応じて、エージェント接続を確立し、EM2 上のスレッド ダンプを有効にして設定します。

   EM2 ユーザは、エージェント ノードを選択し、［スレッド ダンプ］タブの［以前のデータをロード］ボタンをクリックできます。 EM1 から移動されたスレッド ダンプがリストに表示されます。

分布統計メトリックを収集するようにエージェントを設定する方法

Average Response Time メトリックを作成するために、BlamePointTracer によって分析された応答時間の分布情報を収集するようにエージェントを設定できます。 分布統計メトリックは、特定のオペレーションがどのように変化するかについて詳細な情報を提供します。 監視する応答時間値の分布統計の詳細なデータは、getExtendedMetricData Web サービスを介して利用できます。

CA APM エージェントは、特定のオペレーションの応答時間情報を収集するように設定できます。 応答時間は分布統計メトリックに格納されます。分布統計メトリックは、選択したオペレーションの Average Response Time メトリックとペアになります。

以下の手順に従います。

1. <*Agent_Home*>/wily/core/config ディレクトリの *IntroscopeAgent.profile* ファイルを開きます。
2. ペアとなる Distribution Statistics メトリックを作成する Average Response Time メトリックを指定するために、introscope.agent.distribution.statistics.components.pattern のコメント化を解除して編集します。たとえば、以下の一致パターンを使用します。

   ```
   Servlets¥¥|LoginServlet:.*
   ```

   LoginServlet の応答時間の分布統計情報が生成されます。
3. （オプション）以下のガイドラインを利用して、独自の一致パターンを作成します。
   a. 縦棒とピリオドは正規表現において特別な意味を持つため、メトリック ノード区切り記号の前に円記号を付加します。
   b. 円記号はエージェント プロファイルで特別な意味を持つため、円記号の前に別の円記号を付加します。

      正規表現はエージェント ログで特殊文字の後に表示されます。例：

      ```
      Servlets¥|Login Servlet:.*
      ```
   c. サマリ レベルおよび個別のメトリックの正規表現に一致します。メトリックが一致しない場合は、分布統計情報がサマリ レベルに要約されるか、あるいは作成されません。

      以下の例では、一致する表現を示します。

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

      ```
      Servlets.*
      ```
4. IntroscopeAgent.profile ファイルを保存して閉じます。
5. エージェントを再起動します。

分布統計メトリックの例

設定プロパティの設定に応じて、以下に対する分布統計情報を収集できます。

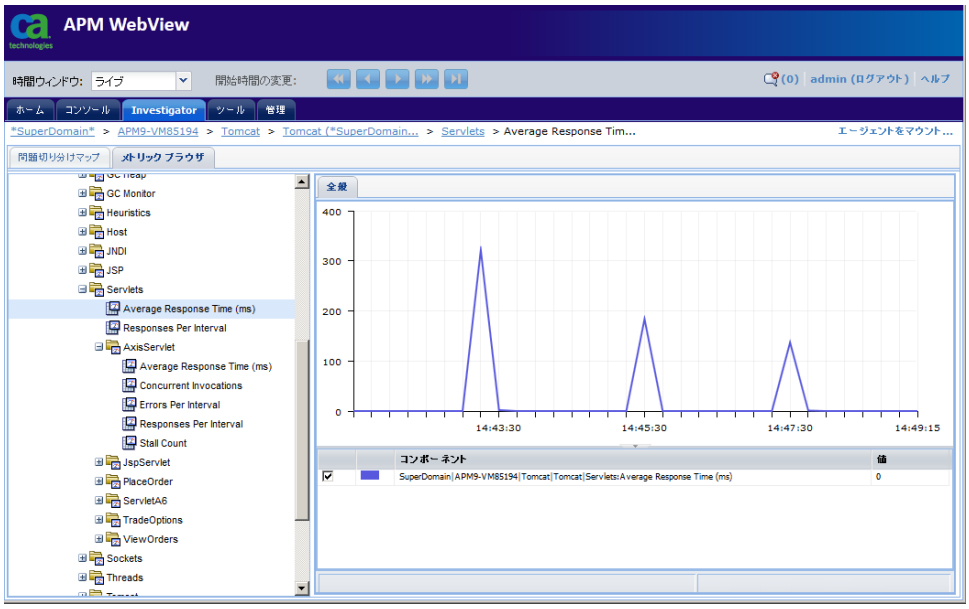
- 個別およびサマリ レベルのサーブレットおよび JSP 分布メトリックの例 (P. 89)
- サーブレットおよび JSP 分布メトリックの例 (P. 90)

個別およびサマリ レベルのサーブレットおよび JSP 分布メトリックの例

個別およびサマリ レベルのサーブレットおよび JSP の分布統計情報を収集するには、以下のパターンを使用します。

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

以下の図は、Java エージェント ノードのサーブレットおよびサマリ レベルの分布統計メトリックを収集する Investigator を示しています。

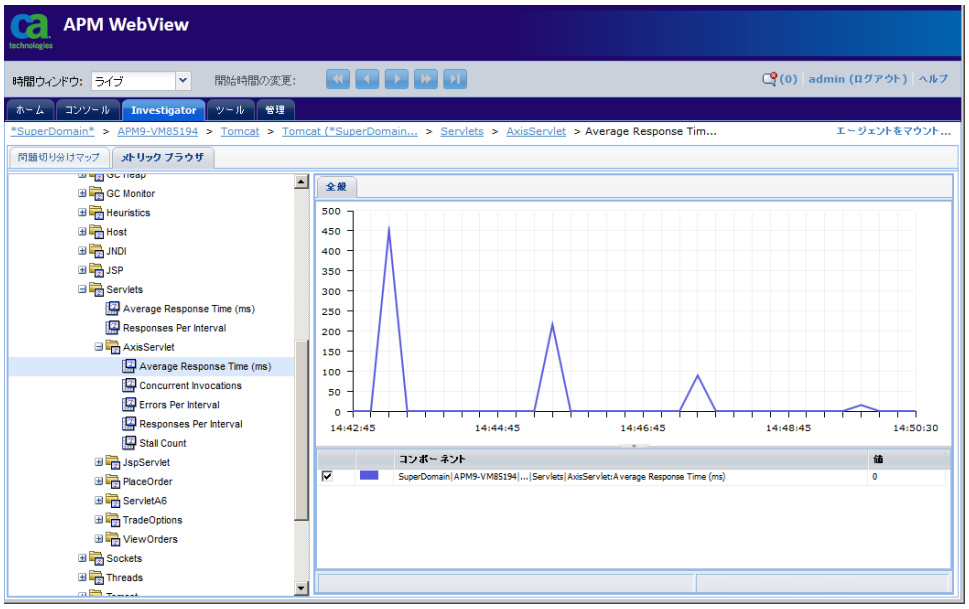


サーブレットおよび JSP 分布メトリックの例

サーブレットまたは JSP サマリ レベルではない、サーブレットおよび JSP レベルの分布統計情報を収集するには、以下のパターンを使用します。

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

以下の図は、Java エージェント ノードのサーブレットの（サマリ レベルではない）分布統計メトリックを収集する Investigator を示しています。



第 4 章: AutoProbe および ProbeBuilding オプション

このセクションには、以下のトピックが含まれています。

AutoProbe および ProbeBuilding の概要

Java エージェントは、監視対象アプリケーションのバイトコードにプローブを挿入します。 ProbeBuilding は、ProbeBuilder ディレクティブ（PBD）および ProbeBuilder リスト（PBL）を使用してアプリケーションに挿入するプローブを決定するプロセスです。Java エージェントに含まれるデフォルトの PBD および PBL ファイルは、基本的なレベルのメトリック コレクションを提供します。 お使いの環境に合わせてメトリック コレクションを調整するには、これらのファイル内のデフォルト設定を変更してください。

アプリケーションおよび環境で収集するメトリックのプローブを挿入するように PBD と PBL を設定します。次に、これらのファイルを使用して、JVM AutoProbe でアプリケーションを自動的にインスツルメントするか、ProbeBuilder を使用して手動でインスツルメントします。 アプリケーションへのインスツルメントには、JVM AutoProbe を使用します。ただし、JVM AutoProbe の設定は、アプリケーションの環境により異なります。

注: サポートされている JVM のバージョンについては、「*Compatibility Guide*」を参照してください。

JVM でアプリケーションをインスツルメントするには、以下のいずれかの方法を使用します。

- JVM AutoProbe と -javaagent プロパティ。 CA Technologies では、JVM で JVM AutoProbe を使用してアプリケーションをインスツルメントすることを強くお勧めします。
- Manual ProbeBuilding は高度なインスツルメンテーション テクニックです。 このメソッドを使用する前に CA サポート にお問い合わせください。

重要: アプリケーションをインスツルメントする場合は、1 つのインスツルメンテーション方法のみを使用します。

Java エージェントがインストールおよび設定されている (P. 27)ことを確認します。

サポートされていないインスツルメンテーション方法

Java エージェントでは、インスツルメンテーション時のアプリケーション サーバ用の AutoProbe メソッドはサポートされていません。 以前のバージョンの JVM （JVM 1.4 以前）と以前のバージョンの Java エージェントを使用するアプリケーションをインスツルメントするのに、アプリケーション サーバ用の AutoProbe を使用することができます。 JVM AutoProbe は、JVM 1.5 以降を使用するアプリケーションに対して使用することをお勧めします。

注: JVM のサポートされているバージョンについては、「*Compatibility Guide*」を参照してください。「*Compatibility Guide*」に記載がない場合は、サポートされている JVM はアプリケーション サーバ バージョンに付属している JVM です。

ProbeBuilding の構成

ProbeBuilding テクノロジによってインスツルメンテーション プロセスが実行されます。 ProbeBuilder ディレクティブ（PBD）ファイルで定義されたプローブが、実行時にエージェントが Web アプリケーションと仮想マシンから収集したメトリックを識別します。

デフォルトでは、AutoProbe は Java エージェントで提供される標準の PBD セットを使用します。 このセットアップによって、適度な数のメトリックが収集されます。 以下のオプションを使用すると、メトリック収集レベルのカスタマイズ、および ProbeBuilding の動作の設定が行えます。

完全（Full）または標準（Typical）追跡オプション

Introscope では、ProbeBuilder リスト（PBL）ファイルで、インスツルメンテーション プロセスで使用されるトレーサ グループを制御します。 *introscope.autoprobe.directivesFile* プロパティは、1 つ以上の PBL ファイルを指定します。

Introscope には、デフォルト PBL にそれぞれ 2 つのバージョンが用意されています。1 つは「*フル*」バージョンで、標準バージョンよりも大きなトレーサ グループ セットを指定でき、詳細なメトリック レポートが得られます。もう 1 つは「*標準*」バージョンで、より小さいトレーサ グループ セットを指定できます。 その結果、メトリック レポートが簡潔になり、オーバーヘッドが軽減されます。 デフォルトで、*introscope.autoprobe.directivesFile* は、標準バージョンのデフォルト PBL ファイルを指定します。

フルと標準で追跡レベルを変更する方法

1. 管理対象アプリケーションを停止します。
2. *IntroscopeAgent.profile* ファイルをテキスト エディタで開きます。
3. 使用する PBL ファイルの名前を *introscope.autoprobe.directivesFile* で指定します。

   たとえば、WebLogic Server のフル バージョンの標準 PBL を使用するには、以下のようにプロパティを設定します。

   ```
   introscope.autoprobe.directivesFile=weblogic-full.pbl
   ```
4. 管理対象アプリケーションを再起動します。

動的 ProbeBuilding

CA Introscope® は、動的 ProbeBuilding を使用して、管理対象アプリケーションまたはエージェントを再起動することなく、新規または変更された PBD を実装できます。 動的 ProbeBuilding は、PBD を修正する場合や、問題の切り分けまたは診断中にアプリケーション サービスを中断せずにデータ収集レベルを一時的に変更する場合に便利です。

重要: 動的 ProbeBuilding は、Java 1.5 以降を使用している場合にのみ利用可能です。 動的 ProbeBuilding は Java 1.5 の機能、および -javaagent コマンドに依存しています。

注: Workstation では、トランザクション追跡ビューアから動的インスツルメンテーションを実行できます。 詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

動的 ProbeBuilding によって、CA Introscope® は、新規および変更済みの PBD を定期的に検出します。オーバヘッドを最小化するために、CA Introscope® は、変更された PBD によって影響を受けるクラスだけを選択的に再インストルメントします。 パフォーマンスを向上させるため、エージェントの動的インスツルメンテーションの範囲は、PBD が編集されたときにインスツルメンテーションが変更されたクラスの再ロードに限定されています。

PBD が編集または hotdeploy ディレクトリに追加された場合、ユーザ ディレクティブ（クラスのディレクティブの追加または削除、またはトレーサ グループの切り替え）のみが再インスツルメントされます。

重要: トレーサ グループを使用するディレクティブへの変更だけがサポートされています。たとえば、IfFlagged スイッチを持っている TraceAllMethods などのディレクティブへの変更です。また、CA Introscope® では、変更なしに使用できる、トレーサ グループまたはフラグを持つディレクティブのみが含まれています。 スキップまたは変換に対する変更はサポートされていません。

以下のディレクティブは再インスツルメントされません。

- トレーサ の追加または新しいトレーサ マッピングの変更などのシステム ディレクティブ
- Skip ディレクティブで指定された配列、インターフェース、クラス、および任意の変換。

以下の再インスツルメントプロセスを設定できます。

- 特定のクラスローダがロードするクラスをすべて除外する。
- 範囲を特定のクラス パッケージに制限する。

注: 動的 ProbeBuilding はデフォルトでは無効になっています。

メトリックのデータをレポートしないようにクラスを再インスツルメントしても、メトリックは、Investigator に表示されたままになります。 既存のメトリックは、クラスが再インスツルメントされても、Investigator ウィンドウからは消えません。

重要: Java 1.5 の制限のために、一部のクラス バイトにアクセスすることができず、そのため以下のような影響があります。

- j2ee.pbd ファイルへの変更がピック アップされず、メトリックは引き続き古い名前のままで公開されます。
- エージェント ログにいくつかの例外が表示されます。
- この問題を回避するには、j2ee.pbd ファイルを変更した後に、アプリケーション サーバを再起動します。

動的 ProbeBuilding を設定する場合、トレーサ グループでの変更を基準にすることをお勧めします。

例: トレーサ グループ XYZ のインスツルメンテーションのレベルを制御します。

この例は、トレーサ グループのインスツルメンテーションのレベルを制御する方法を示します。

以下の手順に従います。

1. 以下の 2 つのトレーサ グループを作成します。
   - XYZTracing - 通常の追跡オプション
   - XYZTracingLite - 限定されたコンポーネントだけが追跡されます

2. トレーサ グループの切り替え： XYZTracing をオフにし、XYZTracingLite をオンにします。
3. 動的 ProbeBuilding が環境のパフォーマンスに与える影響を確認します。
4. それに応じてトレーサ グループを調整します。

   調整は、各トレーサ グループの一部として追跡中のすべてのクラスに影響します。

動的 ProbeBuilding の設定

動的 ProbeBuilding を設定するには、IntroscopeAgent.profile を編集します。

以下の手順に従います。

1. <*Agent_Home*>/wily/core/config ディレクトリに移動します。
2. IntroscopeAgent.profile ファイルをテキスト エディタで開きます。
3. プロパティ introscope.autoprobe.enable が true に設定されていることを確認します。
4. 以下のプロパティのコメント化を解除して、設定します。
   - introscope.autoprobe.dynamicinstrument.enabled=true

     このプロパティを使用すると、動的 ProbeBuilding が有効になります。 このプロパティは、管理対象アプリケーションの再起動後に有効になります。
   - introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1

     PBD の変更をチェックするためのポーリング間隔（分単位）。 デフォルトは 1 分間隔に設定されています。 このプロパティは、管理対象アプリケーションの再起動後に有効になります。
   - introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1

     一部のクラスローダの実装では、非常に大きなクラス ファイルを返すように設定されている場合があります。 この動作はメモリ エラーを防ぎます。 このプロパティは、管理対象アプリケーションの再起動後に有効になります。

- introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10

  一度に非常に多くのクラスを再定義すると、CPU に過大な負荷がかかる可能性があります。 PBD の変更によって多くのクラスを再定義することになる場合、このプロパティを使用してプロセスをいくつかにまとめて処理し、適切なレートになるようにできます。

5. IntroscopeAgent.profile ファイルへの変更を保存し、ファイルを閉じます。
6. 管理対象アプリケーションを再起動します（必要な場合）。

動的インスツルメンテーションの IBM JDK のパフォーマンスへの影響

対象: IBM JDK バージョン 5 と組み合わせた CA Introscope®

注: CA Introscope® と IBM JDK バージョン 6 を一緒に使用する場合は、クラスの再定義の使用によるパフォーマンスのオーバーヘッドは発生しません。

症状:

動的インスツルメンテーションでは、クラスの再定義をサポートする必要があります。 クラスの再定義を使用するとパフォーマンスに著しい影響を与える場合があります。IBM は、このパフォーマンスのオーバーヘッドに関する技術情報を「Java Diagnostics Guide」に掲載しています。 CA Introscope® および IBM JDK のユーザが動的インスツルメンテーションを利用する場合、このパフォーマンス オーバーヘッドについて注意する必要があります。

解決方法:

実稼働環境でのみ動的インスツルメンテーション機能を使用することをお勧めします。

詳細:

UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 での WebSphere Application Server 6.1 の設定 (P. 56)
「ソケット メトリック」 (P. 162)

動的 ProbeBuilding と動的インスツルメンテーション

動的 ProbeBuilding と動的インスツルメンテーションは異なります。

- 動的 ProbeBuilding (P. 94) は、PBD ファイルに対して手動で行った変更と、IntroscopeAgent.profile に対して行った手動による設定に基づいています。 PBD ファイルを更新または変更して適切な場所に保存すると、動的 ProbeBuilding はその変更をインスツルメントします。 動的 ProbeBuilding では、IntroscopeAgent.profile を設定し、更新する PBD ファイルを変更する必要があります。 すべての変更は、手動でそのファイルをもう一度更新または変更するまで永続します。

- 動的インスツルメンテーションは Workstation のトランザクション追跡ビューアから実行されます。インターフェースを使用して選択したインスツルメンテーションへの変更は自動的に行なわれ、多くの場合は一時的なものです。 メソッドを動的にインスツルメントするということは、実行時にインスツルメンテーションを挿入することを意味します。 トランザクション追跡セッション中に、1 つ以上またはすべてのメソッドをインスツルメントできます。 その後、新しくインスツルメントされたメソッドが再度表示されます。 これにより、アプリケーションのパフォーマンス チューニングを動的に実行できます。動的インスツルメンテーションでは、IntroscopeAgent.profile の変更は必要ありません。インスツルメンテーションの変更を恒久的なものにする場合は、PBD が適切な場所に作成され、保存されます。

  注: Workstation トランザクション追跡ビューアから動的インスツルメンテーションを使用する方法の詳細については、「*CA APM Workstation ユーザ ガイド*」を参照してください。

  重要: CA Introscope® は、アプリケーション サーバのホスト コンピュータ上に動的インスツルメンテーション キャッシュを格納します。 動的インスツルメンテーションが動作するためには、アプリケーション サーバが Enterprise Manager へアクセスする際に使用するユーザ アカウントが、アプリケーション サーバのホスト コンピュータ上の *<Agent_Home>* および *<Agent_Home>*/logs ディレクトリへの書き込みアクセスを持っている必要があります。

ProbeBuilding のクラス階層

1.5 より前の JVM では、クラス階層の下位レベルにあるクラスに対するインストルメンテーションは自動的には行われません。 CA Introscope® は、プローブされるクラスを明示的に拡張するクラスのみをインスツルメントします。

サポートされている JVM では、プローブされるクラスの複数のサブクラスレベルをインスツルメントするように CA Introscope® を設定できます。 関連する内部ディレクティブのトレーサ グループは適切に更新され、クラスは動的にインスツルメントされます。 ディレクティブの変更は、ログファイルに書き込まれます。

注: サポートされている JVM については、「*Compatibility Guide*」を参照してください。

PBD を手動で更新する場合は、ディレクティブの更新を無効にし、ログファイルを使用して適切な更新を判断できます。

詳細:

インスツルメントおよび継承 (P. 142)

複数レベルのサブクラスのインスツルメンテーションの有効化

内部ディレクティブを動的に更新するように Introscope を設定するには、以下の手順に従います。

複数レベルのサブクラスのインスツルメンテーションを有効にする方法

1. 「動的 ProbeBuilding (P. 94)」の説明に従って、動的インスツルメンテーションが有効になっていることを確認します。
2. *IntroscopeAgent.profile* を開きます。
3. 複数レベルのサブクラスのインスツルメンテーションを有効にするには、このプロパティの設定のコメント化を解除します。
   `introscope.autoprobe.hierarchysupport.enabled=true`
4. *IntroscopeAgent.profile* を保存します。

複数の継承、インターフェース、および抽象メソッドのサポート

Java エージェントは、インターフェースおよび継承によるインスツルメンテーションをサポートします。この機能は、動的インスツルメンテーションで拡張されます。

Java エージェントでは、API の getMethodCalls を使用したサブクラスの呼び出しによって、メソッドのインスツルメンテーションをサポートします。getMethodCalls を使用すると以下の情報が得られるため、このメソッドを使用して継承されたメソッドまたはインターフェース メソッドのインスツルメンテーションの結果をより深く理解できます。

- メソッドを定義するクラスがインターフェースであるかどうか。
- メソッドの実行可能なインスツルメンテーションによって影響を受けるクラスの数。この数は、サブクラスの数または実装クラスの数です。
- メソッドが特定のスタック トレースで呼び出されるかどうか。

以下の構文を使用すると、Java エージェントでインターフェースおよび抽象メソッドをインスツルメントするためのトレーサを使用できます。

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

以下の場合、このトレーサは、インターフェースを実装するクラスまたはスーパークラスの拡張クラスのメソッドをインスツルメントします。

- メソッドがインターフェースに定義されている場合。
- スーパークラスの抽象メソッドである場合。

重要: このトレーサを使用すると、システムのパフォーマンスに著しい影響を及ぼす可能性があります。 さらに大規模なエージェント設定にデプロイする前に、システムの起動時とインスツルメンテーションのプロセスの実行中にこのトレーサが与える影響をテストします。

詳細:

カスタム トレーサの作成 (P. 129)

インスツルメントされていないサブクラスの定期ポーリングの設定

複数レベルのサブクラスのインスツルメンテーションが有効になっている場合、Introscope は、アプリケーションの起動時に、インスツルメントされていないサブクラスをチェックします。

インスツルメントされていないサブクラスのポーリングを Introscope で設定する方法

1. *IntroscopeAgent.profile* を開きます。
2. このプロパティの設定のコメント化を解除します。
   `introscope.autoprobe.hierarchysupport.runOnceOnly=false`
3. Introscope がインスツルメントされていないサブクラスをポーリングする間隔をデフォルト値の 5 分から変更するには、このプロパティのコメント化を解除し、目的のポーリング間隔を設定します。
   `introscope.autoprobe.hierarchysupport.pollIntervalMinutes`
4. オプションで、Introscope がインスツルメントされていないサブクラスをポーリングする回数を制限できます。このためには、このプロパティのコメント化を解除し、目的の制限値を設定します。
   `introscope.autoprobe.hierarchysupport.executionCount`

   このプロパティのデフォルトは 3 分です。
5. *IntroscopeAgent.profile* を保存します。

ディレクティブの更新の無効化

複数レベルのサブクラスのインスツルメンテーションが有効になっている場合、Introscope は、インスツルメントされていないサブクラスを検出すると、デフォルトで、クラスが確実にインスツルメントされるように適切に内部ディレクティブを更新します。 PBD を手動で更新する場合は、*IntroscopeAgent.profile* にあるこのプロパティの設定のコメント化を解除して、内部ディレクティブの更新を無効にできます。

`introscope.autoprobe.hierarchysupport.disableDirectivesChange=true`

ディレクティブのログの制御

複数レベルのサブクラスのインスツルメンテーションを有効にする場合、*IntroscopeAgent.profile* にある以下のプロパティのコメント化を解除し、複数レベルのサブクラスのインスツルメンテーションのログが作成されるようにする必要があります。 これらのプロパティが設定されると、pbdupdate.log という名前のログ ファイルが *<Agent_Home>*/wily ディレクトリ（デフォルト）、またはカスタムの場所（指定した場合）に作成されます。 複数レベルのインスツルメンテーションの詳細が、エージェントのログに書き込まれます。

```
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_
```

これらのプロパティの変更を有効にするには、管理対象アプリケーションを再起動する必要があります。

バイトコード内の行番号の削除

アプリケーションのバイトコードをインスツルメントする場合、デフォルトでバイトコードの行番号が保持されます。 バイトコードの行番号情報を保持すると、デバッガを使用する場合、またはスタック トレース情報を取得する場合に役立ちます。

この機能は、Java コマンド ラインでシステム プロパティを追加することでオフにできます。 この機能をオフにすると、AutoProbe または ProbeBuilder がアプリケーション コードをインスツルメントするときにすべての行番号が削除されます。

AutoProbe または ProbeBuilder を使用するときにバイトコードで行番号を削除する方法

- Java コマンド ラインで、以下のシステム プロパティを **-D** オプションを指定して定義します。

  com.wily.probebuilder.removeLineNumbers=true

第 5 章: ProbeBuilder ディレクティブ

このセクションでは、ProbeBuilder ディレクティブの作成および変更方法について説明します。

このセクションには、以下のトピックが含まれています。

ProbeBuilder ディレクティブの概要

ProbeBuilder ディレクティブ（PBD）ファイルによって、アプリケーションをインスツルメントするために、Introscope ProbeBuilder がどのようにタイマやカウンタなどのプローブを追加するかが指定されます。PBD ファイルにより、エージェントが Introscope Enterprise Manager にレポートするメトリックを管理します。

注: すべてのメトリックは、システム クロックに設定された時刻を使用して計算されます。 システム クロックがトランザクション処理中にリセットされた場合、そのトランザクションでレポートされた経過時間は誤っている可能性があります。

Introscope には、デフォルトの PBD ファイルのセットが含まれています。また、アプリケーション固有の情報を取得するために、クラスまたはメソッドを追跡するカスタムの Introscope PBD ファイルを作成できます。

ProbeBuilder ディレクティブは 2 種類のファイルによって指定されます。

- **ProbeBuilder ディレクティブ（PBD）ファイル**

  ProbeBuilder ディレクティブ（PBD）ファイルには、アプリケーションをインスツルメントする際に ProbeBuilder が使用するディレクティブが含まれています。このファイルによって、エージェントが Enterprise Manager にレポートするメトリックが決定されます。

- **ProbeBuilder リスト（PBL）ファイル**

  ProbeBuilder リスト（PBL）ファイルには、複数の PBD ファイル名のリストが含まれます。異なる PBL ファイルで同じ PBD ファイルを参照できます。

重要: PBD と PBL は ASCII 文字のみをサポートしています。PBD または PBL に他の文字（Unicode 文字など）を挿入すると、AutoProbe で問題が発生する可能性があります。

Introscope AutoProbe を使用する場合、特定のアプリケーション サーバに関連する PBD および PBL ファイルは、Java Agent をインストールするときに追加されます。 これらのファイルは *<Agent_Home>*¥wily¥core¥config ディレクトリに配置されます。

詳細:

デフォルトの PBD ファイル (P. 105)
デフォルトの PBL ファイル (P. 110)
カスタム トレーサの作成 (P. 129)

デフォルト PBD のコンポーネント追跡

デフォルトの Introscope PBD ファイルは、以下の Java コンポーネントの追跡を実装します。

- Oracle JDBC
- JSP タグ ライブラリ
- JSP IO タグ ライブラリ
- JSP DB タグ ライブラリ
- Struts
- サーブレット
- JavaServer Faces (JSF)
- JavaServer Pages (JSP)
- Enterprise JavaBeans （EJB）
- Java Database Connectivity （JDBC）

- ネットワーク ソケット
- リモート メソッド呼び出し（RMI）
- 拡張可能マークアップ言語（XML）
- Java Transaction API （JTA）
- Java Naming and Directory Interface （JNDI）
- JMS（Java Message Service）
- Common Object Request Broker Architecture （CORBA）
- User Datagram Protocol （UDP）
- ファイル システム
- スレッド
- システム ログ
- スローおよびキャッチされた例外（デフォルトではオフ）

場合によっては、あまりにも多くの Java クラスが ProbeBuilder ディレクティブ（PBD）ファイルの中で監視対象として設定されているために、Java Agent の起動が正常に行われなかったり、ハング状態になってしまうことがあります。 このような問題が発生した場合は、*AutoProbe.log* ファイルを使用して、Java Agent のハング状態を引き起こしたクラスを特定し、PBD ファイルにスキップ ディレクティブを追加して、問題の原因となっている可能性のあるクラスをスキップします。 カスタム PBD ファイルへのスキップ ディレクティブの追加の詳細については、「Skip ディレクティブ (P. 94)」を参照してください。

デフォルトの PBD ファイル

Java Agent には、以下のデフォルトの PBD ファイルが含まれます。

appmap.pbd

アプリケーション問題切り分けマップのインスツルメンテーションに使用するトレーサ ディレクティブを提供します。

appmap-ejb.pbd

アプリケーション問題切り分けマップの EJB インスツルメンテーションに使用するトレーサ ディレクティブを提供します。

appmap-soa.pbd

Java SOAP スタックをサポートする SPM のために、アプリケーション問題切り分けマップの SOA インスツルメンテーションに使用するトレーサ ディレクティブを提供します。

注: 詳細については、「CA APM for SOA 実装ガイド」を参照してください。

bizrecording.pbd

エージェント ビジネス記録をセットアップするトレーサ定義およびディレクティブを提供します。

biz-trx-http.pbd

ビジネス セントリックの HTTP インスツルメンテーションに使用するトレーサ ディレクティブを提供します。

di.pbd

Apache Derby 実装クラスをインスツルメントしないようにするための ProbeBuilder 向けのディレクティブを提供します。

errors.pbd

重大なエラーを発生させるコード レベルのイベントを指定して、Error Detector を設定します。 デフォルトでは、フロントエンドおよびバックエンドのエラーのみが重大と見なされます。 すなわち、ユーザにエラー ページとして表示されるようなエラーやバックエンド システム（ADO.NET、メッセージング、など）の問題を示しているようなエラーのみです。

j2ee.pbd

一般的な Java Enterprise Edition コンポーネント用のトレーサ グループを提供します。 特定の追跡を TurnOn にするには、toggles-full.pbd または toggles-typical.pbd を使用します。

java2.pbd

一般的な Java 2 コンポーネント用のトレーサ グループを提供します。特定の追跡を *TurnOn* にするには、*toggles-full.pbd* または *toggles-typical.pbd* を使用します。

jsf.pbd

Java Server Face （JSF）コンポーネント用のトレーサ グループを提供します。

jsf-toggles-full.pbd

jsf.pbd で提供されている追跡に対して、*TurnOn* ディレクティブの形式でオン/オフを切り替えます。 ほとんどのトレーサ グループがオンになります。

jsf-toggles-typical.pbd

jsf.pbd で提供されている追跡に対して、*TurnOn* ディレクティブの形式でオン/オフを切り替えます。

jvm.pbd

さまざまな Java 仮想マシンに対するサポートを実装するディレクティブを提供します。 Introscope デフォルト ファイルと一緒に使用します。

leakhunter.pbd

リーク検出ユーティリティである CA APM LeakHunter 用のインスツルメンテーション設定を提供します。 通常、このファイルの内容を変更することはありません。

lisa.pbd

CA APM 統合のための CA LISA インスツルメンテーションに使用するトレーサ ディレクティブを提供します。

oraclejdbc.pbd

Oracle JDBC コンポーネント用のトレーサ グループを提供します。 *TurnOn* ディレクティブをコメント化またはコメント化を解除して、追跡対象の Oracle JDBC コンポーネントのセットを変更できます。

ServletHeaderDecorator.pbd

CA CEM 製品との統合の一部であるサーブレット ヘッダ デコレータを有効にします。

smwebagentext.pbd

SiteMinder Web エージェント Introscope プラグイン用のトレーサを提供します。

soaagent.pbd

CA SOA セキュリティ マネージャ（Web サーバとアプリケーション サーバ用の SOA エージェント）の一部である TransactionMinder エージェント用のトレーサを提供します。

spm-correlation.pbd

コンポーネント間にまたがってトランザクション追跡の相関関係付けを制御するディレクティブを提供します。このファイルは、CA APM for SOA を使用するときに、プロセス間にまたがるトランザクション追跡を可能にするのに必要です。

struts.pbd

Apache Struts を監視するディレクティブを提供します。 Introscope デフォルト ファイルと一緒に使用します。

summary-metrics-6.1.pbd

JSP の追跡、サーブレットの追跡、および 7.0 より前の Introscope インスタンスでの EJB の追跡に必要なディレクティブを提供します。

taglibs.pbd

JSP タグ ライブラリ、Jakarta I/O ライブラリおよび DGTags タグ ライブラリとして追跡されるべきクラスを監視するディレクティブを提供します。

TIBCO.pbd

Java Agent は、SOA 拡張機能を使用した TIBCO の監視に関連するいくつかの PBD と一緒にインストールされます。

注: 詳細については、「CA APM for SOA 実装ガイド」を参照してください。

toggles-full.pbd

ほかのディレクティブ ファイルで指定されている追跡について（*TurnOn* ディレクティブの形式で）オン/オフを切り替えます。 ほとんどのトレーサ グループがオンになります。

toggles-typical.pbd

ほかのディレクティブ ファイルで指定されている追跡について（*TurnOn* ディレクティブの形式で）オン/オフを切り替えます。トレーサ グループのごく一部のみがオンになります。

webMethod の PBD

Java Agent は、CA APM for webMethods Broker を使用した webMethods の監視に関連するいくつかの PBD と一緒にインストールされます。

注: 詳細については、「CA APM for SOA 実装ガイド」を参照してください。

WebSphere MQ の PBD

Java Agent は、CA APM for IBM WebSphere MQ を使用した WebSphere MQ コネクタおよびメッセージング システムの監視に関連するいくつかの PBD と一緒にインストールされます。

注: 詳細については、「CA APM for IBM WebSphere MQ ガイド」を参照してください。

Java Agent は、アプリケーション サーバ特有の PBD もインストールします。これは監視しているアプリケーション サーバによって異なります。

詳細:

デフォルトのトレーサ グループおよびトグル ファイル (P. 110)
トレーサ グループのオンまたはオフ (P. 121)

以前のリリースのデフォルト PBD ファイル

エージェントはデフォルトで、現在のリリースの PBD および PBL ファイルを使用します。 ただし、以前のリリースの PBD ファイルおよび PBL ファイルが <*Agent_Home*>/wily/examples/legacy ディレクトリに用意されています。 このディレクトリ内のファイル名には、それぞれ、default-full-legacy.pbl のように -legacy サフィックスが付いています。

デフォルトの PBL ファイル

各エージェントで利用できる PBL ファイルは、以下の 2 つです。

default-full.pbl（デフォルト）

ほとんどのトレーサ グループがオンに設定されている PBD ファイルを参照します。 Introscope は、このセットをデフォルトで使用して、Introscope のフル機能を発揮します。

default-typical.pbl

参照される PBD ファイル内のトレーサ グループのサブセットがオンになります。 標準セットには共通の設定が含まれます。これは、特定の環境に合わせてカスタマイズできるセットです。

また、Java Agent は、アプリケーション サーバ特有の PBL もインストールします。これは監視しているアプリケーション サーバによって異なります。

デフォルトのトレーサ グループおよびトグル ファイル

トレーサ グループは PBD ファイルに定義します。 トレーサ グループによって、クラス セットについての情報がレポートされます。 PBD ファイルでは、トレーサ グループは用語 *flag* によって参照されます。たとえば、TraceOneMethodIfFlagged または SetFlag は、トレーサ グループの情報を定義します。

トレーサ グループは、クラスのセットに適用されるトレーサのセットで構成されます。 たとえば、すべての RMI クラスの応答時間および速度をレポートするトレーサ グループがあります。

特定のトレーサ グループをオンまたはオフにして、システムでのメトリックの収集を細かく設定することができます。この方法は、トレーサ グループの設定方法によって、オーバーヘッドの増減に影響を与えます。

トレーサ グループは toggles-full.pbd ファイルおよび toggles-typical.pbd ファイルで変更できます。これらは default-full.pbl ファイルおよび default-typical.pbl ファイルによって参照されます。 以下に、デフォルトのトレーサ グループおよびそれらのデフォルト設定のリストを示します。

AgentInitialization

エージェントの初期化設定

フル： オン

標準： オン

ApacheStandardSessionTracing

HTTP セッションの設定

フル： オン

標準： オン

AuthenticationTracing

認証の設定

フル： オン

標準： オン

CorbaTracing

CORBA メソッド呼び出し

フル： オン

標準： オン

DBCPTracing

DBCP Configuration

フル： オン

標準： オン

DBCPv55Tracing

DBCP Configuration

フル： オン

標準： オン

EJB2StubTracing

EJB 2.0 の設定

フル： オン

標準： オン

EJB3StubTracing

EJB 3.0 の設定

フル： オン

標準： オン

EntityBean3Tracing

エンティティ EJB 3.0 メソッド呼び出し

フル： オン

標準： オン

EntityBeanTracing

エンティティ EJB メソッド呼び出し

フル： オン

標準： オン

HTTPServletTracing

HTTP サーブレット サービス応答

フル： オン

標準： オン

アプリケーション サーバ AutoProbe を使用している場合は、トレーサ グループ HTTPAppServerAutoProbeServletTracing をオンにします。

InstanceCounts

トレーサ グループで識別されるオブジェクト タイプのインスタンスの数をカウントします。

フル： オン

標準： オン

このトレーサ グループでクラスが特定されるまで、どのグループの追跡も行われません。

J2eeConnectorTracing

J2EE 接続情報

フル： オン

標準： オン

JavaMailTransportTracing

メール送信回数

フル： オン

標準： オン

JDBCQueryTracing

JDBC クエリ

フル： オン

標準： オン

JDBCUpdateTracing

JDBC 更新

フル： オン

標準： オン

JMSConsumerTracing

JMS メッセージ処理回数

フル： オン

標準： オン

JMSListenerTracing

JMS メッセージ処理回数

フル： オン

標準： オン

JMSPublisherTracing

JMS メッセージ ブロードキャスト回数

フル： オン

標準： オン

JMSSenderTracing

JMS メッセージ ブロードキャスト回数

フル： オン

標準： オン

JSPTracing

JSP サービス応答

フル： オン

標準： オン

MessageDrivenBean3Tracing

メッセージ駆動型の EJB 3.0 メソッド呼び出し

フル： オン

標準： オン

MessageDrivenBeanTracing

メッセージ駆動型の EJB メソッド呼び出し

フル： オン

標準： オン

NIOSocketTracing

［NIO］-［Channels］-［Sockets］ノードの下の各ソケット接続に一連のメトリックを生成し、［Backends］ノードの下にメトリックを追加します。

フル： オン

標準： オン

NIOSocketSummaryTracing

すべての NIO ソケット接続を対象とする 1 つのセットのメトリックを生成します。

フル： オン

標準： オン

これらのメトリックには、エージェント プロパティによって NIOSocketTracing メトリックから除外される接続が含まれます。 これらのメトリックには、NIOSocketTracing メトリックから常に除外される内部 NIO ソケットも含まれます。

NIOSelectorTracing

特定の内部 JVM による NIO チャンネルの使用が、NIO チャネル メトリックにカウントされないようにします。

ユーザは、このオプションを変更することはできません。

NIODatagramTracing

各データグラムの接続用の一連のメトリックを生成します。

フル： オン

標準： オン

詳細については、269 ページの NIODatagramTracing メトリックを参照してください。

NIODatagramSummaryTracing

すべての NIO データグラム アクティビティをすべて測定する 1 つのセットのメトリックを生成します。

フル： オン

標準： オン

これらのメトリックには、エージェント プロパティによって NIODatagramTracing メトリックから除外される接続が含まれます。 これらのメトリックには、NIODatagramTracing メトリックから常に除外される内部 NIO ソケットも含まれます。

PersistentSessionTracing

HTTP セッションの設定

フル： オン

標準： オン

RMIClientTracing

RMI クライアント メソッド呼び出し

フル： オン

標準： オン

RMIServerTracing

RMI サーバ メソッド呼び出し

フル： オン

標準： オン

ServerInfoTracing

サーバ情報の設定

フル： オン

標準： オン

SessionBean3Tracing

セッション EJB 3.0 メソッド呼び出し

フル： オン

標準： オン

SessionBeanTracing

セッション EJB メソッド呼び出し

フル： オン

標準： オン

SocketTracing

ネットワーク ソケット帯域幅および SSL トラッキング

フル： オン

標準： オン

StrutsTracing

Struts フレームワークのアクションの実行回数

フル： オン

標準： オン

SuperpagesSessionTracing

HTTP セッションの設定

フル： オン

標準： オン

ThreadPoolTracing

スレッド プールの設定

フル： オン

標準： オン

UDPTracing

ユーザ データグラム プロトコル（UDP）ソケット帯域幅

フル： オン

標準： オン

UnformattedSessionTracing

HTTP セッションの設定

フル： オン

標準： オン

EJB3MethodLevelTracing

メソッド レベルでの EJB 3.0 アクティビティ

フル： オン

標準： オフ

EJBMethodLevelTracing

メソッド レベルでの EJB アクティビティ

フル： オン

標準： オフ

FileSystemTracing

書き込みおよび読み取りされたファイル システムのバイト数

フル： オン

標準： オフ

JAXMListenerTracing

JAXM メッセージの送信

フル： オン

標準： オフ

JNDITracing

JNDI の参照回数

フル： オン

標準： オフ

JSPDBTagsTagLibraryTracing

SQL データベースとの間で読み書きするための Jakarta DB タグのカスタム タブ ライブラリ

フル： オン

標準： オフ

JSPIOTagLibraryTracing

さまざまな入出力タスクの Jakarta IO カスタム タグ ライブラリ

フル： オン

標準： オフ

JTACommitTracing

JTA を使用するコミット回数

フル： オン

標準： オフ

ThreadTracing

クラス別のアクティブ スレッド数

フル： オン

標準： オフ

XMLSAXTracing

XML ドキュメントの解析にかかる時間

フル： オン

標準： オフ

XSLTTracing

XML 変換時間

フル： オン

標準： オフ

CatchException

例外の設定

フル： オフ

標準： オフ

FormattedSessionTracing

HTTP セッションの設定

フル： オフ

標準： オフ

HTTPAppServerAutoProbeServletTracing

HTTP サーブレットの設定

フル： オフ

標準： オフ

HTTPSessionTracing

HTTP セッションの設定

フル： オフ

標準： オフ

JSPTagLibraryTracing

カスタム JSP タグの処理時間

フル： オフ

標準： オフ

ManagedSocketTracing

ネットワークの設定

フル： オフ

標準： オフ

ThrowException

例外の設定

フル： オフ

標準： オフ

通常は、デフォルトの *トグル*PBD ファイルを編集しないでください。 しかし、特定のトレーサ グループをオンまたはオフにすることによって、メトリックの収集を細かく指定することができます。 Tracer グループは、*トグル*ファイルで以下の方法によって変更できます。

- システム オーバーヘッドを低減するためにトレーサ グループをオン/オフにする
- トレーサ グループへのクラスの追加

トレーサ グループは、オンに設定されており *TurnOn* キーワードによってアクティブ化されている（コメント解除されている）場合のみ情報をレポートします。

注: Java エージェントは EJB （2.0 以降）のインスツルメンテーションをサポートしています。EJB と関連付けられたトレーサ グループをオンまたはオフにして、メトリック収集を調整します。 アプリケーション問題切り分けマップでの EJB のサポートは、セッション Bean とエンティティ Bean のみを対象とします。 メッセージ駆動型 Bean はサポートされていません。

追加メトリック情報を収集するためのトグルの設定

以下のトグルは、オンになったときに、すべての API で、有効になっている CA Technologies 提供の Tracer グループの追加メトリックを収集します。設定を変更するには、Full または Typical のトグル ファイルに以下のトグルを追加する必要があります。

DefaultStalledMethodTracing

停止したメソッド追跡

フル： オン

標準： オン

DefaultConcurrentInvocationTracing

同時進行中の呼び出し情報

フル： オン

標準： オフ

DefaultRateMetrics

呼び出し速度のメトリック

フル： オフ

標準： オフ

トレーサ グループのオンまたはオフ

特定のトレーサ グループをオンまたはオフにして、システムでのメトリックの収集を細かく設定することができます。

トレーサ グループをオンにする方法

1. toggles-full.pbd ファイルまたは toggles-typical.pbd ファイルを見つけます（どちらを使用するかは、AutoProbe または Java Agent が、*<appserver>-full.pbl* と *<appserver>-typical.pbl* のどちらの種類のファイルを使用しているかによって決まります）。 これらのファイルは、*<appserver home>*wily/core/config ディレクトリまたは *<Introscope_Home>*/core/config/systempbd ディレクトリにあります。
2. オンにするトレーサ グループを見つけて、行頭のシャープ記号を削除して、その行のコメント化を解除します。 以下の例のディレクティブをオンにすると、すべての HTTP サーブレットが追跡されます。

   ```
   TurnOn: HTTPServletTracing
   ```

   注: トレーサ グループのディレクティブのコメント化を解除する（ディレクティブをオンにする）と、そのトレーサ グループが使用されます。

トレーサ グループをオフにする方法

- トレーサ グループをオフにするには、以下の例のように行頭にシャープ記号を挿入することによって、その行をコメントにします。

  ```
  #TurnOn: HTTPServletTracing
  ```

トレーサ グループへのクラスの追加

特定のクラスの追跡をオンにするには、そのクラスを既存のトレーサ グループに追加します。 クラスをトレーサ グループに含めるには、いずれかの「Identify」キーワードを使用します。

たとえば、クラス *com.myCo.ejbentity.myEJB1* をトレーサ グループ *EntityBeanTracing* に追加するには、以下のように指定します。

