CA Application Performance Management

.NET 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: .NET エージェントの概要

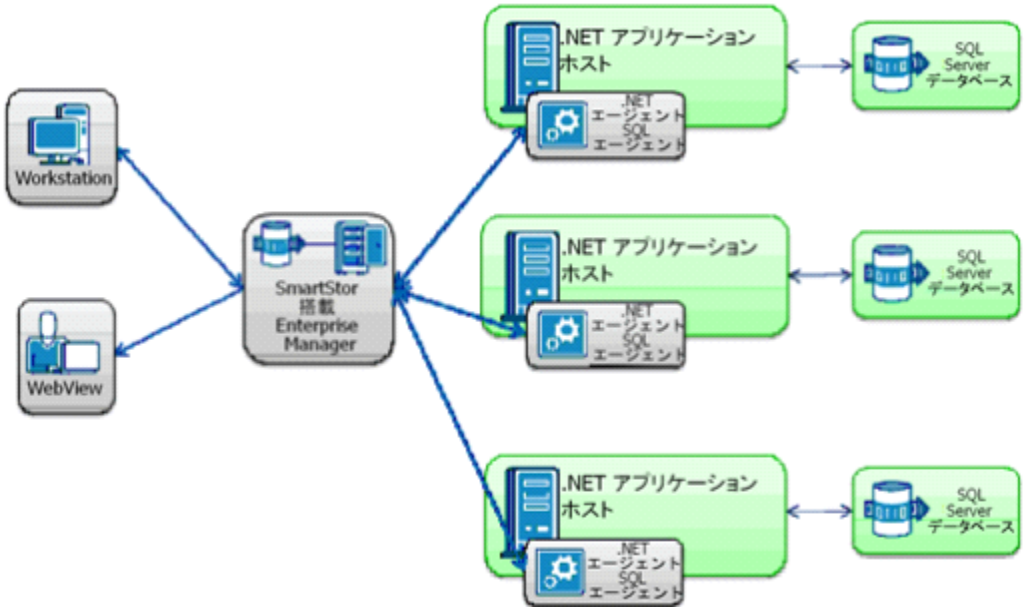
このセクションには、以下のトピックが含まれています。

Introscope デプロイでの .NET Agent について

.NET Agent は、.NET アプリケーションを対象とするアプリケーション管理ソリューションです。.NET Agent は、Microsoft の共通言語ランタイム(CLR)環境で実行されるミッション クリティカルな .NET アプリケーションを監視します。アプリケーションをコンポーネント レベルまで把握できます。

Introscope がデプロイされたシステムでは、エージェントがアプリケーションおよび環境のメトリックを収集し、Enterprise Manager に渡します。Introscope エージェントにメトリックをレポートするアプリケーションは、*インスツルメントされたアプリケーション*と呼ばれます。システムに .NET エージェントをインストールして構成すると、そのシステムで実行されるアプリケーションは起動時に自動的にインスツルメントされます。

以下の図では、.NET アプリケーションが SQL Server データベースに接続するシンプルな Introscope デプロイを示します。 各アプリケーション サーバが .NET および SQL エージェントをホストしています。エージェントは、アプリケーションのアクティビティを監視し、メトリックを Enterprise Manager にレポートします。メトリックは Workstation と WebView で表示できます。 より大規模で複雑なデプロイ環境では、エージェントの数が増え、複数の Enterprise Manager が使用される場合があります。



デフォルトでは、Microsoft インターネット インフォメーション サービス（IIS）を使用してデプロイされ、システム上でアクティブになっている ASP.NET アプリケーションのみがインスツルメントされます。 また、スタンドアロンの .NET 実行可能ファイルをインスツルメントしたり、アプリケーションの一部のみをインスツルメントしたりすることもできます。

トレーサによって実行されるインスツルメンテーション プロセスは、ProbeBuilder ディレクティブ（PBD）ファイルに定義されています。 PBD ファイル内のディレクティブは、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、それらのコンポーネントが CLR で実行される際にエージェントが収集するメトリックを識別します。 その後、.NET エージェントから Enterprise Manager にこのメトリック情報が送信されます。

Enterprise Manager は、複数のエージェントからレポートされたメトリックを保存します。 その後は、Introscope Workstation または WebView クライアント アプリケーションを使用して、アプリケーション アクティビティを監視し、パフォーマンスの問題のソースを調査し、問題を診断することができます。

詳細:

デフォルト インスツルメンテーションの変更 (P. 55)

アプリケーション環境での .NET エージェントの動作

Introscope のデプロイで、監視対象のアプリケーションを実行する各コンピュータに .NET エージェントをインストールします。 インストール後、.NET Agent は Microsoft インターネット インフォメーション サービス（IIS）によって制御されます。 IIS がユーザからアプリケーションへのリクエストを受信するまで、.NET エージェントはアクティブではありません。アプリケーション コード（.aspx、.asmx など）が実行されると、.NET Agent が起動し、NativeProfiler がコードをインスツルメントします。

IIS による .NET エージェントの制御方法

デフォルトでは、.NET エージェントは、IIS によって管理され、IIS ワーカ プロセス下で実行されているアプリケーションのみを監視します。 .NET アプリケーションの開始時に、IIS が .NET エージェントおよびインスツルメント処理をどのように制御するかを以下の手順で示します。

1. IIS がアプリケーションへのユーザ要求を受け取ります。
2. IIS が .NET ワーカ プロセスを開始します。
3. リクエストされた.NET アプリケーションが開始します。
4. 共通言語ランタイム（CLR）が NativeProfiler を開始します。
5. NativeProfiler がグローバル アセンブリ キャッシュ（GAC）から .NET エージェントをロードします。

6. .NET エージェントが IntroscopeAgent.profile を読み込み、インスツルメントに使用する PBL および PBD ファイルを決定します。
7. NativeProfiler が PBL および PBD ファイルの情報を使用してプローブをバイトコードに挿入し、アプリケーション コンポーネントから適切なメトリックを収集できるようにします。アプリケーションがインスツルメントされます。
8. インスツルメントされたアプリケーションが、.NET エージェントへのメトリックのレポートを開始します。