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

「Identify」キーワードは以下のとおりです。

- IdentifyInheritedAs
- IdentifyClassAs
- IdentifyCorbaAs

EJB サブクラスの追跡

デフォルトで、エンティティおよびセッション EJB に関連するディレクティブは、エンティティ、セッション、またはメッセージ駆動型の EJB インターフェースを直接、明示的に実装する EJB のみに Probe を追加します。

通常、アプリケーションの EJB は、エンティティまたはセッション EJB インターフェースを直接、明示的に実装するクラスのサブクラスです。Introscope では、これらはデフォルトでは追加されません。

Introscope によって EJB サブクラスを追跡する場合は、それらを適切なトレーサ グループに追加する必要があります。 これを行うには、追跡する EJB サブクラスの直接の上位クラスを参照するエントリを追加します。

以下のモデルで、*<entity.bean.ancestor.class>* または *<session.bean.ancestor.class>* をインスツルメントされる EJB の直接の上位クラスの完全修飾名に置き換えます。

エンティティ EJB の場合：

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

セッション EJB の場合：

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

以下の例は、次のクラス階層に基づいています。

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

mySessionEJBsubclass1a extends mySessionEJBsubclass1

mySessionEJBsubclass1b extends mySessionEJBsubclass1

```
    mySessionEJBsubclass2 extends mySessionEJB
```

トレーサ グループ *SessionBeanTracing* では、*mySessionEJB* が追跡されます。

以下のトレーサは、*mySessionEJBsubclass1* および *mySessionEJBsubclass2* を追跡します。

IdentifyInheritedAs: mySessionEJB SessionBeanTracing

以下のトレーサは、*mySessionEJBsubclass1a* および *mySessionEJBsubclass1b* を追跡します。

IdentifyInheritedAs: mySessionEJBsubclass1 SessionBeanTracing

注: この例では、パッケージを使用していません。 コードがパッケージ内にある場合は、パッケージ名をクラス名に含める必要があります。

EJB 3.0 アノテーション

以下のディレクティブを使用すると、トレーサ グループに指定されたクラス レベルのアノテーションが含まれている任意のクラスをグループ化することができます。 このディレクティブは EJB 3.0 をサポートしています。 3.0 仕様に準拠する EJB は明示的に既知のインターフェースを実装しませんが、代わりにアノテーションを介することによって完全に識別可能になっています。 EJB 3.0 クラスを簡単に識別するには、以下のディレクティブを使用します。

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

このディレクティブを使用するには、新しいディレクティブに対してディレクティブ クラスおよびディレクティブ パーサ クラスを作成します。次に matcher クラスを追加して、クラスが指定のアノテーションが含まれているかどうかを判定するためにバイトコードを検査します。

注: このディレクティブは、メソッドレベルのアノテーションはサポートしていません。

アプリケーション問題切り分けマップでの EJB のサポート

CA Introscope® では、EJB（2.0 以降）のセッション Bean およびエンティティ Bean の付属の追跡機能をサポートしています。具体的には、Workstation のアプリケーション問題切り分けマップで使用します。 付属の機能を設定するとエージェントの起動時間に影響するため、この機能をテスト環境のみで使用することをお勧めします。

実稼働環境にこの機能をデプロイする場合は、EJB トレーサの対象を特定して設定します。 これはデフォルトの機能では対象範囲が広すぎる場合があるためです。

スーパークラスまたはインターフェースを継承または実装しているクラスにフラグを設定するように ProbeBuilder に指示するには、以下のディレクティブを使用します。

```
IdentifyDeepInheritedAs
```

EJB 2.0 のアプリケーション問題切り分けマップをサポートするために、以下のディレクティブが j2ee.pbd ファイルにあります。

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

これらのディレクティブを使用すると、ProbeBuilder は、クライアント側の EJB スタブと、アプリケーション問題切り分けマップで使用するサーバ側の Bean を識別することができます。

EJB 3.0 のアプリケーション問題切り分けマップをサポートするために、以下のディレクティブが j2ee.pbd ファイルにあります。

```
IdentifyInheritedAnnotatedClassAs
```

このディレクティブは、インターフェースを直接実装するすべてのクラス、またはスーパー インターフェースを通して実装するすべてのクラスを照合します。

アプリケーション問題切り分けマップのコンテキストでは、以下の追加のディレクティブが j2ee.pbd 内に設定されます。

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

EJB の名前付け

呼び出されたバックエンド、汎用フロントエンド、および EJB を処理する監視対象コンポーネントには名前を付けることができます。 名前付けのフォーマッタを使用すると、EJB（2.0 以降）のクライアント スタブや Bean の実装にふさわしい名前を設定できます。

EjbNameFormatter クラスでは、EJB 関連のメトリック名、アプリケーション問題切り分けマップのアプリケーション名、またはノード名を定義します。 以下のプレースホルダを使用します。

- EJB クライアント スタブの場合： *{classname}*、*{interface}*、および *{method}*
- EJB bean の場合： *{classname}*、*{bean}*、*{interface}*、および *{method}*

以下のメトリック名がデフォルトで使用されます。

- EJB Bean フロントエンド： *EJB|{interface}*
- EJB クライアント スタブ バックエンド： *EJB|{interface}*
- EJB Bean 用のアプリケーション問題切り分けマップの所有者名： *{interface}*
- EJB クライアント スタブ用のアプリケーション問題切り分けマップのノード名： *Client {interface}*
- EJB Bean 用のアプリケーション問題切り分けマップのノード名： *Server {interface}*

これらの名前はデフォルトの EJB 名フォーマッタです。このフォーマッタは、j2ee.pbd および appmap-ejb.pbd ファイルで使用されます。 同じ名前フォーマッタで別のメトリック名を使用します。たとえば、既存のトレーサ ディレクティブを変更してより適切な名前を使用しますが、同じフラグを維持したい場合は、以下のように実行します。

```
...
# デフォルトはコメント化されている
#TraceComplexMethodsIfFlagged: EJB2StubTracing EJB2BackendTracer "{interface}"
# EJB アプリケーション名を、呼び出されたメソッドに加えてバックエンド マーカに追加する
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
```

注: EJB コンテキストのトレーサは、EJB 2.0 Bean の setContext() メソッドで設定されます。 このトレーサは、EJB 2.0 Bean の名前フォーマッタ用の CA Introscope® 内部トレーサです。これにより、名前フォーマッタが正しく機能できます。

IntroscopeAgent.profile、PBL、および PBD の同時使用

Java Agent を最初にインストールするときに、インスツルメンテーションのレベルをフルまたは標準に設定します。 この設定では、ProbeBuilder リスト（PBL）ファイル *default-typical.pbl* および *default-full.pbl* を参照します（詳細については、「デフォルトの PBL ファイル (P. 110)」を参照してください）。

ProbeBuilder ディレクティブの適用

PBD を適用する方法は、使用するメソッドによって異なります。 JVM AutoProbe を使用して PBD を実装することを勧めします。 ProbeBuilder ウィザード、またはコマンド ライン ProbeBuilder を使用して、PBD を実装することもできます。

JVM AutoProbe の使用

PBD ファイルの実装の準備が整ったら、これを *hotdeploy* ディレクトリに追加します。 AutoProbe は、*IntroscopeAgent.profile* ファイルを含むディレクトリ（デフォルトでは、*<Agent_Home>*/wily/core/config ディレクトリ）および *<Agent_Home>*/wily/core/config/hotdeploy ディレクトリで PBD ファイルを検索します。 AutoProbe は、これらのディレクトリに対する相対パスによってファイル名を解決します。*wily* ディレクトリの場所を移動した場合は、ファイル パスを正しいディレクトリにマッピングしてください。

AutoProbe を使用して PBD を実装する方法

1. 変更済みの標準の PBD または PBL を *<Agent_Home>*/wily/core/config ディレクトリに保存します。
2. カスタムの PBD を *<Agent_Home>*/wily/core/config/hotdeploy ディレクトリにコピーします。 このディレクトリに追加された PBD はすべて、*IntroscopeAgent.profile* の *introscope.autoprobe.directivesFile* プロパティを更新または変更する必要なく実装されます。

   注: 動的インスツルメンテーションを有効にしている場合、*hotdeploy* ディレクトリ内の PBD はフォルダからそのままピックアップされます。再起動は必要ありません。動的インスツルメンテーションの詳細については、「動的 ProbeBuilding (P. 94)」参照してください。
3. *IntroscopeAgent.profile* を保存します。
4. アプリケーションを再起動します。

ProbeBuilder ウィザードまたはコマンドライン ProbeBuilder の使用

PBD ファイルの実装の準備が整ったら、これを *hotdeploy* ディレクトリに追加します。 コマンドライン ProbeBuilder は、ProbeBuilder の実行元と同じディレクトリ、および *<Agent_Home>*/wily/core/config/hotdeploy ディレクトリで、任意のカスタム ディレクティブ ファイルを探します。 コマンドライン ProbeBuilder は、これらのディレクトリに対する相対パスによってファイル名を解決します。

ProbeBuilder ウィザードまたはコマンド ライン ProbeBuilder を使用して ProbeBuilder ディレクティブを実装する手順は、JVM AutoProbe を使用する場合と同じです。 詳細については、「JVM AutoProbe の使用 (P. 127)」を参照してください。

新規および変更済みの PBD を使用したインスツルメント

新規および変更済みのディレクティブを有効にするには、最新の PBD を使用して、アプリケーションをインスツルメントする必要があります。 この手順は、使用している ProbeBuilding メソッドによって異なります。

-javaagent を介して JVM AutoProbe を使用する JVM 1.5 システム

動的インスツルメンテーションを設定すると、アプリケーションまたは Java Agent を再起動しなくても変更済みの PBD を有効にできます。これによって、アプリケーションのサービスを中断しなくても、PBD の修正を実行したり、切り分け駆動型（triage-driven）のインスツルメンテーションを実行できます。 詳細については、「動的 ProbeBuilding (P. 94)」を参照してください。

新規および変更済みの ProbeBuilder ファイル

対象: 1.5 より前の JVM または -Xbootclasspath を使用するインストール

新規および変更済みの ProbeBuilder ディレクティブ ファイルまたは ProbeBuilder リスト ファイルは、次回アプリケーション サーバがアプリケーション クラスをロードしたときに有効になります。

ディレクティブを追加または変更したときに管理対象アプリケーションが実行されていない場合は、次回アプリケーションを起動したときに、更新済みのディレクティブを使用してインスツルメントされます。

管理対象アプリケーションが実行されている場合は、管理対象アプリケーション クラスをロードまたは再ロードする必要があります。

クラスを再ロードする方法は、使用するアプリケーション サーバによって異なります。 ほとんどのアプリケーション サーバでは再起動する必要があります。

ProbeBuilder ウィザードの使用

ProbeBuilder ウィザードを使用する方法

1. ［カスタム ディレクティブ］画面には、「ProbeBuilder ウィザードまたはコマンドライン ProbeBuilder の使用 (P. 127)」で説明した *hotdeploy* ディレクトリに配置されている PBD ファイルが一覧表示されます。
2. 使用するカスタム ディレクティブ ファイルを選択します。

コマンドライン ProbeBuilder の使用

重要: 最新の PBD を Introscope で使用できるように、コマンドライン ProbeBuilder を最終的なオプションとして使用することをお勧めします。

コマンドライン ProbeBuilder を使用する方法

1. マネージド アプリケーションを停止します。
2. コマンドライン ProbeBuilder または ProbeBuilder ウィザードを実行し、カスタムの PBD および PBL ファイルをコマンド ラインに入力します。
3. 新規ファイルを使用できるようにアプリケーションを設定します。
4. マネージド アプリケーションを起動します。
5. Enterprise Manager および Workstation を実行していない場合は実行します。

カスタム トレーサの作成

カスタム PBD ファイルを作成して、メトリック コレクションを細かく設定することができます。 アプリケーション特有の情報を取得するため追跡を作成して、カスタム ディレクティブを作成するには、特定の構文およびキーワードを使用する必要があります。 カスタム追跡を作成するには、以下のことを定義する必要があります。

- ディレクティブの タイプ（通常は、追跡するクラスまたはメソッドの数を示します）
- 追跡する特定のクラスまたはメソッド
- クラスまたはメソッドで追跡する情報の種類（時間、レート、またはカウントなど）
- この情報が存在するメトリックの完全修飾名（リソース パスを含む）

カスタム PBD は、*<Agent_Home>*/wily/core/config/hotdeploy ディレクトリに保存されています。 このディレクトリに追加された PBD はすべて、*IntroscopeAgent.profile* の *introscope.autoprobe.directivesFile* プロパティを更新または変更する必要なく実装されます。動的インスツルメンテーションを有効にしている場合、*hotdeploy* ディレクトリ内の PBD はフォルダからそのままピックアップされます。 再起動は必要ありません。 動的インスツルメンテーションの詳細については、「動的 ProbeBuilding (P. 94)」参照してください。

カスタム PBD が作成されると、Introscope はこれをすぐに使える PBD として扱います。 作成したメトリックにアラートを設定して SmartStor に保存したり、Introscope Workstation でカスタム ダッシュボードを作成するときに使用したりすることができます。

注: 追跡するメソッドが多くなるとオーバーヘッドも大きくなるため、追跡するメソッドは慎重に選択する必要があります。

共通メトリック用のカスタム BlamePointTracer トレーサの使用

BlamePointTracer は、最も一般的に使用されるトレーサです。 このトレーサは、関連付けられているメソッドまたはクラス用に個々に 5 つのメトリックを生成します。

- Average Response Time (ms)
- Concurrent Invocations
- Errors Per Interval
- Responses Per Interval
- Stall Count

以下に、*BlamePointTracer* の例を示します。 *BlamePointTracer* は、クラス *petshop.catalog.Catalog* で *search* と呼ばれるメソッド用に設定されています。 *PetShop|Catalog|search* は、Introscope Investigator で BlamePoint メトリックが表示されるノードの名前です。

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

トレーサの構文で使用するディレクティブ名および引数

トレーサをグループに関連付けたり、グループを有効化/無効化する単純なキーワードのほかに、PBD ファイルにはトレーサの定義が含まれています。 Introscope でトレーサを認識および処理できるようにするには、カスタム トレーサを作成するときに特定の構文を使用する必要があります。トレーサは、ディレクティブと、追跡するメソッドまたはクラスについての情報を、以下の形式で指定して構成します。

```
<ディレクティブ>: [arguments]
```

ここで、*[arguments]* はディレクティブ固有のリストです。

注: 使用するディレクティブによっては、これらのパラメータのサブセットだけが必要な場合もあります。

<ディレクティブ>

最も一般的に使用するディレクティブは以下の追跡ディレクティブです。

TraceOneMethodOfClass

指定されたクラスの指定されたメソッドを追跡します。

TraceAllMethodsOfClass

指定されたクラスのすべてのメソッドを追跡します。

TraceOneMethodIfInherits

指定されたクラスまたはインターフェースの、すべての直接サブクラス、または直接インターフェースのすべての実装で、1 つのメソッドを追跡します。

TraceAllMethodsIfInherits

指定されたクラスまたはインターフェースの、すべての直接サブクラス、または直接インターフェースのすべての実装で、すべてのメソッドを追跡します。

TraceOneMethodIfFlagged

指定されたクラスが、*TurnOn* キーワードによって有効化されたトレーサ グループに含まれている場合、1 つのメソッドを追跡します。

TraceAllMethodsIfFlagged

指定されたクラスが、*TurnOn* キーワードによって有効化されたトレーサ グループに含まれている場合、すべてのメソッドを追跡します。

注: 具象メソッドが実装された場合のみ、追跡対象となり、実行中にメトリック データをレポートできます。 カスタム トレーサで抽象メソッドを指定しても、メトリック データはレポートされません。

トレース ディレクティブ用の構文は通常、以下の引数から構成されます。

<トレーサ グループ>

トレーサが関連付けられているグループ。

<クラス>

追跡するクラスまたはインターフェースの完全修飾名。 クラスの完全修飾名には、名前だけでなくクラスの完全なアセンブリ名が含まれます。 たとえば、以下のようになります。

[MyAssembly]com.mycompany.myassembly.MyClass

アセンブリ名は [] かっこで囲む必要があります。

<メソッド>

メソッド名（*MyMethod* など）

または

戻り値の型およびパラメータが指定された完全なメソッド シグネチャ（たとえば、*myMethod;[mscorlib]System.Void([mscorlib]System.Int32)*。 メソッド シグネチャの詳細については、「シグネチャの区別 (P. 137)」を参照してください）。

<トレーサ名>

使用するトレーサのタイプを指定します。 たとえば、*BlamePointTracer* となります。 トレーサ名の説明については、以下のトレーサ名の表を参照してください。

<メトリック名>

収集されたデータを Introscope Workstation に表示する方法を制御します。

以下の例は、メトリック ツリーのさまざまなレベルにあるメトリックの名前および場所を指定する 3 つの方法を示します。

metric-name - メトリックは エージェント ノードのすぐ内側に表示されます。

resource:metric-name - メトリックは、エージェント ノード の下の 1 つのリソース（フォルダ） 内に表示されます。

resource|sub-resource|sub-sub-resource:metric-name - メトリックは、エージェント ノード の 1 レベル下よりも深いリソース（フォルダ）に表示されます。 リソースを区切るには、パイプ文字（|）を使用します。

一般的に使用されるトレーサ名および例

以下のリストでは、最も一般的に使用されるトレーサ名とその追跡対象を示します。

BlamePointTracer

追跡対象のコンポーネントの平均応答時間、指定間隔あたりのカウント、並行処理、ストール、およびエラーを含む、標準のメトリックのセットを提供します。

ConcurrentInvocationCounter

すでに開始されているが、まだ終了していないメソッドの回数をレポートします。 結果は、Investigator ツリー内のトレーサに指定されているメトリック名 *<metric-name>* の下にレポートされます。 このトレーサの使用例としては、データベースの同時クエリ数のカウントがあります。

DumpStackTraceTracer

スタックの追跡データを、その適用先のメソッドのインスツルメント済みアプリケーションの標準エラーにダンプします。 Dump Stack Tracer によってスローされた例外スタック追跡は、本当の例外ではありません。これは、メソッド スタックの追跡データを出力するためのメカニズムです。

この機能は、メソッドの呼び出しパスを調べるときに便利です。

重要: ただし、この機能では、大きなシステム オーバーヘッドが発生します。 このトレーサの使用は、急激なオーバーヘッドの増加が問題にならない、診断目的時に限ることを強くお勧めします。

MethodCPUTimer

メソッド実行中に使用される CPU 時間（ミリ秒単位）の平均で算出し、それをメトリック ツリーの *<metricname>* の下にレポートします。

注: このトレーサには、サポートされるプラットフォームのプラットフォーム モニタが必要です。

MethodTimer

メソッド実行時間の平均をミリ秒単位で算出し、それをメトリック ツリーのトレーサに指定されているメトリック名 *<metric-name>* の下にレポートします。

PerIntervalCounter

間隔あたりの呼び出し数。 この間隔は、データのコンシューマ（Investigator のビュー ペインなど）の表示期間に基づいて変更されます。これは Investigator ツリーで、トレーサに指定されているメトリック名 *<metric-name>* の下にレポートされます。

カスタム トレーサ定義での引用符の使用

カスタム トレーサにはスペースを持つメトリック名を使用できます。 カスタムのメトリック名にスペースを使用する場合に、すべてのメトリック名の前後に引用符（""）を使用することをお勧めします。

重要: クラス名は引用符で囲まないでください。 引用符を使用すると、カスタム トレーサが誤動作を起こします。 例：

正

```
IdentifyClassAs: MyClass MyTracers
```

誤

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

クラス名を含むメトリック名を作成する場合は、メトリック名全体を引用符で囲む必要があります。 メトリック名にはスペースを使用できます。 また、メトリック名の中にあるすべてのスペースが引用符の中に含まれている必要があります。たとえば、メトリック名「{classname}|Test One Node」は、以下のように表す必要があります。

正

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

誤

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

重要: Introscope では、クラス ファイル名が無効であるクラスを監視しません。 たとえば以下のようなクラス ファイルがある場合、

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

_[B: によってクラス ファイル名が無効になります。 Java クラス ファイル名には開いている角かっこ（[）を使用できません。 Introscope が無効なクラス名を持つクラスを見つけた場合、そのクラスのインスツルメントは失敗し、その内容はエージェント ログにエラー メッセージとしてレポートされます。

以下に、メソッド トレーサの例を示します。 以下の例では、メトリック名の前後に引用符（""）が使用されています。 CA Technologies では、カスタムのメトリック名を作成するときに、すべてのメトリック名の前後を引用符で囲むことを強くお勧めしています。

Average トレーサの例

このトレーサは、指定したメソッドの平均実行時間（ミリ秒単位）を追跡します。

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

Rate トレーサの例

このトレーサは、1 秒あたりのメソッドの呼び出し回数をカウントし、この速度を、指定したメトリック名の下にレポートします。

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

Per Interval Counter トレーサの例

このメソッド トレーサは、指定した間隔あたりのメソッド呼び出し回数をカウントし、そのカウントを、指定したメトリック名の下にレポートします。

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

間隔は、グラフの頻度など、Enterprise Manager での監視ロジックによって決まります。

Investigator のプレビュー ペインでは、デフォルト間隔は 15 秒です。

Counter トレーサの例

このトレーサは、メソッドの呼び出し回数の合計をカウントします。

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

Counter トレーサの結合例

これらのトレーサは、実行時カウントを保持するために、増分および減分を行うトレーサを結合しています。

```
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"
```

高度な単一メトリック トレーサ

ディレクティブおよびトレーサが、メソッド、クラス、およびクラス セットを追跡します。 単一メトリック トレーサでは、特定のメソッドの特定のメトリックについてレポートが実行されます。これは Introscope で追跡可能な最小の単位です。単一メトリック トレーサは、メソッド シグネチャ、キーワードの代入、メトリック名パラメータの使用など、いくつかの方法によって作成できます。

シグネチャの区別

トレーサを、メソッド署名に基づいてメソッドに適用できます。

特定の署名を持つメソッドの単一インスタンスを追跡するには、内部メソッド記述子形式を使用して指定されたメソッド名（戻り値の型も含む）の最後に署名を付加します。

たとえば、*myMethod(Ljava/lang/String;)V* は、文字列引数および戻り値の型 void を持つメソッドのインスタンスを追跡します。

この形式に関する詳細については、「*Sun Java Virtual Machine Specification*」を参照してください。

メトリック名キーワード ベースの代入

キーワード ベースの代入では、実行時に値をメトリック名に代入できます。

トレーサ内のメトリック名のパラメータは、実行時に実際の値と置き換えられてメトリック名に代入されます。 この機能は、どのディレクティブでも使用できます。

{method}

追跡されるメソッドの名前

{classname}

追跡されるクラスの実行時クラス名

{packagename}

追跡されるクラスの実行時パッケージ名

{packageandclassname}

追跡されるクラスの実行時パッケージ名およびクラス名

注: パッケージを持たないクラスを処理する場合、Introscope は {packagename} を文字列「<Unnamed Package>」と置き換えます。

キーワード ベースの代入: 例 1

pbd ファイル内のトレーサのメトリック名が以下であるとします。

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

また、トレーサが、パッケージ myPackage 内にある myClass の実行時クラスを持つメソッド myMethod に適用されるとします。結果のメトリック名は以下のようになります。

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

キーワード ベースの代入: 例 2

.pbd ファイルに以下のメトリック名を持つトレーサがあるとします。

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

これが同じメソッドに適用されると、結果のメトリック名は以下のようになります。

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

注: パッケージとクラスの間には、最初の例で使用されていた | ではなく、ピリオドが使用されている点に注意してください。

メトリック名ベースのパラメータ

このフォーマットを使用し、「*TraceOneMethodWithParametersOfClass*」キーワードを使用して、メソッドに渡されたパラメータに基づいてメトリック名を作成する単一メソッド トレーサを作成できます。

TraceOneMethodWithParametersOfClass: <クラス名> <メソッド> <トレーサ名> <メトリック名>

パラメータをメトリック名で使用できます。 これは、メトリック名のプレースホルダ文字列をパラメータの値と置き換えることで実現できます。使用するプレースホルダ文字列は、「{#}」です。# は、代入するパラメータのインデックスです。 インデックスのカウントはゼロから始まります。パラメータの代入は、いくつでも、またどのような順序でも使用できます。すべてのパラメータは、メトリック名に代入される前に文字列に変換されます。 文字列以外のオブジェクト パラメータは、toString() メソッドを使用して変換されるので、使用する際は注意が必要です。

重要: パラメータがどんな文字列に変換されるかがはっきりしない場合は、メトリック名を使用しないでください。

メトリック名ベースの例

Web サイトが、「order」という名前のクラスを、「process」という名前のクラスとともに使用します。 メソッドは、異なる種類の order のパラメータ「book」または「music」を保持します。

この場合、以下のようなトレーサを作成できます。

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

このトレーサは、以下のようなメトリックを作成します。

Order

BookOrder

- Average Response Time (ms)

MusicOrder

- Average Response Time (ms)

また、「TraceOneMethodWithParametersIfInherits」キーワードも使用できます。

Skip ディレクティブ

Skip ディレクティブを使用すると、AutoProbe または ProbeBuilder によって特定のパッケージ、クラス、またはメソッドをスキップさせることができます。 デフォルトでは、Java Agent、および基本的な Java クラスとパッケージは、AutoProbe または ProbeBuilder によってスキップされます。

オブジェクト インスタンスのカウント

トレーサ グループ「InstanceCounts」は、関連付けられている特定のオブジェクトの種類のインスタンス数をカウントします（標準の IdentifyClassAs ディレクティブおよび IdentifyInheritedAs ディレクティブを使用した、オブジェクトの種類と InstanceCounts トレーサ グループとの関連付けについては、「トレーサ グループへのクラスの追加 (P. 121)」を参照してください）。 コードに明示的に割り当てられたインスタンスは、すべてカウントされます。 また、サブタイプもカウントされます。 逆シリアル化や複製など、異なる手段で作成されたオブジェクトは、カウントされないことがあります。 カウントされる全体的なインスタンス数によっては、このトレーサ グループを使用して追跡することで、徐々にパフォーマンス（およびメモリ量）が低下する可能性があります。

注: CA Technologies のテストでは、実際にはインスタンス数が極めて大きくならないと大きく影響しないことがわかっています。

InstrumentPoint ディレクティブのオン

キーワード「*InstrumentPoint:*」で識別されるディレクティブには、例外を追跡するもの、および（最初にプローブを実行したときではなく）アプリケーションの起動時にエージェントを初期化させるものの 2 種類があります。

例外

以下のディレクティブは、例外がスローまたはキャッチされた場合にその追跡をオンにするために使用されます。 これらは、パフォーマンスを低下させることがあるので、デフォルトではオンになっていません。 これらのいずれかをオンにするには、以下の該当行のコメント化を解除します。

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

エージェントの初期化

エージェント初期化の InstrumentPoint ディレクティブでは、追加のオーバーヘッドは発生しないので、このディレクティブは、デフォルトでフルおよび標準の PBD セットの両方でオンになっています。

```
InstrumentPoint: AgentInitialization
```

複数の ProbeBuilder ディレクティブ ファイルが使用されている場合は、いずれかのファイルでオンになっている設定（トレーサ グループ、Skip、InstrumentPoint、カスタム メソッド トレーサなど）が有効になります。

カスタム トレーサの結合

同じメトリックに影響を及ぼす複数のトレーサを実質的に結合して使用できます。結合は、増分および減分のトレーサで最もよく使用されます。

以下の例では、「*Total Purchases*」という名前のメトリックを作成します。クラス「*cart*」、メソッド「buyBook」および「buyCD」を使用して、以下のトレーサを作成します。

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

誰かが商品を 1 つ買うと、「*Total Purchases*」というメトリックが 1 ずつ増加します。

インスツルメントおよび継承

対象: 1.5 より前の JVM

1.5 より前の JVM では、クラス階層の下位レベルにあるクラスに対するインストルメンテーションは自動的には行われません。 プローブされるクラスの複数レベル下のサブクラスがロードされている場合は、新しいメソッドおよび優先メソッドは、自動的にはインスツルメントされません。プローブされるインターフェースに実装中に明示的に名前を付けるクラスは、インターフェースを間接的に実装する場合でもインスツルメントされます。

たとえば、ClassB が ClassA を、ClassC が ClassB を継承するといったようなクラス階層があるとします。

```
Interface/ClassA
    ClassB
        ClassC
```

ClassA をインスツルメントすると、ClassA を明示的に継承する ClassB もインスツルメントされます。しかし、ClassC はインスツルメントされません。これは、ClassC が ClassA を明示的に拡張しないためです。 ClassC をインスツルメントするには、明示的に ClassC を指定します。

1.5 より前の Java 環境で、サブクラスが確実にインスツルメントされるようにするには、「EJB サブクラスの追跡 (P. 122)」の手順に従います。

JVM 1.5 を使用する場合は、プローブされるクラス (P. 99)の複数のサブクラス レベルをインスツルメントするように CA Introscope® を設定できます。

Java のアノテーション

CA Introscope® では、カスタム メトリックの作成時に、Java 1.6 のアノテーションを使用することができます。

- **注**: Java のアノテーションの詳細については、Java デベロッパー サイトのドキュメントを参照してください。

IdentifyAnnotatedClassAs を使用してクラスをトレーサ グループに配置し、次に TraceAllMethodsIfFlagged ディレクティブを使用してクラスでメソッドをインスツルメントします。 例 :

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

例では、com.test.MyAnnotation がアノテーション名になります。独自のアノテーションを作成する場合は、コード内の用語を使用してください。アノテーション名を含むクラスが、特定されます。

Blame トレーサを使用した Blame ポイントのマーク付け

CA Introscope® の Blame テクノロジを管理対象 Java アプリケーションで使用すると、アプリケーション層（アプリケーションのフロントエンドおよびバックエンド）でのメトリックを表示できるようになります。 この Boundary Blame と呼ばれる機能を使用すると、問題をアプリケーションのフロントエンドかバックエンドに切り分けることができます。 この情報は、アプリケーションの境界線をマーク付けするために Workstation のアプリケーション問題切り分けマップで使用されます。

CA Introscope® がフロントエンドおよびバックエンドを決定する方法について、また URL グループを設定してフロントエンドのメトリックを集約する方法を制御するオプションについては、「Boundary Blame の設定 (P. 195)」を参照してください。

以下のセクションでは、トレーサを使用してアプリケーションのフロントエンドおよびバックエンドに明示的にマークを付ける方法について説明します。

Blame トレーサ

Introscope は、フロントエンドおよびバックエンドのメトリックをキャプチャするためのトレーサ（FrontendMarker および BackendMarker）を提供しています。 これらのトレーサは、フロントエンドおよびバックエンドに、それぞれ明示的にマークを付けます。

FrontendMarker および BackendMarker を使用して、バックエンドにアクセスするコードなど、独自のコードをインスツルメントして、Introscope でカスタム コンポーネントのメトリックをキャプチャしたり Investigator ツリーに表示したりできます。

FrontendMarker トレーサ（またはそのサブクラス HttpServletTracer および PageInfoTracer）を使用してコンポーネントをインスツルメントしない場合は、フロントエンドのメトリックは生成されません。また、コンポーネントはトランザクション用のフロントエンドとしてマーク付けされません。

1 つのトランザクションで複数のコンポーネントが FrontendMarker トレーサ（またはそのサブクラス）を使用してインスツルメントされる場合、最初に指定されたコンポーネントのみがフロントエンド メトリックを生成します。

注: フロントエンド トレーサを使用する場合、フロントエンド トレーサに指定されたアプリケーションの名前は、アプリケーション問題切り分けマップのトレーサに指定された名前と一致する必要があります。また、両方とも大文字と小文字が区別されることに注意してください。 たとえば、フロント トレーサに **AppOne** という名前を付け、アプリケーション問題切り分けマップのトレーサがこのトレーサを **APPONE** として参照した場合、Workstation のアプリケーション問題切り分けマップでは AppOne の情報が正しく表示されません。

特定のクラスがフロントエンドとしてマーク付けされないようにするには、PBD パラメータ **is.frontend.unless** を指定します。 PBD ディレクティブ *is.frontend.unless* の詳細については、「カスタム FrontendMarker ディレクティブ (P. 146)」を参照してください。

BackendMarker が設定されていない場合、Introscope は、バックエンドを推測します。明示的にマークが付けられているものがない場合は、クライアント ソケットを開くコンポーネントをデフォルト バックエンドとします。

BackendMarker は、以下の場合に使用すると便利です。

- Introscope がバックエンドとして検出する項目に、適切な名前を割り当てる。
- Introscope がインスツルメントしないカスタムの Java ソケットにマークを付ける。
- Java Native Interface （JNI）を通じて呼び出されるネイティブ ソケットの場合に、Java/JNI ブリッジ メソッドをバックエンドとして識別する。

FrontendMarker および BackendMarker は、追跡対象のコンポーネントの平均応答時間、指定間隔あたりのカウント、並行処理、ストール、およびエラーなどのメトリックを提供する BlamePointTracer のインスタンスです。BlamePointTracer は、より詳細な Blame スタックのための中間コンポーネントに適用されます。

複雑にネストされたフロントエンド トランザクションによるエージェント CPU の高オーバーヘッド

Introscope では、サーブレットをフロントエンドとみなすよう設定することができます。 標準的なトランザクションは、サーブレットで開始します。 これは、EJB を呼び出すことがあり、さらにこれがバックエンドを呼び出します。 サーブレットは、他のサーブレットをネストする形で呼び出すことができます。このような場合、Introscope からはネストされたフロントエンドとして見えます。 大抵の場合、これによって Agent CPU のオーバヘッドが増加することはありません。

ただし、フロントエンド レベルでネストされた複雑なトランザクション（例：階層が 40 レベル）は、CPU に大きなオーバーヘッドをもたらす可能性があります。 たとえば、サーブレットがトランザクションでサーブレット自身を繰り返し呼び出したり（連続的な再帰呼び出し）、他の複数のサーブレットを呼び出すと、エージェント CPU のオーバーヘッドが増加する場合があります。オーバヘッドが許容範囲を超えている場合は、CA サポート にご連絡ください。

カスタム FrontendMarker ディレクティブ

PBD パラメータ is.frontend.unless を使用すると、一部のクラスがフロントエンド コンポーネントとしてマークされないようにすることができます。これらのクラスは、FrontendMarker（または HttpServletTracer などのサブクラス）によってインスツルメントされます。 このパラメータは、絶対クラス名のカンマ区切りのリストで設定します。 初期コンポーネントが一般的なディスパッチャである場合、このパラメータが役立ちます。 このディスパッチャは、受信した要求タイプを処理する特定のコンポーネントに要求を転送します。 そのため、2 番目のコンポーネントの方がフロントエンドにより適したマーカになります。 デフォルトは空のリストです。PBD パラメータは動的ではありません。このパラメータの値を変更した場合は、インスツルメントされたアプリケーション サーバを再起動する必要があります。

重要: スペースではなくカンマでクラス名を区切ります。スペースを使用すると SetTracerParameter ディレクティブが無効になります。

パラメータ リストに指定された任意のクラスが、このパラメータを適用するトレーサによってインスツルメントされる場合

- フロントエンドとして指定されます。
- Investigator の［Frontends］ノードの下にメトリックを生成しません。

たとえば、NotAFrontend および AnotherNonFrontend というクラスが、com.ABCCorp パッケージでフロントエンドとして扱われないようにするとします。 これらのクラスは MyFrontendTracer という名前の FrontendMarker によってインスツルメントされます。 以下の PDB ディレクティブを使用します。

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

標準 PBD での Blame トレーサ

Introscope で提供されている 2 つの標準 PBD（*j2ee.pbd* および *sqlagent.pbd*）は、Boundary Blame の追跡を実装しています。

- *j2ee.pbd* 内の *HttpServletTracer* は、FrontendMarker のインスタンスです。
- *sqlagent.pbd* 内の *SQLBackendTracer* は、BackendMarker のインスタンスです。

以前のバージョンの Introscope で使用されていた以下の Blame トレーサも引き続き存在しますが、Introscope PBD ではあまり使用されません。

- BlamedMethodTimer
- BlamedMethodRateTracer
- BlamedMethodTraceIncrementor
- BlamedMethodTraceDecrementor

Boundary Blame および Oracle バックエンド

現在のバージョンの Introscope では、ソケット接続に基づいた Oracle データベースの検出は行われません。Oracle バックエンドを自動検出するには、Introscope で SQL エージェントを使用可能にする必要があります。

SQL エージェントがない場合に Introscope で Oracle バックエンドを検出できるようにするには、oraclejdbc.pbd を以下のように変更します。

oraclejdbc.pbd の以下の部分：

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

以下の例のようにスキップをコメント化します。

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

注: 詳細については、http://ca.com/support で、ナレッジベースの記事「*Disabling Database Name Formatting in 7.1 (KB 1240)*」を参照してください。

第 6 章: Java Agent の名前付け

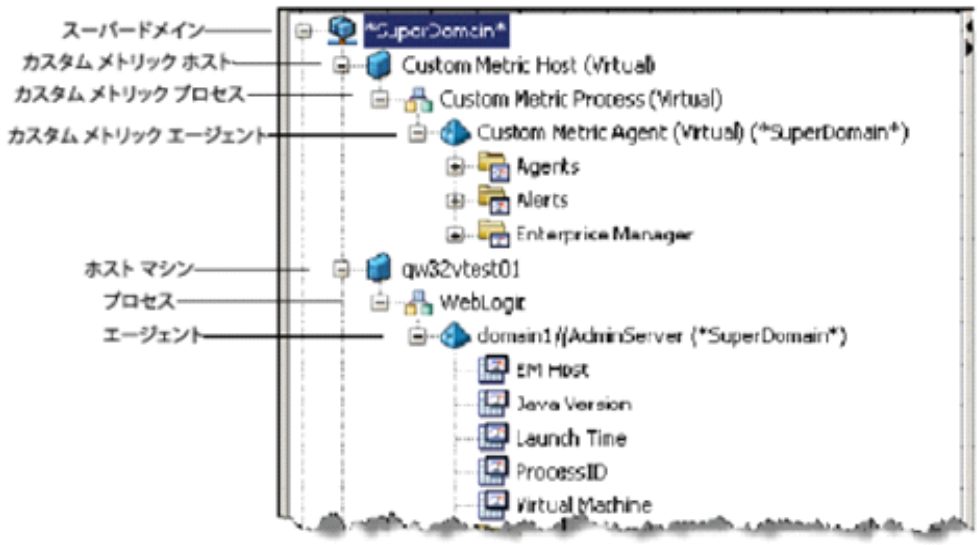
このセクションでは、エージェントの名前付け、関連する環境およびデプロイ実施における考慮点、エージェントに自動的に名前を付けるオプションについて説明します。

このセクションには、以下のトピックが含まれています。

Java Agent 名の理解

Introscope 環境で動作している各 Java Agent には名前が割り当てられています。名前の割り当ては、明示的に割り当てる、自動的に名前を割り当てる方法を設定する、または Java Agent が監視するインスツルメントされたアプリケーションを起動する、といういずれかの方法で行われます。 Java Agent 名は、Introscope Workstation および Investigator 内の多くのビューの中心となります。また、この名前は、対象となるアプリケーションと監視ロジックを関連付けるプロセスにおいて重要な役割を果たします。

エージェントが Enterprise Manager にレポートするときは、Investigator ツリーにそのエージェントのノードが作成されます。 Workstation でダッシュボード、アラート、アクションなどの管理ロジックを設定する場合、エージェント名は管理ロジックの適用対象となるアプリケーションを識別する正規表現のコンポーネントになります。 以下の Investigator ツリーには、WebLogic プロセスの下のホスト *qw32vtest01* で実行している *domain1//Adminserver* という名前のエージェントが表示されています。



エージェントの名前付けの方法

Java エージェントは、以下の順に名前を決定します。

1. 1 つ目の方法で名前が見つかった場合はその名前が採用され、Enterprise Manager に接続されます。
2. 1 つ目の方法を使用しても名前が見つけられなかった場合は、2 つ目の方法が試されます。その後も同様です。
3. どの方法を使用しても名前が見つけられなかった場合は、その Java エージェントに「UnnamedAgent」という名前が付けられます。

方法 1: Java システム プロパティに指定されたエージェント名

エージェント名は、コマンド ラインで Java システム プロパティを使用して定義されます。 この方法は、ほかのエージェント名前付けの方法より優先されます。 「Java システム プロパティを使用したエージェント名の指定 (P. 154)」を参照してください。

方法 2: IntroscopeAgent.profile のシステム プロパティ キーに指定されたエージェント名

エージェント名は、*IntroscopeAgent.profile* 内のプロパティで指定される Java システム プロパティから取得されます。 「システム プロパティ キーを使用したエージェント名の指定 (P. 154)」を参照してください。

方法 3: アプリケーション サーバから自動的に取得されたエージェント名

特定のバージョンの WebLogic または WebSphere を使用している場合、エージェント自動名前付け機能を使用して、エージェント名を自動的にアプリケーション サーバから取得できます。 遅延を設定して、Enterprise Manager に接続する前に名前を決定するのに必要な時間をエージェントに与えることができます。 「アプリケーション サーバからのエージェント名の取得 (P. 154)」を参照してください。

方法 4: エージェント プロファイルで明示的に指定されたエージェント名

エージェント名は、*IntroscopeAgent.profile* のプロパティ *introscope.agent.agentName* で定義されています。 初期のバージョンの Introscope では、この方法がエージェントの名前付けの標準でした。 すでにアプリケーション別のエージェント プロファイルがある場合は、このオプションを使用します。

方法 5: 未知のエージェントであると判断されたエージェントの名前

上記のいずれかの方法を使用してもエージェントの名前を判断できなかった場合は、そのエージェントに「UnnamedAgent」という名前が付けられます。

Introscope によるエージェント名前付けの競合の解決方法

ホスト名、プロセス名、およびエージェント名で構成されるエージェントの完全修飾名は、通常、Introscope 環境内の各エージェントで一意です。ほとんどの場合、ホスト名とプロセス名は異なるため、複数のエージェントが同じエージェント名を持つ場合は、一意の「*完全修飾*」エージェント名が付けられます。 複数のエージェントが同じ完全修飾エージェント名を持つのは、それらが同じホスト上にあり、同じプロセスのモニタを行い、同じエージェント名を持っている場合のみです。

エージェントが Enterprise Manager に接続しようとする際に、同じ完全修飾エージェント名を持つ別のエージェントがすでに接続されている場合、Enterprise Manager は、新しく接続するエージェントの名前の最後に一意の識別子を追加します。 識別子は、パーセント（%）文字と数字で構成されます。 このメカニズムによって、同じ完全修飾名を使用して接続する複数のエージェントを、接続中に一意に識別できるようにします。 Enterprise Manager は、接続しているエージェントのうち、重複する 1 つ目のエージェントの名前を、その末尾に「%1」を追加して変更します。

たとえば、以下の完全修飾エージェント名を持つ 2 つのエージェントがあるとします。

```
hostPA|processNIM|PodAgent
```

Enterprise Manager に 1 つずつ接続します。 Enterprise Manager は 2 つ目のエージェントの名前を以下のように変更します。

```
PodAgent%1
```

同じ完全修飾名の別のエージェントが接続されると、順に *PodAgent%2*、*PodAgent%3*、*PodAgent%4* のように名前が変更されます。パーセント文字の後の数字は、続き番号です。

名前が変更されたエージェントが切断されると、割り当てられていた末尾の文字列を再使用できます。 たとえば、*PodAgent* の接続中に *PodAgent%1* が切断された場合、次に接続される、完全修飾名 *hostPA|processNIM|PodAgent* のエージェントの名前は、*PodAgent%1* に変更されます。

末尾の識別子を再使用することで、Enterprise Manager は、接続のたびに特定のエージェント名の末尾に同じ文字列を追加することが可能です。 しかし、連続する接続で、対象となるエージェントの名前変更をこの方法で行うことはお勧めしません。 エージェントの名前が接続のたびに変わるのは、履歴データをクエリする際に問題になります。 したがって、Enterprise Manager がエージェントの名前を変える必要がないような名前付けの方策を講じることをお奨めします。

クラスタ化されたアプリケーション用のエージェントの名前付けに関する考慮事項

同じアプリケーションの複数のインスタンスを実行する場合、Introscope はカスタム メトリック エージェントを*含む*同一のエージェント名を、名前に文字と乱数を追加することによって解決しようとします。 ただし CA Technologies では、Introscope が別の方法で名前付けを解決するように設定することをお勧めします。

同一 エージェント名を解決するためのオプションには以下のものがあります。

- 重複するエージェントの名前付けを有効にすることによって（「クラスタ化された環境での重複したエージェントの名前付けの有効化 (P. 159)」を参照）、問題のエージェントが重複するエージェントであることを Introscope に通知します。
- 一意のエージェント名を自分で定義し、エージェント別のエージェント プロファイルを用意します（「アプリケーション インスタンスに対する一意の名前の設定 (P. 160)」を参照）。
- Introscope 独自の名前付け方法を使用して、各エージェントに一意の名前を付けるように Introscope を設定します（「Introscope によるエージェント名前付けの競合の解決方法 (P. 152)」を参照）。

Java システム プロパティを使用した エージェント名の指定

Java システム プロパティを使用してエージェント名を指定する方法

- Java コマンド ラインから、このプロパティを使用して任意の名前を指定します。
  `-Dcom.wily.introscope.agent.agentName=`

システム プロパティ キーを使用したエージェント名の指定

この方法は、エージェントが名前の検索に使用する 2 番目の方法です。導入環境での既存の Java システム プロパティの値を使用してエージェントに名前を付ける場合は、この方法を使用します。

システム プロパティ キーを使用してエージェント名を指定する方法

1. *IntroscopeAgent.profile* を開きます。
2. 「Agent Name 」セクションにある以下のプロパティで、エージェント名を提供する Java システム プロパティを指定します。
   `introscope.agent.agentNameSystemPropertyKey`

   注: ここで指定された Java システム プロパティが存在しない場合、このプロパティは無視されます。
3. アプリケーション サーバを再起動します。

アプリケーション サーバからのエージェント名の取得

エージェントがアプリケーション サーバからアプリケーション サーバ インスタンス名を自動的に取得し、その情報をもとにエージェント自体の名前付けを行うように設定することができます。 これによって、エージェント プロファイル ファイルごとに個別のエージェント名を設定する手間が省けます。アプリケーション サーバ環境で変更があった場合、エージェントは、自身の名前を変更できます。これにより、アプリケーション サーバ プラットフォームが混在している可能性がある多数の環境に、単一のエージェント プロファイルを展開できます。

エージェントの名前付けをサポートするアプリケーション サーバ

エージェント自動名前付け機能がサポートされるのは、以下のサポートされているアプリケーション サーバのバージョンで Introscope を使用する場合です。

- JBoss
- WebLogic 9.x
- WebSphere 6.1.x 分散型
- WebLogic 10.0
- WebSphere 7.0.x 分散型
- WebLogic 10.3

Introscope Workstation に表示されるアプリケーション サーバの名前は、Java J2EE API によって決定されます。すべてのアプリケーション サーバで API の実装方法が異なるため、それが原因でアプリケーション サーバの名前が Workstation 内で異なって表示されることがあります。Workstation 内では、複数のアプリケーション サーバの名前が異なる方法でフォーマットされており、アプリケーション サーバの名前が同じでも、リリースごとに異なる方法でフォーマットされている可能性があります。

エージェント自動名前付け

エージェント自動名前付け機能が有効になっている場合、エージェントは、起動時にアプリケーション サーバからの名前情報を検索します。 エージェントは、エージェント名を取得してから Enterprise Manager に接続します。

エージェントが名前付けに必要な情報を見つけると、Introscope は、その情報を編集して、エージェント名をエージェント名前付け規則に準拠させます。

サポートされているアプリケーション サーバでのエージェント名は、いくつかの情報の要素から構成され、それらはアプリケーション サーバごとに異なります。

- JBoss の場合、エージェント名はサーバの起動時に指定される設定名に基づいています。
- WebLogic の場合、エージェント名は以下のようになります。

  ドメイン（データ センター） + クラスタ + （WLS の）インスタンス
- WebSphere の場合、エージェント名は以下のようになります。

  セル（ドメイン） + プロセス（WAS のインスタンス）

情報が取得されると、セグメントは、以下の例のようにスラッシュで区切られます。

```
medrec/MyCluster/MedRecServer
```

個々のセグメントの名前に含まれるスラッシュはすべて下線に変換されます。 たとえば、ドメインの名前が Petstore/West である場合は、Petstore_West と変換されます。

注: Introscope は、以下の規則に従って、エージェント名の作成に使用される情報を編集します。

- パイプ、コロン、パーセンテージ記号などの文字は、下線に置き換えられます。
- 英字以外の文字で始まる名前には、先頭に「A」が追加されます。
- 空の名前は、「UnknownAgent」である状況と区別するために「UnnamedAgent」に置き換えられます。

エージェント自動名前付け機能を有効にする方法

1. *IntroscopeAgent.profile* で、*introscope.agent.agentAutoNamingEnabled* を *true* に設定します。
2. 以下のアプリケーション サーバ固有の変更を行います。
   - WebLogic の場合、Introscope 起動クラスを作成します。「WebLogic 用の起動クラスの設定 (P. 49)」を参照してください。
   - WebSphere の場合、Introscope カスタム サービスを作成します。「WebSphere でのカスタム サービスの設定 (P. 61)」を参照してください。
   - JBoss の場合、XML ファイルを作成します。 「JBoss の設定 (P. 45)」を参照してください。

エージェント自動名前付けおよび名前変更されたエージェント

エージェント自動名前付け機能を使用すると、エージェントは常に、各アプリケーション サーバ固有の最新のエージェント名を取得しようとします。このために、エージェントは定期的に新しい名前をチェックします。

アプリケーション サーバ設定を変更した結果としてエージェント名が変更される場合、エージェントは自動的に自身の名前を変更します。 このとき、このエージェントは、Investigator ツリーでは切断されているものとして表示されます。 切断されたエージェントは、引き続き Investigator ツリーに表示され、定義されているマウント解除までの待機時間が経過した後、自動的にマウント解除されます。また、手動でもマウント解除できます。

名前変更されたエージェントは、Enterprise Manager に再接続され、Investigator ツリーに表示されます。 エージェントは、これらの変更を記録します。

Enterprise Manager の接続遅延のエージェント自動名前付けプロパティの設定、および名前変更のチェック間隔の設定の詳細については、「エージェントの高度な自動名前付けオプション (P. 157)」を参照してください。

エージェントの高度な自動名前付けオプション

ご使用の環境に合わせてエージェント自動名前付け機能を制御するために変更できるプロパティがいくつかあります。

Enterprise Manager の初期接続の遅延

エージェント自動名前付け機能を使用する場合、エージェントは、Enterprise Manager に接続する前に、エージェント名情報を検索する長さとして設定された時間まで待機します。 デフォルトの遅延は 120 秒です。

遅延の値を変更する方法

1. *IntroscopeAgent.profile* を開きます。
2. 「**Agent Name**」セクションのプロパティ *introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds* で必要な遅延を設定します。
3. アプリケーション サーバを再起動します。

エージェントの名前変更チェック間隔

エージェント自動名前付け機能を使用している場合、エージェントはアプリケーション サーバの名前付け情報が変更されているかどうかを定期的にチェックします。 デフォルト間隔は 10 分です。

この間隔を変更する方法

1. *IntroscopeAgent.profile* を開きます。
2. 「**Agent Name**」セクションのプロパティ *introscope.agent.agentAutoRenamingIntervalInMinutes* で、間隔を設定します。
3. アプリケーション サーバを再起動します。

エージェント ログ ファイルの自動名前付けの無効化

デフォルトでは、Java システム プロパティまたはアプリケーション サーバから提供された情報によってエージェント名が自動的に検出されたときに、そのエージェントに関連付けられているログ ファイルにも、同じ情報を使用して自動的に名前が付けられます。 ただし、このログの自動名前付けをオフにして、*IntroscopeAgent.profile* で指定されたエージェント ログ名を引き続き使用することもできます。

エージェント ログ ファイルの自動名前付けを無効にする方法

1. *IntroscopeAgent.profile* を開きます。
2. プロパティ *introscope.agent.disableLogFileAutoNaming* の値を *true* に設定します。
3. *IntroscopeAgent.profile* を保存します。
4. アプリケーション サーバを再起動します。

クラスタ化された環境での重複したエージェントの名前付けの有効化

同じホストおよびプロセスのモニタを行う同じ名前のエージェントが 2 つ存在し、これらにユーザが一意の名前を付けていない場合、名前に数が追加されます。 重複するエージェントの名前付けを有効にすると、エージェントとクラスタ化されたアプリケーション内の特定のアプリケーション インスタンスを互いに関係付けることができます。

以下の場合は重複するエージェントを実行しています。

- ホスト、プロセス、または Java Agent 名を、1 つ以上のほかのエージェントと共有しているエージェントを実行している。
- 同じエージェント プロファイルを使用している複数のエージェントを実行している。

エージェントの重複した名前付け機能を有効にする方法

1. 管理対象アプリケーションおよび Java Agent を停止します。
2. *IntroscopeAgent.profile* を開き、以下のプロパティを true に設定します。
   `introscope.agent.clonedAgent=true`
3. *IntroscopeAgent.profile* を保存します。
4. 管理対象アプリケーションおよび Java Agent を再起動します。

重複するエージェントの名前付けシナリオ

Java Agent の重複プロパティがオンになっている場合、名前がすべて *AgentX* である 4 つの Java Agent があると、Enterprise Manager はこれらのエージェントに *AgentX-1*、*AgentX-2*、*AgentX-3*、および *AgentX-4* という名前を付けます。 *AgentX-1* が切断されて再接続された場合は、再接続時に、名前としてまた *AgentX-1* を使用します。 この名前付けでは、データベース内の Java Agent 名の数が、元の重複した Java Agent の数を超えることはなくなります。

アプリケーション インスタンスに対する一意の名前の設定

同じマシン上にあるアプリケーションの複数のインスタンスを監視する場合、以下のように一意のエージェント名を明示的に設定できます。

一意のエージェント名を設定する方法

1. 各アプリケーションに別々のエージェント プロファイルを作成します。
2. エージェント プロファイルで各エージェントに一意の名前を付けます。
3. 各アプリケーションが使用するエージェント プロファイルを指定します。

アプリケーション問題切り分けマップとエージェント名

Workstation のアプリケーション問題切り分けマップでは、Introscope データベースにアプリケーション コンポーネントの情報を保存する場合に加えて、フロントエンドおよびバックエンドのアプリケーションを定義する場合にも、いくつかの一意の識別子の一部としてエージェント名を使用します。 エージェント名が変更されると、アプリケーション問題切り分けマップの一部の内容も変更される場合があります。 たとえば、エージェントを初めて登録するときに、Enterprise Manager は、*%<sequence number>* をエージェント名に追加して（*MyAgent%1* など）、重複したエージェント名を一意の名前として割り当てる場合があります。 アプリケーション問題切り分けマップの一部が、重複したエージェント名に情報を依存している場合、マップの内容が変わる場合があります。

これはエージェントや Enterprise Manager の正常な機能に影響しませんが、システム全体のキャパシティを低下させる場合があります。そのため、Introscope 環境で使用する命名規則には十分注意することをお勧めします。この問題を回避するために、明確なエージェント名を指定することもできます。

第 7 章: Java Agent のモニタリングおよびログ記録

Introscope でアプリケーションを監視する際は、Java Agent 自体の稼働状況およびアクティビティも監視できます。 このセクションでは、エージェントの稼働状況の監視、および Java Agent のログ記録オプションについて説明します。

このセクションには、以下のトピックが含まれています。

エージェントの接続メトリックの設定

デフォルトで、Introscope は、Enterprise Manager に接続されているエージェントの接続ステータスについて監視可能なメトリックを生成します。エージェントの接続メトリックを監視すれば、エージェントと Enterprise Manager 間の接続の現在の状態を判断できます。

エージェントの接続メトリックは、Workstation Investigator の Enterprise Manager プロセス（カスタム メトリック ホスト）の下に表示されます。
`Custom Metric Host (Virtual) ¥ Custom Metric Process(Virtual) ¥ Custom Metric Agent (Virtual) (*SuperDomain*) ¥ Agents ¥ <HostName> ¥ <Agent Process Name> ¥ <Agent Name> ¥ ConnectionStatus`

接続メトリックに設定される値を以下に示します。

- 0： エージェントに関して利用できるデータがありません。
- 1： エージェントは接続されています。
- 2： エージェントは、レポートを行うために速度が低下しています。
- 3： エージェントは切断されています。

エージェントの切断によって、「注目点」イベントも生成されます。 ほかのイベントと同様に、ユーザは、履歴クエリ インターフェースを使用して、エージェントの切断についてクエリを実行できます。 エージェント切断イベントは、アプリケーションの稼働状況の評価に使用されるデータの一部となり、Workstation コンソールの［概要］タブに表示されます。

エージェントが Enterprise Manager から切断された後、Introscope はエージェントがタイムアウトするまで切断状態のメトリックの生成を続けます。 エージェントがタイムアウトすると、接続メトリックは生成されなくなり、Enterprise Manager にもレポートされません。

以下の手順に従います。

1. Enterprise Manager がインストールされているコンピュータで、*<Introscope_Home>/config* ディレクトリにある *IntroscopeEnterpriseManager.properties* ファイルを開きます。
2. 以下のプロパティを変更します。
   introscope.enterprisemanager.agentconnection.metrics.agentTimeoutInMinutes
   時間は分単位で指定します。
3. *IntroscopeEnterpriseManager.properties* を保存します。

注: Enterprise Manager のプロパティについては、「CA APM 設定および管理ガイド」を参照してください。

「ソケット メトリック」

ソケットおよび Secure Sockets Layer（SSL）のメトリック コレクションは、エージェントではデフォルトで有効になっています。

注: *Agent.NoRedef.jar* を使用している JVM では、ソケット メトリックがレポートされません。 詳細については、「WebSphere 6.1 用の AutoProbe (P. 56)」を参照してください。

ソケットおよび SSL メトリック コレクションの制限

ソケットと SSL のメトリック コレクションはデフォルトでは有効になっています。ただし、一部のメトリック コレクションを制限して、より関連性の高い情報を対象にしたり、オーバーヘッドを削減することができます。

ソケットおよび SSL のメトリック コレクションを制限する方法

1. *IntroscopeAgent.profile* ファイルをテキスト エディタで開きます。
2. Agent I/O Socket Metrics セクションで以下のプロパティの値を編集し、メトリックが必要とするホストのまたはポートのリストを追加します。

   パラメータの値に無効なホストまたはポートが含まれていると、エージェント ログに警告メッセージが書き込まれ、その値は無視されます。その結果としてリストにエントリがない場合、制限は適用されません。

   - *introscope.agent.io.socket.client.hosts*
     カンマ区切りのホストのリスト。指定されたホストの「クライアント」のソケット メトリックのみが生成されます。 ホストは、名前または IP アドレスのテキスト表現（IPv4 または IPv6 形式のいずれ）を使用して指定できます。

     注: 重複したホスト名は無視されます。複数のホスト名が 1 つの IP にマップされている場合、1 つの名前のみが保持されます。 ただし、プロパティは、同じ意味の名前のいずれかのセットとクライアント接続を照合します。

   - *introscope.agent.io.socket.client.ports*
     カンマ区切りのポート番号のリスト。指定されたポートの「クライアント」ソケット メトリックのみが生成されます。

     注: 重複したポート名は無視されます。

   - *introscope.agent.io.socket.server.ports*
     カンマ区切りのポート番号のリスト。指定されたポートの「サーバ」ソケット メトリックのみが生成されます。

   上記のプロパティは動的です。変更を有効にするためにアプリケーションを再起動する必要はありません。
3. *IntroscopeAgent.profile* を保存します。

ソケットおよび SSL メトリック コレクションの微調整

「ソケットおよび SSL メトリック コレクションの制限 (P. 163)」で説明したように、ソケットおよび SSL メトリック コレクションを制限できる一方で、特定のトレーサ グループをオンまたはオフにすることで、メトリック コレクションをさらに調整できます。 これによってオーバーヘッド コストを削減できるだけでなく、必要な情報を対象にすることができます。

ソケットおよび SSL メトリック コレクションを微調整する方法

1. *<Agent_Home>*/wily/core/config ディレクトリの *java2.pbd* ファイルを開きます。
2. *java2.pbd* の I/O Socket Tracer Group または Network Tracer Group セクションで、オンまたはオフにするトレーサを見つけて、そのトレーサをコメント化するか、コメント化を解除します。 たとえば、入力帯域幅のメトリックを抑制するには、以下をコメント化します。

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

   注: *MappingTracer* で終わる名前を持つトレーサはコメント化しないでください。
3. *java2.pbd* ファイルを保存します。

アプリケーション問題切り分けマップでの SSL、NIO、およびソケットの追跡

Workstation Investigator のアプリケーション問題切り分けマップには、インスツルメントされたクライアント ソケット接続を表示できます。 エージェントは、トランザクションで使用されている外部システムの詳細を記録し、その情報を Enterprise Manager に送信します。 この情報は、アプリケーション問題切り分けマップにグラフィカルに表示されます。

<Agent_Home>/wily/core/config ディレクトリの appmap.pbd にあるトレーサは、既存の SSL、NIO、およびソケット インスツルメンテーションを拡張します。このため、エージェントはアプリケーション問題切り分けマップにより多くのコンポーネント情報を送信できます。 これらのトレーサは、デフォルトで有効になっています。appmap.pbd にある Trace Sockets および Trace NIO Sockets セクションの特定のトレーサをコメント化することで、無効にすることができます。

アプリケーション問題切り分けマップのコンポーネント名の変更

アプリケーション問題切り分けマップに表示される場合、コンポーネント名には宛先ホストとポート ID が含まれます。クライアントのソケット メトリック名も同様に表示されます。 コンポーネント名のホスト ID には、ホスト名またはホストの IP アドレスのどちらかを使用するように設定できます。 デフォルトではホスト名を使用します。 表示されるコンポーネント名を変更できます。

以下の手順に従います。

1. *<Agent_Home>*/wily/core/config ディレクトリの appmap.pbd ファイルを開きます。
2. Trace Sockets および Trace NIO Sockets セクションで、変更するトレーサを選択します。
3. 関連するトレーサの *{hostname}* を *{hostip}* に変更します。

   たとえば、以下のように、元のトレーサはデフォルトの *{hostname}* を使用しています。

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

   ホスト IP を表示するには、代わりに *{hostip}* を使用します。

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

4. appmap.pbd ファイルを保存します。

ソケットおよび SSL メトリック コレクションの無効化

ソケットおよび SSL のメトリック コレクションが不要な場合は、これを完全にオフにすることができます。

ソケットおよび SSL メトリック コレクションをオフにする方法

1. *toggles-full.pbd* または *toggles-typical.pbd* ファイルを開きます（デプロイ時に使用したファイルを開きます）。
2. 以下のように、*SocketTracing* をコメント化します（シャープ記号またはハッシュ記号（#）を行頭に挿入します）。
   ```
   #TurnOn: SocketTracing
   ```
3. 変更したファイルを保存します。

toggles-full.pbd および *toggles-typical.pbd* ファイルでのトレーサ グループのオンまたはオフの詳細については、「デフォルトのトレーサ グループおよびトグル ファイル (P. 110)」および「トレーサ グループのオンまたはオフ (P. 121)」を参照してください。

下位互換性

対象：リリース 9.0 より前のトレーサ

Java エージェント ソケット トレーサは、9.0 より前のリリースのトレーサよりも多くの内容を設定できます。 ただし、以前のトレーサに戻す（およびソケット追跡機能と設定オプションを無効にする）ことができます。

9.0 より前のリリースのトレーサを使用している場合、Java エージェントを設定できます。

- ソケット メトリックおよび以前のトレーサを収集する (P. 167) 方法。
- 入出力帯域幅メトリックを収集する (P. 167) 方法。

ソケット メトリックの収集

対象: リリース 9.0 より前のトレーサ

トレーサを使用して、ソケット メトリックを収集するには、Java エージェントを設定します。

以下の手順に従います。

1. toggles-full.pbd または toggles-typical.pbd ファイルを開きます（デプロイ時に使用したファイルを開きます）。
2. 以下のように、SocketTracing をコメント化します（シャープ記号またはハッシュ記号（#）を行頭に挿入します）。
   `#TurnOn: SocketTracing`
3. ManagedSocketTracing のコメント化を解除します。
   `TurnOn: ManagedSocketTracing`
4. ファイルを保存します。

入出力帯域幅メトリックを収集する方法

対象: リリース 9.0 より前のソケット トレーサ

ソケット トレーサを使用しており、入出力帯域幅メトリックが必要な場合は、Java エージェントを設定します。

以下の手順に従います。

1. IntroscopeAgent.profile を開きます。
2. Agent Socket Rate Metrics セクションを見つけて、以下のプロパティを true に変更します。
   `introscope.agent.sockets.reportRateMetrics=true`

   注: ManagedSocketTracing が有効で、SocketTracing が無効である場合にのみ、機能します。
3. IntroscopeAgent.profile を保存します。

ログ記録オプションの設定

アプリケーション サーバに Java Agent をインストールして、サーバを起動すると、ログ ディレクトリが *<Agent_Home>*/wily/logs に作成されます。アプリケーション サーバ プロセスは、*<Agent_Home>* ディレクトリに対して完全な読み取り/書き込み/実行権限を持っている必要があります。これを実現するには、アプリケーション サーバ プロセスを実行しているオペレーティング システム上の同じユーザで Java Agent をインストールします。 または、Java Agent を別のユーザとしてインストールし、*chmod* コマンドを使用して必要な権限を付与します。

Java Agent には冗長モードで実行するオプションが用意されています。 冗長モードでは、アクションおよびご利用の環境とのエージェントのやり取りに関してより詳細なレベルで記録します。この情報は、環境またはエージェントの機能についての問題を解決する場合に役立ちます。

Introscope では、これらに Log4J 機能が使用されます。ほかの Log4J 機能を使用する場合は、Log4J のマニュアルを参照してください。

冗長(verbose)モードでのエージェントの実行

エージェントを冗長モードで実行すると、エージェント ログにはより詳細なレベルの情報が記録されます。

エージェントを冗長モードで実行する方法

1. *IntroscopeAgent.profile* ファイルをテキスト エディタで開きます。
2. このプロパティを変更して、既存の *INFO* を *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel* に置き換えます。

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel, console, logfile
   ```

3. *IntroscopeAgent.profile* を保存します。

   注: このプロパティへの変更は、1 分以内に有効となり、管理対象アプリケーションを再起動する必要はありません。

エージェント出力のファイルへのリダイレクト

冗長モードでのエージェントのログ記録を制御するプロパテは、エージェント ログが出力される場所およびこのログ ファイルの場所も制御します（詳細については、「冗長（verbose）モードでのエージェントの実行 (P. 168)」を参照）。

エージェント出力をファイルにリダイレクトする方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. プロパティ *log4j.logger.IntroscopeAgent* を探します。

   このプロパティのオプションは以下のようになります。

   - *console*： ログファイルの情報はコンソールに送信されます。
   - *logfile*： ログファイルの情報はログファイルに送信されます。 これを選択した場合、ログ ファイルの場所は *log4j.appender.logfile.File* プロパティを使用して設定されます。「エージェントのログ ファイルの名前または場所の変更 (P. 170)」を参照してください。

   たとえば、エージェントが冗長モードでログ ファイルのみにレポートするように設定する場合、プロパティは以下のように設定します。

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel,logfile
   ```

   エージェントがログファイルとコンソールの両方にレポートするように設定する場合、プロパティには *logfile* と *console* の両方を含めます。

   注: デフォルトでは、エージェント ログ *IntroscopeAgent.log* は *<Agent_Home>*/wily/logs ディレクトリに書き込まれます。 エージェントの自動名前付けオプションを設定している場合、エージェントのログ ファイルも「エージェントのログ ファイルおよびエージェントの自動名前付け (P. 170)」に説明されているように自動的に名前が付けられます。
3. *IntroscopeAgent.profile* を保存します。

エージェントのログ ファイルの名前または場所の変更

プロパティを変更してログ ファイルの場所および名前を変更することもできます。

ログ ファイルの名前または場所を変更する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *log4j.appender.logfile.File* プロパティを探します。

   logfile が *log4j.logger.IntroscopeAgent* プロパティで指定されている場合、ログ ファイルの場所は *log4j.appender.logfile.File* プロパティを使用して設定されます。 詳細については、「エージェント出力のファイルへのリダイレクト (P. 169)」の手順 2 を参照してください。

   注: システム プロパティ (Java コマンド ライン -D オプション) は、ファイル名の一部に含まれます。 たとえば、Java コマンドが、*-Dmy.property=Server1* で起動されると、*log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* は *log4j.appender.logfile.File=../../logs/Introscope-Server1.log* に拡張されます。
3. 新しい場所とファイル名の完全修飾パスを使用して、ログ ファイルの場所と名前を設定します。 例：

   ```
   log4j.appender.logfile.File=C:/Logs/AgentLog1.log
   ```
4. *IntroscopeAgent.profile* を保存します。

エージェントのログ ファイルおよびエージェントの自動名前付け

エージェント自動名前付け機能を使用している場合は、デフォルトで、そのエージェントに関連付けられているログファイルにも、エージェントの名前付けと同じ情報を使用して自動的に名前が付けられます。

エージェントの自動名前付けは、以下の方法でログ ファイルにも影響があります。

- ログ ファイルの元の名前が *.log* で終わっていない場合は、末尾にピリオドおよび *log* が追加されます。

- 英字または数字以外のすべての文字は下線で置き換えられます。
- 高度な Log4J 機能が使用されている場合、エージェント ログ ファイルの自動名前付け機能は動作しません。

以下の例では、エージェントのログ ファイルの名前付けの方法を示しています。この例では、エージェント名 *DOM1//ACME42*（*DOM1* は WebLogic ドメイン、*ACME42* はインスタンス）を使用します。

エージェント ログ ファイル（デフォルト名は *AutoProbe.log*）が作成されるときに、まだエージェント名が利用できない場合は、次のようにファイル名にタイムスタンプが含まれます。

```
AutoProbe.20040416-175024.log
```

エージェント名を利用できるようになると、エージェントの自動名前付けを使用してログ ファイル名は以下のように変更されます。

```
AutoProbe.DOM1_ACME42.log
```

ログの自動名前付けを無効にすることもできます。詳細については、「エージェントの高度な自動名前付けオプション (P. 157)」を参照してください。

日付またはサイズによるログのロール アップ

サイズまたは日付に基づいてログをロール アップし、指定された日数の情報を保持して残りをパージすることができます。

ログ ファイルをロール アップする方法

1. *IntroscopeAgent.profile* を開き、Logging Configuration セクションを探します。
2. 以下のプロパティを変更します。

```
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
```

注: このプロパティの変更を有効にするには、管理対象アプリケーションを再起動する必要があります。

3. *IntroscopeAgent.profile* を保存します。

   たとえば、以下の設定では、バックアップ ログまたはロールアップされたログを最大 3 つまで維持します。また、各ログのサイズは 2 キロバイト以内にします。

```
log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel, 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.AutoNamingRollingFileAppender
log4j.appender.logfile.MaxFileSize=2KB
log4j.appender.logfile.MaxBackupIndex=3
```

ProbeBuilder ログの管理

ProbeBuilder は、参照するすべてのクラス、インスツルメントするすべてのクラス、さらにインスツルメンテーションを追加しないすべてのクラスをログ記録します。また、インスツルメンテーション プロセス中に追加したプローブ、および使用した PBD も記録します。 さらに、スキップしたためにインスツルメントされなかったクラスもログ記録します。

コマンド ライン ProbeBuilder および ProbeBuilder ウィザードのログの名前および場所

コマンドライン ProbeBuilder と ProbeBuilder ウィザードのログ ファイルの場所は、ProbeBuilder ウィザードまたはコマンド ライン ProbeBuilder で Java クラスを指定した場所によって決まります。ディレクトリの場合、ログ ファイルは、出力先ディレクトリ内に置かれます。 ファイルの場合、ログ ファイルは、出力先ファイルの近くに置かれます。

ProbeBuilder ログ ファイルは以下のような名前になります。

```
<original-directory-or-original-file>.probebuilder.log
```

<original-directory> または *<original-file>* は、ProbeBuilder ウィザードまたはコマンド ライン ProbeBuilder で指定した Java クラスの場所です。

最新のログのみが維持され、以前のログ ファイルはすべて上書きされます。

AutoProbe ログの名前および場所

AutoProbe は常に変更を記録しようとしています。 デフォルトの AutoProbe ログ ファイルの名前は *AutoProbe.log* です。

AutoProbe ログの名前または場所を変更する方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *introscope.autoprobe.logfile* プロパティを探し、完全修飾されたファイル パスを使用して、ログの名前および場所を変更します。絶対名以外で指定すると、I*ntroscopeAgent.profile* ファイルを起点とする相対的な指定と見なされます。

   注: エージェント プロファイルをクラスパスのリソースからロードする場合、*IntroscopeAgent.profile* ファイルがリソース内に置かれているため、AutoProbe が自分のログ ファイルに書き込むことができなくなります。

   このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
3. *IntroscopeAgent.profile* を保存します。

第 8 章: LeakHunter および ErrorDetector の設定

このセクションでは、LeakHunter と ErrorDetector を有効にして設定する方法について説明します。

このセクションには、以下のトピックが含まれています。

LeakHunter

Introscope LeakHunter は、メモリ リークの可能性があるソースの特定を支援するアドオン コンポーネントです。メモリ リークの可能性のあるソースの特定は、時間の経過と共にサイズが増加している（つまり、コレクションに格納されるオブジェクトの数が時間の経過と共に増加している）コレクション インスタンスの監視によって行います。

短時間（数分から数時間）で実行されるプログラムでメモリ リークが発生しても、大きな問題にならない可能性が高いと考えられます。しかし、Web サイトのように 1 日 24 時間実行されるアプリケーションでは、小さなメモリ リークがすぐに大きな問題に発展する可能性があります。

Introscope LeakHunter は、起動し、コレクション クラスを検出し、情報の収集を終えると停止することで、メモリ リークに関する情報を追跡するように設計されています。LeakHunter をこのように動作させることにより、発生するオーバーヘッドはわずかで、一時的なものに抑えられます。

LeakHunter の仕組み

LeakHunter を有効にしたら、LeakHunter が新たな潜在的リークを探索するタイムアウト期間も定義します。 AutoProbe を使用する場合、必要な操作は管理対象アプリケーションの再起動のみです。 ProbeBuilder ウィザードまたはコマンドライン ProbeBuilder を使用する場合は、（すでに使用している PBD ファイルに加えて）leakhunter.pbd を使用してアプリケーションを再インスツルメントする必要があります。

LeakHunter は、時間の経過と共にサイズが増加しているコレクションを検出すると、以下の処理を行います。

- コレクションを一意の ID で特定する
- コレクションに関する情報をメトリック データとして Enterprise Manager にレポートする
- エージェント マシン上のログ ファイルにコレクションに関する情報をレポートする
- そのコレクションに関するデータの追跡とレポートを続行する

コレクションのリークが収まったと思われる場合、LeakHunter はその旨を Enterprise Manager とログ ファイルにレポートしますが、そのコレクションに関するデータの追跡とレポートは継続します。

LeakHunter は、タイムアウトになるまで、潜在的なリークの検出を継続すると共に、特定済みの潜在的リークを監視します。 タイムアウトになると、LeakHunter は新しく割り当てられたコレクションにおける潜在的リークの検出は停止して、リークの可能性があると識別済みのコレクションに対してのみチェックを継続します。 これにより、LeakHunter のオーバーヘッドが大幅に減り、他の潜在的リークの監視をさらに行うことができます。LeakHunter は、管理対象アプリケーションがシャットダウンされるまで、特定済みの潜在的リークを継続して監視します。

メモリ リークのソースを特定するには、Introscope Investigator のメトリック データを参照するか、ログ ファイルを確認します。

Java 環境での LeakHunter の追跡対象

Java 実装環境では、Introscope LeakHunter は以下のコレクションを追跡します。

- java.util.Collection の実装
  - java.util.ArrayList
  - java.util.LinkedList
  - java.util.TreeSet
  - java.util.HashSet
  - java.util.LinkedHashSet
  - java.util.Vector
  - java.util.Stack
- java.util.Map の実装
  - 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 で追跡されない対象

Introscope LeakHunter で追跡されない対象は以下のとおりです。

- コレクションが原因でないリーク。
- カスタム コレクションの実装、または参照の数が増えるその他のデータ構造
- リークを発生させているインスツルメントされていないコレクション

このほかに、Java 実装環境では、LeakHunter は以下の原因によるリークを追跡しません。

- 「Java 環境での LeakHunter の追跡対象 (P. 177)」で説明したコレクションで作成されたサブクラス。 ただし、ProbeBuilder ディレクティブ（PBD）ファイルをアップデートすると、この情報を取得できます。詳細については、*leakhunter.pbd* ファイルを参照してください。

注: アプリケーション サーバ AutoProbe を使用している場合、LeakHunter はアプリケーション サーバが割り当てたコレクションを自動的には追跡しません。 このようなコレクションを追跡するには、アプリケーション サーバを静的にインスツルメントするか、JVM AutoProbe を使用する必要があります。

システムおよびバージョン要件

Introscope LeakHunter のシステム要件は、Java Agent のシステム要件と同じです。

インストール後、デフォルトでは、LeakHunter は有効化されていません。LeakHunter の機能を使用するには、LeakHunter を有効化する必要があります。

LeakHunter の有効化および無効化

LeakHunter はエージェントの拡張機能として実行されるため、クラスパスをアップデートする必要はありません。 インストール後、デフォルトでは、LeakHunter は有効化されていません。LeakHunter の機能を使用するには、LeakHunter を有効化する必要があります。

LeakHunter を有効にする方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。

2. *LeakHunter Configuration* の見出しの下で、プロパティ *introscope.agent.leakhunter.enable* を探して、値 *true* を入力します。
3. エージェント プロファイル を保存します。
4. **.typical.pbl* または **.full.pbl* を開きます（*IntroscopeAgent.profile* のプロパティ *introscope.autoprobe.directivesFile* にリストしたファイルを開きます）。そして、*leakhunter.pbd* のコメント化を解除します。

   注: ProbeBuilder ウィザードを使用する場合は、*leakhunter.pbd* ファイルを *<Agent_Home>¥wily¥core¥config¥hotdeploy* ディレクトリにコピーします。
5. アプリケーションを再起動します。

重要: デフォルトでは、LeakHunter のようなエージェント拡張機能は、*<Agent_Home>*¥wily¥core¥ext ディレクトリにあり、ここから参照されます。ただし、エージェント拡張機能のディレクトリの場所は、エージェント プロファイルで変更することができます。*¥ext* ディレクトリの場所を変更する場合は、*¥ext* ディレクトリの内容も同様に移動する必要があります。

LeakHunter を無効にする方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. *LeakHunter Configuration* の見出しの下で、プロパティ *introscope.agent.leakhunter.enable* を探して、値 *false* を入力します。
3. エージェント プロファイル を保存します。
4. アプリケーションを再起動します。

LeakHunter プロパティの設定

LeakHunter の設定プロパティは、エージェント プロファイル *IntroscopeAgent.profile* にあります。

LeakHunter を設定する方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。

2. 必要に応じて、以下の LeakHunter プロパティを設定します。

   introscope.agent.leakhunter.logfile.location

   LeakHunter.log ファイルの場所を指定します。 ファイル名は、*<IntroscopeAgent.profile>* ディレクトリを基準にした相対的なパスで指定します。 このプロパティがコメント化されている場合、または値が指定されていない場合には、ログ ファイルへの書き込みは行われません。

   デフォルト値は *logs/LeakHunter.log* です。

   introscope.agent.leakhunter.logfile.append

   アプリケーションが再起動された後、既存のログ ファイルを新しいファイルで置き換えるか（値を false に設定した場合）、または既存のログ ファイルにログを追記するか（値を true に設定した場合）を指定します。

   デフォルト値は False です。

   introscope.agent.leakhunter.leakSensitivity

   メモリ リークを検出するための感度レベルを指定します。 リークの感度を高く設定するとレポートされる潜在リークの数が増え、感度を低く設定するとレポートされる潜在リークの数が減ります。

   プロパティの値は、1〜10 の整数である必要があります。

   デフォルトの感度レベルは 5 です。

   introscope.agent.leakhunter.timeoutInMinutes

   LeakHunter を使用して新たな潜在的リークを検出する期間（分単位）を指定します。 プロパティ値には負数以外の整数値を指定する必要があります。 値を 0 に設定すると「タイムアウトなし」になります。

   デフォルト値は 120 分です。

introscope.agent.leakhunter.collectAllocationStackTraces

割り当てスタック トレース データを収集するかどうかを指定します。 このオプションをオンにすると、システムの CPU の使用率とメモリ使用量が増加する可能性があります。 このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

デフォルト値は False です。

introscope.agent.leakhunter.ignore.*n*

LeakHunter が無視する特定のコレクションを指定します。

ジェネリック コレクションの場合は、たとえば System.Collections.Generic.List`1 のように、ジェネリック タイプの修飾子が含まれる構文を使用してください。

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

このプロパティのデフォルト値は以下のとおりです（*n*=0～4）。

```
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. エージェント プロファイルへの変更を保存します。

重要: *IntroscopeAgent.profile* には、LeakHunter が無視するパッケージを制御するプロパティが含まれています。 これらのプロパティは、デフォルトで有効化されています。これらのプロパティをコメント化すると、エージェント ログに例外がレポートされます。

パフォーマンスの低下を引き起こすコレクションの無視

size() メソッドがコレクション内のオブジェクトの数に時間比例して実行されるようなコレクションは、パフォーマンスを低下させます。つまり、*size()* メソッドの実行時間が長くなっていくような（たとえば、リストのサイズを取得するのにリストの各エレメントを移動してカウントするような、不適切に実装された LinkedList）コレクションは、アプリケーションのパフォーマンスに悪い影響を与えます。

このようなコレクションは、「LeakHunter プロパティの設定」に示すように、IntroscopeAgent.profile の ignore プロパティを使用して無視します。

LeakHunter の実行

LeakHunter は、エージェント拡張機能として実行されます。

LeakHunter を実行する方法

- JVM AutoProbe を使用する場合： LeakHunter を有効にしたら、アプリケーションを再起動します。
- コマンドライン ProbeBuilder を使用する場合： *leakhunter.pbd*（および以前から使用している既存の .pbd）を使用して、管理対象アプリケーションを再インスツルメントします。 管理対象アプリケーションを再起動します。
- ProbeBuilder ウィザードを使用する場合： ProbeBuilder ウィザードを実行して、カスタム pbd リストから *leakhunter.pbd* を選択します。 使用する新しい .jar を実装したら、管理対象アプリケーションを再起動します。

コレクション ID による潜在的リークの特定

LeakHunter では、潜在的なリークはそれぞれ一意のコレクション ID で特定されます。コレクション ID により、Investigator ツリーのメトリック データとログ ファイルのデータを相関させることや、アプリケーション間で不変的な名前を提供することができます。

一意のコレクション ID には、以下のいずれかのタイプに基づいた特定の構文があります。

<メソッド>-<4 桁のハッシュ コード>#<一意の番号>

<フィールド>-<4 桁のハッシュ コード>#<一意の番号>

- *<メソッド>*： コレクションが割り当てられたメソッドの名前
- *<フィールド>*： コレクションの割り当て先であるフィールドの名前
- *<4 桁のハッシュ コード>*： メソッド名またはフィールド名を含むクラスのフル ネームのハッシュ コード
- *#<一意の番号>*： メソッドとハッシュ コードが同じ潜在的リークに追加される番号。この番号により、エージェントが実行されている間、一意のコレクション ID が確保されます。

潜在的リークのコレクション ID は、アプリケーションの実行中は不変です。

上記のコレクション ID の例を以下に示します。

```
theLookupTable-6314#1
getLoginID-1234#1
getLoginID-1234#2
getLoginID-1234#3
verifyCart-5678#1
verifyCart-0012#1
```

LeakHunter のログ ファイル

LeakHunter ログ ファイルには、Introscope LeakHunter によって特定された、管理対象アプリケーションの潜在的リークに関する情報が含まれています。 各エントリに、1 つの潜在的リークに関する情報が含まれています。LeakHunter のログ ファイルには、以下の 4 つの状況に関するエントリがあります。

- 潜在的なリークが初めて特定されたとき：「潜在的なリークが初めて特定されたときのログ エントリ (P. 184)」を参照してください。
- 特定済みのリークが停止したとき：「特定済みの潜在的リークのリークが停止したときのログ エントリ (P. 185)」を参照してください。
- 以前に特定済みのリークが再びリークし始めたとき：「特定済みの潜在的リークが再びリークしているときのログ エントリ」を参照してください。
- LeakHunter のタイムアウトが発生したとき：「LeakHunter のタイムアウトのログ エントリ」を参照してください。

潜在的なリークが初めて特定されたときのログ エントリ

このタイプの LeakHunter のログ エントリには、以下のような、潜在的なリークが始めて特定された際のそのリークに関する情報が含まれています。

- 現在のタイムスタンプ（ログへの書き込み時間）
- コレクション ID
- コレクションのクラス
- コレクションの割り当てメソッド
- コレクションの割り当て時間
- コレクションの割り当てスタック トレース
- コレクションの割り当て先であるフィールドの名前
- コレクションの現在のサイズ

注: LeakHunter ログ ファイルに記録された、リークしたコレクションの現在のサイズは動的には更新されません。 ログ ファイルには、リークが最初に確認されたときのリーク サイズが使用されます。 リークした接続のサイズに関する最新情報を参照するには、Introscope Workstation で［リーク］タブをクリックします。

例: 潜在的なリークが検出された場合のログ ファイル エントリ

以下の例は、潜在的なリークが初めて特定されたときの Java ログ ファイルのエントリを示しています。

```
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:
不明
Field Name(s):
sonOfLH_test.v3
sonOfLH_test.v4
sonOfLH_test.v5
Current Size: 44
```

特定済みの潜在的リークのリークが停止したときのログ エントリ

このタイプの LeakHunter のログ エントリには、以下のような、リークが停止した潜在的リークに関する情報が含まれています。

- 現在のタイムスタンプ（ログへの書き込み時間）
- コレクション ID
- コレクションのクラス
- コレクションの現在のサイズ

例: 潜在的なリークのリークが停止した場合のログ ファイル エントリ

以下の例は、潜在的なリークがリークを停止したときの Java ログ ファイルのエントリを示しています。

```
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
```

特定済みの潜在的リークが再びリークしているときのログ エントリ

このタイプの LeakHunter のログ エントリには、以下のような、再びリークし始めた潜在的リークに関する情報が含まれています。

- 現在のタイムスタンプ（ログへの書き込み時間）
- コレクション ID
- コレクションのクラス
- コレクションの現在のサイズ

例: 潜在的なリークがリークを再開した場合のログ ファイル エントリ

以下の例は、潜在的なリークが再びリークし始めたときの Java ログ ファイルのエントリを示しています。

```
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 のタイムアウト時のログ エントリ

このタイプの LeakHunter のログ エントリには、継続して追跡される潜在的リークの数が含まれています。

例: タイムアウトが発生した場合のログ ファイル エントリ

```
LeakHunter timeout occurred at 4/27/10 1:32:12 PM PDT
LeakHunter will only continue to track the 3 potential leaks
```

LeakHunter の使用

LeakHunter の使用方法の詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

ErrorDetector

Introscope ErrorDector を使用すると、アプリケーション サポート担当者は、ユーザの Web トランザクションを妨げるおそれのある重大なエラーの原因を検出および診断することができます。 このようなアプリケーションの可用性に関する問題が生じた場合、通常はユーザに対して「404 ページが見つかりません」などのエラー メッセージが表示されることになります。しかし、これらのエラー メッセージには IT 担当者が問題の原因を特定するために必要である具体的な情報が含まれていません。 Introscope ErrorDetector を使用すると、ライブ アプリケーションでこれら重大なエラーの発生時にすぐに対応できるように監視して、そのエラーの頻度および性質を判断し、根本原因に関する具体的な情報を開発者に伝えることができます。

ErrorDetector は、ユーザに優れたユーザ エクスペリエンスを保証し、より完全なトランザクションを提供するための最適なアプリケーション管理ソリューションです。それらは、重大なアプリケーション エラーの根本原因の分析を可能にすることで実現されます。

Introscope ErrorDetector を使用すると、IT チームは以下の処理を行うことができます。

- 異常なトランザクションの頻度を調べる
- ログに記録された例外がユーザに影響を与えるかどうかを調べる
- トランザクション パスのどの部分でエラーが発生したのかを正確に確認する
- 重大なエラーを再現し、診断し、取り除くために必要な情報を取得する

Introscope ErrorDetector は Introscope と統合されているため、Introscope Workstation でエラーを監視できます。 アプリケーション エラーが発生したときに、ライブ エラー ビューアを使用して、それぞれのエラーに関する詳細情報を調べることもできます。

エラーの種類

CA Technologies では一定の基準を定義して、J2EE 仕様に含まれる情報に基づいて、深刻なエラーについて規定しています。ErrorDetector では、エラーおよび例外の両方がエラーであると見なされます。 エラーの種類の中で最も一般的なものは、スローされた例外です。

一般的なエラーのいくつかの例を以下に示します。

- HTTP エラー（404、500 など）

  注: 場合によっては、HTTP 404 エラーはアプリケーション サーバではなく Web サーバで発生することがあります。 このエラーが発生した場合、ErrorDetector はエージェントから Web サーバ エラーの検出を検出できません。

- SQL ステートメント エラー
- ネットワーク接続性エラー（タイムアウト エラー）
- バックエンド エラー（例： JMS を通じてメッセージを送信できない、メッセージ キューにメッセージを書き込めない）

CA Technologies によって重大であると見なされたエラーが、ユーザにとっても重大であるとは限りません。 ErrorDetector が追跡するエラーの中で、重要でないエラーと見なされるエラーがある場合は、これらのエラーを無視するよう選択することができます。 追跡する必要のあるエラーを追加する場合は、エラー トレーサを使用して、これらのエラーを追跡するための新たなディレクティブを作成できます。

ErrorDetector の仕組み

Introscope では、エージェントのインストールに *errors.pbd* と呼ばれる ProbeBuilder ディレクティブ（PBD）ファイルが含まれています。 この PBD のトレーサが、重大なエラーをキャプチャします。

ErrorDetector は、エージェントがインストールされると自動的にインストールされます。 ErrorDetector がインストールされたら、*errors.pbd* を使用して ErrorDetector の機能を有効にするように Introscope を設定します。 ProbeBuilder ウィザードまたはコマンドライン ProbeBuilder を使用している場合は、（すでに使用している PBD ファイルに加えて）*errors.pbd* を使用してアプリケーションを再インスツルメントする必要があります。

エージェントは、*errors.pbd* ファイルの定義に従ってエラー情報を収集します。

Workstation では、以下の内容を表示できます。

- Investigator のエラー メトリック データ
- Live Error Viewer のライブ エラー
- エラー スナップショットのエラー詳細情報。エラーがどのように発生したかに関する、コンポーネント レベルの情報を示します。

ErrorDetector は Transaction Tracer と統合されているため、トランザクション パスのコンテキスト内で、なぜ、またどのようにして重大なエラーが発生したのかを正確に知ることができます。 さらに、すべてのエラーおよびトランザクションはトランザクション イベント データベースに維持されるため、履歴データの分析を通してトレンドを見極めることができます。

Introscope では、トランザクションはサービスの呼び出しと処理として定義されています。 Web アプリケーションのコンテキストでは、Web ブラウザから送信された URL の呼び出しと処理を指します。 Web サービスのコンテキストでは、SOAP メッセージの呼び出しと処理を指します。

Java Agent での ErrorDetector の有効化

エラー データのキャプチャ用に Java Agent を有効にするには、プロパティ *introscope.agent.errorsnapshots.enable* を *IntroscopeAgent.profile* で true に設定する必要があります。デフォルトで、Java Agent はエラー データのキャプチャが有効になっています。

Java Agent で ErrorDetector データ キャプチャを有効または無効にする方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. *introscope.agent.errorsnapshots.enable* プロパティが *true* であることを確認します。

   注: ErrorDetector を無効にするには、このプロパティを *false* に設定します。
3. ProbeBuilder ウィザードを使用している場合は、*errors.pbd* ファイルを *<EM_Home>*/config/custompbd ディレクトリにコピーして、アプリケーションを再インスツルメントします。
4. エージェント プロファイル を保存します。

ErrorDetector オプションの設定

エージェントが Enterprise Manager に送信するエラーの最大数を制限したり、無視するエラーを指定できるように ErrorDetector を設定できます。

ErrorDetector を有効にすると、オーバーヘッドをあまり発生させずに、エラー データをキャプチャできます。 付属のスロットルは、15 秒につき 10 個のエラーに設定されています。 この間隔でより多くのエラーをキャプチャする場合は、スロットルを増やすことができますが、オーバーヘッドがより多く発生することになるため注意してください。

ErrorDetector スロットルを変更する方法（オプション）

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. *introscope.agent.errorsnapshots.throttle* プロパティの新しい値を入力します。
3. エージェント プロファイル を保存します。

   注: このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

追跡しないエラーがある場合は、エージェントでエラーを無視するように設定できます。 エラーを「タグ付け」する（エラーを識別する）ために指定する情報には、完全一致のエラー メッセージ、またはワイルドカードのアスタリスク記号を使って表したメッセージの一部を使用できます。

特定のエラーを無視する方法（オプション）

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. *introscope.agent.errorsnapshots.ignore* プロパティで、対象となるエラーの種類を特定する情報を考え、値を設定します。

   たとえば、以下の ignore プロパティでは、プロパティで指定された「IOException」という用語が含まれるすべてのエラーが無視されます。

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   ```

3. その他のエラーを無視するには、そのエラーに対応したエラーを無視するプロパティを順番に追加します。 たとえば、2 つの種類のエラーを無視するには、プロパティは以下のようになります。

```
introscope.agent.errorsnapshots.ignore.0=*IOException*
introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code *500*
```

4. エージェント プロファイルへの変更を保存します。

   注: このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

高度なエラー データ キャプチャ

ErrorDetector は、ほとんどの一般的なエラーをデフォルトでキャプチャしますが、CA Technologies では、ユーザの状況に合わせてエラー検出メカニズムをカスタマイズできるオプションを用意しています。

以下のエラー関連のトレーサを使用して、エラーをキャプチャするための ProbeBuilder ディレクティブ（PBD）を作成できます。

- **ExceptionErrorReporter** は、標準の例外を報告します。
- **ThisErrorReporter** は、現在のオブジェクトをエラーとして報告します。
- **HTTPErrorCodeReporter** は、HTTP エラー コードおよび関連するエラー メッセージをキャプチャします。
- **MethodCalledErrorReporter** は、特定のメソッドがコールされると報告します。

これらのトレーサで作成する新しいディレクティブは、*<Agent_Home>*/wily ディレクトリ以下の *errors.pbd* ファイル内に配置する必要があります。

新たなエラー タイプの定義

ErrorDetector 用のエラーは PBD を使用して定義します。 エラーの存在をチェックして、エラー メッセージをキャプチャ（またはコンストラクト）する特別なトレーサーがいくつかあります。 これらのトレーサを正しい場所に配置して、ご使用のアプリケーションまたはそのインフラストラクチャでの新たなエラー タイプを ErrorDetector に通知することができます。

ExceptionErrorReporter

ExceptionErrorReporter トレーサは、インスツルメントされたメソッドからスローされる例外をチェックするために使用されます。 例外がスローされると、このトレーサはその例外をエラーとして処理し、その例外からエラー メッセージを取得します。 これは最も一般的なエラー定義です。

エラー メッセージをキャプチャするには、ExceptionErrorReporter トレーサを「...WithParameters」ディレクティブと共に使用する必要があります。例:

TraceOneMethodWithParametersOfClass: com.bank.CustomerAccount getBalance ExceptionErrorReporter "CustomerAccount:Errors Per Interval"

このディレクティブは、CustomerAccount の getBalance() メソッドからスローされる例外がすべてエラーとなるように指定しています。

間隔ごとのエラー数のメトリックをインクリメントするには、「...WithParameters」ディレクティブを使用する必要がありますが、メソッド上のすべてのトレーサでパラメータが利用できるように、「...WithParameters」ディレクティブはいずれのメソッドについても 1 回のみ指定するようにしてください。たとえば、以下のように指定します。

TraceOneMethodWithParametersOfClass: com.myClass myMethod BlamePointTracer

このディレクティブは com.myClass myMethod メソッド用のパラメータを ExceptionErrorReporter トレーサも含めた他のトレーサで利用できるようにしています。

MethodCalledErrorReporter

MethodCalledErrorReporter トレーサは、メソッドがコールされるという行為そのものがエラーが発生したことを意味するメソッドに対して使用されます。 例 :

TraceOneMethodOfClass: com.bank.CheckingAccount cancelCheck MethodCalledErrorReporter "CustomerAccount:Canceled Checks Per Interval"

このディレクティブは、cancelCheck() メソッドが（何らかの理由で）コールされると、それがエラーであることを示しています。 エラー メッセージは、クラスおよびコールされたメソッドを簡単に示します。

例外またはエラーをスローするメソッドがわからない場合は、ThisErrorReporter トレーサを使用してみます。

ThisErrorReporter

ThisErrorReporter トレーサは *MethodCalledErrorReporter* と類似していますが、インスツルメントされたオブジェクトで *toString()* をコールすることによって、エラー メッセージをコンストラクトします。このトレーサは、例外クラスのコンストラクタに配置すると非常に便利です。 例：

```
TraceOneMethodWithParametersOfClass: ezfids.util.exception.EasyFidsException [set the init variable for your book] ThisErrorReporter "Exceptions|{packageandclassname}:Errors Per Interval"
```

注: エラー メッセージをキャプチャするには、ThisErrorReporter トレーサを「...WithParameters」ディレクティブと共に使用する必要があります。

このディレクティブは、InvalidPINException のコンストラクタ（「init」または「.ctor」）がコールされると、エラーが構成されることを示しています。エラー メッセージは、InvalidPINException で toString() をコールすることによって決定されます。この処理では通常、アプリケーション開発者が指定したエラー メッセージが返されます。

このトレーサーは、独自の例外タイプに基づいたカスタムのエラー管理システムを使用している場合に活用することをお勧めします。

注: Introscope では、java.* パッケージのコードをインスツルメントできないため、このトレーサを java.lang.Exception または java.sql.SQLException に配置しても機能しません。

HTTPErrorCodeReporter

HTTPErrorCodeTracer トレーサは、サーブレットおよび JSP のエラー コードを報告します。 これは、以下のインシデントをカウントする間隔ごとのカウンタです。

- HTTP 応答コード 400 以上
- Java 環境でのコード 400 以上に対する、*sendError* または *setStatus* の *javax.servlet.http.HttpServletResponse* サブクラス呼び出し

使用例については、「*errors.pbd*」を参照してください。

エラートレーサ ディレクティブと警告の併用

前のセクションで説明されているトレーサーは慎重に使用してください。最良の運用方法は、オーバーヘッドの原因となるエラー追跡を、バックエンド システムで発生する回復不可能なエラーなどのような非常に重大な問題を報告するものに限定することです。

デフォルトの *errors.pbd* は、オーバーヘッドを最小限に抑えながら、重大なエラーを報告するように設計されています。 監視してしているすべてのメソッドに ExceptionErrorReporter を適用するなど、エラー追跡を過度に使用すると「誤検出」が多く発生することになります。たとえばこの場合、ユーザが数値フィールドに「California」と入力することで、*NumberFormatException* を発生させる可能性がありますが、この例外を重大な問題として報告することはお勧めできません。

ErrorDetector の使用

ErrorDetector の使用方法の詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

第 9 章: Boundary Blame の設定

このセクションでは、Java Agent Blame レポートのデフォルトの動作および関連する設定オプションついて説明します。

このセクションには、以下のトピックが含まれています。

Boundary Blame の理解

Blame テクノロジを使用すると、管理対象 Java アプリケーション内で作業して、アプリケーションのフロントエンドおよびバックエンドでメトリックを表示できるようになります。 この Boundary Blame と呼ばれる機能によって、問題の原因を切り分けることができます。

Introscope がバックエンドを検出する方法は、アプリケーションによって異なります。データベース アクティビティの場合、Introscope は SQL エージェントを使用してバックエンドを検出します。SQL エージェントを使用できない場合でも、Introscope はバックエンド アクティビティを検出します。これは、クライアント/サーバ データベース、JMS サーバ、LDAP サーバなどのバックエンドは、ソケットを通じてアクセスされるためです。Oracle のバックエンドがある環境で Introscope SQL エージェントを使用していない場合は、「Boundary Blame および Oracle バックエンド (P. 147)」を参照してください。

Boundary Blame が Introscope Investigator にどのように表示されるかについては、「CA APM Workstation ユーザ ガイド」を参照してください。

URL グループの使用

URL グループを使用して、パスのプレフィクスがユーザ定義の文字列で始まる要求セットに対するブラウザの応答時間をモニタできます。 パスのプレフィックスは、URL のホスト名に続く部分です。たとえば、以下の URL があるとします。

http://burger1.com/testWar/burgerServlet?ViewItem&category=11776&item=5550662630&rd=1

ここでは、以下がパスのプレフィクスです。

/testWar

1 つの URL のパス プレフィクスから派生する可能性がある要求の有用なカテゴリに対して URL グループを定義できます。 たとえば、アプリケーションの URL の形式に応じて、アプリケーションでサポートする顧客ごと、主要なアプリケーションごと、またはサブアプリケーションごとに URL グループを定義できます。 これによって、コミットされたサービスレベルのコンテキストでの性能、またはアプリケーションのミッション クリティカルな部分のパフォーマンスをモニタできます。

例: URL グループのプロパティを定義する方法

以下の例は、Java Agent プロファイルの抜粋で、URL グループの定義方法を示しています。

```
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 グループのキーの定義

introscope.agent.urlgroup.keys プロパティを使用して、URL グループのすべての ID またはキーのリストを定義します。URL グループのキーは、URL グループの属性を宣言する、ほかのプロパティ定義で参照されます。 以下の例では、3 つの URL グループのキーを定義します。

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
```

複数のグループに分類される URL をいくつか含んでいる URL グループを定義する場合は、プロパティでその URL グループのキーをリストした順序が重要になります。 絞り込んだメンバシップを持つ URL グループは、幅広いメンバシップを持つ URL グループに先行します。 たとえば、**alpha** というキーを使用する IP グループが
パス プレフィックス */examplesWebAp* を持ち、また、**delta** というキーを持つ URL グループ
が */examplesWebApp/cleverones* のパス プレフィックスを持つ場合、*keys* パラメータの中で、delta が *alpha* に先行している必要があります。

各 URL グループのメンバシップの定義

*introscope.agent.urlgroup.group.***groupKey***.pathprefix* プロパティは、URL のパスのプレフィクスと照合するパターンを指定して、URL グループ内に含める要求を定義します。

例: URL のパス プレフィックスに対するグループ キーのマッピング

以下のプロパティ定義では、URL のパス部分が */testWar* で始まるすべての要求を、キーが *alpha* の URL グループに割り当てています。

```
introscope.agent.urlgroup.group.alpha.pathprefix=/testWar
```

たとえば、以下の要求は、指定したパスのプレフィックスと一致します。

```
http://burger1.com/testWar/burgerServlet?ViewItem&category=
11776&item=5550662630&rd=1
http://burger1.com/testWar/burgerServlet?Command=Order&item=5550662630
```

例: アプリケーション パスによる URL グループの作成

コール センタ サービスを提供する企業は、アプリケーションの機能ごとに URL グループを設定することによって、機能分野ごとの応答時間のモニタを行うことができます。 顧客が以下の URL でサービスにアクセスするとします。

*http://genesystems/us/*application_function/

この例の *application_function* は、*OrderEntry*、*AcctService*、および *Support* などのアプリケーションに対応し、各 URL グループの *pathprefix* プロパティは、適切な *application_function* を指定します。

注: *pathprefix* のプロパティには、アスタリスク記号（*）をワイルド カードとして使用できます。

URL グループの名前の定義

プロパティ *introscope.agent.urlgroup.group.groupKey.format* では、キーが *groupKey* である URL グループの応答時間メトリックが、Introscope Workstation ではどの名前で表示されるかを決定します。

通常、*format* プロパティは、テキスト文字列を URL の名前として割り当てるために使用されます。以下の例では、キー *alpha* を持つ URL グループのメトリックが、*Alpha Group* という名前で Workstation に表示されます。

introscope.agent.urlgroup.group.alpha.format=Alpha Group

URL グループの高度な名前付け技術（オプション）

必要に応じて、URL グループ名を、サーバ ポート、プロトコルなどの要求エレメント、または要求 URL のサブ文字列から派生できます。 これは、要求を調べればアプリケーション モジュールを簡単に区別できる場合に便利です。 このセクションでは、*format* プロパティの高度な形式について説明します。

URL グループ名にホストを使用

URL グループのメトリックを要求に関連付けられている HTTP サーバのホスト名を反映した名前でまとめるには、*format* パラメータを以下のように定義します。

introscope.agent.urlgroup.group.alpha.format={host}

format={host} とすると、以下の要求の統計は、それぞれメトリック名 *us.mybank.com* および *uk.mybank.com* の下に表示されます。

https://us.mybank.com/mifi/loanApp.....

https://uk.mybank.com/mifi/loanApp.....

URL グループ名にプロトコルを使用

URL グループの統計情報を、要求に関連付けられているプロトコルを反映した名前でまとめるには、*format* パラメータを以下のように定義します。

introscope.agent.urlgroup.group.alpha.format={protocol}

format={protocol} の場合、統計は、Investigator で、要求 URL のプロトコルの部分に対応したメトリック名でグループ化されます。 たとえば、以下の要求の統計情報は、メトリック名 *https* の下に表示されます。

https://us.mybank.com/cgi-bin/mifi/scripts.....

https://uk.mybank.com/cgi-bin/mifi/scripts.....

URL グループ名にポートを使用

URL グループの統計情報を、要求に関連付けられているポートを反映した名前でまとめるには、*format* パラメータを以下のように定義します。

introscope.agent.urlgroup.group.alpha.format={port}

format={port} の場合、統計は、要求 URL のポートの部分に対応した名前で、グループ化されます。 たとえば、以下の要求の統計は、*9001* という名前の下に表示されます。

https://us.mybank.com:9001/cgi-bin/mifi/scripts.....

https://uk.mybank.com:9001/cgi-bin/mifi/scripts.....

URL グループ名にパラメータを使用

Investigator 内で URL グループの統計を要求に関連付けられているパラメータの値を反映したメトリック名でまとめるには、*format* パラメータを以下のように定義します。

```
introscope.agent.urlgroup.group.alpha.format={query_param:param}
```

*format={query_param:*param*}* の場合、統計は、Investigator で、指定したパラメータの値に対応したメトリック名でグループ化されます。 パラメータのない要求は、<empty> の下に一覧表示されます。 たとえば、以下のパラメータ定義があるとします。

```
introscope.agent.urlgroup.group.alpha.format=
{query_param:category}
```

以下の要求の統計は、メトリック名「734」で表示されます。

```
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=3772&tc=photo
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=8574&tc=photo
```

URL グループ名に要求パスのサブ文字列を使用

URL グループの統計情報を要求 URL のパス部分のサブ文字列を反映した名前でまとめるには、*format* パラメータを以下のように定義します。

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:m:n}
```

ここで、「*m*」は、最初の文字のインデックスで、「*n*」は、最後の文字のインデックス +1 です。 文字列の選択は、*java.lang.String.substring()* メソッドのように機能します。 たとえば、以下のように設定されているとします。

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:0:3}
```

以下の要求の統計は、メトリック ノード「*/ht*」の下に表示されます。

```
http://research.com/htmldocu/WebL-12.html
```

URL グループ名に要求パスの区切り文字列部分を使用

URL グループの統計情報を要求 URL パスの文字区切りの部分を反映した名前でまとめるには、*format* パラメータを以下のように定義します。

```
introscope.agent.urlgroup.group.alpha.format=
{path_delimited:delim_char:m:n}
```

ここで、*delim_char* は、パス内のセグメントを区切る文字です。「*m*」は、選択する最初のセグメントのインデックスです。「*n*」 は、選択する最後のセグメントのインデックス +1 です。 たとえば、以下のように設定されているとします。

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/:2:4}
```

以下の形式の要求に対する統計情報を考えます。

```
http://www.buyitall.com/userid,sessionid/pageid
```

これは、メトリック名 */pageid* で表示されます。

以下のことに注意してください。

- 区切り文字（この例では /（スラッシュ））もセグメントとしてカウントされます。
- セグメントのカウントは 0 から始まります。

URL の例で、スラッシュ文字で区切られたセグメントを以下に示します。

0=*/*, 1=*userid,sessionid*, 2=*/*, and 3=*pageid*

必要に応じて、複数の区切り文字を指定できます。 たとえば、以下のように設定されているとします。

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/,:3:4}
```

上記の形式の要求に対する統計情報は、メトリック名 *sessionid* の下に表示されます。URL の例でスラッシュおよびカンマで区切られたセグメントを以下に示します。

0=*/*, 1=*userid*, 2=*,* 3=*sessionid*, 4=*/*, and 5=*pageid*

URL グループに対する複数の命名メソッドの使用

以下のように、複数の命名メソッドを 1 つの *format* 文字列に組み合わせることができます。

```
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 の実行

URLGrouper は、一般的な形式での Web サーバ ログ ファイルを分析するコマンド ライン ユーティリティです。 URL グループを定義する出発点として URLGrouper を使用できます。

注: URLGrouper ユーティリティを使用して、Web サーバ ログ ファイルを分析できます。URLGrouper は、Web サーバ ログ ファイルの内容に基づき、潜在的な URL グループに対してプロパティ 設定のセットを出力します。アスタリスク記号（*） は、ワイルドカードとして、URLGrouper で使用できます。

URLGrouper を実行する方法

1. コマンド シェルを開きます。
2. 以下のコマンドを入力します。

   ```
   java -jar urlgrouper.jar logfile
   ```

   ここで、*logfile* は、Web サーバ ログ ファイルへのフル パスです。
3. URL グループ セットのプロパティ定義が、STDOUT に出力されます。
4. 提案された URL グループを設定するには、URLGrouper が生成したプロパティ ステートメントを *IntroscopeAgent.profile* にコピーします。

Blame トレーサの使用

トレーサを使用して、アプリケーションのフロントエンドおよびバックエンドを明示的にマークできます。 詳細については、「Blame トレーサを使用した Blame ポイントのマーク付け (P. 144)」を参照してください。

第 10 章: トランザクション追跡オプションの設定

このセクションでは、デフォルトのトランザクション追跡の動作および関連する設定オプションについて説明します。

このセクションには、以下のトピックが含まれています。

トランザクション追跡の新しいモード

CA APM Introscope 9.1 では、新しいエージェント トランザクション アーキテクチャと新しいトレーサが導入されました。 このトランザクション追跡用の新しいモードは、以前のリリースで古いトレーサが使用していたトランザクション Blame スタックを置き換えます。 この新しいモードの実装では、計算と処理が最適化され、エージェントのパフォーマンスが向上します。

CA APM Introscope 9.1 では、以前と同じトランザクション Blame スタックでエージェントを設定および実行できます。 エージェントのこの「レガシー」モードは完全にサポートされていますが、将来のリリースにおけるエージェントの拡張機能は新しいモードにのみ実装されます。 「レガシー」モードでエージェントを実行した場合、以下の機能は利用できません。

- バージョン 9.1 でのエージェントの CPU 使用率と応答時間の最適化
- .NET エージェントの動的インスツルメンテーション
- SQL エージェントのプロパティ

  `introscope.agent.sqlagent.sql.artonly`

  `introscope.agent.sqlagent.sql.turnoffmetrics`

重要: レガシーのトレーサと CA APM エージェントを新しいモードで実行する場合、この設定は「混合モード」と呼ばれます。 メモリ消費量が増加してサービス障害が発生する可能性があるため、混合モードでは実行**しない**でください。

レガシー モードのトレーサを実行している場合は、エージェントをレガシー モードで実行します。 互換性のないレガシーのトレーサがあることをユーザに知らせるために、以下のメッセージがエージェント ログに書き込まれます。

```
"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"
```

新しいモードのトレーサを実行するまで、レガシー モードでエージェントを実行するように設定を戻してください。

以下の標準装備の拡張機能を使用している場合は、エージェントをレガシー モードで実行するように設定してください。

- 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

レガシー モードのトランザクション追跡を使用するためのエージェントの設定

レガシー モードのトランザクション追跡への切り替えは、新しいモードがある CA APM のバージョンにのみ適用されます。 バージョン 9.1 より前の CA APM には新しいモードはありません。

エージェントがレガシー モードを使用するように設定するには、2 つのオプションがあります。

- 事前設定されたレガシー エージェント パッケージをデプロイします。これらのパッケージはファイルのみで構成されるエージェントのパッケージとして提供され、インストーラはありません。 たとえば、UNIX 上の Oracle WebLogic エージェント用のレガシー パッケージは以下のとおりです。

  `IntroscopeAgentFiles-Legacy-NoInstaller<`*バージョン番号*`>weblogic.unix.tar`

- 手動によるレガシー モードの設定

以下の手順に従います。

1. 監視対象のアプリケーションを停止します。
2. ログ ファイルを新たに作成するために、<Agent_Home>/logs ディレクトリ内の既存のログ ファイルをアーカイブして削除します。
3. <Agent_Home>/core/config ディレクトリ内の既存の .pbl および .pbd ファイルをバックアップします。
4. 既存の <Agent_Home>/core/config/IntroscopeAgent.profile をバックアップします。
5. レガシーの .pbl ファイルと .pbd ファイルを <Agent_Home>/examples/legacy ディレクトリから <Agent Home>/core/config ディレクトリにコピーします。
6. <Agent_Home>/core/config/IntroscopeAgent.profile を開き、以下の変更を行います。
   - プロパティ introscope.agent.configuration.old=true を追加します。
   - コピーされたレガシーの .pbl ファイルおよび .pbd ファイルを適切にポイントするようにエージェント プロパティ introscope.autoprobe.directivesFile を更新します。たとえば、spm.pbl を spm-legacy.pbl で置き換えます。
7. 管理対象アプリケーションを再起動します。

特定の CA APM 拡張機能を設定して標準インストールでレガシー モードを使用するには、以下の表に示す .pbl および .pbd ファイルを参照します。

拡張機能名	レガシー標準 pbl または pbd ファイル
CA APM for Oracle Weblogic Server	ppweblogic-legacy.pbd spm-legacy.pbl
CA APM for Oracle Weblogic Portal	powerpackforweblogicportal-legacy.pbl spm-legacy.pbl
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

自動トランザクション追跡の動作の制御

自動トランザクション追跡によって、明示的にトランザクション追跡を実行しなくても、問題になりそうなトランザクションのタイプの履歴分析が可能になります。 Introscope では、以下の 2 種類の自動トランザクション追跡が提供されます。

- URL グループ化に基づいて、デフォルトで有効になるトランザクション追跡サンプリング。
- URL グループ化にかかわらず、追跡情報を収集する設定可能な自動追跡サンプリング。

Transaction Trace コンポーネント クランプ

Introscope では、追跡のサイズを制限するクランプが設定されます。 デフォルトは 5,000 コンポーネントです。 この制限に達すると、警告がログに出力され、追跡が停止します。

サーブレットが数百のオブジェクト インタラクションおよびバックエンドの SQL コールを実行する場合など、予想されるコンポーネント数を超えるコンポーネント トランザクションをクランプすることができます。 クランプがない場合、Transaction Tracer は、これを 1 つのトランザクションと見なし、無限に実行し続けます。 特定の極限状態にあるときに適所でクランプが実施されない場合、JVM は追跡を完了する前にメモリ不足に陥ってしまいます。

トランザクションをクランプするための以下のプロパティは、IntroscopeAgent.profile ファイルの中にあります。

introscope.agent.transactiontrace.componentCountClamp=*5000*

クランプされたコンポーネントを生成している追跡は、アスタリスクを付けて示されます。このような追跡の表示方法の詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

重要: トランザクション追跡のコンポーネント クランプ サイズが大きくなると、トランザクション追跡に必要なメモリも増えます。 詳細については、「CA APM 設定および管理ガイド」を参照してください。

トランザクション追跡サンプリング

トランザクション追跡のサンプリングはデフォルトで有効になっています。 必要に応じて、この動作を無効にすることもできます。 トランザクション追跡の詳細については、「トランザクション追跡 (P. 376)」を参照してください。

自動追跡サンプリングを設定する際、指定した間隔の間に追跡するトランザクションの数を指定します。

注: これらのプロパティは Enterprise Manager のプロパティ ファイルに保存されています。 *sampling.perinterval* および *sampling.interval* プロパティのデフォルトを変更する前に、サンプリング レートが上がったために Enterprise Manager 内での負荷が増大する可能性を考慮してください。 Enterprise Manager は、自身に接続しているすべてのエージェントに対して、この設定をプッシュします。 また、エージェント内でこれらのプロパティを設定することもできます。 エージェント内でこれらのプロパティを設定すると、Enterprise Manager によって個別のエージェント向けに行われた設定が上書きされます。

自動追跡サンプリングを設定するには、以下のプロパティを変更します。

- *introscope.agent.transactiontracer.sampling.enabled*

  false に設定すると、トランザクション追跡のサンプリングが無効になります。 デフォルト値は true です。

- *introscope.agent.transactiontracer.sampling.perinterval.count*

  指定した間隔の間に追跡するトランザクションの数を指定します。

  トランザクションのデフォルトの数は 1 です。

- *introscope.agent.transactiontracer.sampling.interval.seconds*

  指定したトランザクション数を追跡する時間の長さを指定します。

  デフォルト間隔は 2 分ごとです。

エージェントのヒープのサイジング

Agent は、収集したデータの保存に Java のヒープ メモリを使用します。エージェントの GC ヒープ使用量については、［GC Heap Overview］から参照することができます。

ヒープ使用量に関する説明を以下に示します。

- ヒープ使用率が高い場合は、エージェントのインストール時にヒープの割り当てを増やす必要があります。
- 監視中のアプリケーションに非常に複雑なトランザクションや、時間のかかるトランザクションが見られる場合は、トランザクション追跡のサンプリングに多くのヒープ メモリが必要となります。

注: GC ヒープ使用量の詳細については、「*CA APM Workstation ユーザ ガイド*」を参照してください。 高性能の CA Introscope® 環境を運用している場合は、エージェントの適切な JVM ヒープ設定に関して CA プロフェッショナル サービスにお問い合わせください。

詳細:

Transaction Trace コンポーネント クランプ (P. 207)
トランザクション追跡サンプリング (P. 208)

プロセスにまたがるトランザクション追跡

CA Introscope® Java エージェントを使用して、WebLogic Server または WebSphere で JVM の境界を超えてトランザクションを追跡できます。 環境は、同じベンダーのアプリケーション サーバの互換バージョンで構成されている必要があります。 プロセスにまたがるトランザクション追跡は、サーブレットから EJB などの同期トランザクション、および非同期トランザクションでサポートされています。

重要: プロセスにまたがるトランザクション追跡は、CA Introscope® 9.0 以降のエージェントを使用する場合にのみ利用可能です。

詳細:

WebLogic Server でのプロセスにまたがる追跡の有効化 (P. 50)
WebSphere でのプロセスにまたがる追跡の有効化 (P. 66)

トランザクション追跡データ収集の拡張

ほかのデータ（サーブレット呼び出しや JSP 呼び出しのユーザ ID データ）やその他のトランザクション追跡データ（HTTP 要求ヘッダ、要求パラメータ、セッション属性など）も取得するように、Introscope Transaction Tracer を設定することができます。 これらのデータを取得するには、IntroscopeAgent.profile に条件を定義する必要があります。

ユーザ ID データについて

サーブレット呼び出しや JSP 呼び出しのユーザ ID を取得できるように Java Agent を設定するには、管理対象アプリケーションでユーザ ID の特定に使用されている方法についての情報を入手する必要があります。 この情報は管理対象アプリケーションを開発したアプリケーション設計者から取得できる可能性があります。

Introscope Transaction Tracer は、以下の手段のいずれかを使用してユーザ ID を保存している管理対象アプリケーションからであれば、ユーザ ID を取得することができます。

- HttpServletRequest.getRemoteUser()
- HttpServletRequest.getHeader (String key)
- HttpSession.getValue (String key): 返されるオブジェクトは、ユーザ ID を表すストリングか、ユーザ ID を返す toString() を備えたオブジェクトです。

これらのメソッドのうちの 1 つを使用して管理対象アプリケーションがユーザ ID を保存する場合は、「追加のトランザクション追跡データを収集するためのエージェントの設定 (P. 211)」を参照して、ユーザ ID データを収集するように Java Agent を設定します。

サーブレット要求データについて

Introscope を使用して、ユーザ定義のパラメータに一致するトランザクション追跡データを収集することが可能です。 たとえば、HTTP 要求ヘッダにユーザ エージェント（User-Agent）情報を含むトランザクションのトランザクション追跡データを、エージェントで収集するように設定することができます。

Introscope では、以下のサーブレット要求情報を記録できます。

- 要求ヘッダ
- 要求パラメータ
- セッション属性

ご使用の管理対象アプリケーションのサーブレット要求情報を記録するには、「追加のトランザクション追跡データを収集するためのエージェントの設定 (P. 211)」を参照して、このデータを収集するように Java Agent を設定します。

追加のトランザクション追跡データを収集するためのエージェントの設定

ほかのトランザクション追跡データ（ユーザ ID、HTTP 要求ヘッダ、HTTP 要求パラメータ、または HTTP セッション属性など）も収集するように、Java Agent を設定することができます。

以下の手順に従います。

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. トランザクション追跡のプロパティを見つけます。これは、「Transaction Tracer Configuration」という見出しの下にあります。

ユーザ ID データの収集

ユーザ ID を識別できるように Java Agent を設定する方法

1. 管理対象アプリケーションでユーザ ＩＤ の保存に使用されているメソッドに対応するプロパティを設定します。

   注: 1 つまたは 1 組のプロパティのみコメント化を解除してください。そうしないと、不適切なプロパティが使用されるおそれがあります。

2. HttpServletRequest.getRemoteUser() が使用されている場合は、以下のプロパティのコメント化を解除します。

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getRemoteUser
   ```

3. HttpServletRequest.getHeader (String key) が使用されている場合は、以下の 1 対のプロパティのコメント化を解除し、2 番目のプロパティにキー ストリングを定義します。

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getHeader
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```

4. HttpSession.getValue (String key) が使用されている場合は、以下の 1 対のプロパティのコメント化を解除し、2 番目のプロパティにキー ストリングを定義します。

   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getValue
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```

サーブレット要求データの収集

HTTP 要求ヘッダやパラメータなどのサーブレット要求データを記録する方法

1. トランザクション追跡データの収集対象にする HTTP 要求ヘッダを指定する場合は、以下のプロパティのコメント化を解除し、追跡する HTTP 要求ヘッダをカンマ区切りのリストの形式で指定します。

   ```
   #introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent
   ```

2. トランザクション追跡データの収集対象にする HTTP 要求パラメータを指定する場合は、以下のプロパティのコメント化を解除し、追跡する HTTP 要求パラメータをカンマ区切りのリストの形式で指定します。

   ```
   #introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter1,parameter2
   ```

3. データ追跡の対象にする HTTP セッション属性を指定する場合は、以下のプロパティのコメント化を解除し、追跡する HTTP セッション属性を以下の例のようにカンマ区切りのリストの形式で指定します。
   ```
   #introscope.agent.transactiontracer.parameter.httpsession.attributes=cartID,deptID
   ```
4. 管理対象アプリケーションを再起動します。

コンポーネント ストール レポートの設定

Application Performance Management のストールは、定義された期間、プローブされるコンポーネントから応答がない状態を表します。 デフォルトでは、APM Introscope エージェントはこの状態を検出し、ストール メトリックをレポートします。

エージェントがストールを確認する度に、メソッド スタックの最上位にあるインスツルメントされたコンポーネントがすべて確認されます。 コンポーネントがストールした場合、ストール メトリックおよびストール スナップショットが作成されます。 ストール メトリックは、ダウンストリーム監視をサブスクライブする各コンポーネントに対しても作成されます。

ストール メトリックは、メソッド スタックの最上位のインスツルメントされたコンポーネントに対してまず作成されます。 これらのメトリックは、*introscope.agent.stalls.thresholdseconds* 値より長くかかるコンポーネントに対して作成されます。

ダウンストリーム サブスクライバ コンポーネントのストール

ダウンストリーム監視をサブスクライブするコンポーネントが、ストールしているダウンストリーム コンポーネントを呼び出すとストールが発生します。

デフォルトでは、以下のコンポーネントがダウンストリーム ストール監視をサブスクライブします。

- FrontendTracer でプローブされるコンポーネント
- BackendTracer でプローブされるコンポーネント
- WebServiceBlamepointTracer@Servicelevel でプローブされるコンポーネント
- WebServiceBlamepointTracer@Opertationlevel でプローブされるコンポーネント

例

フロントエンド コンポーネントは、ストールしているバックエンド コンポーネントを呼び出す Web サービスを呼び出します。

フロントエンド--> Web サービス --> バックエンド

デフォルトではそれらすべてがダウンストリーム監視をサブスクライブするため、ストール メトリックは、バックエンド、Web サービス、およびフロントエンド コンポーネントに対して作成されます。

アップストリーム継承コンポーネントのストール

ストール チェッカは、まずコンポーネント スタックの最上位のメンバがストールしているかどうかを確認します。 最上位のコンポーネントがストールしていない場合、ストール チェッカはストールしている親コンポーネントを探します。 親がストールしている場合、スタック内の該当コンポーネントおよびアップストリームの各親に対してストール メトリックが作成されます。

コンポーネント M1() が複数のコンポーネント M2() を呼び出し、それぞれに数秒しかかからない場合でも、ストール メトリックが M1() に対してレポートされる場合があります。

例:

ストールしきい値は、15 秒に設定されています。メソッド M1() は、メソッド M2() を 100 回のループで呼び出します。 応答にはわずか 2 秒しかかからないので、M2() はストールしません。 ただし、M1() は結果的にストールします。

イベントとしてのストールのキャプチャの無効化

Introscope では、デフォルトで、トランザクション ストールをトランザクション イベント データベース内のイベントとしてキャプチャし、検出したイベントからストール メトリックを生成します。 ストール メトリックは、トランザクションの最初と最後のメソッドで生成されます。 ユーザは、ストール イベントと関連付けられているメトリックを Workstation の履歴イベント ビューアで表示できます。

注: 生成されたストール メトリックは、いつでも利用できますが、ストール イベントを表示できるのは、Introscope Error Detector をインストールしている場合のみです。 ストールは、通常のエラーとして保存され、エラー TypeView で表示できます。または、「type:errorsnapshot」のクエリを実行することによって履歴クエリ ビューアで表示できます。

イベントとしてのストールのキャプチャを無効にしたり、ストールしきい値を変更したり、エージェントがストールをチェックする頻度を変更したりすることができます。 以下のプロパティを使用します。

- *introscope.agent.stalls.thresholdseconds* は、トランザクションがストールしていると見なされる応答時間しきい値の最小値を指定します。
- *introscope.agent.stalls.resolutionseconds* は、エージェントがストールをチェックする頻度を指定します。

第 11 章: Introscope SQL エージェントの設定

このセクションでは、Introscope SQL エージェントを設定する方法について説明します。

このセクションには、以下のトピックが含まれています。

SQL エージェントの概要

SQL エージェントは、詳細なデータベース パフォーマンス データを Enterprise Manager にレポートします。 SQL エージェントは、管理対象アプリケーションとデータベース間のやり取りを追跡することによって、アプリケーションの個別の SQL ステートメントのパフォーマンスを詳しく調べることができます。

Java Agent がアプリケーションを監視するのと同じように、SQL エージェントは SQL ステートメントを監視します。 SQL エージェントは目立たず、非常に低いオーバーヘッドでアプリケーションまたはデータベースを監視します。

個別の SQL ステートメント レベル単位までの有意義なパフォーマンス測定を行うために、SQL エージェントは収集するパフォーマンス データを要約します。これは、トランザクション固有のデータを除外し、元の SQL ステートメントを Introscope 固有の正規化されたステートメントに変換することによって実現します。正規化されたステートメントには、クレジット カード番号などの機密情報が含まれないので、この処理でもデータのセキュリティは守られます。

たとえば、SQL エージェントが以下の SQL クエリを変換するとします。

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

以下は、変換後の正規化されたステートメントです。

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

同様に、SQL エージェントが以下の SQL 更新ステートメントを変換するとします。

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES ('Atwood', 'The Robber Bride')
```

以下は、変換後の正規化されたステートメントです。

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES (?, ?)
```

注: 引用符内のテキスト（'xyz'）のみが正規化されます。

正規化されたステートメントのメトリックは集約されます。また、Workstation Investigator の JDBC ノードに表示することができます。

SQL エージェント ファイル

SQL エージェントは、すべてのエージェント インストールに含まれています。 SQL エージェント機能を提供するファイルは以下のとおりです。

- wily/core/ext/SQLAgent.jar
- wily/core/config/sqlagent.pbd

注: デフォルトで、SQLAgent.jar ファイルなどのエージェントの拡張は、wily/core/ext ディレクトリにインストールされます。エージェントの拡張ディレクトリの場所は、エージェント プロファイルの introscope.agent.extensions.directory プロパティで変更できます。 */ext* ディレクトリの場所を変更する場合は、*/ext* ディレクトリの内容も同様に移動する必要があります。

SQL ステートメントの正規化

アプリケーションの中には非常に大量の独自の SQL ステートメントを生成してしまうものがあります。 EJB 3.0 のようなテクノロジが使用されている場合、長い独自の SQL ステートメントが増えてしまう可能性があります。 長い SQL ステートメントはエージェント内のメトリック増加の原因となり、他のシステム上の問題だけでなくパフォーマンスを低下させることになります。

不適切に記述された SQL ステートメントがメトリック増加を引き起こす仕組み

通常、SQL エージェント メトリックの数は、一意の SQL ステートメントの数とほぼ一致します。 アプリケーションで小規模な SQL ステートメント セットを使用しているにもかかわらず、一意の SQL メトリックが大規模かつ急激に増加していることが SQL エージェントによって示される場合は、SQL ステートメントの作成方法に問題がある可能性があります。

SQL ステートメントがメトリックの急増を引き起こす場合の共通の状況がいくつかあります。 たとえば、コメントが埋め込まれていたり、一時テーブルを作成したり、値リストを挿入したりする SQL ステートメントは、実行するたびに個別のメトリック名を不必要に作成する可能性があります。

例 : SQL ステートメントのコメント

SQL ステートメント内でのコメントの使用は、メトリックが急増する一般的な原因の 1 つです。たとえば、以下のような SQL ステートメントがある場合、

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

SQL エージェントは、以下のように、メトリック名の一部としてコメントを使用したメトリックを作成します。

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

SQL ステートメントに埋め込まれているコメントは、誰がどのクエリを実行しているのかをデータベース管理者が確認するのに役立ちますが、クエリを実行しているデータベースはそのコメントを無視します。 一方で、SQL エージェントは、SQL ステートメントをキャプチャしても、そのコメント文字列を解析しません。そのために、SQL エージェントは、一意なユーザ ID ごとに独自のメトリックを作成し、メトリック増加を引き起こす原因となります。

この問題は、SQL コメントを一重の引用符で囲むことで回避することができます。 例 :

```
"/*' John Doe, user ID=?, txn=? '*/ select * from table..."
```

すると SQL エージェントは、以下のようなメトリックを作成し、コメントによって一意のメトリック名が作成されなくなります。

```
"/* ? */ select * from table..."
```

例: 一時表または自動的に生成された表名

メトリックが急増するその他の潜在的な原因は、一時表、または SQL ステートメントで自動的に生成された名前を持つ表を参照するアプリケーションである場合があります。 たとえば、Investigator を開いて Backends|{backendName}|SQL|{sqlType}|sql の下のメトリックを表示した場合、 以下のようなメトリックが表示される場合があります。

```
SELECT * FROM TMP_123981398210381920912 WHERE ROW_ID = ?
```

この SQL ステートメントは、表名に一意の識別子が追加されている一時表にアクセスしています。 *TMP_* という表名に追加された数字によって、ステートメントが実行されるたびに一意のメトリック名が作成され、メトリックが急増します。

例: 値のリストを生成する、または値を挿入するステートメント

メトリックが急増するその他の原因には、値のリストを生成するステートメント、または値を大量に変更するステートメントがあります。 たとえば、潜在的なメトリック増加に関する警告をすでに受け取っていて、調査の結果、以下の SQL ステートメントを見直す必要があると仮定します。

```
#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)
```

コードを調べてみると、"CHANGE CITY FROM CARROLTON, TO CAROLTON, _ " が都市名の配列を生成していることが分かります。

同様に、潜在的なメトリックの急増を調査している場合は、これに似た SQL ステートメントを確認することをお勧めします。

```
CHANGE COUNTRY FROM US TO CA _ CHANGE EMAIL ADDRESS FROM TO BRIGGIN @ COM _ "
```

コードを調べてみると、CHANGE COUNTRY により国名が大量にリストされることがわかります。 また、国名に引用符を付けると、ユーザの電子メール アドレスが SQL ステートメントに挿入され、メトリックの急増の原因となり得る一意のメトリックが作成されます。

SQL ステートメントの正規化オプション

長い SQL ステートメントを解決するには、以下のような SQL エージェントに使用するノーマライザを含めます。

- デフォルト SQL ステートメント ノーマライザ (P. 221)
- カスタム SQL ステートメント ノーマライザ (P. 222)
- 正規表現の SQL ステートメントのノーマライザ (P. 224)
- コマンドラインの SQL ステートメントのノーマライザ (P. 228)

デフォルト SQL ステートメント ノーマライザ

SQL エージェントでは、デフォルトで標準 SQL ステートメント ノーマライザがオンになっています。 これは一重の引用符内のテキストを正規化します（'xyz'）。 たとえば、SQL エージェントが以下の SQL クエリを変換するとします。

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

以下は、変換後の正規化されたステートメントです。

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

正規化されたステートメントのメトリックは集約され、Workstation の Investigator に表示することができます。

カスタム SQL ステートメント ノーマライザ

SQL エージェントでは、カスタムの正規化を実行するための拡張を追加できます。 これを行うため、SQL エージェントによって実装されている正規化スキーマを含む JAR ファイルを作成します。

SQL ステートメントのノーマライザ拡張を適用する方法

1. 拡張 JAR ファイルを作成します。

   注: SQL ノーマライザ拡張ファイルのエントリ ポイント クラスは、*com.wily.introscope.agent.trace.ISqlNormalizer* インターフェースを実装している必要があります。

   JAR 拡張ファイルの作成には、SQL ノーマライザ拡張用の特定のキーを含むマニフェスト ファイルの作成が含まれます。これについては、以下の手順 2 で詳しく説明します。ただし、拡張を動作させるためには、その他の一般キーも必要です。 これらのキーは、すべての拡張ファイルを作成するときに使用するタイプです。 作成する拡張ファイルは、データベース SQL ステートメント テキスト正規化、たとえば *Backends|{backendName}|SQL|{sqlType}|{actualSQLStatement}* ノードの下のメトリックと関連します。 *{actualSQLStatement}* は、SQL ノーマライザによって正規化されます。

2. 作成した拡張のマニフェストに、以下のキーを配置します。

   - *com-wily-Extension-Plugins-List:testNormalizer1*

     注: このキーの値は任意です。 このインスタンスでは、例として *testNormalizer1* が使用されています。 このキーの値として指定するものはすべて、以下のキーでも同様に使用してください。

   - *com-wily-Extension-Plugin-testNormalizer1-Type: sqlnormalizer*
   - *com-wily-Extension-Plugin-testNormalizer1-Version: 1*
   - *com-wily-Extension-Plugin-testNormalizer1-Name: normalizer1*

     ノーマライザの一意の名前を含んでいる必要があります。たとえば normalizer1 となります。

   - *com-wily-Extension-Plugin-testNormalizer1-Entry-Point-Class: <Thefully-qualified classname of your implementation of ISQLNormalizer>*

3. 作成した拡張ファイルを、*<Agent_Home>/wily/core/ext* ディレクトリに配置します。

4. *IntroscopeAgent.profile* で、以下のプロパティを配置および設定します。
   `introscope.agent.sqlagent.normalizer.extension`

   作成した拡張のマニフェスト ファイルからプロパティを *com-wily-Extension-Plugin-{plugin}-Name* に設定します。このプロパティの値は大文字と小文字を区別しません。 例：
   `introscope.agent.sqlagent.normalizer.extension=normalizer1`

   重要: これはホット プロパティです。 拡張名を変更すると、拡張を再登録する必要があります。

5. *IntroscopeAgent.profile* で、オプションで以下のプロパティを追加してエラー スロットル カウントを設定できます。
   `introscope.agent.sqlagent.normalizer.extension.errorCount`

   エラーおよび例外の詳細については、「例外 (P. 223)」を参照してください。

   注: カスタム ノーマライザ拡張によってスローされたエラーがエラー スロットル カウントを超えたとき、拡張は無効となります。

6. *IntroscopeAgent.profile* を保存します。
7. アプリケーションを再起動します。

例外

作成した拡張によって、あるクエリに対してエラーが発生した場合、デフォルトの SQL ステートメント ノーマライザはそのクエリに対してデフォルトの正規化スキーマを使用します。 このようなことが発生した場合は、拡張によって例外が発生したことを示す ERROR メッセージが記録され、スタック トレース情報と一緒に DEBUG メッセージも記録されます。ただし、このようなエラーが 5 個発生すると、デフォルトの SQL ステートメント ノーマライザは作成した拡張を無効にし、ノーマライザが変更されるまで今後のクエリで作成した拡張が使用されるのを停止します。

Null または空の文字列

クエリに対して、作成した拡張が Null 文字列または空の文字列を返す場合、StatementNormalizer はそのクエリに対してデフォルト正規化スキーマを使用し、拡張が Null 値を返したことを示す INFO メッセージが記録されます。 ただし、このような Null または空の文字列が 5 個返されると、StatementNormalizer はメッセージのログ記録を停止しますが、引き続き拡張の使用を試みます。

正規表現の SQL ステートメントのノーマライザ

SQL エージェントには、設定可能な正規表現（regex）に基づいて SQL ステートメントを正規化する拡張機能が用意されています。 この *RegexNormalizerExtension.jar* ファイルは *<Agent_Home>/wily/core/ext* ディレクトリにあります。

正規表現 SQL ステートメント ノーマライザの使用例については、「正規表現 SQL ステートメント ノーマライザの例 (P. 226)」を参照してください。

正規表現拡張機能を適用する方法

1. *IntroscopeAgent.profile* を開きます。
2. 以下のプロパティで設定を行います。
   - *introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer*

     事前設定されているノーマライザ スキーマを上書きするのに使用される SQL ノーマライザ拡張機能の名前を指定します。 正規表現拡張機能を有効にする場合、このプロパティを RegexSqlNormalizer に設定します。
   - *introscope.agent.sqlagent.normalizer.regex.keys=key1*

     このプロパティは、リストされた順番に評価される regex グループ キーを指定します。 正規表現拡張機能を有効化するためには、このプロパティが必要です。 デフォルトの値はありません。
   - *introscope.agent.sqlagent.normalizer.regex.key1.pattern=A*

     このプロパティは、SQL ステートメントと照合するのに使用される regex パターンを指定します。 *java.util.Regex* パッケージクラスで許可されている有効な正規表現は、すべてここで使用可能です。正規表現拡張機能を有効化するためには、このプロパティが必要です。 デフォルトの値はありません。
   - *introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=B*

     このプロパティは、置換文字列の形式を指定します。*java.util.Regex* パッケージクラスで許可されている有効な regex は、すべてここで使用可能です。 正規表現拡張機能を有効化するためには、このプロパティが必要です。 デフォルトの値はありません。

- *introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false*

  このプロパティが true に設定されている場合、SQL 文字列はすべての regex キー グループに対して評価されます。 実装は連鎖されます。 したがって、SQL 文字列が複数のキー グループと一致する場合、group1 からの正規化された SQL 出力は group2 の入力となり、同じように続きます。

  プロパティが false に設定されている場合、キー グループが SQL 文字列と一致するとすぐに、そのグループから正規化された SQL 出力が返されます。 MatchFallThrough property によって拡張機能が有効化または無効化されることはありません。

  たとえば、Select * from A where B、という SQL 文字列があるとすると、以下のプロパティを設定できます。

  ```
  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 が *false* の場合、SQL は key1 regex に対して正規化されます。その regex からの出力は、次のようになります： *Select * from X where B*。 この SQL は返されます。

introscope.agent.sqlagent.normalizer.regex.matchFallThrough が *true* の場合、SQL は最初に key1 regex に対して正規化されます。その regex からの出力は、次のようになります。*Select * from X where B*. 次にこの出力は、*key2 regex* の入力となります。*key2 regex* からの出力は、*Select * from X where Y* です。 これは SQL が返されます。

注: 正規表現拡張機能を有効化する場合、このプロパティは必要ありません。

- *introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false*

  このプロパティは、パターン マッチで大文字小文字を区別するかどうかを指定します。 デフォルト値は False です。 正規表現拡張機能を有効化するために、このプロパティは必要ありません。

- *introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false*

  このプロパティが false に設定されていると、SQL 内の一致パターンの最初の出現箇所が置換文字列に置き換えられます。 このプロパティが true に設定されている場合、SQL 内の一致パターンのすべての出現箇所が置換文字列に置き換わります。

  たとえば、Select * from A where A like Z、という SQL ステートメントがあるとすると、以下のようにプロパティを設定します。

  ```
  introscope.agent.sqlagent.normalizer.regex.key1.pattern=A
  introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=X
  ```

introscope.agent.sqlagent.normalizer.regex.key1.replaceAl が *false* の場合、SQL ステートメントは次のように正規化されます： *Select * from X where A like Z*。

introscope.agent.sqlagent.normalizer.regex.key1.replaceAl が *true* の場合、SQL ステートメントは次のように正規化されます： *Select * from X where* **X** *like Z*。

デフォルト値は False です。 正規表現拡張機能を有効化するために、このプロパティは必要ありません。

注: いずれの正規表現パターンも入力 SQL と一致しない場合、RegexNormalizer は Null の文字列を返します。 ステートメント ノーマライザは次に、デフォルトの正規化スキーマを使用します。

3. *IntroscopeAgent.profile* を保存します。

重要: 上記のプロパティはすべてホット プロパティです。つまりこれらのプロパティに対する変更は *IntroscopeAgent.profile* を保存した時点で有効になります。

正規表現 SQL ステートメント ノーマライザの例

以下の 3 つの例は、正規表現 SQL ステートメント ノーマライザの実装方法を理解するのに役立ちます。

例 1

これは正規表現 SQL ステートメントを正規化する前の SQL クエリです。

```
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)
```

理想的な正規化された SQL ステートメントは以下のようになります。

```
INSERT INTO COMMENTS (COMMENT_ID, ...) VALUES (?, ?, ?, ?, ?, ?,?, CHANGE CITY FROM
( )
```

IntroscopeAgent.profile ファイルを上記の正規化された SQL ステートメントにするために必要な設定は以下のとおりです。

```
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
```

例 2

これは正規表現 SQL ステートメントを正規化する前の SQL クエリです。

```
SELECT * FROM TMP_12398139821038192O912 WHERE ROW_ID =
```

理想的な正規化された SQL ステートメントは以下のようになります。

```
SELECT * FROM TMP_ WHERE ROW_ID =
```

IntroscopeAgent.profile ファイルを上記の正規化された SQL ステートメントにするために必要な設定は以下のとおりです。

```
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
```

例 3

SQL ステートメントを次のように正規化したいとします。*Select .... ResID1, CustID1* where *ResID1=.. OR ResID2=.. n times OR CustID1=.. OR n times* プロパティを以下のように設定する必要があります。

```
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
```

コマンドラインの SQL ステートメントのノーマライザ

正規表現 SQL ノーマライザが使用されておらず、where 句で値が二重引用符（" "）で囲まれている場合、以下のコマンドライン コマンドを使用して SQL ステートメントを正規化します。

```
-DSQLAgentNormalizeDoubleQuoteString=true
```

重要: このコマンドの代わりに正規表現 SQL ノーマライザを使用して二重引用符の SQL ステートメントを正規化することができます。 詳細については、「正規表現の SQL ステートメントのノーマライザ (P. 224)」を参照してください。

ステートメント メトリックの無効化

アプリケーションの中には大量の独自の SQL ステートメントを生成するものがあり、その結果、SQL エージェントにおいてメトリックが爆発的に増加してしまう場合があります。SQL エージェントの中で SQL ステートメント メトリックをオフにできます。

注: ステートメント メトリックをオフにした場合でも、バックエンドまたはトップレベルの JDBC メトリックは失われません。

ステートメント メトリックをオフにする方法

1. sqlagent.pbd ファイルを開き、SQL ステートメントを探します。 例：
   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements
   executeQuery(Ljava/lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends|{database}|SQL|{commandtype}|Query|{sql}"
   ```
2. 無効化する追跡ディレクティブから *{sql}* を削除します。 例：
   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements executeQuery(Ljava/
   lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends|{database}|SQL|{commandtype}|Query"
   ```
3. *sqlagent.pbd* ファイルを保存します。

   SQL エージェント内の SQL ステートメント メトリックがオフになります。

SQL メトリック

SQL エージェントのメトリックは Introscope Workstation Investigator の［バックエンド］ノードの下に表示されます。 SQL ステートメント メトリックは、Backends|<バックエンド名>|SQL ノードの下に表示されます。

注: *［平均応答時間（ミリ秒）］*は、*ExecuteReader()* メソッドを介して実行されたクエリなどのデータ リーダを返すクエリのみを表示します。 このメトリックは、データ リーダの *Close()* メソッドにかかった平均時間を表します。

SQL データに特有のメトリック タイプには以下のものがあります。

- Connection Count： メモリ内のライブ接続オブジェクトの数。

  接続は、ドライバの *connect()* メソッドが呼び出されると開始され、その接続が *close()* メソッドで閉じられると終了します。 SQL エージェントは、接続への弱い参照をセットで保持します。接続がガベージ コレクトされた場合、カウントには変更が反映されます。

- Average Result Processing Time (ms)： クエリの平均処理時間。

  このメトリックは、*executeQuery()* 呼び出しの最後から結果セットの *close()* メソッドの呼び出しまでの、結果セットの処理にかかる平均時間を示します。

注: インスツルメントされた XADataSource は、コミットまたはロールバック メトリックをレポートしない可能性があります。 ほかのインスツルメントされたデータソースは、メトリックにデータが含まれていない限り、コミットまたはロールバック メトリックをレポートしない可能性があります。

第 12 章: JMX レポートの有効化

このセクションでは、Java Agent が JMX データをレポートできるようにする方法について説明します。

このセクションには、以下のトピックが含まれています。

Java エージェントの JMX サポート

CA Introscope® は、アプリケーション サーバまたは Java アプリケーションが JMX 互換の MBean として公開する管理データを収集できます。 CA Introscope® は、それらの JMX データを Investigator メトリック ツリーに表示します。CA Introscope® は、以下のアプリケーション サーバを使用する、JMX 情報の収集をサポートしています。

- Glassfish - JMX メトリックをレポートするために、Glassfish に必要な追加設定はありません。
- JBoss - JMX メトリックをレポートするために JBoss を設定する手順は、「Java エージェントを使用するための JBoss の設定 (P. 45)」に説明されています。
- Tomcat - JMX メトリックをレポートするために Tomcat を設定する手順は「Java Agent を使用するための Apache Tomcat の設定 (P. 44)」に説明されています。
- WebLogic - JMX レポートを使用するために WebSphere および WebLogic を設定 (P. 236) できます。
- WebSphere - JMX レポートを使用するために WebSphere および WebLogic を設定 (P. 236) できます。

注: CA Introscope® は、Sun JMX 仕様に組み込まれた MBean をサポートします Sun JMX 仕様の詳細については、「Java Management Extensions」（英語）を参照してください。

CA Introscope® は JMX データを CA Introscope® のメトリック形式に変換して、Investigator 内の以下のノードの下に表示します。

<*ドメイン*>|<*ホスト*>|<*プロセス*>|<*エージェント*>|JMX|

Introscope での WebLogic JMX メトリックのサポート

WebLogic は、JMX メトリックのソースとして以下の MBean を提供します。

- RuntimeServiceMBean： サーバごとの実行時メトリック。アクティブで有効な設定を含みます。
- DomainRuntimeServiceMBean： ドメイン全体の実行時メトリック。
- EditServiceMBean： ユーザによる永続コレクションの編集を可能にします。

CA Introscope® は、以下の理由から RuntimeServiceMBean だけをポーリングします。

- RuntimeServiceMBean はローカル アクセスをサポートしている（効率の問題）。
- RuntimeServiceMBean には妥当と考えられるデータのほとんどが含まれている。

デフォルトの JMX メトリック変換プロセス

CA Introscope® では、デフォルトで Investigator に表示するために JMX メトリックが変換されます。 CA Introscope® は、以下の場合に MBean を変換します。

- WebLogic を使用する場合。
- JMX メトリックの簡素化のためのプライマリ キーの変換を設定 (P. 233) していない場合。

注: 一致する MBean がないプライマリ キーを指定している場合、CA Introscope® は、デフォルトの変換方法を使用します。

デフォルトの変換方法では、CA Introscope® は、属性の名前と値の両方を表示し、このペアをメトリック ツリーにアルファベット順に一覧表示します。

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

たとえば、WebLogic MBean に以下の特性があるとします。

- ドメイン名： WebLogic
- キーと値のペア： category=server、type=jdbc
- メトリック名： connections

プライマリ キーが IntroscopeAgent.profile の introscope.agent.jmx.name.primarykeys プロパティに指定されていない場合、MBean 属性は、以下の CA Introscope® メトリックに変換されます。

<ドメイン>|<ホスト>|<プロセス>|<エージェント>|JMX|Weblogic|*category=server|type=jdbc:connections*

キーと値のペアは、CA Introscope® メトリックではアルファベット順に表示されます。

プライマリ キーの変換を使用した JMX メトリックの簡素化

メトリックが JMX ノードの下に表示される順序を必要に応じて設定できます。エージェント プロファイルでプライマリ キーを定義できます。プライマリ キーは、MBean の ObjectName を識別します。

プライマリ キーの変換を設定しない場合、CA Introscope® は JMX データの変換 (P. 232)を行います。デフォルト変換では、メトリックは、Investigator の JMX ノードの下にアルファベット順に一覧表示されます。 JMX データをメトリックに変換するこの方法により、簡素化されたメトリック名が作成されます。 生成されたメトリックのキー/値のペアの情報の順序を制御します。

動作は、エージェント プロファイルの introscope.agent.jmx.name.primarykeys プロパティで設定されます。primarykeys プロパティの値は、一意に MBean を識別する MBeans JMX ObjectName の部分を指定します。 たとえば、WebLogic MBean の ObjectName には以下のキーが含まれます。

- MBean の種類を指定する Type キー。
- MBean が示すリソースの名前を指定する Name キー。

ObjectName のキー/値のペアは、MBean のタイプに応じてさまざまです。

CA Introscope® は、以下の規則に従って、introscope.agent.jmx.name.primarykeys プロパティの値で識別される MBean を変換および表示します。

- 表示されるのはキー値情報だけです。キー名は表示されません。
- 値は primarykeys プロパティで定義された順になります。
- 値の大文字と小文字が区別されます。

たとえば、WebLogic MBean に以下の特性があるとします。

- ドメイン名： WebLogic
- MBean ObjectName のキーと値のペア： category=server、type=jdbc
- メトリック名： connections

以下のように設定するとします。

introscope.agent.jmx.name.primarykeys=type,category connections 属性は、以下の構造で Investigator ツリーに表示されます。

<Introscope ドメイン>|*<ホスト>*|*<プロセス>*|*<エージェント>*|JMX|Weblogic|jdbc|server:connections

注: WebLogic 9.0 には普遍的に利用できるプライマリ キーがありません。デフォルトの変換方法にある、メトリックのキー/値のペアの名前付け規則を使用します。 このため、WebLogic 9.0 の JMX Metric ツリーは異なる構造を持ちます。

詳細:

デフォルトの JMX メトリック変換プロセス (P. 232)

JMX フィルタによるメトリックの量の管理

JMX フィルタの定義によって、CA Introscope® で収集および表示される JMX MBean 情報が決まります。 フィルタが設定されていない場合は、すべての JMX MBean 情報がエージェントによって Enterprise Manager にレポートされるため、システム オーバーヘッドが増加します。

フィルタは、エージェント プロファイル ファイル IntroscopeAgent.profile の introscope.agent.jmx.name.filter プロパティで設定されます。フィルタは、プロパティにカンマ区切りの文字列として入力されるキーワードです。CA Introscope® では、アスタリスク（*）および疑問符（?）のワイルドカード文字を使用したフィルタ文字列がサポートされています。

CA Introscope® は、JMX で生成されたメトリックとフィルタ文字列を照合します。 一致するものが見つかった場合、一致するメトリックが CA Introscope® にレポートされます。

返されるメトリックの量を制限するには、できるだけ絞り込むようにフィルタ文字列を定義します。 たとえば、1 つの MBean 属性と一致し、複数の MBean に存在するフィルタ文字列を定義します。 それらの MBean の *個々*のメトリックがレポートされます。 特定の MBean の属性だけが必要である場合は、フィルタ文字列内の MBean 名で属性名を修飾できます。

たとえば、JMSDestinationRuntime MBean の MessagesCurrentCount 属性値を取得するとします。

MessagesCurrentCount のメトリックの完全修飾名が以下の場合を考えます。

```
*SuperDomain*|host-name|Process|Agent-name|JMX|comp-1|
JMSDestinationRuntime|comp-2:MessagesCurrentCount
```

IntroscopeAgent.profile で introscope.agent.jmx.name.filter を以下のように定義します。

```
JMX|comp-1|JMSDestinationRuntime|comp-2:MessagesCurrentCount
```

WebLogic の JMX フィルタ

WebLogic 用の *IntroscopeAgent.profile* ファイルには、以下のキーワードがすでに定義されています。

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

JMX レポートを使用するための WebSphere および WebLogic の設定

JMX をサポートするように CA Introscope® を設定する方法は、使用するアプリケーション サーバによって異なります。ここでは、WebLogic Server および WebSphere から JMX データを収集して表示するように CA Introscope® を設定する方法について説明しています。

以下の手順に従います。

1. 管理対象アプリケーションをシャットダウンします。
2. WebSphere エージェントのみの場合は、IntroscopeAgent.profile で、introscope.agent.jmx.enable を true に設定します

   注: WebSphere エージェント プロファイルでのデフォルト値は false です。
3. IntroscopeAgent.profile でプライマリ キーを設定します。
   - WebLogic 9.0 の場合は、以下をコメント化します。
     `introscope.agent.jmx.name.primarykeys`
   - その他のバージョンの WebLogic の場合は、以下のコメント化を解除します。
     `introscope.agent.jmx.name.primarykeys`

   注: プロパティの値を変更する場合、値では大文字と小文字を区別する必要があります。また、複数のキーはカンマで区切る必要があります。

4. IntroscopeAgent.profile で、introscope.agent.jmx.name.filter プロパティのコメント化が解除されていることを確認します。

   注: フィルタには、バージョン番号などの MBean 属性を含める必要があります。 たとえば、以下の完全メトリック名があるとします。

   ```
   *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
   ```

   バージョン番号属性を含む、フィルタされたバージョンは以下のようになります。

   ```
   PlantsByWebSphere|name=PlantsByWebSphere#PlantsByWebSphere.war|node=LODVMAPM033Node01|platform=dynamicproxy|process=server1|spec=1.0|type=SessionManager|version=6.1.0.0
   ```

   または、単純に以下のようにします。

   ```
   Plant*re|*version*
   ```

5. プロパティに、目的の文字列をカンマで区切って入力します。

   CA Introscope® でフィルタ文字列を照合するには、正確なスペルで、大文字/小文字を区別して文字列を入力します。

6. 変更を保存します。
7. マネージド アプリケーションを起動します。
8. WebLogic Server の CA Introscope® 起動クラス、または WebSphere のカスタム サービスを設定して、JMX データを有効にします。

詳細:

WebLogic 用の起動クラスの作成 (P. 49)

WAS での JSR-77 データの有効化および JMX メトリックの表示

WebSphere で JSR-77 JMX MBean オブジェクトを収集、保持、およびレポートするように Introscope を設定できます。 J2EE 管理仕様である JSR-77 は、J2EE アーキテクチャの管理可能な部分を抽象化し、管理情報にアクセスするためのインターフェースを定義します。

注: JSR-77 をサポートするには、JVM のバージョン 1.4 以降が必要です。JVM が 1.4 である場合、アプリケーション サーバも JSR-77 をサポートする必要があります。

以下の手順に従います。

1. 管理対象アプリケーションをシャットダウンします。
2. WebSphere でカスタム サービスを設定します (P. 61)。
3. IntroscopeAgent.profile で、以下のプロパティが true であることを確認します。

   ```
   introscope.agent.jmx.enable=true
   ```

4. IntroscopeAgent.profile で、以下のプロパティを設定して JSR-77 を有効にします。

   ```
   introscope.agent.jmx.name.jsr77.disable=false
   ```

5. IntroscopeAgent.profile で以下のプロパティのコメント化を解除して、メトリック変換のプライマリ キー メソッドを設定します。

   ```
   introscope.agent.jmx.name.primaryKeys=J2EEServer,Application,
   j2eeType,JDBCProvider,name,mbeanIdentifier
   ```

   注: このプロパティ定義が含まれているのは、Introscope で提供されている WebSphere 用の IntroscopeAgent.profile のみです。

6. レポートする JSR-77 メトリックを指定するには、以下のプロパティのコメント化を解除して設定します。

   ```
   introscope.agent.jmx.name.filter
   ```

   フィルタは必須ではありませんが、使用することを強くお勧めします。

7. JSR-77 メトリックで、特定の Mbean 属性を除外するように指定するには、以下のプロパティのコメント化を解除し、必要に応じて更新します。

   ```
   introscope.agent.jmx.ignore.attributes=server
   ```

注: 詳細については、KB 記事 TEC534087「WebSphere Performance Viewer Data vs. Introscope PMI Data」（http://ca.com/support）を参照してください。

詳細:

プライマリ キーの変換を使用した JMX メトリックの簡素化 (P. 233)
JMX フィルタによるメトリックの量の管理 (P. 235)

第 13 章: プラットフォーム モニタの設定

このセクションでは、Introscope プラットフォーム モニタの設定方法について説明します。

このセクションには、以下のトピックが含まれています。

プラットフォーム モニタ

プラットフォーム モニタは、CPU 統計を含むシステム メトリックを Java Agent が Enterprise Manager にレポートできるようにします。 プラットフォーム モニタは、Introscope エージェント インストーラに含まれています。

以下の情報を考慮します。

- Windows Server 2003 および AIX 以外のすべてのオペレーティング システムのプラットフォーム モニタは、Java Agent のインストール時に自動的に有効になります。 Windows Server 2003 および AIX のプラットフォーム モニタを機能させるには、若干の設定が必要です。
- プラットフォーム モニタのバイナリは、アプリケーション サーバやオペレーティング システムのビット モードには依存しません。 また、プラットフォーム モニタのバイナリは、JVM アーキテクチャだけに依存しています。
- HP-UX 用の Java エージェント プラットフォーム モニタは、プロセスが 1 つ以上存在する場合に、CPU の使用率をレポートします。たとえば、プロセスが 4 つ存在する場合、CPU の最大使用率は 400 % になります（4 つのプロセスが CPU を 100 % 使用）。 1 つのプロセスが 110% の使用を示している場合は、このプロセスが CPU を 1.1 個使用していることを表します。

注: システム要件については、「*Compatibility Guide*」を参照してください。

Java エージェントは、以下のプラットフォーム メトリックを生成します。

- **ProcessID**
- **Processor Count** — CPU 数を示します。
- **Utilization % (process)**： Java エージェント プロセスの場合、すべてのプロセッサの総容量のうち、このプロセスが使用しているパーセンテージを示します。 プロセッサの数に関係なく、このメトリックは 1 つの数字だけを生成します。
- **Utilization % (aggregate)** — このプロセッサの場合の、システムのすべてのプロセスによる合計使用率（パーセンテージ）を示します。 各プロセッサは、Investigator ツリーのリソースとして表示されます。

Windows Server 2003 でのプラットフォーム モニタの有効化

Windows Server 2003 でプラットフォーム モニタを実行するには、「*admin*」アクセス権限が必要です。

Windows でプラットフォーム モニタリングを動作させるには、システム オブジェクトを有効にする必要があることに注意してください。

システム オブジェクトが有効化されているかどうかを確認する方法

1. ［スタート］ -［ファイル名を指定して実行］を選択します。
2. 「*perfmon*」と入力して、［実行］をクリックします。
3. ダイアログ ボックスで、［追加］をクリックします。
4. ［カウンタの追加］ダイアログ ボックスで、ドロップダウン リスト ボックスに「プロセス」および「プロセッサ」パフォーマンス オブジェクトがあれば、システム オブジェクトは有効化されています。

システム オブジェクトを有効にする方法

1. ［スタート］ -［アクセサリ］ -［コマンド プロンプト］を右クリック > ［別のユーザーとして実行］ -［Administrator］を選択します。
2. コマンド *lodctr /r* を実行します。

   「プロセス」および「プロセッサ」オブジェクトが有効化されます。

AIX でのプラットフォーム モニタの有効化

AIX でプラットフォーム モニタを有効にすることができます。

以下の手順に従います。

1. Java エージェントのインストール後に、wily/core/ext ディレクトリに以下のファイルがインストールされていることを確認します。
   - introscopeAIXPSeries32Stats.jar
   - introscopeAIXPSeries64Stats.jar
   - libIntroscopeAIXPSeries32Stats.so
   - libIntroscopeAIXPSeries64Stats.so
2. Perfstat ライブラリをインストールします。IBM の FTP サイトから以下のパッケージをインストールします。
   - bos.perf.libperfstat
   - bos.perf.perfstat
3. コンピュータを再起動します。

   パッチが有効になり、プラットフォーム モニタが有効になります。

プラットフォーム モニタの無効化

corresponding.jar ファイルを別のディレクトリに移動することによって、プラットフォーム モニタを無効にすることができます。

以下の手順に従います。

1. /wily/core/ext ディレクトリに移動します。
2. ご使用のプラットフォームに対応する、introscope<*プラットフォーム*>.jar ファイルを選択します（例： introscopeSolarisAmd32Stats.jar）。
3. /wily/core/ext ディレクトリから別のディレクトリに .jar ファイルを移動します。

HP-UX でのプラットフォーム モニタに対するアクセス権限の設定

.zip または .tar インストーラを使用して HP-UX にプラットフォーム モニタをインストールするには、以下のファイルの読み取り/書き込み/実行の権限を 777 に変更する必要があります。

```
wily/ext/introscopeHpuxItaniumStats.jar
wily/ext/introscopeHpuxItaniumStats.so
wily/ext/introscopeHpuxParisc32Stats.jar
wily/ext/introscopeHpuxParisc32Stats.so
```

例（root ユーザ）：

```
chmod 777 /<Agent_Home>/wily/ext/introscopeHpuxItaniumStats.jar
```

権限の変更は、エージェントのインストール後で、エージェント を起動する前に行ってください。

上記の最初の 2 つのファイルを使用する場合は、エージェントの起動前に gcc もインストールする必要があります。C および C++ 用のコンパイラ gcc は、HP のサポート Web サイトで探してダウンロードできます。

プラットフォーム モニタのトラブルシューティング

ほとんどの場合、プラットフォーム モニタは、オペレーティング システムを検出して、そのオペレーティング システムがサポートされている場合は動作します。 万が一検出しない場合は、Java エージェント プロファイルで明示的にオペレーティング システムを指定すれば、プラットフォーム モニタを確実に動作させるのに役立ちます。

以下の手順に従います。

1. IntroscopeAgent.profile を開きます。
2. 「Platform Monitor Configuration」という見出しの下で、ご使用のオペレーティング システムに完全に一致する値を探し、そのプロパティのコメント化を解除します。 たとえば、Sun SPARC ハードウェアを使用する Sun Solaris 64 ビット オペレーティング システムについては、以下の値を見つけてコメント化を解除します。

   ```
   #introscope.agent.platform.monitor.system=SolarisSparc64
   ```

3. 管理対象アプリケーションを再起動します。

Windows でのプラットフォーム モニタリングのトラブルシューティング

Windows プラットフォームでは、Java Agent について以下のエラーが表示される場合があります。

```
11/28/06 08:29:55 AM PST [ERROR] [IntroscopeAgent] An error occurred polling for platform data
```

エラーが頻繁ではない場合は、Windows 自身の一時的なエラーが原因で発生していると考えられ、害はありません。Windows 以外のプラットフォームでエラーが発生する場合、またはエラーが頻繁に発生する場合は、比較的重大なエラーである可能性があるので、Introscope の CA サポート に報告してください。

SAP NetWeaver

SAP NetWeaver では、SAP ユーザ アカウントの権限が不足しているために、プラットフォーム モニタが正しく機能しない場合があります。

デフォルトでは、SAP ユーザ アカウントは Performance Monitor Users に登録されません。これは、［コンピュータの管理］ -［システム ツール］ -［ローカル ユーザとグループ］-［グループ］-［Performance Monitor Users］の順にアクセスすると見つかります。 Performance Monitor Users に登録されたユーザには、Perfmon 関連のデータ（HKEY_PERFORMANCE_DATA）へのアクセス権があります。

SAP NetWeaver および Introscope パフォーマンス モニタリングに関する問題が発生した場合は、Performance Monitor Users グループに SAP ユーザ アカウントを追加して Windows マシンを再起動すると、この問題を解決できる場合があります。

第 14 章: CA APM と CA LISA の統合

このセクションには、以下のトピックが含まれています。

CA APM と CA LISA を統合する方法 (P. 245)

CA APM と CA LISA を統合する方法

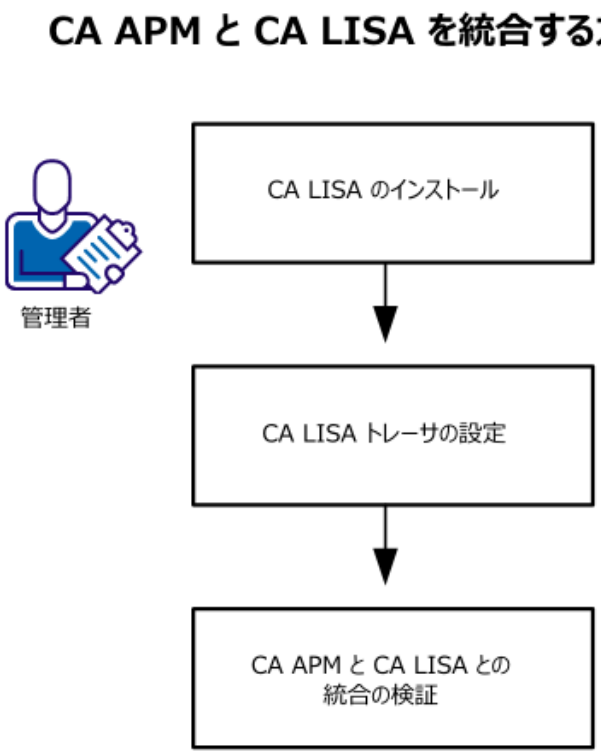
CA APM と CA LISA を統合し、運用前環境の CA LISA からのロードおよび再帰テストから生成された合成トランザクションを使用して、アプリケーションのパフォーマンス問題の監視、検出、切り分け、および診断を行えます。

合成トランザクションは、実際のトランザクション パフォーマンスを表します。以下の用途に利用できます。

- アプリケーションを実運用環境にリリースする前に、テスト環境でデータを監視します。 合成データの監視によって、アプリケーションは定義済みの制限内で動作することが確認され、また期待されるサービス レベル アグリーメント、リソース消費、およびベースライン パフォーマンス特性が認定されます。
- ソフトウェア開発ライフサイクルのアプリケーション パフォーマンスの問題またはボトルネックの根本原因を早期に検出および識別します。その結果、ユーザは実稼働環境に、より高品質のアプリケーションを配布できます。
- ガイダンスとして実稼働キャパシティ プラニングに使用するために、運用前環境でアプリケーションのパフォーマンスおよびリソース使用状況レベルのレポートを作成します。

- 監視する実運用アプリケーションの重要な側面を把握できるようにメトリックを収集します。

以下の図では、管理者が CA LISA と CA APM を統合するために実行するタスクについて説明します。



1. CA LISA のインストール (P. 250)
2. CA LISA トレーサの設定 (P. 251)。
3. CA APM と CA LISA の統合の確認 (P. 251)。

CA LISA のインストール

以下の方法で、CA LISA をインストールできます。

- CA LISA ファイルのダウンロードおよび抽出 (P. 247)
- CA LISA をインストールする対話型インストーラの使用 (P. 250)

CA LISA ファイルの手動でのダウンロードおよび抽出

CA APM と統合するために CA LISA ファイルをダウンロードおよび抽出します。

重要: CA LISA 統合をインストールする前に、JRE がインストールされていることを確認します。

以下の手順に従います。

1. CA サポート の CA APM ソフトウェア ダウンロード セクションから、CALISAIntegrationNoInstaller<*バージョン番号*>.windows.zip または CALISAIntegrationNoInstaller<*バージョン番号*>.unix.tar をダウンロードします。
2. ほかのエージェントの配布が含まれていないディレクトリへ配布ファイルの内容を抽出します。

   ファイルは指定されたディレクトリに抽出されます。

CA LISA からのテスト イベント メトリックおよびデータを表示するための CA APM のインスツルメント

CA APM をインスツルメントし、以下の CA LISA プロセスの CA LISA からのテスト イベント メトリックおよびデータを表示します。

- CA LISA Coordinator
- CA LISA Test Runner
- CA LISA Workstation

追加の CA LISA プロセスをインツルメントできます。 ただし、CPU、メモリ使用率、およびエージェントに関連する最小のメトリック、および JVM アイデンティティのみがレポートされます。

以下の手順に従います。

1. Windows または Linux プロセス用の使用可能な実行可能ファイルまたはシェル スクリプト ファイルと同じ名前の対応する .vmoptions ファイルを作成し、*<LISA_Home>*¥bin ディレクトリに保存します。 たとえば、CA LISA Workstation をインスツルメントするには、*<LISA_Home>*¥bin ディレクトリに LISAWorkstation.vmoptions ファイルを作成します。

   以下のリストは、インスツルメントできる、使用可能な実行可能ファイルまたはシェル スクリプト ファイルの名前とプロセスを示しています。

   - Coordinator プロセス -- CoordinatorServer.exe または CoordinatorServer.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Registry プロセス -- Registry.exe、Registry.sh、TestRegistry.exe、または TestRegistry.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Simulator プロセス -- Detached_Simulator.exe、Detached_Simulator.sh、Simulator.exe、または Simulator.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - TestRunner プロセス -- TestRunner.exe または TestRunner.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Virtual Service Environment プロセス -- VirtualServiceEnvironmentService.exe または VirtualServiceEnvironmentService.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。

- Workstation プロセス -- LISAWorkstation.exe または LISAWorkstation.sh ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。

2. （オプション）Windows サービス用の使用可能な実行可能ファイルまたはシェル スクリプト ファイルと同じ名前の対応する .vmoptions ファイルを作成し、*<LISA_Home>*¥bin ディレクトリに保存します。

   以下のリストは、インスツルメントできる実行可能ファイルの名前とサービスを示しています。

   - Coordinator サービス -- CoordinatorService.exe ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Simulator サービス -- SimulatorService.exe ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Registry サービス -- TestRegistryService.exe ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。
   - Virtual Service Environment サービス -- VirtualServiceEnvironmentService.exe ファイルに対応するファイルを .vmoptions ファイル拡張子で作成します。

3. Java の com.wily.introscope.agent.agentName システム プロパティを定義する行を追加して、エージェント ノード名を変更します。たとえば、ノードに CA LISA Workstation Agent と名前を付けるには、以下の行を追加します。

   ```
   -Dcom.wily.introscope.agent.agentName=CA LISA Workstation Agent
   ```

4. 以下の行を .vmoptions ファイルに追加します。

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

   ここで *<<Agent_Home>* は CA LISA 固有のエージェント インストールへのパスです。 通常、*<Agent_Home>* パスは絶対パスですが、CA LISA プロセスが実行されるカレント ディレクトリを基準にした相対パスも可能です。

5. （オプション）追加する JVM コマンド ライン オプションまたはシステム プロパティごとに個別の行を入力します。

6. インスツルメントされた CA LISA プロセスが Java 1.7 で実行される場合は、-XX:UseSplitVerifier を追加します。

7. アプリケーション サーバを再起動します。

   インスツルメント設定が適用されます。

インストーラを使用した CA LISA のインストール

インストーラを使用して、CA APM と CA LISA を統合できます。

以下の手順に従います。

1. お使いのオペレーティング システム用のインストーラを選択します。
   - CALISAIntegrationInstaller<*バージョン番号*>.unix.tar
   - CALISAIntegrationInstaller<*バージョン番号*>windows.zip
2. CA LISA サーバで選択したインストーラを実行します。
   a. インストール中に、CA LISA インストールのホーム ディレクトリおよびインストールに使用されるディレクトリを指定します。
   b. インスツルメントする CA LISA プロセスを選択します。
   c. EM およびポートを指定します。

      「CA LISA からのテスト イベント メトリックおよびデータを表示するための CA APM のインスツルメント (P. 248)」の説明のとおり、インストーラは、*<LISA_Home>*¥bin ディレクトリに必要な vmoptions ファイルを作成します。

      注: インスツルメントされた CA LISA アプリケーションが Java 1.7 で実行される場合は、インストール後に vmoptions ファイルを手動で編集して、-XX:UseSplitVerifier を追加します。
3. EM のインストール中に、監視オプションとして、［CA APM Integration for CA LISA］を選択します。

   管理モジュールおよびタイプ ビューに、CA LISA データが表示されます。

   注: インストーラの実行時に［CA APM Integration for CA LISA］監視オプションを選択しない場合、EM インストール ディレクトリにある <EM_Home>¥examples¥CAAPMIntegrationForCALISA フォルダからファイルをコピーすることで、手動でこの手順を実行できます。

CA LISA トレーサの設定

CA LISA トレーサを設定して、監視する CA LISA システム コンポーネントをカスタマイズします。 CA LISA トレーサでは、メトリックが以下のノードにレポートされる最低のレベルを制御することによって、CA APM にレポートされるメトリックの数を制限できます。

- Test Case
- Simulator
- Test Steps

メトリックがレポートされる最低レベルは、［Test Case］ノードです。

以下の手順に従います。

1. *<LISA_Home>*¥core¥config に移動し、lisa.pbd 設定ファイルを開き、環境の要件に合わせて必要な設定を行い、ファイルを保存します。
2. *<LISA_Home>*¥core¥config に移動し、IntroscopeAgent.profile 設定ファイルを開き、環境の要件に合わせて必要な設定を行い、ファイルを保存します。
3. アプリケーション サーバを再起動します。

   設定が適用されます。

CA APM と CA LISA の統合の確認

CA LISA プロセスを呼び出し、インスツルメントされたノードが Investigator ツリーに存在することを確認することによって、CA APM と CA LISA の統合を確認できます。

注: CA LISA テストを使用して呼び出され、Investigator ツリーの［<CA LISA>］-［Test Case］ノードの下に表示されるメトリックは、CA LISA Coordinator、CA LISA Test Runner、または CA LISA Workstation から実行された後にのみ、Enterprise Manager にレポートされます。

以下の手順に従います。

1. CA LISA Workstation を使用して、テストを実行します。 CA LISA の使用方法の詳細については、CA LISA のドキュメントを参照してください。
2. まだ実行されていない場合は、Enterprise Manager を起動します。
3. まだ実行されていない場合は、CA LISA プロセスを開始します。

   CA LISA プロセスを再起動するとエージェントが起動されます。
4. Workstation を起動し、Investigator を開きます。
5. 以下のノード下のデータを探します。

   SuperDomain | <*ホスト名*> | CA LISA | <*CA LISA Workstation*> | CA *LISA* | Test Case | <*テスト ケース名*>

   - <ホスト名> は、CA LISA エージェントがインストールされたコンピュータが含まれるノードです。
   - <CA LISA Workstation> は、LISAWorkstation.vmoptions ファイルで指定されたエージェント名のノードです。 このノードの設定の詳細については、「CA LISA からのテスト イベント メトリックおよびデータを表示するための CA APM のインスツルメント (P. 248)」を参照してください。
   - <テスト ケース名> は、検証の最初の手順で実行されたテスト ケースの名前です。
6. フォルダを展開します。

   アクティブなメトリックが、設定されたノードの下に表示されます。

第 15 章: CA APM Cloud Monitor の CA APM への統合

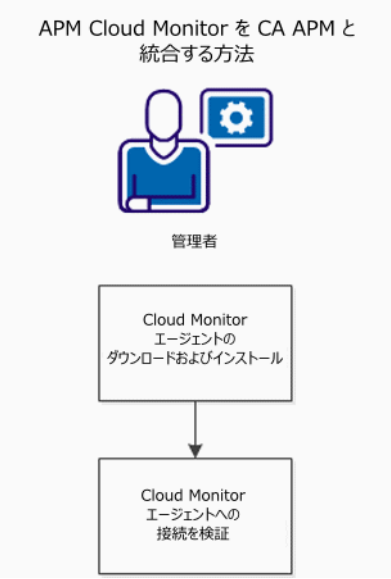
CA APM Cloud Monitor を使用すると、以下のタスクを実行できます。

- 40 以上の国々の 60 以上の監視ステーションからすべてのユーザ エクスペリエンスを把握する。
- 実際のブラウザで監視を行い、ユーザ エクスペリエンスを正確に計測する。
- SaaS ベンダーおよび MSP が提供するアプリケーションの監視により、SLA の遵守を図る。
- ファイアウォールの外部からのアプリケーション応答時間を（合成トランザクションによって）テストし、エンド ユーザ エクスペリエンスをグローバルに把握する。また、ユーザ トラフィックが発生していない場合にもパフォーマンスを監視する。
- 実際のユーザ トランザクションをレプリケートして、アプリケーション インフラストラクチャ全体にわたってパフォーマンスを監視することで、問題を迅速に特定、診断、および解決する。

CA APM 導入環境に CA APM Cloud Monitor を統合する方法

APM 管理者は、CA APM の導入環境に CA APM Cloud Monitor を統合して、監視および問題切り分けの機能を拡張できます。

以下のフローチャートは、オンプレミスの CA APM 導入環境に CA APM Cloud Monitor を統合する方法を示しています。



1. CA APM Cloud Monitor エージェントをダウンロードしてインストールします。（P. 254）
2. エージェント接続を検証します。（P. 255）

CA APM Cloud Monitor エージェントのダウンロードおよびインストール

このタスクでは、CA APM Cloud Monitor エージェントをダウンロードしてインストールします。このエージェントは、CA APM Cloud Monitor と CA APM を統合するために必要です。

注: CA APM Cloud Monitor エージェントは、ネットワーク上の任意のコンピュータにインストールできます。 その機能は、インターネットを介して CA APM Cloud Monitor データを受信し、Enterprise Manager にそのデータを転送することです。任意のコンピュータにインストールすることができますが、サーバ レベルの CPU/RAM を持ったコンピュータを選択する必要があります。

CA APM Cloud Monitor エージェントをダウンロードしてインストールする方法

1. CA APM Cloud Monitor エージェント アーカイブ ファイルをダウンロードします。 そのファイルをコンピュータに保存します。

   Windows、Linux、および Solaris 用のインストーラは、CA APM ソフトウェア ダウンロード Web サイトにあります。

2. エージェントをインストールします。

   a. CA APM Cloud Monitor インストーラ ファイルを起動します。

   b. 使用許諾の条項に同意します。

   c. *<Agent_Home>* ディレクトリを指定します。

   d. エージェントが接続する Enterprise Manager ホストおよびポートを指定します。

   e. ［Cloud Monitor Account Settings］画面で、CA APM Cloud Monitor のユーザ ID とパスワードを入力します。 これらは、CA APM Cloud Monitor Web サイトへのログインに使用するのと同じクレデンシャルです。

CA APM Cloud Monitor エージェントのインストールが完了します。

注: エージェントの起動および接続の検証の手順については、「CA APM Cloud Monitor エージェント接続の検証 (P. 255)」を参照してください。

CA APM Cloud Monitor エージェント接続の検証

CA APM Cloud Monitor エージェントが正しくインストールされていることを検証するには、エージェントを起動し、CA APM WebView または Workstation を使用してデータをチェックします。CA APM Cloud Monitor からの受信データが最新であることを確認することによって、接続を検証します。

注: WebView または Workstation で CA APM Cloud Monitor データを確認するには、フォルダと監視を CA APM Cloud Monitor にセットアップしておく必要があります。 この手順を実行していない場合は、「*CA APM Workstation ユーザ ガイド*」で CA APM Cloud Monitor の使用に関するセクションを参照してください。

以下の手順に従います。

1. CA APM Cloud Monitor エージェントがインストールされたディレクトリに移動し、以下のいずれかのファイルを実行してエージェントを起動します。
   - Windows の場合、APMCloudMonitor.bat をダブルクリックします。
   - Linux または Solaris の場合、APMCloudMonitor.sh を実行します。

   エージェントがコンソール ウィンドウで起動します。
2. CA APM WebView または Workstation を起動します。
3. CA APM Cloud Monitor のデータを見つけます。
   a. Investigator ウィンドウが開いていない場合は、［ファイル］-［新規 Investigator］を選択します。
   b. ［メトリック ブラウザ］タブを参照します。
   c. 以下のノードを展開します。

      SuperDomain | *<Host_Name>* | APMCloudMonitor | APMCloudMonitorAgent | APM Cloud Monitor

      <Host_Name> は、通常は CA APM Cloud Monitor エージェントがインストールされているコンピュータですが、［メトリック ブラウザ］ツリーに表示されるものが APMCloudMonitor.properties ファイルの apmcm.agent.hostName プロパティの値になります。
   d. フォルダを展開して個別のメトリックを参照し、メトリックが最新であることを確認します。

トラブルシューティング

Enterprise Manager のインストール処理中に監視オプションとして「Cloud Monitor」を選択しなかった場合、WebView または Workstation は CA APM Cloud Monitor データの一部を表示しません。

問題の解決方法

- *<EM_Home>*¥examples¥CAAPMIntegrationForCloudMonitor¥ ディレクトリの内容を <EM_Home>/config/modules/ および <EM_Home>/ext/xmltv にコピーします。

  これにより、Cloud Monitor メトリック データを正しく表示するのに必要な管理モジュールおよびその他のファイルが使用可能になります。

データを制限する方法

CA APM Cloud Monitor エージェントから Enterprise Manager に送信されるデータの量を制限して、パフォーマンスを向上させることができます。

CA APM Cloud Monitor プロパティの設定によるデータの制限

APMCloudMonitor.properties ファイル内のプロパティを使用して、CA APM Cloud Monitor エージェントが Enterprise Manager に送信するデータをフィルタできます。

CA APM Cloud Monitor エージェント プロパティの設定によりデータをフィルタするには、*<CloudMonitor_Agent_Home>*/CloudMon/conf/APMCloudMonitor.properties 内の「Metric filters」セクションを編集します。

これらの設定の詳細については、APMCloudMonitor.properties のプロパティ リファレンスを参照してください。

チェックポイントの削除によるデータの制限

CA APM Cloud Monitor では、五大陸の 60 を超えるチェックポイント ステーションにアクセスできます。 Cloud Monitor は、これらのステーションの中からランダムに選択し、ステーションから Web サイトまたはアプリケーションへの可用性とパフォーマンスをチェックします。 時間と共に、有効なすべてのステーションがこのチェックを実行し、その結果、60 を超える各サイトからデータがログ記録されます。

利用可能なチェックポイント ステーションの一部を削除することで、CA APM Cloud Monitor が CA APM に送信するデータの量を制限できます。

以下の手順に従います。

1. CA APM Cloud Monitor の Web サイト（cloudmonitor.ca.com）にログインします。
2. ［Subscriptions］ - ［Preferences］を選択します。

   デフォルトでは、すべてのステーションが選択されています。

3. 以下のように、デフォルトの選択を変更します。
   - 個別のステーションのチェック ボックスをオフにします。*または*、
   - ［Clear］をクリックしてすべてをクリアし、リストの先頭にあるグループの中から選択します。

     たとえば、北米にあるステーションだけを選択するには、以下の手順に従います。

     a. ［Clear］をクリックします。

     b. ［North America］をクリックします。
4. ページ下部にある［Change］をクリックします。

スケジュールの調整によるデータの制限

デフォルトでは、監視は定期的（毎日 1 時間おき）に可用性とパフォーマンスをチェックします。 この結果、時間が経過すると必要以上のデータが CA APM に返される場合があります。

以下の手順に従います。

1. Cloud Monitor の Web サイト（cloudmonitor.ca.com）にログインします。
2. ［Settings］ > ［Monitors］を選択します。
3. 個別の監視を選択し、［More Options］を選択します。
4. 以下の設定をリセットします。
   - Delay between checks
   - Check period
   - Check on these days only
   - Maintenance schedule

付録 A: Java Agent のプロパティ

IntroscopeAgent.profile の場所の設定

エージェントは、基本的な接続および名前付けのプロパティについて、IntroscopeAgent.profile 内のプロパティを参照します。エージェントをインストールすると、エージェント プロファイルが *<Agent_Home>*/wily/core/config ディレクトリにインストールされます。

Introscope は、記載されている順に、以下の場所でエージェント プロファイルを探します。

- システム プロパティ com.wily.introscope.agentProfile で定義されている場所
- com.wily.introscope.agentResource で定義されている場所
- *<作業ディレクトリ>*/wily/core/config ディレクトリ

注: Windows コンピュータでパスを追加する場合、各円記号（¥）にもう 1 つ円記号を付加してエスケープ処理します。たとえば、「C:¥¥Introscope¥¥lib¥¥Agent.jar」のようにします。

IntroscopeAgent.profile の場所を変更する方法

1. 以下のいずれかの方法で、新しい場所を定義します。
   - Java コマンド ラインで、**-D** オプションを使って、以下の IntroscopeAgent.profile ファイルの場所への完全パスを指定して、システム プロパティを定義します。
   - -D com.wily.introscope.agentProfile.

- クラスパスのリソースで IntroscopeAgent.profile を利用できるようにします。 *com.wily.introscope.agentResource* を、エージェント プロファイルを含むリソースへのパスを指定するよう設定します。

  注: IntroscopeAgent.profile の場所を変更する場合は、AutoProbe ログの場所も変更する必要があります。 詳細については、「ProbeBuilder ログの管理 (P. 172)」を参照してください。

2. ProbeBuilder ディレクティブ（PBD および PBL ファイル）をエージェント プロファイルと同じ場所に移動します。これらは、プロファイルの場所を基準にして参照されます。

Sun ONE を使用する場合は、エージェント プロファイルの新しい場所を Sun One の *server.xml* ファイルに追加します。

IntroscopeAgent.profile for Sun ONE の場所を変更する方法

1. Introscope 情報を Sun ONE 7.0 の起動スクリプトに追加するため、管理者または root としてログインします。
2. *<SunOne_Home>*/domains/domain1/server1/config/ にある server.xml ファイルを開きます。
3. server.xml の jvm-options スタンザに以下の行を追加します。

```
<jvm-options>
-Dcom.wily.introscope.agentProfile=SunOneHome/wily/core/config/IntroscopeAgent.profile
</jvm-options>
```

コマンドライン プロパティの上書き

Introscope では、コマンド ラインを使用して、Enterprise Manager、エージェント、Workstation、および WebView の特定のプロパティを上書きすることができます。 Java Agent に関して言えば、この機能は、共有されている複数のエージェントのコピーを持つクラスタ環境を使用しており、監視対象のアプリケーションごとにいくつかのエージェントの設定を調整する場合に便利です。

以下の手順は、監視対象のアプリケーション サーバにエージェントをインストールおよび設定しており、エージェントが正しく Enterprise Manager に接続していることを前提としています。

コマンド ラインを使用してエージェント プロパティを上書きする方法

1. Java コマンドを変更したファイルを開いて、エージェントを起動します。

   このファイルの場所は、ご利用の環境で使用しているアプリケーション サーバによって異なります。

2. *-D* コマンドを追加して、プロパティを上書きします。 たとえば、以下のコマンドを追加すると、エージェントが *weblogic-full.pbl* ファイルも使用するようにできます。

   ```
   -Dintroscope.autoprobe.directivesFile=weblogic-full.pbl
   ```

   このコマンドを、開いているファイルの他の *-D* コマンドの横に配置します。

   注: このコマンドを使用してホット デプロイ可能なプロパティを上書きすると、そのプロパティはホット デプロイ可能ではなくなります。また、設定ファイルのプロパティを後で変更すると、Workstation によって、上書きしたプロパティを変更したため変更は有効でないことを示す警告メッセージが表示されます。 これを回避するためには、設定ファイルのプロパティを変更する前に上書きコマンドを削除します。

3. ファイルを保存します。

4. エージェントを再起動します。

   上記で使用されている例では、Workstation のエージェント ノードに WebLogic メトリックが追加されているのがわかります。

   重要: システム プロパティは、Introscope プロパティのプロパティ スペースの一部となり、IndexedProperties を使用して *java.io.tmpdir* などすべてのものを表示できるようになります。

エージェント フェールオーバ

Java Agent がプライマリ Enterprise Manager から切断された場合に備えて、エージェント フェールオーバ プロパティでは、どの Enterprise Manager に対してエージェントがフェールオーバするか、およびエージェントがプライマリ Enterprise Manager に再接続を試行する回数を指定します。

introscope.agent.enterprisemanager.connectionorder

エージェントがデフォルト Enterprise Manager から切断された場合に、エージェントが使用するバックアップ Enterprise Manager との接続順序を指定します。

プロパティ設定

エージェントが接続可能な他の Enterprise Manager の名前。

デフォルト

ホスト名、ポート番号およびソケット ファクトリの DEFAULT プロパティによって定義された Enterprise Manager。

例

```
introscope.agent.enterprisemanager.connectionorder=DEFAULT
```

注

- カンマ区切りリストを使用します。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds

以下の Enterprise Manager への再接続を拒否されたエージェントが再接続を試行する間隔（秒単位）を指定します。

- エージェント プロファイルの introscope.agent.enterprisemanager.connectionorder プロパティ値で設定された順序に基づく Enterprise Manager。
- loadbalancing.xml の設定に基づいて許可されたすべての Enterprise Manager

Enterprise Manager に接続できないエージェントは、以下の方法で接続を処理します。

- その次に接続を許可されている Enterprise Manager への接続を試みます。
- 許可されていない Enterprise Manager へは接続しません。

注: loadbalancing.xml の設定、およびエージェントから Enterprise Manager への接続の設定については、「CA APM 設定および管理ガイド」を参照してください。

デフォルト

デフォルトの間隔は 120 秒です。

例

```
introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120
```

注

このプロパティの変更を有効にするには、管理対象アプリケーションを再起動します。

このプロパティは、デフォルトでコメント化されています。

このプロパティは、エージェントが以下の CA APM コンポーネントにまたがって接続することを許可されている環境で役立ちます。

- クラスタ
- コレクタおよびスタンドアロン Enterprise Manager
- クラスタ、コレクタ、およびスタンドアロン Enterprise Manager の任意の組み合わせ

エージェントが別のクラスタ内の Enterprise Manager に接続できる場合に、このプロパティが設定されていないと何が発生するかを以下の例に示します。

1. クラスタ内の Enterprise Manager に接続しているエージェントが切断されます。
2. エージェントは、クラスタ 2 内の Enterprise Manager に拒否モードで接続します。
3. エージェントは、クラスタ 1 内の許可された Enterprise Manager がいつ利用可能になるのかわかりません。

このプロパティによって、エージェントは Enterprise Manager が接続可能になるまで、許可された Enterprise Manager への接続を試行し続けるよう強制されます。

エージェント HTTP トンネル

トンネリング技術を使用して情報を送信できるようにエージェントを設定できます。これにより、エージェントから Enterprise Manager にリモート接続できるようになります。 この機能を使用するには、HTTP トンネル Web サービスのホストとなる Enterprise Manager 上の埋め込み Web サーバにエージェントを接続するように設定する必要があります。

新しいエージェント接続として *IntroscopeAgent.profile* に HTTP トンネル通信を設定するには、以下のように指定します。

- Enterprise Manager を実行しているマシンのホスト名。詳細については、introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT（P. 266）を参照してください。
- Enterprise Manager Web サーバへの接続ポート。 これは、エージェントが接続する Enterprise Manager の *IntroscopeEnterpriseManager.properties* で指定されている *introscope.enterprisemanager.webserver.port* プロパティの値です。
- HTTP トンネル ソケット ファクトリ。 以下のクライアント ソケット ファクトリを指定します。
  `com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory`

プロキシ サーバ用のエージェント HTTP トンネル

以下のプロパティは、エージェントが HTTP 上でトンネルを行うように設定されている場合にのみ適用され、プロキシ サーバを使用して Enterprise Manager に接続する必要があります。

- introscope.agent.enterprisemanager.transport.http.proxy.host（P. 265）
- introscope.agent.enterprisemanager.transport.http.proxy.port（P. 265）
- introscope.agent.enterprisemanager.transport.http.proxy.username（P. 265）
- introscope.agent.enterprisemanager.transport.http.proxy.password（P. 266）

詳細については、「HTTP トンネルのためのプロキシ サーバの設定 (P. 73)」を参照してください。

introscope.agent.enterprisemanager.transport.http.proxy.host

プロキシ サーバのホスト名を指定します。

デフォルト

指定なし（コメント化されています）。

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.http.proxy.port

プロキシ サーバのポート番号を指定します。

デフォルト

指定なし（コメント化されています）。

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.http.proxy.username

プロキシ サーバがエージェントに認証を要求する場合は、認証用のユーザ名を指定します。

デフォルト

指定なし（コメント化されています）。

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.http.proxy.password

プロキシ サーバがエージェントに認証を要求する場合は、認証用のパスワードを指定します。

デフォルト

指定なし（コメント化されています）。

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

エージェント HTTPS トンネル

HTTPS を使用して情報を送信できるようにエージェントを設定できます。これにより、エージェントから Enterprise Manager にリモート接続できるようになります。

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT（P. 266）
- introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT（P. 267）
- introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT（P. 267）

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータのホスト名を指定します。

デフォルト

localhost

例

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

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

Enterprise Manager をホストするコンピュータ上のポート番号を指定します。 HTTPS トンネリングを使用している場合、エージェントからの接続をリスンするデフォルトのポートは 8444 です。 このプロパティは、デフォルトでコメント化されています。

デフォルト

8444

例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

HTTPS 使用時にエージェントから Enterprise Manager への接続に使用する、クライアント ソケット ファクトリを指定します。

デフォルト

com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory

例

```
introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

エージェントのメモリ オーバーヘッド

エージェントで重大なメモリ オーバーヘッドが発生するのは極端な場合のみです。 メモリ消費量を抑えると、応答時間が遅くなる可能性があります。 ただし、アプリケーションはそれぞれに異なっているため、メモリ使用量と応答時間との間のトレードオフはアプリケーション自体によって異なる可能性があります。

introscope.agent.reduceAgentMemoryOverhead

このプロパティは、使用するエージェント設定を指定します。 エージェントのメモリ オーバーヘッドを軽減する場合は、コメント化を解除します。 デフォルトでは、このプロパティは true に設定されていますが、コメント アウトされています。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.reduceAgentMemoryOverhead=true
```

注

このプロパティの変更を有効にするには、管理対象アプリケーションを再起動します。

Agent メトリックのエイジング

エージェント メトリック エイジングは、エージェントのメモリ キャッシュから定期的にデッド メトリックを削除します。 デッド メトリックとは、設定された時間内に新しいデータをレポートしていないメトリックを表します。 古いメトリックを削除することで、エージェントのパフォーマンスを改善し、急増メトリックの危険性を回避することができます。

注: 急増メトリックは、不注意から、システムの処理能力を超える量のメトリックをレポートするようにエージェントが設定されている場合に発生します。 非常に多くのメトリックがレポートされると、エージェントがアプリケーション サーバの性能に影響を与える可能性があり、極端な場合では、サーバがまったく機能しなくなる可能性があります。

同一のグループに存在するメトリックは、グループ内のすべてのメトリックが削除の対象であると判断される場合にのみ削除されます。 現在、*BlamePointTracer* および *MetricRecordingAdministrator* メトリックのみがグループとして削除されます。 その他のメトリックは個別に削除されます。

MetricRecordingAdministrator には、メトリック グループを作成、取得、または削除するための以下のインターフェースがあります。

- *getAgent().IAgent_getMetricRecordingAdministrator.addMetricGroup*

  文字列のコンポーネント、収集メトリックです。コンポーネント名は、メトリック グループのメトリック リソース名です。 複数のメトリックがグループであるとみなされるためには、同じメトリック ノード下にある必要があります。 メトリックは、*com.wily.introscope.spec.metric.AgentMetric* データ構造体のコレクションです。 このコレクションには *AgentMetric* データ構造体のみ追加することができます。

- *getAgent().IAgent_getMetricRecordingAdministrator.getMetricGroup*

  文字列のコンポーネントです。メトリック リソース名であるコンポーネント名に基づいて、メトリックのコレクションを取得することができます。

- *getAgent().IAgent_getMetricRecordingAdministrator.removeMetricGroup*

  文字列のコンポーネントです。メトリック グループは、メトリック リソース名であるコンポーネントに基づいて削除されます。

- *getAgent().IAgent_getDataAccumulatorFactory.isRemoved*

  メトリックが削除されたかどうかを確認します。このインターフェースは、拡張機能にアキュムレータのインスタンスを保持する場合に使用します。メトリック エイジングによってアキュムレータが削除された場合、このインターフェースを使用して無効な参照を保持することを防ぎます。

重要: *MetricRecordingAdministrator* インターフェースを使用する拡張機能を作成する場合は(たとえば、ほかの CA Technologies 製品で使用するため)、必ずアキュムレータの独自のインスタンスを削除してください。 メトリックが長い間呼び出されなかったためにエイジアウトし、その後データがそのメトリックで使用できるようになった場合、古いアキュムレータ インスタンスは新しいメトリック データ ポイントを作成しません。 このような状況を回避するには、アキュムレータの独自のインスタンスを削除せず、*getDataAccumulatorFactory* インターフェースを代わりに使用してください。

エージェント メトリック エイジングの設定

エージェント メトリック エイジングは、デフォルトでオンになっています。 プロパティ introscope.agent.metricAging.turnOn (P. 271) を使用して、この機能をオフにするよう選択することもできます。 *IntroscopeAgent.profile* からこのプロパティを削除すると、エージェント メトリック エイジングはデフォルトでオフになります。

エージェント メトリック エイジングは、エージェントのハートビートで実行されます。 ハートビートはプロパティ introscope.agent.metricAging.heartbeatInterval (P. 272) を使用して設定します。ハートビートの頻度は低く設定するようにしてください。ハートビートの頻度が高いと、エージェントおよび CA Introscope® のパフォーマンスに影響を与えます。

各ハートビートの間に、メトリックの特定のセットがチェックされます。これは、プロパティ introscope.agent.metricAging.dataChunk (P. 272) を使用すると設定できます。 また、高い値はパフォーマンスに影響を与えるため、この値を低くすることも重要なことです。ハートビートごとにチェックされるメトリック数のデフォルト値は 500 です。削除の対象となるメトリックがないかどうか、500 メトリックのそれぞれがチェックされます。たとえば、このプロパティを 1 回のハートビートにつき 500 メトリックずつチェックするよう設定し、エージェント メモリには合計で 10,000 のメトリックがある場合、10,000 メトリックすべてをチェックするのに時間はかかりますが、パフォーマンスへの影響は小さくなります。 しかし、このプロパティを大きな数字に設定すると、10,000 メトリックすべてをチェックする時間は短くなりますが、オーバーヘッドが大きくなってしまう可能性があります。

メトリックが一定期間新しいデータを受信していないと、そのメトリックは削除の候補になります。 この期間はプロパティ introscope.agent.metricAging.numberTimeslices (P. 273) を使用して設定できます。 デフォルトでは、このプロパティは 3000 に設定されています。 メトリックが削除の条件と一致すると、グループ内のすべてのメトリックがメトリック削除の候補になっているかどうかのチェックが実行されます。この要件も満たした場合、メトリックは削除されます。

introscope.agent.metricAging.turnOn

エージェント メトリック エイジングをオンまたはオフにします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.metricAging.turnOn=true
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.metricAging.heartbeatInterval

メトリックを削除するかどうかをチェックする間隔（秒単位）を指定します。

デフォルト

1800

例

```
introscope.agent.metricAging.heartbeatInterval=1800
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.agent.metricAging.dataChunk

各間隔ごとにチェックされるメトリックの数を指定します。

デフォルト

500

例

```
introscope.agent.metricAging.dataChunk=500
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.metricAging.numberTimeslices

このプロパティは、削除の候補にする前に、新しいデータを受信しなかったかどうかをチェックする間隔を指定します。

デフォルト

3000

例

```
introscope.agent.metricAging.numberTimeslices=3000
```

注

このプロパティへの変更はただちに有効になります。管理対象アプリケーションを再起動する必要はありません。

introscope.agent.metricAging.metricExclude.ignore.0

指定したメトリックを削除対象から除外します。エイジングから 1 つ以上のメトリックを除外するには、メトリック名またはメトリック フィルタをリストへ追加します。

プロパティ設定

カンマ区切りのメトリックのリスト。 メトリック名には、アスタリスク（*）をワイルド カードとして使用できます。

デフォルト

デフォルトは「Threads」で始まるメトリック名です（*Threads**）。

例

```
introscope.agent.metricAging.metricExclude.ignore.0=Threads*
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

エージェント メトリック クランプ

エージェントが Enterprise Manager に送信するメトリックをクランプ（制限）するおよその数は設定できます。 生成されたメトリックの数がプロパティの値を超えている場合、エージェントは新しいメトリックの収集と送信を停止します。

introscope.agent.metricClamp

エージェントが Enterprise Manager に送信するメトリックをクランプするおよその数を設定します。

デフォルト

5000

例

```
introscope.agent.metricClamp=5000
```

注

- このプロパティが設定されていない場合、メトリック クランプは発生しません。 古いメトリックが引き続き値をレポートします。
- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。
- このクランプ プロパティは、apm-events-thresholds-config.xml ファイル内にある introscope.enterprisemanager.agent.metrics.limit プロパティと連携して動作します。

  注: introscope.enterprisemanager.agent.metrics.limit プロパティについては、「CA APM 設定および管理ガイド」を参照してください。

  introscope.enterprisemanager.agent.metrics.limit クランプ値が introscope.agent.metricClamp 値の前にトリガされた場合、Enterprise Manager はエージェント メトリックを読み取りますが、それを Investigator メトリック ブラウザ ツリー内でレポートしません。

  introscope.agent.metricClamp クランプ値が introscope.enterprisemanager.agent.metrics.limit クランプ値の前にトリガされた場合、エージェントは Enterprise Manager へのメトリックの送信を停止します。

エージェント ネーミング

アプリケーション サーバの Java Agent 名などといった情報を取得するためにプロパティを設定できます。

詳細:

Java Agent 名の理解 (P. 149)

introscope.agent.agentAutoNamingEnabled

サポートされているアプリケーション サーバの Java Agent 名を、エージェントの自動名前付け機能を使用して取得するかどうかを指定します。

プロパティ設定

True または False

デフォルト

アプリケーション サーバによって異なります。以下の「注」を参照してください。

例

```
introscope.agent.agentAutoNamingEnabled=false
```

注

- WebLogic、WebSphere および JBoss のアプリケーション サーバでは、エージェントの自動名前付け機能が**有効**になっています。
- このプロパティは、アプリケーション サーバが WebLogic の場合は起動クラスを指定する必要があり、WebSphere の場合はカスタム サービスを指定する必要があります。
- Oracle Application Server、Interstage、Sun ONE、および Tomcat のアプリケーション サーバでは、エージェントの自動名前付け機能は**無効**になっています。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

重要: WebLogic、WebSphere、および JBoss では、プロパティ *introscope.agent.agentAutoNamingEnabled* がデフォルトで **TRUE** に設定されています。

introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds

エージェントから Enterprise Manager に対して接続が行われる前で、エージェントが名前付け情報を受信待ちする時間の長さ（秒単位）を指定します。

デフォルト

120

例

```
introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds=120
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.agentAutoRenamingIntervalInMinutes

エージェントの名前が変更されたかどうかをエージェントが確認する間隔（分単位）。

デフォルト

10

例

```
introscope.agent.agentAutoRenamingIntervalInMinutes=10
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.agentName

ほかのエージェント名前付けの方法が失敗するときは、このプロパティのコメント化を解除し、デフォルトのエージェント名を付けます。

プロパティ設定

どのシステムでも、このプロパティの値が無効である場合、またはプロファイルから削除されている場合は、エージェント名は *UnnamedAgent* になります。

例

```
#introscope.agent.agentName=AgentName
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- アプリケーション サーバ固有のエージェント インストーラで提供されるエージェント プロファイルでは、デフォルトは、たとえば *WebLogic Agent* など、アプリケーション サーバを反映したものとなります。
- デフォルトのエージェント インストーラで提供されるエージェント プロファイルでは、このプロパティの値は、*AgentName* になり、行がコメント化されます。

introscope.agent.agentNameSystemPropertyKey

Java システム プロパティの値を使用してエージェント名を指定する場合は、このプロパティを使用します。

デフォルト

指定なし。

例

```
introscope.agent.agentNameSystemPropertyKey
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.disableLogFileAutoNaming

AutoNaming オプションを使用している場合に、エージェントのログ ファイルの自動名前付けを無効にするかどうかを指定します。

このプロパティを **true** に設定すると、エージェント、AutoProbe、および LeakHunter のログファイルで、エージェント名またはタイムスタンプを使用した自動名前付けが無効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.disableLogFileAutoNaming=false
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- ログ ファイルの自動名前付けは、Java システム プロパティまたはアプリケーション サーバ カスタム サービスを使用してエージェント名を決定できる場合にのみ、有効になります。

introscope.agent.clonedAgent

アプリケーションの同一コピーを同じマシン上で実行することを可能にします。 アプリケーションの同一のコピーを同じマシンで実行する場合は、このプロパティを True に設定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.clonedAgent=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.customProcessName

Introscope Enterprise Manager および Workstation に表示されるプロセス名を指定します。

デフォルト

アプリケーション サーバによって異なります。

例

```
introscope.agent.customProcessName=CustomProcessName
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- アプリケーション サーバ固有のエージェント インストーラで提供されるエージェント プロファイルのデフォルトでは、たとえば「WebLogic」など、アプリケーション サーバを反映したものとなります。
- デフォルトのエージェント インストーラで提供されるエージェント インストーラでは、このプロパティはコメント化されています。

introscope.agent.defaultProcessName

カスタム プロセス名が提供されず、エージェントが、メイン アプリケーション クラスの名前を判断できない場合は、この値がプロセス名として使用されます。

デフォルト

UnknownProcess

例

```
introscope.agent.defaultProcessName=UnknownProcess
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.display.hostName.as.fqdn

このプロパティは、エージェント名を完全修飾ドメイン名（fqdn）として表示するかどうかを指定します。 完全修飾ドメイン名を有効にするには、このプロパティ値を 「true」に設定します。デフォルトでは、エージェントはホスト名を表示します。

注: Catalyst 統合では、このプロパティを true に設定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.display.hostName.as.fqdn=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

エージェントの記録(ビジネスの記録)

エージェントがビジネス トランザクションの記録を処理する方法を制御することができます。

注: エージェント ビジネス記録の詳細については、「CA APM トランザクション定義ガイド」を参照してください。

introscope.agent.bizRecording.enabled

エージェントでのビジネス トランザクション記録を有効または無効にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.bizRecording.enabled=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

エージェントでのビジネス トランザクション記録でさらに設定が必要な場合は、アプリケーション問題切り分けマップの追加プロパティを参照してください。

詳細:

アプリケーション問題切り分けマップ (P. 286)
アプリケーション問題切り分けマップのビジネス トランザクションの POST パラメータ (P. 291)
アプリケーション問題切り分けマップの管理されたソケットの設定 (P. 295)

エージェント スレッドの優先順位

以下のプロパティは、エージェント スレッドの優先順位を制御します。

- introscope.agent.thread.all.priority（P. 283）

introscope.agent.thread.all.priority

エージェント スレッドの優先順位を制御します。

プロパティ設定

1 （最低）～ 10 （最高）の範囲の値を設定できます。

デフォルト

5 （コメント化されています）

例

```
#introscope.agent.thread.all.priority=5
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

エージェントの Enterprise Manager への接続

エージェントから Enterprise Manager への接続方法は制御することができます。

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータのホスト名を指定します。

デフォルト

localhost

例

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

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

エージェントからの接続をリスンしている Enterprise Manager をホストするコンピュータ上のポート番号を指定します。

デフォルト

デフォルトのポートは、設定する通信チャネルのタイプによって異なります。 エージェントと Enterprise Manager 間の直通通信では、デフォルト ポートは 5001 です。

例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- HTTPS （HTTP over SSL）を使用して Enterprise Manager に接続する場合は、デフォルト ポートは 8444 です。SSL を使用して Enterprise Manager に接続する場合は、デフォルト ポートは 5443 です。 ただし、これらのデフォルト設定はデフォルトではコメントアウトされています。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

エージェントから Enterprise Manager への接続に使用する、デフォルトのクライアント ソケット ファクトリを指定します。

デフォルト

デフォルトのソケット ファクトリは、設定する通信チャネルのタイプによって異なります。 エージェントと Enterprise Manager 間の直通通信では、デフォルトのソケット ファクトリは以下のとおりです。

com.wily.isengard.postofficehub.link.net.DefaultSocketFactory

例

```
introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.DefaultSocketFactory
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータの IP アドレスを指定します。

デフォルト

このプロパティは、デフォルトでは定義されていません。

例

```
introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT=<address>
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータのローカル ポートを指定します。

デフォルト

このプロパティは、デフォルトでは定義されていません。

例

```
introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT=<ポート>
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

アプリケーション問題切り分けマップ

アプリケーション問題切り分けマップ データを設定できます。

注: アプリケーション問題切り分けマップの使用方法については、「CA APM ワークステーション ユーザ ガイド」を参照してください。

introscope.agent.appmap.enabled

アプリケーション問題切り分けマップの監視対象のコードの追跡を有効または無効にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.appmap.enabled=true
```

注

デフォルトで有効です。

introscope.agent.appmap.metrics.enabled

アプリケーション問題切り分けマップのノードのメトリック追跡を有効または無効にします。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.appmap.metrics.enabled=false
```

注

このプロパティは、デフォルトでコメント化されています。

introscope.agent.appmap.queue.size

アプリケーション問題切り分けマップのバッファ サイズを設定します。

プロパティ設定

正の整数。

デフォルト

1000

例

```
introscope.agent.appmap.queue.size=1000
```

注

- 値は正の整数である必要があります。
- 値を 0 に設定すると、バッファは制限されません。
- このプロパティは、デフォルトでコメント化されています。

introscope.agent.appmap.queue.period

Enterprise Manager にアプリケーション問題切り分けマップ データを送信するための頻度をミリ秒で設定します。

プロパティ設定

正の整数。

デフォルト

1000

例

```
introscope.agent.appmap.queue.period=1000
```

注

- 必ず正の整数となるようにしてください。
- 値を 0 に設定すると、デフォルト値が使用されます。
- このプロパティは、デフォルトでコメント化されています。

introscope.agent.appmap.intermediateNodes.enabled

アプリケーションのフロントエンドおよびバックエンド ノードの間の中間ノードを含める機能を有効または無効にします。

プロパティ設定

True または False

デフォルト

False

例

```
#introscope.agent.appmap.intermediateNodes.enabled=true
```

注

- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。
- このプロパティを true に設定すると、エージェントのパフォーマンスが低下する場合があります。
- このプロパティは、デフォルトでコメント化されています。

アプリケーション問題切り分けマップと Catalyst の統合

アプリケーション問題切り分けマップ データは、Catalyst と統合するための設定が可能です。

注: アプリケーション問題切り分けマップの使用方法については、「CA APM ワークステーション ユーザ ガイド」を参照してください。

情報を送信する機能の設定

このプロパティは、Catalyst との統合のための追加情報を送信するエージェント機能を有効または無効にします。

以下の手順に従います。

1. デフォルトの IntroscopeAgent.profile ファイルをテキスト エディタで開きます。

introscope.agent.appmap.catalystIntegration.enabled=<false|true> の行を探し、以下のように値を設定します。

true

Catalyst との統合のための追加情報を送信するエージェント機能を有効にします。

false

設定を無効にします。

指定方法の例を以下に示します。

```
introscope.agent.appmap.catalystIntegration.enabled=false
```

注: このプロパティは、デフォルトでコメント化されています。

2. ファイルを保存して閉じます。

エージェントは、この設定を使用するようにセットアップされます。

利用可能なネットワークのリストの設定

introscope.agent.primary.net.interface.name プロパティは、Catalyst 統合用のエージェントによって使用されるホスト コンピュータのプライマリ ネットワーク インターフェース名を指定します。 このプロパティの設定は変更することができ、変更は自動的に適用されます。

注: エージェント ロギング レベルが DEBUG に設定されている場合、設定に利用可能なネットワーク インターフェース名に関する情報がログ ファイル内に表示されます。 あるいは、ネットワーク インターフェース ユーティリティを使用して、このプロパティのプライマリ ネットワークインターフェース名を指定できます。

以下の手順に従います。

1. デフォルトの IntroscopeAgent.profile ファイルをテキスト エディタで開きます。
2. introscope.agent.primary.net.interface.name=<false|true> の行を探し、名前の値を指定します。

   名前の指定方法の例を以下に示します。

   ```
   introscope.agent.primary.net.interface.name=eth4
   ```

   注: デフォルト値は未定義です。 このプロパティが設定されていない場合、エージェントはプライマリ インターフェースとして利用可能な最初のネットワーク インターフェースを割り当てます。ネットワーク インターフェース ユーティリティを使用して、このプロパティのプライマリ ネットワークインターフェース名を指定できます。
3. （オプション）複数のネットワーク アドレスを許可するには、サブインターフェース番号（開始番号は 0）を指定します。

   サブインターフェース番号の指定方法の例を以下に示します。

   ```
   introscope.agent.primary.net.interface.name=eth4.1
   ```
4. ファイルを保存して閉じます。

   この設定を使用するプロファイルがセットアップされます。

詳細:

ネットワーク インターフェース ユーティリティの使用 (P. 423)

アプリケーション問題切り分けマップのビジネストランザクションの POST パラメータ

一致した POST パラメータによってさらに複雑な監視を実行できるように、Local Product Shorts を設定できます。

introscope.agent.bizdef.matchPost

このプロパティは、POST パラメータが照合を行う時期を決定します。

プロパティ設定

このプロパティの有効な設定値は、「never」、「before」、「after」です。

- エージェントの完全な機能を使用し、パフォーマンスを改善するには、このプロパティを **never** に設定します。 この設定により、アプリケーションは、URL、cookie、およびヘッダ パラメータを使用して、すべてのビジネス トランザクションを識別できます。ただし、POST パラメータのみを使用して識別されるビジネス トランザクションの照合はできません。
- エージェントのパフォーマンスを十分に発揮させるには、このプロパティを **before** に設定します。 この設定により、アプリケーションは POST パラメータを使用して一部またはすべてのトランザクションを識別できます。ただし、HTTP フォーム要求のサーブレット ストリームには直接アクセスしません。このプロパティが *before* に設定されている場合、デプロイされる新しいアプリケーションも標準の API に準拠する必要があります。

  重要: このプロパティを *before* に設定すると、場合によってはアプリケーションに悪影響が及ぶおそれがあります。 このプロパティの設定については、実装前に CA Technologies の担当者に確認してください。

- ビジネス トランザクションと POST パラメータを正確に照合するには、このプロパティに **after** を設定します。ただし、エージェント機能は制限されます。 このプロパティに *after* を設定した場合、エージェントは、プロセス間で POST パラメータによって識別されたビジネス トランザクションをマップしたり、完全なセットのメトリックを作成することができません。また、この設定は他のオプションよりも CPU 時間を少し多く消費します。ただし、POST パラメータ機能が必要な場合には一番確実な設定であると考えられます。 この設定により、アプリケーションは POST パラメータを使用して一部またはすべてのビジネス トランザクションを識別できますが、サーブレット ストリームに直接アクセスしないという保証はできません。

例

```
introscope.agent.bizdef.matchPost=after
```

注

- *never* - POST パラメータとの照合を試行しません。 これは一番高速のオプションですが、的確でないビジネス トランザクションと一致する場合があります。
- *before* - サーブレットが実行される前に POST パラメータを照合します。
- *after* - サーブレットが実行された後に POST パラメータ パターンを照合します。 プロセス間のマッピングと一部のメトリックは使用できません。 このパラメータのデフォルト設定です。

既知の制限

エージェント記録を使用して定義されたメトリックは、Investigator のアプリケーション問題切り分けマップに表示されます。 エージェント記録を設定するときに正規表現を使用する場合、いくつかの既知の制限があります。 ほとんどの制限は POST パラメータに関係するものです。

既知の制限は以下のとおりです。

- 行の終了文字（.）は、POST パラメータ値ではサポートされていません。
- POST パラメータの定義がビジネス トランザクション定義に依存している場合、ビジネス トランザクション コンポーネントに提供されるのは 3 つのメトリックのみです。 提供されるメトリックは以下のとおりです。
  - Average Response Time
  - Responses Per Interval
  - Errors Per Interval
- POST パラメータの定義がビジネス トランザクション定義に依存している場合、トランザクション追跡コンポーネントのビジネス コンポーネント名は汎用的な名前になります。ビジネス サービス、ビジネス トランザクション、およびビジネス トランザクション コンポーネントに固有の名前ではありません。また、これは、照合しない POST パラメータの定義に依存しているビジネス トランザクション定義にも適用されます。
- JBoss および Tomcat の一部のバージョンは、ヘッダ キーを小文字の値として保存する場合があるため、*HEADER_TYPE* で *caseSensitiveName* 属性が正しく機能しません。

注: エージェント記録の詳細については、「CA APM トランザクション定義ガイド」を参照してください。

アプリケーション問題切り分けマップの管理されたソケットの設定

以下のプロパティを使用すると、アプリケーション問題切り分けマップでのソケット メトリックの表示を有効または無効にできます。

- introscope.agent.sockets.managed.reportToAppmap（P. 295）
- introscope.agent.sockets.managed.reportClassAppEdge（P. 296）
- introscope.agent.sockets.managed.reportMethodAppEdge（P. 296）
- introscope.agent.sockets.managed.reportClassBTEdge（P. 297）
- introscope.agent.sockets.managed.reportMethodBTEdge（P. 297）

アプリケーション問題切り分けマップの使用方法の詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

introscope.agent.sockets.managed.reportToAppmap

管理されたソケットをアプリケーション問題切り分けマップに表示するようにします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.sockets.managed.reportToAppmap=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.sockets.managed.reportClassAppEdge

管理されたソケットがクラス レベルのアプリケーションの境界線をアプリケーション問題切り分けマップにレポートできるようにします。

プロパティ設定

True または False

デフォルト

False

例

introscope.agent.sockets.managed.reportClassAppEdge=false

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.sockets.managed.reportMethodAppEdge

管理されたソケットがメソッド レベルのアプリケーションの境界線をアプリケーション問題切り分けマップにレポートできるようにします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.sockets.managed.reportMethodAppEdge=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.sockets.managed.reportClassBTEdge

管理されたソケットがクラス レベルのビジネス トランザクションの境界線をアプリケーション問題切り分けマップにレポートできるようにします。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.sockets.managed.reportClassBTEdge=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.sockets.managed.reportMethodBTEdge

管理されたソケットがメソッド レベルのビジネス トランザクションの境界線をアプリケーション問題切り分けマップにレポートできるようにします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.sockets.managed.reportMethodBTEdge=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

AutoProbe

以下のプロパティは、AutoProbe を設定します。

- introscope.autoprobe.directivesFile（P. 298）
- introscope.autoprobe.enable（P. 298）

introscope.autoprobe.directivesFile

AutoProbe の ProbeBuilder ディレクティブ ファイルを指定します。

デフォルト

インストーラによって異なります。

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- このプロパティに 1 つ以上のディレクトリが含まれ、かつ動的インスツルメンテーションが有効である場合、エージェントはアプリケーションを再起動せずに指定されたディレクトリからディレクティブ ファイルをロードします。

introscope.autoprobe.enable

このプロパティは、バイトコードへのプローブの自動挿入を有効または無効にします。

プロパティ設定

true または false

デフォルト

true

例

```
introscope.autoprobe.enable=true
```

注: このプロパティを false に設定すると、アプリケーションのバイトコードへのプローブの自動挿入が無効になります。ただし、エージェントおよびエージェントが行うレポートは無効になりません。 変更を有効にするには、JVM を再起動する必要があります。

introscope.autoprobe.logfile

Introscope AutoProbe は、変更を常にログに記録しようとします。ログ ファイルの場所をデフォルト以外の場所に移動するには、このプロパティを設定します。

プロパティ設定

絶対ファイル パス、または絶対以外のパス。 絶対名以外で指定すると、このプロパティ設定ファイルを起点とする相対的な指定と見なされます。

デフォルト

../../logs/AutoProbe.log

例

```
introscope.autoprobe.logfile=../../logs/AutoProbe.log
```

ログ記録を無効にするには、以下のようにログ ファイルをコメント化します。

```
introscope.autoprobe.logfile=logs/AutoProbe.log
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

ブートストラップ クラス インスツルメンテーション マネージャ

以下のプロパティは、ブートストラップ クラス インスツルメンテーション マネージャを設定します。

- introscope.bootstrapClassesManager.enabled（P. 300）
- introscope.bootstrapClassesManager.waitAtStartup（P. 301）

ブートストラップ クラス インスツルメンテーション マネージャは、エージェントのブートストラップの後に、一連のクラスをインスツルメントします。これにより、エージェントのパフォーマンスが改善されるほか、Java NIO および Secure Sockets Layer（SSL）用のトレーサを簡単に実装できます。*IntroscopeAgent.profile* でこのプロパティをコメント化すると、このプロパティを無効にすることができます。

introscope.bootstrapClassesManager.enabled

ブートストラップ マネージャを有効または無効にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.bootstrapClassesManager.enabled=true
```

注

- このプロパティは、Java 1.5 以降を実行している JVM のみで機能します。
- false に設定されている場合、システム クラスはインスツルメントされません。
- このプロパティが設定されていない場合、デフォルト値は false になります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.bootstrapClassesManager.waitAtStartup

ブートストラップ クラスをインスツルメントするために、エージェントが起動後に待機する時間を秒単位で設定します。

プロパティ設定

時間（秒）

デフォルト

- HP-UX、Interstage、WebLogic、または WebSphere アプリケーション サーバを使用している場合は 240 秒。
- JBoss、Oracle、Sun、または Tomcat を使用している場合は 5 秒。

例

```
introscope.bootstrapClassesManager.waitAtStartup=5
```

注

- このプロパティは、Java 1.5 以降を実行している JVM のみで機能します。
- このプロパティが有効である場合、スキップ済みとして指定されたクラスを解除できます。スキップ済みのクラスがインスツルメントされている場合は、CA Technologies の担当者、または CA サポート に連絡してください。

CA CEM エージェント プロファイル プロパティ

CA CEM 関連の IntroscopeAgent.profile プロパティは設定することができます。 CA Introscope® エージェント プロファイル ファイルは *<Agent_Home>*¥wily ディレクトリ内にあります。

すべての CA CEM 関連プロパティは、CA CEM および CA Introscope® との統合が機能するために必要なオプションにあらかじめ設定されています。

introscope.autoprobe.directivesFile

directivesFile プロパティ設定によって ServletHeaderDecorator / HTTPHeaderDecorator および CEMTracer を有効にする必要があります。

ディレクティブ ファイル プロパティは、AutoProbe のディレクティブ ファイル（PBD）またはディレクティブ リスト（PBL）を検索する場所を指定します。

AutoProbe は、ディレクティブを使用することで、ユーザ アプリケーションを Introscope に対応させ、エージェントが Enterprise Manager にレポートするメトリックを指定します。

設定

インストールされたエージェント アプリケーション サーバに応じて、<*appserver*>-full.pbl または <*appserver*>-typical.pbl の形式を使用します。

デフォルト

```
default-typical.pbl
```

例

```
introscope.autoprobe.directivesFile=weblogic-typical.pbl
```

注

このプロパティ リストの最後に「ServletHeaderDecorator.pbd」または「httpheaderdecorator.pbd」を単純に追加することもできますが、以下の操作の方がより推奨されます。

1. プロパティ内で指定された PBL ファイルを見つけます（上記の例で weblogic-typical.pbl）。
2. PBL ファイルをテキスト エディタで開きます。
3. Java Agent の場合は、ServletHeaderDecorator.pbd 行のコメント化を解除して有効にします。
4. .NET Agent の場合は、httpheaderdecorator.pbd 行のコメント化を解除して有効にします。
5. PBL ファイルへの変更を保存します。

introscope.agent.remoteagentconfiguration.allowedFiles

このプロパティは、任意のマシンからエージェント ディレクトリにリモートでコピーできるファイルを識別します。

Enterprise Manager はこのプロパティ内のファイル名を使用して、エージェントに送信する有効な CA CEM ドメイン構成ファイルを識別します。ドメイン構成ファイルには CA CEM ビジネス サービスおよびトランザクション定義が含まれます。

設定

有効なファイル名を使用します。

デフォルト

domainconfig.xml

例

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

注

このプロパティは、Introscope コマンド ライン Workstation（CLW）の send config file コマンドにも適用されます。

詳細については、「コマンドライン Workstation の使用」を参照してください。

このプロパティは CA CEM リリースで有効です。

introscope.agent.remoteagentconfiguration.enabled

このブール値が true に設定されている場合、別のコンピュータからエージェントへのリモート ファイル コピーが可能です。

Enterprise Manager では、CA CEM ドメイン構成ファイルをエージェントに送信するために、このプロパティを true に設定する必要があります。 ドメイン構成ファイルには CA CEM ビジネス サービスおよびトランザクション定義が含まれます。

設定

true または false

デフォルト

- Java エージェントの場合は true
- .NET エージェントの場合は false

例

```
introscope.agent.remoteagentconfiguration.enabled=true
```

注

リモート ユーザは、CA Introscope® コマンド ライン Workstation（CLW）の send config file コマンドを使用して、introscope.agent.remoteagentconfiguration.allowedFiles プロパティで指定されたファイル（複数可）をエージェント ディレクトリにコピーすることもできます。

このプロパティは CA CEM 4.0 と 4.1 リリースで有効です。このプロパティは、［Introscope 設定］ページ上の［CEMTracer 4.0 / 4.1 サポート］オプションを選択した場合にのみ、CA CEM 4.2 / 4.5 でも有効です。

［CEMTracer 4.0 / 4.1 サポート］オプションを使用すると、一定時間をかけて 4.0 または 4.1 から 4.2 / 4.5 にエージェントを徐々に移行することができます。 このオプションは必要な場合にのみ使用してください。

互換性がないエージェント（サポートされていない .NET エージェント、EPA エージェント、またはその他の非 Java エージェント）の場合、introscope.agent.remoteagentconfiguration.enabled プロパティを false に設定します。

introscope.agent.decorator.enabled

このブール値が true に設定されている場合、エージェントは HTTP 応答ヘッダに追加のパフォーマンス モニタリング情報を追加するように設定されます。 ServletHeaderDecorator / HTTPHeaderDecorator は、各トランザクションに GUID を付与し、その GUID を HTTP ヘッダ、x-apm-info に挿入します。

これによって、CA CEM と CA Introscope® の間のトランザクションの相関関係付けが可能になります。

設定

true または false

デフォルト

- Java エージェントの場合は false
- .NET Agent の場合は true

例

```
introscope.agent.decorator.enabled=false
```

introscope.agent.decorator.security

このプロパティは、CA CEM に送信される修飾済みの HTTP 応答ヘッダの形式を決定します。

設定

- clear： クリア テキスト エンコーディング
- encrypted： ヘッダ データが暗号化されます

デフォルト

clear

例

```
introscope.agent.decorator.security=clear
```

注

デフォルト設定の clear は、初期テストに適しています。 ただし、この設定によって、ファイアウォールの外部に知られたくないトランザクション ヘッダ内の情報が開示される可能性があります。 このプロパティを encrypted に設定すと、実稼働環境のセキュリティが強化されます。

このプロパティを encrypted に設定するには、サポートされている JVM を使用します。

注: JVM サポート情報については、「*Compatibility Guide*」を参照してください。

introscope.agent.cemtracer.domainconfigfile

CA CEM ビジネス サービスおよびトランザクション階層を指定する CA CEM ドメイン構成ファイルの名前。 CEMTracer は、インストール ディレクトリ内でこの名前のファイルを探します。

CA CEM 管理者が CA CEM で[すべての監視を同期]をクリックするごとに、ドメイン構成ファイルが Enterprise Manager にプッシュされ、次に接続された各エージェントへファイルがプッシュされます。

CA CEM のリリース固有情報については、以下の「**注**」を参照してください。

設定

任意の有効なファイル名を指定できます。

デフォルト

`domainconfig.xml`

例

```
introscope.agent.cemtracer.domainconfigfile=domainconfig.xml
```

注

このプロパティは CA CEM 4.0 と 4.1 リリースで有効です。 ［Introscope 設定］ページ上の［CEMTracer 4.0 / 4.1 サポート］オプションを選択した場合にのみ、CA CEM 4.2 / 4.5 でも有効です。

［CEMTracer 4.0 / 4.1 サポート］オプションを使用すると、一定時間をかけて 4.0 または 4.1 から 4.2 / 4.5 にエージェントを徐々に移行することができますが、これは必要な場合にのみ使用してください。

- エージェントが Enterprise Manager に接続されていない場合、ドメイン構成ファイルは送信されません。

- エージェント ディレクトリが読み取り専用である場合、ドメイン構成ファイルを書き込むことはできません。
- CEMTracer 4.0 / 4.1 がエージェント上で有効になっていない場合、いったん送信されたドメイン構成ファイルに対しては何の操作も行われません。

introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes

エージェントがドメイン構成ファイルを再ロードする頻度（分単位）。（4.0 / 4.1 のエージェントは、ドメイン構成ファイルを Enterprise Manager が送信するごとに自動的に再ロードするわけではありません。 変更がなければ、エージェントは再ロードしません）。

CA CEM のリリース固有情報については、以下の「**注**」を参照してください。

設定

数値

デフォルト

1

例

```
introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes=1
```

注

このプロパティは CA CEM 4.0 と 4.1 リリースで有効です。 ［Introscope 設定］ページ上の［CEMTracer 4.0 / 4.1 サポート］オプションを選択した場合にのみ、CA CEM 4.2 / 4.5 でも有効です。

［CEMTracer 4.0 / 4.1 サポート］オプションを使用すると、一定時間をかけて 4.0 または 4.1 から 4.2 にエージェントを徐々に移行することができますが、これは必要な場合にのみ使用してください。

introscope.agent.distribution.statistics.components.pattern

BlamePointTracer から応答時間の配布情報を収集するには、このプロパティのコメント化を解除して編集します。 この応答時間情報は、Average Response Time （平均応答時間）メトリックの作成に使用できます。

詳細情報

分布統計メトリックを収集するようにエージェントを設定する方法 (P. 87)

セッション ID の収集の設定

introscope.agent.transactiontracer.parameter.capture.sessionid プロパティは、Transaction Tracer データ内のセッション ID の収集を有効または無効にします。 デフォルトでは、このプロパティは有効で、Transaction Tracer データ内に記録されます。 このプロパティを無効にすると、フィルタにデータを使用できません。

以下の手順に従います。

1. IntroscopeAgent.profile ファイルをテキスト エディタで開きます。

   以下の行を探します。

   ```
   # 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. 指示に従って行をコメント化またはコメント化解除することで、プロパティを有効または無効にします。

   ```
   # introscope.agent.transactiontracer.parameter.capture.sessionid=true
   ```

3. ファイルを保存して閉じ、エージェントを再起動します。

   エージェント設定では、セッション ID の収集について指定した値が使用されます。

ChangeDetector の設定

ローカル エージェントと ChangeDetector との連携方法は制御することができます。

注: ChangeDetector の使用方法の詳細については、「CA APM ChangeDetector ユーザ ガイド」を参照してください。

introscope.changeDetector.enable

ChangeDetector を有効または無効に指定します。ChangeDetector を有効にするには、プロパティを true に設定します。 デフォルトではコメント化され、false に設定されます。 ChangeDetector を有効にする場合、追加の ChangeDetector 関連プロパティを設定する必要があります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.changeDetector.enable=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.changeDetector.agentID

ローカル エージェントを識別するために ChangeDetector によって使用されるテキスト文字列を指定します。 このプロパティは、デフォルトでコメント化されています。 ChangeDetector を有効にする場合は、このプロパティのコメント化を解除して適切な値を設定してください。

デフォルト

デフォルト値は SampleApplicationName です。

例

```
introscope.changeDetector.agentID=SampleApplicationName
```

introscope.changeDetector.rootDir

ChangeDetector ファイルのルート ディレクトリを指定します。 ルート ディレクトリは、ChangeDetector がローカル キャッシュ ファイルを作成するフォルダです。

プロパティ設定

ChangeDetector ファイルのルート ディレクトリへのフル パス（テキスト文字列）。

デフォルト

デフォルト パスは *c://sw//AppServer//wily//change_detector* です。

例

```
introscope.changeDetector.rootDir=c://sw//AppServer//wily//change_detector
```

注

円記号をエスケープするには、この例のように円記号を使用します。

introscope.changeDetector.isengardStartupWaitTimeInSec

エージェントが起動してから ChangeDetector が Enterprise Manager への接続を試行するまでの待機する秒数を設定します。 このプロパティは、デフォルトでコメント化されています。

デフォルト

デフォルトは 15 秒です。

例

```
introscope.changeDetector.isengardStartupWaitTimeInSec=15
```

introscope.changeDetector.waitTimeBetweenReconnectInSec

Enterprise Manager への接続を再試行する前に、ChangeDetector が待機する秒数を指定します。 このプロパティは、デフォルトでコメント化されています。

デフォルト

デフォルトは 10 秒です。

例

```
introscope.changeDetector.waitTimeBetweenReconnectInSec=10
```

introscope.changeDetector.profile

ChangeDetector データソース設定ファイルへの絶対パスまたは相対パスを設定します。 このプロパティは、デフォルトでコメント化されています。

デフォルト

デフォルトは ChangeDetector-config.xml です。

例

```
introscope.changeDetector.profile=CDConfig¥¥ChangeDetector-config.xml
```

注

円記号をエスケープするには、この例のように円記号を使用します。

introscope.changeDetector.profileDir

データ ソース設定ファイルを含むディレクトリへの絶対パスまたは相対パスを指定します。 このプロパティが設定される場合、このディレクトリ内のデータ ソース設定ファイルはすべて *introscope.changeDetector.profile* プロパティによって指定された任意のファイルに加えて使用されます。 このプロパティは、デフォルトでコメント化されています。

デフォルト

デフォルトは changeDetector_profiles です。

例

```
introscope.changeDetector.profileDir=c:¥¥CDconfig¥¥changeDetector_profiles
```

注

円記号をエスケープするには円記号を使用します。

introscope.changeDetector.compressEntries.enable

ChangeDetector データ バッファの圧縮を許可するかどうかを指定します。パフォーマンスを向上させるため、起動時にメモリ消費が発生する場合は、このプロパティを true に設定できます。

プロパティ設定

True または False

デフォルト

プロパティがエージェント プロファイルで設定されていない場合、またはコメント化されている場合、デフォルト値は false です。

例

```
introscope.changeDetector.compressEntries.enable=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.changeDetector.compressEntries.batchSize

このプロパティには、前述の introscope.changeDetector.compressEntries.enable で設定された圧縮ジョブのバッチ サイズを定義します。

デフォルト

1000

例

```
introscope.changeDetector.compressEntries.batchSize=1000
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

WebLogic Server でのプロセス間トレースの有効化

以下のプロパティは、WebLogic Server でのプロセスにまたがる追跡を設定します。

- introscope.agent.weblogic.crossjvm（P. 313）

introscope.agent.weblogic.crossjvm

プロパティ設定

True または False

デフォルト

True （コメント化されています）

例

```
introscope.agent.weblogic.crossjvm=true
```

プロセスにまたがるトランザクション追跡

以下のプロパティのコメント化を解除して、末尾フィルタのためにダウンストリームの追跡の自動収集を有効にします。

- introscope.agent.transactiontracer.tailfilterPropagate.enable（P. 314）

このプロパティを有効にして、かつ末尾フィルタを使用してトランザクション追跡を長時間実行していると、不要な追跡が多数発生したまま Enterprise Manager に送信される可能性があります。

introscope.agent.transactiontracer.tailfilterPropagate.enable

末尾フィルタが存在するために、ダウンストリーム エージェントからの追跡の自動収集がトリガされるかどうかを制御します。 このプロパティは、先頭フィルタの使用によるダウンストリームの追跡の自動収集には影響しません。

プロパティ設定

True または False

デフォルト

False、コメント化されています。

例

```
introscope.agent.transactiontracer.tailfilterPropagate.enable=false
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

動的インスツルメンテーション

カスタム PBD の作成、アプリケーション サーバの再起動、またはエージェントの再起動を行うことなく、クラスとメソッドが動的にインスツルメントされるようにすることができます。

注: Introscope Workstation からトランザクション追跡および動的インスツルメンテーションを使用する方法の詳細については、「CA APM Workstation ユーザ ガイド」を参照してください。

introscope.autoprobe.dynamicinstrument.enabled

対象: JDK 1.5 で動作し、AutoProbe を使用するエージェント

このプロパティを使用すると、エージェントの動的 ProbeBuilding が有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.autoprobe.dynamicinstrument.enabled=false
```

詳細:

動的 ProbeBuilding (P. 94)

autoprobe.dynamicinstrument.pollIntervalMinutes

対象: JDK 1.5 で動作し、AutoProbe と動的 ProbeBuilding を使用するエージェント

このプロパティは、エージェントが新規および変更済みの PBD をポーリングする間隔を決定します。

デフォルト

1

例

```
autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs

いくつかのクラスローダの実装では非常に大きなクラス ファイルを返すように設定されている場合があります。これはメモリ エラーを防ぐためのものです。

デフォルト

1

例

```
introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo

一度に非常に多くのクラスを再定義すると、CPU に過大な負荷がかかる可能性があります。PBD の変更によって多数のクラスの再定義が発生する場合、これによってプロセスは適度なレートでバッチ処理されます。

デフォルト

10

例

```
introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10
```

introscope.agent.remoteagentdynamicinstrumentation.enabled

動的インスツルメンテーションのリモート管理を有効または無効にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.remoteagentdynamicinstrumentation.enabled=true
```

注

- このプロパティの変更を有効にするには、管理対象アプリケーションを再起動する必要があります。
- 動的インスツルメンテーションは、CPU への負荷が高い操作です。 インスツルメントされるクラスを最小限にする設定を使用します。

introscope.autoprobe.dynamicinstrument.pollIntervalMinutes

PBD の変更をポーリングするためのポーリング間隔（分単位）を定義します。

デフォルト

1

例

```
introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

ErrorDetector

エージェントと ErrorDetector との連携方法は制御することができます。

introscope.agent.errorsnapshots.enable

エージェントが、重大なエラーが発生したトランザクションの詳細をキャプチャできるようにします。 Introscope ErrorDetector は、エージェントと共にデフォルトでインストールされます。 エラー スナップショットを表示可能にするには、このプロパティを true に設定する必要があります。

プロパティ設定

True または False

デフォルト

True

注

これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.errorsnapshots.throttle

エージェントが 15 秒間に送信できるエラー スナップショットの最大数を指定します。

デフォルト

10

例

```
introscope.agent.errorsnapshots.throttle=10
```

注

これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.errorsnapshots.ignore.<index>

1 つ以上のエラー メッセージ フィルタを指定します。 フィルタは、プロパティ名に付加されるインデックス ID （例、.0、.1、.2 ...）を使用して、必要な数だけ作成できます。 ワイルドカード（*）を使用でき、指定した条件に一致するエラー メッセージは無視されます。 定義したフィルタに一致するエラーに対してスナップショットは生成されません。また、エラー イベントは該当する Enterprise Manager に送信されません。

重要: このプロパティは、SOAP エラー メッセージのフィルタには使用できません。

デフォルト

以下のように、定義の例が記述されており、コメント化されています。

例

```
introscope.agent.errorsnapshots.ignore.0=*com.company.HarmlessException*
introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code: 404*
```

注

これは動的プロパティです。 このプロパティの設定は実行時に変更することができ、変更は自動的に反映されます。

拡張機能

エージェントの拡張機能の場所を設定できます。

introscope.agent.extensions.directory

エージェントによってロードされるすべての拡張機能がある場所を指定します。 ディレクトリへの絶対パスまたは相対パスを指定できます。 絶対パスを指定しない場合、指定する値が *IntroscopeAgent.profiles* ファイルの場所を起点とする相対パスに解決されます。

デフォルト

デフォルトの場所は *<Agent_Home>*/ext ディレクトリの *ext* ディレクトリです。

例

```
introscope.agent.extensions.directory=../ext
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.agent.common.directory

エージェント拡張機能の関連ファイルの場所を設定します。

デフォルト

デフォルトの場所は <Agent_Home>/common ディレクトリ内の common フォルダです。

例

```
introscope.agent.common.directory=../../common
```

GC Monitor

GC 監視ノード下のメトリックは、ガベージ コレクタとメモリ プールに関する情報をレポートします。 このようなメトリックは、パフォーマンスに悪影響を及ぼしているメモリ関連の問題を検出するのに役立ちます。 メトリックの収集は、エージェント プロファイル内で手動で有効にする必要があります。

introscope.agent.gcmonitor.enable

このプロパティは、ガベージ コレクタとメモリ プールのメトリックを有効または無効にします。

プロパティ設定

True または False

デフォルト

デフォルト値は true です。

例

```
introscope.agent.gcmonitor.enable=true
```

注

これは動的プロパティです。 このプロパティの設定は実行時に変更することができ、変更は自動的に反映されます。

Sun または IBM JVM を監視するエージェントについて、GC 監視メトリックのみをレポートできます。

Java NIO

Java エージェントでは、Java New I/O （Java NIO、または NIO）の機能をサポートしています。Java NIO とは、近年のオペレーティング システムの低レベル I/O 操作にアクセスするために設計された API のコレクションです。Java NIO のメトリックでは、インスツルメントされたアプリケーションが Java NIO を使用する方法に関する情報を収集します。

注: CA Introscope® の Java NIO メトリックおよび Java NIO 情報のメトリック収集は、Java 1.5 以降の JVM でのみ利用可能です。

CA Introscope® Java エージェントは、NIO チャネル用のメトリックを収集します。

特定の NIO メトリックの生成を制限できます。

Java NIO トレーサ グループは、デフォルトでは有効になっています。これらのトレーサ グループをオフにして、メトリックの生成をさらに制限することができます。

詳細:

チャネル (P. 322)
Java NIO メトリックの制限 (P. 324)
デフォルトのトレーサ グループおよびトグル ファイル (P. 110)

チャネル

Java NIO チャネルは、外部システムに加えて NIO バッファへの一括データ転送を行います。 これは、標準的な Java I/O のパフォーマンスと拡張性の問題に対処するために特別に設計された、低レベルのデータ転送メカニズムです。

チャネルは、バッファと外部システムの間でバイトを移動するためのメカニズムを提供します。 Introscope チャネル メトリックの特長は、チャネルからのデータ フローの割合です。収集された NIO チャネル メトリックは、Java I/O の標準的なテクニックを使用して、ファイルおよびソケット I/O 用に現在作成されたメトリックに対応します。 以下のチャネル タイプのメトリックは別々に収集され、Workstation Investigator に表示されます。

- データグラム チャネル
- ソケット チャネル

NIODatagramTracing メトリック

UDP はコネクションレス プロトコルですが、「接続」という用語は、NIODatagramTracing の説明の中で、Java Agent がデータグラム メトリックを収集する方法を説明するために使用されます。Java Agent は、「送信先」または「送信元」であるリモートのエンドポイント データグラムのメトリックを別々に収集します。 接続は、「送信先」または「送信元」のそれぞれのエンドポイントから見た最初のデータグラムの方向によって、クライアントまたはサーバに分類されます。

たとえば、最初のデータグラムがエンドポイントから「着信」した場合、Java Agent はそのエンドポイントに対するそれ以降のデータグラムは、すべて *NIO|Channels|Datagrams|Server|Port {PORT}* の下に分類されます。ここで、*{PORT}* はローカル ポートを表します。

最初のデータグラムがエンドポイントへの「発信」である場合、そのエンドポイントに対するそれ以降のデータグラムは、すべて *NIO|Channels|Datagrams|Client|{HOST}|Port {PORT}* に分類されます。ここで、*{HOST}* および *{PORT}* はリモートのエンドポイントを表します。

また、Java Agent は、データグラムが「受信」メソッドによって読み取られた場合を除き、クライアントの「接続」用のバックエンド メトリックを生成します。 DatagramChannel の connect メソッドを使用して作成されたデータグラム チャネルは、監視対象の最初のデータグラムの方向にかかわらず、クライアント接続であるとみなされます。

注: UDP の接続を使用して（connect メソッドを使用して）作成されたデータグラム チャネルは、クライアント接続であるとみなされます。

Java NIO メトリックの制限

Java NIO インスツルメンテーションの動作方法を制御するためにプロパティを設定できます。 データグラムおよびソケット メトリックの生成を制限することもできます。 これらのプロパティは、NIOSocketTracing および NIODatagramTracing トレーサ グループによって生成された詳細メトリックのみに影響します。

注: トレーサ グループの詳細については、「デフォルトのトレーサ グループおよびトグル ファイル (P. 110)」を参照してください。

introscope.agent.nio.datagram.client.hosts

指定されたホストで「クライアント」の UDP 「接続」にレポートするメトリックを制限します。

プロパティ設定

カンマ区切りのホストのリスト。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.datagram.client.hosts=hostA,hostB
```

注

- リストを空のままにすると、ホストの制限は適用されません。
- ホストは、名前または IP アドレスのテキスト表現（IPv4 または IPv6 形式のいずれ）を使用して指定できます。
- 無効なホスト名はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。
- 重複したホスト名は無視されます。複数のホスト名が 1 つの IP にマップされている場合、1 つの名前のみが保持されます。 ただし、プロパティは、同じ意味の名前のいずれかのセットとクライアント接続を照合します。

introscope.agent.nio.datagram.client.ports

NIO メトリックをレポートするポートを一覧表示します。指定されたポートの「クライアント」データグラム メトリックのみが生成されます。

プロパティ設定

カンマ区切りのポート番号のリスト。 *ポート*は、データグラムが送受信されるリモートのポートです。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.datagram.client.ports=123,456,789
```

注

- リストを空のままにすると、ポートの制限は適用されません。
- 無効なポート番号はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。
- 重複したポートは無視されます。複数のポートが 1 つの IP にマップされている場合、1 つのポートのみが保持されます。

introscope.agent.nio.datagram.server.ports

NIO メトリックをレポートするポートを一覧表示します。指定されたポートの「サーバ」データグラム メトリックのみが生成されます。

プロパティ設定

カンマ区切りの*ポート*番号のリスト。 *ポート*はデータグラムが送受信されるローカル ポートです。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.datagram.server.ports=123,456,789
```

注

- リストを空のままにすると、ポートの制限は適用されません。
- 無効なポート番号はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.nio.socket.client.hosts

指定されたホストで「クライアント」の TCP 「接続」にレポートするメトリックを制限します。

プロパティ設定

カンマ区切りのホストのリスト。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.socket.client.hosts=hostA, hostB
```

注

- リストが空の場合、ホストの制限は適用されません。
- ホストは、名前または IP アドレスのテキスト表現（IPv4 または IPv6 形式のいずれ）を使用して指定できます。
- 無効なホスト名はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.nio.socket.client.ports

NIO メトリックをレポートするポートを一覧表示します。指定されたポートの「クライアント」ソケット メトリックのみが生成されます。

プロパティ設定

カンマ区切りのポート番号のリスト。 *ポート*は、データグラムが送受信されるリモートのポートです。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.socket.client.ports=123,456,789
```

注

- リストが空の場合、ポートの制限は適用されません。
- 無効なポート番号はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.nio.socket.server.ports

NIO メトリックをレポートするポートを一覧表示します。指定されたポートの「サーバ」ソケット メトリックのみが生成されます。

プロパティ設定

カンマ区切りの*ポート*番号のリスト。 *ポート*はデータグラムが送受信されるローカル ポートです。

デフォルト

未定義（値はありません）。

例

```
introscope.agent.nio.socket.client.ports=123,456,789
```

注

- リストが空の場合、ポートの制限は適用されません。
- 無効なポート番号はエージェント ログにレポートされ、無視されます。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

Java NIO メトリックは、Workstation Investigator のエージェントのトップレベル ノードの下にある［NIO］ノードの下に表示されます。「クライアント」接続用の追加の NIO メトリックは、［バックエンド］ノードの下に表示されます。

個々の NIO メトリックは、メトリックを表示しないように、*TraceOneMethodIfFlagged* または *TraceOneMethodWithParametersIfFlagged* ディレクティブをコメント化して抑制されている場合があります。 ただし、名前が *BackendTracer* または *MappingTracer* で終わるトレーサをコメント化する必要はありません。

たとえば、データグラムの並行処理リーダ メトリックを抑制するには、以下の行をコメント化します。

```
TraceOneMethodWithParametersIfFlagged: NIODatagramTracing read
NIODatagramConcurrentInvocationCounter "Concurrent Readers"
```

JMX

以下のプロパティは、JMX メトリックを設定します。

- introscope.agent.jmx.enable（P. 330）
- introscope.agent.jmx.ignore.attributes（P. 331）
- introscope.agent.jmx.name.filter（P. 332）
- introscope.agent.jmx.name.jsr77.disable（P. 333）
- introscope.agent.jmx.name.primarykeys（P. 333）
- introscope.agent.jmx.excludeStringMetrics（P. 335）

introscope.agent.jmx.enable

JMX 関連メトリックの収集を有効にするかどうかを指定します。

プロパティ設定

true または false

デフォルト

エージェントのバージョンによって異なります。

例

```
introscope.agent.jmx.enable=false
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.agent.jmx.ignore.attributes

無視する必要がある JMX MBean 属性がある場合に、その属性を定義します。

プロパティ設定

カンマ区切りのキーワードのリスト。

デフォルト

server（コメント化されています）

例

```
introscope.agent.jmx.ignore.attributes=server
```

注

- MBean 属性の名前がこのリストに含まれる語に一致した場合、その属性は無視されます。
- どの MBean 属性も無視しない場合は、このリストを空にしてください。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.jmx.name.filter

Introscope によって収集および表示される JMX データの種類を決定するための、フィルタ文字列のカンマ区切りのリストを指定します。

Introscope は、フィルタ文字列に一致する JMX 生成メトリックをレポートします。フィルタ文字列では、アスタリスク（*）および疑問符（?）のワイルドカード文字を使用できます。

- * は、ゼロまたは複数の文字に該当します。
- ? は、1 つの文字に該当します。

文字列として使用されている * または ? に一致させるには、¥¥ を共に使用してエスケープします。

例：

- 「*ab¥¥*c*」は、「*ab*c*」という文字を含むメトリック名が該当します。
- 「*ab*c*」は、「*abc*」、「*abxc*」、「*abxxc*」などの文字を含むメトリック名が該当します。
- 「*ab?c*」は、「*abxc*」という文字を含むメトリック名が該当します。
- 「*ab¥¥?c*」は、「*ab?c*」という文字を含むメトリック名が該当します。

デフォルト

コメント化されています。

WebLogic 用：

```
ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType
```

例

```
#introscope.agent.jmx.name.filter=ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType
```

注

- システムで利用可能なすべての MBean データを含めるには、このリストを空にしてください。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.jmx.name.jsr77.disable

このプロパティは、Introscope が複雑な JMX データを含む完全な JSR77 データを収集およびレポートするかどうかを制御します。

このプロパティは、WebLogic および WebSphere の *IntroscopeAgent.profile* ファイルで使用する場合にのみ、利用可能です。

プロパティ設定

True または False

デフォルト

True

注

- このプロパティを有効にするには、アプリケーション サーバで JSR-77 管理機能をサポートする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.jmx.name.primarykeys

MBean 情報のユーザ定義の順序。この順序を指定すると名前変換が簡素化されます。

プロパティ設定

カンマ区切りのキーの順序付けリスト。キーは、特定の MBean を一意に識別する必要があります。

デフォルト

デフォルトの *IntroscopeAgent.profile* ファイル（コメント化されています）。

例

```
introscope.agent.jmx.name.primarykeys=J2EEServer
```

注

- WebLogic 用のプロパティ設定
  - Type
  - Name
- WebLogic Server 9.0 を使用している場合、このプロパティはコメント化されます。
- WebSphere 用のプロパティ設定
  - J2EEServer
  - Application
  - j2eeType
  - JDBCProvider
  - name
  - mbeanIdentifier
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.jmx.excludeStringMetrics

文字列の値を持つメトリックを含めるかどうかを制御します。 文字列の値を持つメトリックを有効にするには、このプロパティを false にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.jmx.excludeStringMetrics=true
```

注

- 文字列の値を持つメトリックを除外すると、全体的なメトリック数が減り、エージェントおよび EM のパフォーマンスが改善されます。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

LeakHunter

以下のプロパティは、エージェントの LeakHunter とのやり取りを設定します。

- introscope.agent.leakhunter.collectAllocationStackTraces（P. 336）
- introscope.agent.leakhunter.enable（P. 337）
- introscope.agent.leakhunter.leakSensitivity（P. 338）
- introscope.agent.leakhunter.logfile.append（P. 339）
- introscope.agent.leakhunter.logfile.location（P. 340）
- introscope.agent.leakhunter.timeoutInMinutes（P. 340）

introscope.agent.leakhunter.collectAllocationStackTraces

LeakHunter が潜在リークに対して割り当てスタック トレースを生成するかどうかを制御します。 このプロパティを *true* に設定すると、潜在リークの割り当てに関するより詳細なデータを得ることができますが、メモリの追加が必要で、CPU のオーバーヘッドも大きくなります。 このため、デフォルトは *false* です。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.leakhunter.collectAllocationStackTraces=
false
```

注

- このプロパティを *true* に設定すると、CPU 使用率とメモリでシステム オーバーヘッドが増加する可能性があります。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.leakhunter.enable

LeakHunter の有効、無効を設定します。 LeakHunter を有効にするには、値を true に設定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.leakhunter.enable=false
```

注

- このオプションをオンにすると、CPU の使用率とメモリ使用量が増加する可能性があります。他のメトリックによってメモリ リークが示された場合にのみ、この機能を有効にしてください。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.leakhunter.leakSensitivity

LeakHunter リーク検出アルゴリズムの感度レベルを制御します。感度を高く設定するとレポートされる潜在リークの数が増え、感度を低く設定するとレポートされる潜在リークの数が減ります。

プロパティ設定

リーク感度の範囲は、1 （低） ～ 10 （高）の範囲の正の整数値である必要があります。

デフォルト

5

例

```
introscope.agent.leakhunter.leakSensitivity=5
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.agent.leakhunter.logfile.append

アプリケーションの再起動時に既存のログ ファイルを新しいファイルで置き換えるか、または既存のログ ファイルにログを追記するかを指定します。

プロパティ設定

True または False

- False はログ ファイルを置き換えます。
- True は既存のログ ファイルにログ情報を追記します。

デフォルト

False

例

```
introscope.agent.leakhunter.logfile.append=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.leakhunter.logfile.location

LeakHunter.log ファイルの場所を指定します。 ファイル名には絶対パスまたは相対パスが使用できます。相対パスは <Agent_Home> ディレクトリを基準に解釈されます。 LeakHunter でログ ファイルにデータ記録しない場合は、値は空白にするかコメントアウトしてください。

デフォルト

デフォルト パスは *../../logs/LeakHunter.log* です。これは、<Agent_Home>logs ディレクトリのログ ファイルを示しています。

例

```
introscope.agent.leakhunter.logfile.location=../../logs/LeakHunter.log
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.leakhunter.timeoutInMinutes

LeakHunter が新しい潜在リークの検出に費やす時間の長さ（分単位）を制御します。指定した時間が経過すると、LeakHunter は新しい潜在リークの検索を停止します。 以前に特定した潜在リークを継続して追跡します。

プロパティ設定

正の整数である必要があります（負の数値は指定できません）。

デフォルト

デフォルトは 120 分です。

例

```
introscope.agent.leakhunter.timeoutInMinutes=120
```

注

- LeakHunter に常に潜在リークを検出させたい場合は、値にゼロを設定します。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.leakhunter.ignore.<number>

指定されたパターンと一致するクラスを無視するよう指定します。 デフォルトでは 10 個のクラスが提供されています。使用するクラスをコメント化します。

プロパティ設定

クラス一致パターンのカンマ区切りリスト

デフォルト

なし

例

```
introscope.agent.leakhunter.ignore.4=java.util.SubList
introscope.agent.leakhunter.ignore.5=com.sun.faces.context.BaseContextMap$EntrySet
introscope.agent.leakhunter.ignore.6=com.sun.faces.context.BaseContextMap$Key
```

注

- 一部のコレクションは LeakHunter と併用できません。コレクションを LeakHunter セーフにするためには、任意のスレッドから、いつでも *size()* を呼び出せるようにしておくのが安全です。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。
- ワイルドカード文字「*」を使用できます。

ログ

以下のプロパティは、エージェントのログ オプションを設定します。

- log4j.logger.IntroscopeAgent（P. 343）
- log4j.appender.logfile.File（P. 344）
- log4j.logger.IntroscopeAgent.inheritance（P. 344）
- log4j.appender.pbdlog.File（P. 345）

log4j.logger.IntroscopeAgent

このプロパティは、ログ情報のログ レベルと出力場所の両方を制御します。

プロパティ設定

指定できる値の詳細度を以下に示します。

- *INFO*
- *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel*

指定できる出力先の値は以下のとおりです。

- *console*
- *logfile*
- *console* と *logfile* の両方

デフォルト

INFO, console, logfile

例

```
log4j.logger.IntroscopeAgent=INFO,console,logfile
```

エージェントのログを無効にするには、このプロパティから以下のようにオプションを削除します。

変更前

```
log4j.logger.IntroscopeAgent=INFO, console, logfile
```

変更後

```
log4j.logger.IntroscopeAgent=
```

注

- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

log4j.appender.logfile.File

ログ ファイルが log4j.logger.IntroscopeAgent で指定されている場合、IntroscopeAgent.log ファイルの名前と場所を指定します。 ファイル名は、エージェント プロファイルを含むディレクトリを基準にした相対名です。

デフォルト

IntroscopeAgent.log

例

```
log4j.appender.logfile.File=../../logs/IntroscopeAgent.log
```

注

システム プロパティ（Java コマンド ライン -D オプション）は、ファイル名の一部として展開されます。 たとえば、Java が *-Dmy.property=Server1* で起動される場合、*log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* は、*log4j.appender.logfile.File=../../logs/Introscope-Server1.log* に展開されます。

log4j.logger.IntroscopeAgent.inheritance

インスツルメンテーションが必要なクラスについてのログ メッセージのログ レベルおよび出力先を制御します。

プロパティ設定

スーパータイプまたはインターフェースを拡張するのでインスツルメントされていないクラスのログを制御するには、このプロパティを *INFO, pbdlog* に設定します。

継承クラスのログ記録については、「ディレクティブのログの制御 (P. 102)」を参照してください。

デフォルト

なし

例

```
log4j.logger.IntroscopeAgent.inheritance=INFO,pbdlog
```

log4j.appender.pbdlog.File

インスツルメンテーションが必要なクラスについてのメッセージのログファイルを指定します。

プロパティ設定

スーパータイプまたはインターフェースを拡張するためインスツルメントされていないクラスのログを制御するには、*pbdupdate.log* に設定します。

デフォルト

なし

例

```
log4j.appender.pbdlog.File=../../pbdupdate.log
```

log4j.appender.pbdlog

インスツルメンテーションが必要なクラスについてのメッセージのログのパッケージを指定します。

プロパティ設定

スーパータイプまたはインターフェースを拡張するためインスツルメントされていないクラスのログを制御するには、このプロパティを *com.wily.introscope.agent.AutoNamingRollingFileAppender* に設定します。

デフォルト

なし

例

```
log4j.appender.pbdlog=com.wily.introscope.agent.AutoNamingRollingFileAppender
```

log4j.appender.pbdlog.layout

インスツルメンテーションが必要なクラスについてのメッセージのログの規則を指定します。

プロパティ設定

スーパータイプまたはインターフェースを拡張するためインスツルメントされていないクラスのログを制御するには、このプロパティを *com.wily.org.apache.log4j.PatternLayout* に設定します。

デフォルト

なし

例

```
log4j.appender.pbdlog.layout=com.wily.org.apache.log4j.PatternLayout
```

log4j.appender.pbdlog.layout.ConversionPattern

インスツルメンテーションが必要なクラスについてのメッセージのログの規則を指定します。

プロパティ設定

スーパータイプまたはインターフェースを拡張するのでインスツルメントされていないクラスのログを制御するには、このプロパティを以下に設定します。

```
%d{M/dd/yy hh:mm:ss a z} [%-3p] [%c] %m%n
```

デフォルト

なし

例

```
log4j.appender.pbdlog.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z} [%-3p]
[%c] %m%n
```

log4j.additivity.IntroscopeAgent.inheritance

複数のレベルの継承のディレクティブのログが *pbdupdate.log* ファイルのみに記録されるようにします。

プロパティ設定

True または False

デフォルト

True

例

```
log4j.additivity.IntroscopeAgent.inheritance=true
```

注

複数のレベルの継承のディレクティブのログが *pbdupdate.log* のみに記録されるように設定するには、このプロパティをエージェント プロファイルに追加して、*false* に設定します。

メトリック カウント

以下のプロパティは、Investigator でメトリック数メトリックが表示される場所を制御します。

- introscope.ext.agent.metric.count（P. 348）

introscope.ext.agent.metric.count

Investigator でメトリック数メトリックを表示する場所を制御します。 デフォルトでは、 *[Custom Metric Agent]* ノードの下に *[メトリック数]* が表示されます。 *[Agent Stats]* ノードの下に *[Metric Count]* メトリックを表示する場合は、このプロパティを *IntroscopeAgent.profile* に追加します。

プロパティ設定

True または False

デフォルト

False（*IntroscopeAgent.profile* には存在しません）

例

```
introscope.ext.agent.metric.count=true
```

注

［Agent Stats］ノードの下に［Metric Count］メトリックを表示するには、このプロパティを *IntroscopeAgent.profile* に追加し、true に設定します。

複数の継承

インターフェースまたはスーパー クラスに基づいたディレクティブの場合、エージェントは複数の継承を検出できません。そのため、複数を継承したクラスはインスツルメントされません。 アプリケーション サーバまたはエージェント プロセスの起動後にこれらのクラスを見つけるには、以下のプロパティを有効にします。 これらのプロパティによって、インスツルメントしなければならないがまだ実行されず、変更を反映させるのに動的インスツルメンテーションに依存しているクラスがログに記録されます。

- introscope.autoprobe.hierarchysupport.enabled（P. 349）
- introscope.autoprobe.hierarchysupport.runOnceOnly（P. 350）
- introscope.autoprobe.hierarchysupport.pollIntervalMinutes（P. 351）
- introscope.autoprobe.hierarchysupport.executionCount（P. 351）
- introscope.autoprobe.hierarchysupport.disableLogging（P. 352）
- introscope.autoprobe.hierarchysupport.disableDirectivesChange（P. 352）

introscope.autoprobe.hierarchysupport.enabled

JDK 1.5 で動作し、AutoProbe および動的インスツルメンテーションを使用するエージェントの場合、このプロパティを使用して、スーパータイプまたはインターフェースを拡張するクラスのインスツルメンテーションを有効にできます。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.autoprobe.hierarchysupport.enabled=true
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.autoprobe.hierarchysupport.runOnceOnly

スーパータイプまたはインターフェースを拡張するクラスのインスツルメンテーションを有効にしている場合は、このプロパティを使用して、その機能を有効にするユーティリティを 1 回だけ実行するか、または指定された間隔で実行するかを制御できます。

定期的に検出する必要がある場合にのみ、このプロパティを **true** に変更します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.autoprobe.hierarchysupport.enabled=false
```

注

- 動的インスツルメンテーションに関連するログ プロパティは、Logging で定義します。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.autoprobe.hierarchysupport.pollIntervalMinutes

複数の継承のためにインスツルメントできなかったクラスをチェックするポーリング間隔です。ほとんどの場合、ポーリングは 1 回のみ行われます。ただし、アプリケーション サーバの初期化を行う場合は、控えめな値を設定することをお勧めします。

デフォルト

5

例

```
introscope.autoprobe.hierarchysupport.pollIntervalMinutes=5
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.autoprobe.hierarchysupport.executionCount

ポーリング間隔を 1 回のみ実行したり、定期的に実行するのではなく限定された回数だけ実行するようなポーリング間隔が必要な場合は、このプロパティを使用してポーリング間隔を実行する正確な回数を指定します。実行する正確な数を指定するには、常にこのプロパティを使用します。

このプロパティを使用すると、1 回のみ実行する設定は上書きされます。

プロパティ設定

正の整数。

デフォルト

3

例

```
introscope.autoprobe.hierarchysupport.executionCount=3
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.autoprobe.hierarchysupport.disableLogging

検出済みのクラスをログ記録する必要がない場合は、このプロパティのコメント化を解除します。このプロパティは、動的インスツルメンテーションが有効である場合にのみコメント化を解除します。

プロパティ設定

True または False

デフォルト

True

例

```
#introscope.autoprobe.hierarchysupport.disableLogging=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.autoprobe.hierarchysupport.disableDirectivesChange

変更のみをログに記録し、かつ動的インスツルメンテーションのトリガを無効にするには、このプロパティのコメント化を解除します。

プロパティ設定

True または False

デフォルト

True

例

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

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

プラットフォーム モニタリング

以下のプロパティは、プラットフォーム モニタリング メトリックを設定します。

- introscope.agent.platform.monitor.system（P. 353）

introscope.agent.platform.monitor.system

プラットフォーム モニタをロードするオペレーティング システムの名前。

プロパティ設定

このプロパティのオプションの詳細については、「プラットフォーム モニタリングのトラブルシューティング (P. 242)」を参照してください。

デフォルト

コメント化されています（プラットフォームにより異なります）。

例

```
introscope.agent.platform.monitor.system=Solaris
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

リモート設定

以下のプロパティは、Java Agent のリモート設定を許可します。

- introscope.agent.remoteagentconfiguration.enabled（P. 354）
- introscope.agent.remoteagentconfiguration.allowedFiles（P. 354）

introscope.agent.remoteagentconfiguration.enabled

このプロパティは、エージェントのリモート設定を有効または無効にします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.remoteagentconfiguration.enabled=true
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.remoteagentconfiguration.allowedFiles

このプロパティには、このエージェントにリモートから転送できるファイルの正確なリストを記載します。

プロパティ設定

domainconfig.xml

デフォルト

domainconfig.xml

例

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

セキュリティ

以下のプロパティは、CA CEM に送信される HTTP ヘッダのセキュリティを設定します。

- introscope.agent.decorator.security（P. 355）

introscope.agent.decorator.security

このプロパティは、CA CEM に送信される、修飾済みの HTTP 応答ヘッダの形式を決定します。

プロパティ設定

Clear： クリア テキスト エンコーディング

Encrypted： ヘッダ データが暗号化されます

デフォルト

Clear

例

```
introscope.agent.decorator.security=clear
```

サーブレット ヘッダ デコレータ

以下のプロパティは、CA CEM と Introscope 間のトランザクションの相関を有効にします。

- introscope.agent.decorator.enabled（P. 356）

introscope.agent.decorator.enabled

このブール値が true に設定されている場合、エージェントは HTTP 応答ヘッダに追加のパフォーマンス モニタリング情報を追加するように設定されます。 ServletHeaderDecorator は、各トランザクションに GUID を付与し、その GUID を x-apm-info などの HTTP ヘッダに挿入します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.decorator.enabled=false
```

「ソケット メトリック」

I/O ソケット メトリックの生成は、以下のパラメータによって制限される場合があります。

- introscope.agent.sockets.reportRateMetrics（P. 357）
- introscope.agent.io.socket.client.hosts（P. 357）
- introscope.agent.io.socket.client.ports（P. 358）
- introscope.agent.io.socket.server.ports（P. 358）

introscope.agent.sockets.reportRateMetrics

各ソケットの入出力（I/O）帯域幅レート メトリックのレポートを有効にします。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.sockets.reportRateMetrics=false
```

注

- ManagedSocketTracing が有効で、SocketTracing が無効である場合にのみ、機能します。 詳細については、「下位互換性（P. 166）」を参照してください。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.io.socket.client.hosts

指定されたリモート ホストに対してインスツルメントされるソケット クライアント接続を制限します。

プロパティ設定

カンマ区切りの値のリスト。

例

```
introscope.agent.io.socket.client.hosts=
```

注

- 個々の値が無効である場合、その値は無視されます。
- パラメータが定義されていない場合、または無効な値を除外するとリストが空になる場合、そのパラメータに制限は適用されません。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.io.socket.client.ports

指定されたリモート ポートに対してインスツルメントされるソケット クライアント接続を制限します。

プロパティ設定

カンマ区切りの値のリスト。

例

```
introscope.agent.io.socket.client.ports=
```

注

- 個々の値が無効である場合、その値は無視されます。
- パラメータが定義されていない場合、または無効な値を除外するとリストが空になる場合、そのパラメータに制限は適用されません。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.io.socket.server.ports

指定されたローカル ポートを使用しているものに対してインスツルメントされるソケット クライアント接続を制限します。

プロパティ設定

カンマ区切りの値のリスト。

例

```
introscope.agent.io.socket.server.ports=
```

注

- 個々の値が無効である場合、その値は無視されます。
- パラメータが定義されていない場合、または無効な値を除外するとリストが空になる場合、そのパラメータに制限は適用されません。
- これは動的プロパティです。実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

SQL エージェント

SQL エージェントのさまざまな要素は設定することができます。

詳細:

introscope.agent.sqlagent.normalizer.extension

このプロパティは、事前に設定済みの正規化スキーマより優先して使用される SQL ノーマライザ拡張機能の名前を指定します。

カスタムの正規化拡張機能を機能させるには、マニフェスト属性 *com-wily-Extension-Plugin-{pluginName}-Name* の値がこのプロパティに指定された値と一致する必要があります。

名前のカンマ区切りリストを指定した場合、エージェントはデフォルトのノーマライザ拡張機能を使用します。

たとえば、以下の設定では、RegexSqlNormalizer が正規化に使用されます。

```
introscope.agent.sqlagent.normalizer.extension=ext1, ext2
```

このプロパティは、SQL エージェント メトリック用に Investigator ツリーに表示される SQL ステートメントをバイト単位で制限します。

プロパティ設定

事前に設定済みの正規化スキーマより優先して使用される SQL ノーマライザ拡張機能の名前。

デフォルト

RegexSqlNormalizer

例

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
```

注

デフォルト設定を使用する場合、以下の正規表現 SQL ステートメント ノーマライザ プロパティも設定する必要があります。

- introscope.agent.sqlagent.normalizer.regex.matchFallThrough（P. 360）
- introscope.agent.sqlagent.normalizer.regex.keys（P. 361）
- introscope.agent.sqlagent.normalizer.regex.key1.pattern（P. 362）
- introscope.agent.sqlagent.normalizer.regex.key1.replaceAll（P. 363）
- introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat（P. 364）
- introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive（P. 365）

このプロパティへの変更はただちに有効になります。管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.matchFallThrough

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。このプロパティが true に設定されている場合、SQL 文字列はすべての regex キー グループに対して評価されます。

実装は連鎖されます。 たとえば、SQL が複数のキー グループと一致する場合、group1 からの正規化された SQL 出力は group2 の入力となり、同じように続きます。

プロパティが false に設定されている場合、キー グループと一致するとすぐに、そのグループから正規化された SQL 出力が返されます。

プロパティ設定

True または False

デフォルト

false

例

```
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.keys

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。 このプロパティは、regex グループのキーを指定します。 キーは順番に評価されます。

デフォルト

key1

例

```
introscope.agent.sqlagent.normalizer.regex.keys=key1
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.key1.pattern

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。 このプロパティは、SQL と照合するのに使用される regex パターンを指定します。

プロパティ設定

java.util.Regex パッケージで使用できる有効な regex エントリは、すべてここで使用可能です。

デフォルト

.*call(.*¥)¥.FOO(.*¥)

例

```
introscope.agent.sqlagent.normalizer.regex.key1.pattern=.*call(.*¥)¥.FOO(.*¥)
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.key1.replaceAll

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。 このプロパティが false に設定されていると、SQL クエリ内の一致パターンの最初の出現箇所が置換文字列に置き換えられます。 true に設定されていると、SQL クエリ内の一致パターンのすべての出現箇所が置換文字列に置き換えられます。

プロパティ設定

True または False

デフォルト

false

例

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。このプロパティは、置換文字列の形式を指定します。

プロパティ設定

java.util.Regex パッケージの *java.util.regex.Matcher* クラスで使用できる有効な regex エントリは、すべてここで使用可能です。

デフォルト

$1

例

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive

正規表現 SQL ステートメント ノーマライザを設定するには、introscope.agent.sqlagent.normalizer.extension (P. 359) と一緒にこのプロパティを使用します。 このプロパティは、パターン マッチで大文字と小文字を区別するかどうかを指定します。

プロパティ設定

true または false

デフォルト

false

例

```
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.sqlagent.sql.artonly

introscope.agent.sqlagent.sql.artonly プロパティを使用して、エージェントが Average Response Time メトリックのみを作成して送信するように設定できます。 バックエンド下のすべての SQL エージェント メトリックが影響を受けます。このプロパティの値が true のとき、SQL メトリックおよびトランザクション追跡でのエージェントのパフォーマンスは向上します。

注: introscope.agent.sqlagent.sql.turnoffmetrics (P. 366)=true の設定は、このプロパティに優先します。

重要: このプロパティ設定が機能するには、以下のトレーサ パラメータを設定する必要があります。

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

このプロパティは、デフォルトでオフになっています。

```
introscope.agent.sqlagent.sql.artonly=false
```

このプロパティへの変更はただちに有効になります。管理インターフェースを使用して変更できます。

注: このプロパティは、Connection Count などのカスタム メトリックを制御しません。

introscope.agent.sqlagent.sql.rawsql

トランザクション追跡の SQL コンポーネントのパラメータとして*正規化されていない* SQL を追加するために、introscope.agent.sqlagent.sql.rawsql プロパティはエージェントを設定します。 このプロパティの値が true のとき、SQL メトリックおよびトランザクション追跡でのエージェントのパフォーマンスは向上します。

このプロパティは、デフォルトでオフになっています。

```
introscope.agent.sqlagent.sql.rawsql=false
```

このプロパティへの変更は、管理対象アプリケーションの再起動後に有効になります。

重要: このプロパティを有効にすると、トランザクション追跡でパスワードと機密情報が開示されてしまう可能性があります。

introscope.agent.sqlagent.sql.turnoffmetrics

エージェントから Enterprise Manager に送信するメトリックを減らすために SQL ステートメント メトリックをオフにするには、introscope.agent.sqlagent.sql.turnoffmetrics プロパティを使用します。 このプロパティの値が true のとき、SQL メトリックおよびトランザクション追跡でのエージェントのパフォーマンスは向上します。

重要: このプロパティ設定が機能するには、以下のトレーサ パラメータを設定する必要があります。

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

このプロパティは、デフォルトでオフになっています。

```
introscope.agent.sqlagent.sql.turnoffmetrics=false
```

このプロパティは、introscope.agent.sqlagent.sql.artonl プロパティをオーバーライドします。

このプロパティへの変更はただちに有効になります。管理ユーザ インターフェースを使用して変更できます。

introscope.agent.sqlagent.sql.turnofftrace

introscope.agent.sqlagent.sql.turnofftrace プロパティは、バックエンド下の SQL ステートメントについて、エージェントがトランザクション追跡コンポーネントを作成し、それを Enterprise Manager に送信するかどうかを制御します。このプロパティの値が true のとき、SQL メトリックおよびトランザクション追跡でのエージェントのパフォーマンスは向上します。

重要: このプロパティ設定が機能するには、以下のトレーサ パラメータを設定する必要があります。

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

このプロパティは、デフォルトでオフになっています。

```
introscope.agent.sqlagent.sql.turnofftrace=false
```

このプロパティへの変更はただちに有効になります。管理ユーザ インターフェースを使用して変更できます。

SSL 通信

エージェントは SSL を介して Enterprise Manager に接続できます。 以下のプロパティを使用して通信の設定を行います。

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (P. 266)
- 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

デフォルトでエージェントが接続する Enterprise Manager を実行するコンピュータのホスト名を指定します。

デフォルト

localhost

例

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

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

エージェントからの接続をリスンしている Enterprise Manager をホストするコンピュータ上のポート番号を指定します。 SSL （Secure Socket Layer）プロトコルを使用している場合、エージェントからの接続をリスンするデフォルトのポートは 5443 です。

デフォルト

5443

例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5443
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

SSL 使用時にエージェントから Enterprise Manager への接続に使用する、クライアント ソケット ファクトリを指定します。

デフォルト

com.wily.isengard.postofficehub.link.net.SSLSocketFactory

例

```
introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.SSLSocketFactory
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT

信頼される Enterprise Manager 証明書が含まれているトラストストアの場所。 トラストストアを指定しないと、エージェントはすべての証明書を信頼します。

プロパティ設定

エージェントの作業ディレクトリへの絶対パスまたは相対パス。

例

```
introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/var/trustedcerts
```

注

Windows では、円記号をエスケープ処理する必要があります。 例：C:¥¥keystore

introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT

トラストストアのパスワード

例

```
introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT=
```

introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT

エージェントの証明書が含まれているキーストアの場所。 Enterprise Manager でクライアントの認証を必要とする場合、キーストアが必要です。

プロパティ設定

エージェントの作業ディレクトリへの絶対パスまたは相対パス。

例

```
introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=c:¥¥keystore
```

注

Windows では、円記号をエスケープ処理する必要があります。 例：C:¥¥keystore

introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT

キーストアのパスワード。

例

```
introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT=MyPassword768
```

introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT

有効な暗号化スイートを設定します。

プロパティ設定

暗号化スイートのカンマ区切りリスト。

例

```
introscope.agent.enterprisemanager.transport.tcp.
ciphersuites.DEFAULT=SSL_DH_anon_WITH_RC4_128_MD5
```

注

指定しない場合、デフォルトの有効な暗号化スイートを使用します。

ストール メトリック

以下のプロパティは、ストール メトリックを設定します。

- introscope.agent.stalls.thresholdseconds（P. 371）
- introscope.agent.stalls.resolutionseconds（P. 371）

ストール メトリック プロパティの詳細については、「イベントとしてのストールのキャプチャの無効化 (P. 215)」を参照してください。

introscope.agent.stalls.thresholdseconds

このプロパティは、実行中のプロセスがストール状態と判断されるまでの秒数を指定します。 Stall Count メトリックの精度を確保するために、ストールのしきい値を 15 秒以上に設定してください。 この設定により、Enterprise Manager が収集サイクルを完了する時間を確保できます。

デフォルト

デフォルトは 30 秒です。

例

```
introscope.agent.stalls.thresholdseconds=30
```

注

これは動的プロパティです。 このプロパティの設定は実行時に変更することができ、変更は自動的に反映されます。

introscope.agent.stalls.resolutionseconds

このプロパティは、エージェントがストールをチェックする頻度を指定します。 Stall Count メトリックの精度を確保するために、ストールの精度を 10 秒未満に設定しないでください。 この設定により、Enterprise Manager が収集サイクルを完了する時間を確保できます。

デフォルト

デフォルトは 10 秒間隔です。

例

```
introscope.agent.stalls.resolutionseconds=10
```

注

これは動的プロパティです。 このプロパティの設定は実行時に変更することができ、変更は自動的に反映されます。

スレッド ダンプ

以下のプロパティは、CA Introscope® スレッド ダンプ機能に関するエージェントのさまざまな要素を有効にして設定します。

- introscope.agent.threaddump.enable（P. 373）
- introscope.agent.threaddump.deadlockpoller.enable（P. 374）
- introscope.agent.threaddump.deadlockpollerinterval（P. 374）
- introscope.agent.threaddump.MaxStackElements（P. 375）

注: スレッド ダンプの設定方法の詳細については、「スレッド ダンプを有効にして設定する方法 (P. 84)」を参照してください。

introscope.agent.threaddump.enable

エージェント JVM 上でのスレッド ダンプの収集を有効にし、ユーザが［スレッド ダンプ］タブを表示できるようにします。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.threaddump.enable=true
```

注

- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。
- このプロパティは、*IntroscopeEnterpriseManager.properties* ファイルの introscope.enterprisemanager.threaddump.enable プロパティと連携して動作します。このプロパティを true に設定すると、Enterprise Manager のスレッド ダンプ機能が有効になります。

introscope.agent.threaddump.deadlockpoller.enable

メトリック ブラウザ ツリー内のデッドロック数メトリックによって、エージェント JVM 内の現在のデッドロック数が表示されるようにします。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.threaddump.deadlockpoller.enable=true
```

注

- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.threaddump.deadlockpollerinterval

デッドロック状態にあるスレッドの数について、CA Introscope® がエージェント JVM のポーリングを行う頻度（ミリ秒単位）。

プロパティ設定

0 より大きい整数

デフォルト

15,000 （ミリ秒）

例

```
introscope.agent.threaddump.deadlockpollerinterval=15000
```

注

- このプロパティの変更を有効にするには、管理対象アプリケーションを再起動する必要があります。

introscope.agent.threaddump.MaxStackElements

スレッド スタック トレース内の行の総数によって、CA Introscope® スレッド ダンプのサイズが決まります。 このプロパティは、スレッド スタックに格納できる行の数を設定します。

プロパティ設定

0 より大きい 25,000 以下の整数

デフォルト

12,000

例

```
introscope.agent.threaddump.MaxStackElements=12000
```

注

このプロパティの変更を有効にするには、管理対象アプリケーションを再起動します。

トランザクション追跡

以下のプロパティは、トランザクション追跡とサンプリング用です。

- introscope.agent.bizdef.turnOff.nonIdentifying.txn（P. 376）
- introscope.agent.transactiontracer.parameter.httprequest.headers（P. 377）
- introscope.agent.transactiontracer.parameter.httprequest.parameters（P. 378）
- introscope.agent.transactiontracer.parameter.httpsession.attributes（P. 378）
- introscope.agent.transactiontracer.userid.key（P. 379）
- introscope.agent.transactiontracer.userid.method（P. 379）
- introscope.agent.transactiontrace.componentCountClamp（P. 380）
- introscope.agent.crossprocess.compression（P. 381）
- introscope.agent.crossprocess.compression.minlimit（P. 382）
- introscope.agent.crossprocess.correlationid.maxlimit（P. 383）
- introscope.agent.transactiontracer.sampling.enabled（P. 384）
- introscope.agent.transactiontracer.sampling.perinterval.count（P. 384）
- introscope.agent.transactiontracer.sampling.interval.seconds（P. 385）
- introscope.agent.transactiontrace.headFilterClamp（P. 385）

詳細については、「トランザクション追跡オプションの設定 (P. 203)」を参照してください。

introscope.agent.bizdef.turnOff.nonIdentifying.txn

非識別トランザクションの追跡を有効または無効にします。

このプロパティを *introscopeAgent.profile* で FALSE に設定すると、非識別トランザクションの追跡が生成されます。

デフォルトでは、CEM UI でこの機能を有効にしても、非識別トランザクションの追跡は生成されません。

プロパティ設定

TRUE または FALSE

デフォルト

TRUE

例

```
introscope.agent.bizdef.turnOff.nonIdentifying.txn=FALSE
```

introscope.agent.transactiontracer.parameter.httprequest.headers

収集する HTTP 要求ヘッダ データを、カンマ区切りリストの形式で指定します。 カンマ区切りリストを使用します。

デフォルト

User-Agent （コメント化されています）

例

```
introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent
```

注

IntroscopeAgent.profile には、コメント化されていますが、このプロパティを null 値に設定したステートメントが含まれています。オプションで、ステートメントのコメント化を解除して、任意のヘッダ名を入力できます。

introscope.agent.transactiontracer.parameter.httprequest.parameters

収集する HTTP パラメータ データを、カンマ区切りリストの形式で指定します。

デフォルト

汎用パラメータ（コメント化されています）

例

```
introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter1,parameter2
```

注

IntroscopeAgent.profile には、コメント化されていますが、このプロパティを null 値に設定したステートメントが含まれています。オプションで、ステートメントのコメント化を解除して、任意のパラメータ名を入力できます。

introscope.agent.transactiontracer.parameter.httpsession.attributes

収集する HTTP セッション属性データを、カンマ区切りリストの形式で指定します。

デフォルト

汎用パラメータ（コメント化されています）

例

```
introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

注

IntroscopeAgent.profile には、コメント化されていますが、このプロパティを null 値に設定したステートメントが含まれています。オプションで、ステートメントのコメント化を解除して、任意のパラメータ名を入力できます。

introscope.agent.transactiontracer.userid.key

ユーザ定義のキー文字列。

デフォルト

汎用パラメータ（コメント化されています）

例

```
#introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

注

IntroscopeAgent.profile には、コメント化されていますが、このプロパティを null 値に設定したステートメントが含まれています。 ご利用の環境で、*HttpServletRequest.getHeader* または *HttpServletRequest.getValue* を使用してユーザ ID にアクセスしている場合、ユーザはオプションでステートメントのコメント化を解除し、適切な値を指定することができます。

詳細については、introscope.agent.transactiontracer.userid.method (P. 379) を参照してください。

introscope.agent.transactiontracer.userid.method

ユーザ ID を返すメソッドを指定します。Agent プロファイルには、左記の 3 つの値それぞれに対する、コメント化されたプロパティ定義が含まれます。

ユーザ ID が、getRemoteUser、getHeader、getValue のどれによってアクセスされているかに対応して、該当するステートメントのコメント化を解除します。

プロパティ設定

設定できる値を以下に示します。

- *HttpServletRequest.getRemoteUser*
- *HttpServletRequest.getHeader*
- *HttpServletRequest.getValue*

デフォルト

コメント化されています。上記のオプションを参照してください。

例

IntroscopeAgent.profile には、以下の 3 つの値それぞれに対する、コメント化されたプロパティ定義が含まれます。 使用するプロパティのコメント化を解除することができます。

```
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

トランザクション追跡で使用できるコンポーネント数を制限します。

デフォルト

5000

重要: クランプ サイズが大きくなると、メモリ要件も高くなります。極端なケースでは、JVM の最大ヒープ サイズを調整する必要があります。そうしないと、管理対象アプリケーションはメモリ不足に陥ってしまいます。

例

```
introscope.agent.transactiontrace.componentCountClamp=5000
```

注

- そのクランプを超えるトランザクション追跡はエージェントによって破棄され、警告メッセージがエージェントのログ ファイルに記録されます。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。
- 制限に達すると、ログに警告が出され、トレースは停止します。
- ゼロは有効な値では**ありません**。 introscope.agent.transactiontrace.componentCountClamp=0 を設定**しないでください**。

introscope.agent.crossprocess.compression

プロセスにまたがるトランザクション追跡データのサイズを小さくするには、このプロパティを使用します。

プロパティ設定

lzma、gzip、none

デフォルト

lzma

例

```
introscope.agent.crossprocess.compression=lzma
```

注

- このオプションによりエージェントの CPU のオーバーヘッドが増加しますが、プロセス間のヘッダ サイズは小さくなります。
- *lzma* 圧縮は *gzip* より効率的ですが、CPU を多く使用する場合があります。
- .NET Agent は *gzip* オプションをサポートしません。そのため、相互運用する必要がある場合は、*gzip* を使用しないでください。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.crossprocess.compression.minlimit

圧縮を適用するプロセス間パラメータ データの最小の長さを設定するには、このプロパティを使用します。

プロパティ設定

最大限度の合計は 0 から 2 倍まで設定できます。これは、introscope.agent.crossprocess.correlationid.maxlimit (P. 383) に設定します。

デフォルトの 1500 より少なく設定すると、圧縮はさらに頻繁に実行され、CPU のオーバーヘッドをさらに消費します。 デフォルト設定の 1500 は、一般的には通常の状態で、CPU に影響を与えません。

デフォルト

1500

例

```
introscope.agent.crossprocess.compression.minlimit=1500
```

注

- 前述の introscope.agent.crossprocess.compression と一緒に使用します。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.crossprocess.correlationid.maxlimit

使用可能なプロセス間パラメータ データの最大サイズ。

プロセス間パラメータ データの合計サイズがこの制限を超えた場合、圧縮を適用した後でも一部のデータはドロップされ、プロセス間の相関機能の一部が正しく機能しません。

ただし、この設定は、ヘッダ サイズが大きすぎるためにユーザのトランザクションがネットワーク転送で失敗することを防ぎます。

デフォルト

4096

例

```
introscope.agent.crossprocess.correlationid.maxlimit=4096
```

注

- 前述の *introscope.agent.crossprocess.compression* および *introscope.agent.crossprocess.compression.minlimit* プロパティと一緒に使用します。
- これは動的プロパティです。 実行時にこのプロパティを設定を変更することができ、変更は自動的に反映されます。

introscope.agent.transactiontracer.sampling.enabled

Transaction Tracer のサンプリングを無効にするには、以下のプロパティのコメント化を解除します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.transactiontracer.sampling.enabled=false
```

注

このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。

introscope.agent.transactiontracer.sampling.perinterval.count

通常、このプロパティは Enterprise Manager に設定されます。 エージェントでこのプロパティを設定すると、Enterprise Manager 内の設定が無効になります。 詳細については、「CA APM 設定および管理ガイド」を参照してください。

デフォルト

1

例

```
introscope.agent.transactiontracer.sampling.perinterval.count=1
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.transactiontracer.sampling.interval.seconds

通常、このプロパティは Enterprise Manager に設定されます。 エージェントでこのプロパティを設定すると、Enterprise Manager 内の設定が無効になります。

注: 詳細については、「CA APM 設定および管理ガイド」を参照してください。

デフォルト

120

例

```
introscope.agent.transactiontracer.sampling.interval.seconds=120
```

注

このプロパティの変更を有効にするには、マネージドアプリケーションを再起動する必要があります。

introscope.agent.transactiontrace.headFilterClamp

先頭フィルタリングで使用できるコンポーネントの最大レベルを指定します。 先頭フィルタリングは、トランザクション全体を収集する可能性のために、トランザクションの最初の部分を調べるプロセスです。 先頭フィルタリングは、1 番目の追跡対象のコンポーネントが終了するまで各コンポーネントをチェックします。 コール スタックが非常に深いトランザクションでは、クランプが適用されない場合、これが問題になる可能性があります。 クランプの値は、固定されたレベルまでエージェントに強制的に参照のみを実行させることにより、この動作によるメモリと CPU 使用率への影響を制限します。

デフォルト

30

警告： クランプ サイズが大きくなると、メモリ要件も高くなります。ガベージ コレクションの動作に影響を与えるため、アプリケーション全体のパフォーマンスに影響します。

例

```
introscope.agent.transactiontrace.headFilterClamp=30
```

注

- このプロパティへの変更はただちに有効となり、管理対象アプリケーションを再起動する必要はありません。
- トランザクション追跡のレベルがクランプを超えた場合、サンプリングやユーザが開始したトランザクション追跡などのほかのメカニズムがアクティブで、コレクションのトランザクションを選択 *しない限り*、そのトランザクション追跡はそれ以上検査されません。

introscope.agent.ttClamp

このプロパティは、レポート サイクルごとにエージェントがレポートするトランザクション数を制限します。

プロパティ設定

整数を指定します。

デフォルト

50

例

```
introscope.agent.ttClamp=50
```

注

- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。
- このプロパティが設定されていない（空白のままである）場合、デフォルト値は 200 になります。

URL のグループ化

以下のプロパティは、フロントエンド メトリック用の URL グループを設定します。

- introscope.agent.urlgroup.keys（P. 387）
- introscope.agent.urlgroup.group.default.pathprefix（P. 388）
- introscope.agent.urlgroup.group.default.format（P. 388）

詳細については、「URL グループの使用 (P. 196)」を参照してください。

introscope.agent.urlgroup.keys

フロントエンド名前付けの設定。

デフォルト

Default

例

```
introscope.agent.urlgroup.keys=default
```

注

URL アドレスが 2 つの URL グループに属している場合、このプロパティによってその URL グループのキーをリストした順序が重要になります。 絞り込んだパターンで定義されている URL グループは、幅広いパターンで指定された URL グループの順番に先行して指定する必要があります。

たとえば、キー alpha を持つ URL グループに、1 つのアドレスが含まれ、キー beta を持つ URL グループに、キー alpha の URL グループ内のアドレスも含めてネットワーク セグメントのすべてのアドレスが含まれている場合、keys パラメータの中では alpha を beta より前に置く必要があります。

introscope.agent.urlgroup.group.default.pathprefix

フロントエンド名前付けの設定。

デフォルト

*

例

```
introscope.agent.urlgroup.group.default.pathprefix=*
```

introscope.agent.urlgroup.group.default.format

フロントエンド名前付けの設定。

デフォルト

Default

例

```
introscope.agent.urlgroup.group.default.format=default
```

WebSphere PMI

以下のプロパティは、WebSphere PMI メトリックを設定します。

- introscope.agent.pmi.enable（P. 390）
- introscope.agent.pmi.enable.alarmManagerModule（P. 390）
- fintroscope.agent.pmi.enable.beanModule（P. 391）
- introscope.agent.pmi.enable.cacheModule（P. 391）
- introscope.agent.pmi.enable.connectionPoolModule（P. 392）
- introscope.agent.pmi.enable.hamanagerModule（P. 392）

- introscope.agent.pmi.enable.j2cModule（P. 393）
- introscope.agent.pmi.enable.jvmpiModule（P. 393）
- introscope.agent.pmi.enable.jvmRuntimeModule（P. 394）
- introscope.agent.pmi.enable.objectPoolModule（P. 395）
- introscope.agent.pmi.enable.orbPerfModule（P. 396）
- introscope.agent.pmi.enable.schedulerModule（P. 397）
- introscope.agent.pmi.enable.servletSessionsModule（P. 397）
- introscope.agent.pmi.enable.systemModule（P. 398）
- introscope.agent.pmi.enable.threadPoolModule（P. 399）
- introscope.agent.pmi.enable.transactionModule（P. 399）
- introscope.agent.pmi.enable.webAppModule（P. 400）
- introscope.agent.pmi.enable.webServicesModule（P. 400）
- introscope.agent.pmi.enable.wlmModule（P. 401）
- introscope.agent.pmi.enable.wsgwModule（P. 401）
- introscope.agent.pmi.filter.objrefModule（P. 402）

これらのプロパティは、*IntroscopeAgent.websphere.profile* ファイル、または WebSphere のインストールのデフォルト エージェント プロファイルにあります。

introscope.agent.pmi.enable

WebSphere PMI からのデータ収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.pmi.enable=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.alarmManagerModule

true に設定すると、PMI アラーム マネージャ データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.alarmManagerModule=false
```

注

- Introscope データとして表示するには、WebSphere でアラーム マネージャ データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.beanModule

WebSphere PMI からのデータ収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.beanModule=false
```

introscope.agent.pmi.enable.cacheModule

true に設定すると、PMI キャッシュ データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.cacheModule=false
```

注

- Introscope データとして表示するには、WebSphere でキャッシュ データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.connectionPoolModule

PMI connectionPool データの収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.pmi.enable.connectionPoolModule=true
```

introscope.agent.pmi.enable.hamanagerModule

true に設定すると、PMI マネージャ データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.hamanagerModule=false
```

注

- Introscope データとして表示するには、WebSphere でマネージャ データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.j2cModule

true に設定すると、PMI J2C データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.pmi.enable.j2cModule=true
```

注

- Introscope データとして表示するには、WebSphere で J2C データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.jvmpiModule

PMI JVMPI データのコレクションを有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.jvmpiModule=false
```

注

WebSphere で JVMPI を有効にして、このモジュールにデータが提供されるようにする必要があります。

introscope.agent.pmi.enable.jvmRuntimeModule

PMI JVM の実行時データの収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.jvmRuntimeModule=false
```

注

- WebSphere で JVMPI を有効にして、このモジュールにデータが提供されるようにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.objectPoolModule

true に設定すると、PMI オブジェクト プール データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.objectPoolModule=false
```

注

- Introscope データとして表示するには、WebSphere でオブジェクト プール データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.orbPerfModule

true に設定すると、PMI orbPerf データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.orbPerfModule=false
```

注

- Introscope データとして表示するには、WebSphere で orbPerf データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.schedulerModule

true に設定すると、PMI スケジューラ データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.schedulerModule=false
```

注

- Introscope データとして表示するには、WebSphere でスケジューラ データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.servletSessionsModule

PMI servletSessions からのデータ収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.pmi.enable.servletSessionsModule=true
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.systemModule

true に設定すると、PMI システム データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.systemModule=false
```

注

- Introscope データとして表示するには、WebSphere でシステム データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.threadPoolModule

true に設定すると、PMI スレッド プール データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

True

例

```
introscope.agent.pmi.enable.threadPoolModule=true
```

注

- Introscope データとして表示するには、WebSphere でスレッド プール データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.transactionModule

PMI トランザクションからのデータ収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.transactionModule=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.webAppModule

PMI webApp からのデータ収集を有効にするかどうかを指定します。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.webAppModule=false
```

注

このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.webServicesModule

true に設定すると、PMI Web サービス データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.webServicesModule=false
```

注

- Introscope データとして表示するには、WebSphere で Web サービス データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.wlmModule

true に設定すると、PMI WLM データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.wlmModule=false
```

注

- Introscope データとして表示するには、WebSphere で WLM データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.enable.wsgwModule

true に設定すると、PMI WSGW データのコレクションが有効になります。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.enable.wsgwModule=false
```

注

- Introscope データとして表示するには、WebSphere で WSGW データのカテゴリをオンにする必要があります。
- このプロパティの変更を有効にするには、マネージド アプリケーションを再起動する必要があります。

introscope.agent.pmi.filter.objrefModule

ハードコードされたフィルタを制御します。

objref フィルタは、「@xxxxx」で終わる名前を除外します。「xxxxx」は数字の文字列です。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.pmi.filter.objrefModule=false
```

注

このプロパティの変更を有効にするには、管理対象アプリケーションを再起動する必要があります。

WLDF メトリック

以下のプロパティは、WLDF メトリックを設定します。

- introscope.agent.wldf.enable（P. 403）

introscope.agent.wldf.enable

このプロパティは、WLDF メトリックの収集を有効にします。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.wldf.enable=false
```

付録 B: インスツルメンテーションの代替方法

このセクションでは、JVM AutoProbe を使用できないときにアプリケーションをインスツルメントするための代替方法について説明します。 あらゆる場合において、このセクションで説明する代替方法よりも JVM AutoProbe を優先して使用することをお勧めします。 ただし、特定のアプリケーション サーバで JVM AutoProbe を使用できない場合は、このセクションで示す手順を使用してアプリケーションをインスツルメントできます。

このセクションには、以下のトピックが含まれています。

その他のアプリケーション サーバでの Java エージェントのデプロイ

JVM AutoProbe は、アプリケーションをインスツルメントするための一般的な方法です。CA Technologies では、JVM AutoProbe を使用してアプリケーションをインスツルメントすることを強くお勧めします。

ただし、以下のアプリケーション サーバで JVM 1.4 よりも前のバージョンを実行している場合は、アプリケーション サーバ AutoProbe を使用できます。

- Sun ONE 7.0

  アプリケーション サーバ AutoProbe は、Sun ONE バージョン 7 のアプリケーション サーバのみでサポートされています。

- Oracle 10g 10.0.3

  アプリケーション サーバ AutoProbe は、Oracle バージョン 10g 10.0.3 のアプリケーション サーバのみでサポートされています。

- WebSphere または WebLogic

重要: アプリケーション サーバの AutoProbe は以下のプラットフォームではサポートされていません。

- 1.5 より後のバージョンの JVM
- OS/400

重要: アプリケーションをインスツルメントする際は、1 つの方法のみを使用します。 すでに JVM AutoProbe を使用している場合は、アプリケーション サーバ AutoProbe を使用しないでください。

アプリケーション サーバを開始するときに、クラス名の識別子としてハイフン (-) を使用しないでください。CA Introscope® はこの文字を解析しないため、これを使用すると、エージェント ログにクラス ロード エラーが出力される場合があります。

詳細:

AutoProbe を使用するための Sun ONE の設定 (P. 407)
AutoProbe を使用するための Oracle の設定 (P. 408)
Java エージェントを使用するための IBM WebSphere の設定 (P. 56)
Java エージェントを使用するための Oracle WebLogic の設定 (P. 49)

AutoProbe を使用するための Sun ONE の設定

対象: Sun ONE 7.0

AutoProbe を使用してアプリケーションをインスツルメントするために、Sun ONE のインストールを設定できます。

以下の手順に従います。

注: 以下の .xml の例で「...」が使用されている箇所は、.xml コードの情報が省略されていることを示します。 これらの情報は、例に関係がありません。

1. 管理者またはルートとしてログインします。
2. 以下の場所に移動し、server.xml ファイルを開きます。
   <Sun ONE インストール ディレクトリ>/domains/*domain1/server1/config/*

   注: アイテムの区切り文字はコロン（:）です。
3. wily/Agent.jar への完全パスを server.xml ファイルの java-config エレメントの「server-classpath」プロパティに追加します。 例：
   ```
   <java-config ... server-classpath="/sw/sun/sunone7/wily/Agent.jar:..." ...>
   ```
4. java-config エレメントに移動します。
   - bytecode-preprocessors プロパティを追加し、それに値 com.wily.introscope.api.sun.appserver.SunONEAutoProbe を設定します。

     例：
     ```
     <java-config ...
     bytecode-preprocessors="com.wily.introscope.api.sun.appserver.SunONEAutoProbe">
     ```
   - jvm-options エレメントを追加して、エージェント プロファイルの場所を定義します。 com.wily.introscope.agentProfile、または com.wily.introscope.agentResource を定義します。

     com.wily.introscope.agentProfile の例を以下に示します。
     ```
     <java-config ...>
     ...
     <jvm-options>-Dcom.wily.introscope.agentProfile=/sw/sun/sunone7/wily/core
     /config/IntroscopeAgent.profile </jvm-options>
     </java-config>
     ```

com.wily.introscope.agentResource の例を以下に示します。

```
<java-config ...>
...
<jvm-options>-Dcom.wily.introscope.agentResource=<仮想パス
>/IntroscopeAgent.profile</jvm-options>
</java-config>
```

- （オプション） com.wily.introscope.agentResource を設定した場合は、リソース ファイルをサーバ クラスパスに追加します。

5. トレーサ グループを設定してサーブレット データを収集します。

AutoProbe を使用するための Oracle の設定

対象: Oracle 10g 10.0.3

AutoProbe を使用してアプリケーションをインスツルメントするために、Oracle のインストールを設定できます。

以下の手順に従います。

1. Agent.jar を、アプリケーション サーバ クラスパスに追加します。
2. システム プロパティ oracle.classpreprocessor.classes に値 com.wily.introscope.api.oracle.OracleAutoProbe を設定します。
3. システム プロパティ oracle.j2ee.class.preprocessing に値 true を設定します。
4. コマンド ラインで以下のコマンドを実行します。

   -Dcom.wily.introscope.probebuilder.oracle.enable=true

5. 以下のコマンドを使用して、Oracle Application Server 10g を再起動します。

```
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
```

   重要: Windows ホスト上で Sun JDK 1.42 を使用して Oracle 10g リリース 2 を実行している場合は、^（カレット）文字を使用してスラッシュをエスケープする必要があります。 例：

```
-Xbootclasspath^/p:<IntroscopeAgent.jar path>
```

6. トレーサ グループを設定してサーブレット データを収集します。

詳細:

HTTP サーブレットの追跡の設定 (P. 410)

WebLogic Server の設定

AutoProbe を使用してアプリケーションをインスツルメントするために、WebLogic Server を設定できます。

以下の手順に従います。

1. アプリケーション起動スクリプト（*startMedRecServer.cmd* など）の classpath を編集して、wily/Agent.jar ファイルを含めます。
2. Java コマンド ラインで、以下のプロパティを -D オプションを指定して設定し、Introscope AutoProbe をアクティブにします。

```
-Dweblogic.classloader.preprocessor=
com.wily.introscope.api.weblogic.PreProcessor
```

3. HTTP サーブレットの追跡を設定 (P. 410) してサーブレット データを収集するようにトレーサ グループを設定します。

HTTP サーブレットの追跡の設定

AutoProbe をアプリケーション サーバと共に使用してアプリケーションをインスツルメントする前に、*toggles-full.pbd* および *toggles-typical.pbd* ファイルの中でトレーサ グループを設定する必要があります。 これにより、サーブレットのデータが収集できるようになります。

1 つのトレーサ グループをオフにしてから別のトレーサ グループをオンにします。

HTTP サーブレットの追跡を設定する方法

1. <*アプリケーション サーバのホーム*>*/wily/core/config/toggles-full.pbd* ファイルに移動し、それを開きます。
2. PBD の *HTTP Servlets Configuration* セクションへ移動します。
3. 行頭にシャープ記号を挿入して *HTTPServletTracing* トレーサ グループを**オフ**にします。 例：
   ```
   #TurnOn: HTTPServletTracing
   ```
4. 行頭のシャープ記号を削除して *HTTPAppServerAutoProbeServletTracing* トレーサ グループを**オン**にします。 例：
   ```
   TurnOn: HTTPAppServerAutoProbeServletTracing
   ```
5. <*アプリケーション サーバのホーム*>*/wily/core/config/toggles-typical.pbd* ファイルに対して、手順 2 ～ 4 を繰り返します。

AutoProbe コネクタ ファイルの作成

廃止された JVM AutoProbe のメソッドを正しく操作するには、コネクタの .jar ファイルが必要です。 AutoProbe コネクタを作成するには、以下の手順に従います。 ご使用の JVM のバージョンが 1.5 の場合は、「JVM AutoProbe」にある説明に従ってください。

以下の手順に従います。

1. 作業ディレクトリを、インストール ディレクトリの下の wily/connectors に変更します。

2. 以下のいずれかのコマンドを使用して、AutoProbe コネクタ作成ツールを実行します。

   - ツールを実行している JVM を使用して JVM を指定する場合

     ```
     java -jar CreateAutoProbeConnector.jar -current
     ```

   - コマンド ラインで JVM ディレクトリを渡して JVM を指定する場合

     ```
     java -jar CreateAutoProbeConnector.jar -jvm <directory>
     ```

   出力は、wily/connectors/AutoProbeConnector.jar の形式のファイルとなります。

3. （オプション）作成された .jar ファイルをより管理しやすく、汎用で使用できるように名前を変更します。 例：

   - wily/connectors/AutoProbeConnector131_02_Sun.jar
   - wily/connectors/AutoProbeConnector130_IBM.jar

詳細:

HTTP サーブレットの追跡の設定 (P. 410)

JVM 用の AutoProbe コネクタの実行

Sun または IBM JVM 用の AutoProbe コネクタを作成したら、作成したファイルを実行してアプリケーションをインスツルメントします。 コネクタを実行する方法は、使用しているアプリケーション サーバによって異なります。 詳細については、ご使用のアプリケーション サーバの該当するセクションを参照してください。

SAP J2EE 6.20 用の AutoProbe コネクタを実行する方法

1. 以下のファイルを開きます。
   <ドライブ>:¥usr¥sap¥<J2EE_ENGINE_ID>¥j2ee¥j2ee_<INSTANCE>¥cluster¥server¥cmdline.properties

2. 以下のコマンドを「**JavaParameters**」セクションに追加します。

   ```
   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>
   -Dcom.wily.introscope.agent.agentName=<エージェントの名前>
   ```

例：

```
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 サーバを再起動します。

NetWeaver 04/SAP J2EE 6.40 用の AutoProbe コネクタを実行する方法

1. **SAP J2EE Configtool** を実行します。
2. 変更するサーバを選択します。
3. ［**Java Parameters**］フィールドに以下の新しい Java パラメータを追加します。

   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>

   例：

   ```
   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
   ```

   注：Windows 上で NetWeaver 6.40 を使用している場合は、これらの java パラメータのスラッシュに「/」を使用する必要があります。
4. ［**Disk**］をクリックして保存します。
5. 各サーバについてステップ 2 ～ 4 を繰り返します。
6. SAP サーバを再起動します。
7. 次のファイルを開いて、Configtool の変更が適用されたことを確認します。
   <ドライブ>:¥usr¥sap¥ccms¥P66¥JC00¥j2ee¥cluster¥instance.properties
8. *ID<server_id>.JavaParameters* で始まる行を探し、入力した行が含まれていることを確認します。

Sun ONE 用の AutoProbe コネクタを実行する方法

1. 管理者またはルートとしてログインします。

   Introscope 情報を Sun ONE 7.0 の起動スクリプトに追加するには、管理者またはルートのアクセス権でログインする必要があります。
2. 以下の場所にある server.xml ファイルを開きます。
   <SunONE install dir>/domains/domain1/server1/config/

3. *server.xml* ファイルに、以下の行を追加します。

```
<jvm-options>
-Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar
</jvm-options>
```

アイテムの区切り文字はコロン（:）です。 例：

```
<jvm-options>
-Xbootclasspath/p:/sw/sun/sunone7/wily/connectors/AutoProbeConnector.jar:/sw/sun/sunone7/wily/Agent.jar
</jvm-options>
```

Oracle 10g 用の AutoProbe コネクタを実行する方法

- AutoProbe コネクタを実行するには、ブートストラップ クラスパスを変更します。

```
-Xbootclasspath/p:wily/connectors/AutoProbeConnectorJar:PathToAgentJar
```

Windows ホスト上で Sun JDK 1.42 を使用して Oracle 10g リリース 2 を実行している場合は、^（カレット）文字を使用してスラッシュをエスケープする必要があります。 例：

```
-Xbootclasspath^/p:<IntroscopeAgent.jar path>
```

WebLogic のバージョンが異なる場合は、違うバージョンの Java を使用して実行します。 Java 1.4 以前のバージョンを使用している場合は、以下の手順を使用して AutoProbe コネクタを実行します。 Java 1.5 以降を使用している場合、詳細については「JVM AutoProbe」を参照してください。

WebLogic 用の AutoProbe コネクタを実行する方法

1. 以下のコマンドを使用して、アプリケーションの起動スクリプトのブートストラップ クラスパスを編集して、作成した *AutoProbeConnector.jar*（たとえば、*startMedRecServer.cmd*）を含めます。

```
-Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar
```

-X スイッチを、スクリプトの最後にある、最後の起動コマンド（*JAVA_VM* および *JAVA_OPTIONS* の後）に追加します。以下は、スイッチの正しい挿入場所を示す抜粋です。

```
"$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. デフォルトのブートストラップ クラスパス以外のパスを使用している場合は、カスタマイズしたブートストラップ クラスパスの先頭に *Agent.jar* および *AutoProbeConnector.jar* ファイルを追加します。

JRockit JVM と組み合わせた WebLogic 用の AutoProbe コネクタを実行する方法

- JVM を起動するとき、以下のコマンドライン オプションを追加します。

```
-Xbootclasspath/a:<Agent.jar のパス>
-Xmanagement:class=com.wily.introscope.api.jrockit.AutoProbeLoader
```

その他のアプリケーション サーバ用の AutoProbe コネクタを実行する方法

- AutoProbe コネクタを実行するには、以下のコマンドを使用して、*Agent.jar* および AutoProbe コネクタをアプリケーション サーバのブートストラップ クラスパスに追加します。

```
-Xbootclasspath/p:wily/connectors/AutoProbeConnector.jar:PathToAgentJar
```

例: Xbootclasspath を使用した WAS のインスツルメント

この例は、-Xbootclasspath オプションを使用して WebSphere Application Server をインスツルメントする方法を示します。このオプションを使用すると、ブートストラップ時に JVM によってデフォルトでロードされるエンティティ（クラス、jar、ディレクトリ、zip）を上書きすることができます。agent.jar ファイルで Xbootclasspath を直接使用できないため、インスツルメントする JVM の AutoprobeConnector.jar ファイルを作成します。

以下の手順に従います。

1. WebSphere Application Server で使用される Java 実行可能ファイルを AppServer/java/jre/bin ディレクトリなどで見つけます。
2. コマンド プロンプトを開き、以下のコマンドを入力します。

```
cd <agent_install_dir>/wily/connectors
<path_to_WAS>/AppServer/java/jre/bin/java –jar CreateAutoProbeConnector.jar
–current
```

AutoprobeConnector.jar が作成されます。

3. 以下のコマンドを入力します。

   ```
   -Xbootclasspath/p:<path_to_created_Autoprobe_jar>/AutoprobeConnector.jar:<path_to_agent>/Agent[NoRedef].jar
   –Dcom.wily.introscope.agentProfile=<path_to_agent>/IntroscopeAgent[NoRedef].profile
   ```

   -Xbootclasspath/p:

   ディレクトリ、JAR アーカイブ、および ZIP アーカイブのパスをコロンで区切って指定し、デフォルトのブートストラップ クラス パスを先頭に追加します。 ブートストラップ時に JVM によってデフォルトでロードされるエンティティを上書きします。

   注: UNIX システムでは Xbootclasspath 内でコロン（ : ）を使用し、Windows システムではセミコロン（;）を使用します。

ProbeBuilder の手動実行について

ProbeBuilder の手動実行は、アプリケーションをインスツルメントするための、動的でない方法です。ProbeBuilder を手動で実行すると、アプリケーション サーバが実行される前にディスク上のクラスがインスツルメントされます。 ご使用の環境が AutoProbe をサポートしないか、AutoProbe を使用しない場合、手動 ProbeBuilding を使用できます。

Manual ProbeBuilding は、他のインスツルメント メソッドと併用しないでください。あくまで 最後の手段として利用してください。

手動 ProbeBuilding の説明では、以下のインストール タスクおよび設定タスクを実行していることを前提としています。

1. Java Agent をインストールしている。詳細については、「Java Agent のインストール」を参照してください。
2. Java Agent の接続プロパティを設定している。 詳細については、「Enterprise Manager への接続 (P. 71)」を参照してください。
3. Java Agent 名を設定している。詳細については、「Java Agent の名前付け (P. 149)」を参照してください。
4. ProbeBuilder オプションを設定している。 詳細については、「AutoProbe および ProbeBuilding オプション (P. 91)」を参照してください。

WebSphere for z/OS の AutoProbe の設定

対象: WebSphere for z/OS 6.1 および 7.0

AutoProbe を使用してアプリケーションをインスツルメントするために、z/OS 上の WebSphere のインストールを設定できます。AutoProbe の詳細については、「AutoProbe および ProbeBuilding オプション (P. 91)」を参照してください。

注: 以下の手順を使用して WebSphere 7.0 for z/OS をインスツルメントすると、z/OS の WAS 7 用の JVM 1.5 AutoProbe メソッドほど詳細なメトリックが提供されません。 たとえば、スレッドのメトリック レベルはインスツルメントされません。

重要: Java エージェント 9.0 以降を使用して z/OS 上の WebSphere 7.0 を監視している場合、アプリケーション サーバ プロセスが繰り返し再起動されることがあります。この問題を回避するには、WAS 7.0 のビルド レベルを 7.0.0.8 以上にアップグレードします。

以下の手順に従います。

1. WebSphere で、Administration Console を起動します。
2. ［Application Servers］-［<サーバ>］-［Process Definition］を選択します。

   ［Control］と［Servant］の項目がリスト表示されます。
3. ［Servant］-［JavaVirtualMachine］をクリックします。
4. ［Generic JVM Argument］フィールドを、クラスローダ プラグイン、および IntroscopeAgent.profile ファイルの場所を指定するよう設定します。 以下のいずれかを設定します。

   ```
   com.wily.introscope.agentProfile
   ```

   または

   ```
   com.wily.introscope.agentResource
   ```

引数は、以下の値を持ちます（1 つの引数で複数のプロパティが設定されています）。

```
-Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
-Dcom.wily.introscope.agentProfile=<IntroscopeAgent.profile へのパス>
```

または

```
-Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
-Dcom.wily.introscope.agentResource=<IntroscopeAgent.profile を含むリソースへのパス>
```

5. *<WebSphere インスタンス ディレクトリ>*/lib/ext ディレクトリに Agent.jar ファイルを配置します。

   注: Agent.jar ファイルを WebSphere インストール ディレクトリに配置しないでください。

   以下に間違ったディレクトリと正しいディレクトリの例を示します。

   誤： `/usr/lpp/zWebSphere/V5R0M0/lib/ext`
   正： `/WebSphere/V5R0M0/AppServer/lib/ext`

6. ./wily ディレクトリ内にある新しく作成されたすべての CA Introscope® ファイルおよびディレクトリに、WebSphere プロセスから読み取りアクセスできることを確認します。

7. すべての *.log ファイルが WebSphere プロセスへの書き込みアクセス権を持つことを確認します。Java エージェントおよび ProbeBuilder は、これらのファイルを ./wily フォルダに書き込みます。 これらのファイルには以下のものが含まれます。

   - すべての CA Introscope® ファイルおよびディレクトリ
   - *<WAS インスタンス ディレクトリ>*/lib/ext 内の CA Introscope® ファイル

8. WebSphere アプリケーション サーバを再起動します。

9. WebSphere によって「open for e-business」が通知され、Administration Console が開きます。

   メトリックのレポートが開始されます。

10. Java2 のセキュリティを有効にした WebSphere 環境で AutoProbe を正しく実行するには、Java2 セキュリティ ポリシーへのアクセス権の追加 (P. 61) を行います。

11. HTTP サーブレットの追跡を設定 (P. 410) してサーブレット データを収集します。

詳細:

AutoProbe および ProbeBuilding オプション (P. 91)

付録 C: PBD Generator の使用

PBD Generator ツールを使用すると、エージェントによって使用されるカスタム Java クラス ファイルをインスツルメントできます。

このセクションには、以下のトピックが含まれています。

CA PBD Generator について

PBD Generator ユーティリティは、Java コードに注釈を付けるのに使用した Javadoc タグから PBD ファイルを作成し、Java Agent が使用するカスタム Java クラス ファイルのインスツルメンテーションを促進できます。

Generator は、Java ソース ファイルのセットを調べ、Javadoc タグ *@instrument* を含むクラスのメソッドをインスツルメントします。

PBD Generator ツールを使用して以下のことが行えます。

- PBD ファイル構築を自動化し、PBD ファイルを手動で作成する際に発生する可能性のあるエラーを解消します。
- ビルド システムに PBD 生成を統合し、PBD ファイルの自動作成および更新を行い、変更内容を Java ソースに組み込みます。

PBDGenerator.jar ファイルを使用して、PBD Generator を Apache Ant ターゲットに組み込んで、Ant Javadoc タスクとして実行することにより、PBD Generator を統合化します。

PBD Generator の設定

このツールは、Ant ベースのビルド システムに、Ant ターゲット内の Javadoc タスクとして組み込むことが予定されています。

以下の Javadoc タスクの例は、Ant の中でこのツールを使用する方法を説明しています。

```
<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>
```

必要な PBD Generator パラメータ

以下の主要 PBD Generator パラメータが必要になります。

sourcepath

Java ソース ツリーのルート ディレクトリ

destdir

ツールから出力される PBD ファイルのディレクトリ パス

packagenames

インスツルメント用に検査される Java パッケージのカンマ区切りのリスト

doclet path

このツールを含んでいる PBD Generator jar ファイルを参照するパス

param name="-d"

このパラメータは *destdir* と同じ値を含む必要があります

PBD Generator の使用

PBD Generator を使用する前に、インスツルメントされる Java ソース ファイルに特別な Javadoc タグを挿入します。

JavaDoc タグの構文は以下のとおりです。

```
@instrument <有効なメトリック プレフィックス> <オプションのトレーサ名>
```

各値は以下のとおりです。

<*有効なメトリック プレフィックス*> には、コロン（:）なしの文字列である、任意の有効な Introscope メトリック プレフィックスを指定します。パイプ文字（|）は使用できます。

<*オプションのトレーサ名*> は BlamePointTracer、FrontendMarker または BackendMarker が指定できます。 トレーサ名が指定されていない場合、デフォルトは BlamePointTracer です。

付録 D: ネットワーク インターフェース ユーティリティの使用

ネットワーク インターフェース ユーティリティは、Catalyst 統合用のエージェントによって使用されるホスト コンピュータのネットワーク インターフェース名の値を指定するために使用します。

このセクションには、以下のトピックが含まれています。

ネットワーク インターフェース名の決定 (P. 423)

ネットワーク インターフェース名の決定

ネットワーク インターフェース ユーティリティは introscope.agent.primary.net.interface.name プロパティに対し、名前とサブインターフェースの値を提供します。 エージェントによって使用されるのと同じ JVM およびアプリケーション サーバ上でこのユーティリティを実行します。

以下の手順に従います。

1. コマンド ラインから、以下のディレクトリに移動します。

   <Agent_Home>/wily/tools

2. 以下のコマンドを実行してユーティリティを起動します。

   ```
   java -jar NetInterface.jar
   ```

   Java によってサポートされるネットワーク インターフェース名のリストを含む［ネットワーク インターフェース］タブがブラウザに表示されます。

詳細:

利用可能なネットワークのリストの設定 (P. 290)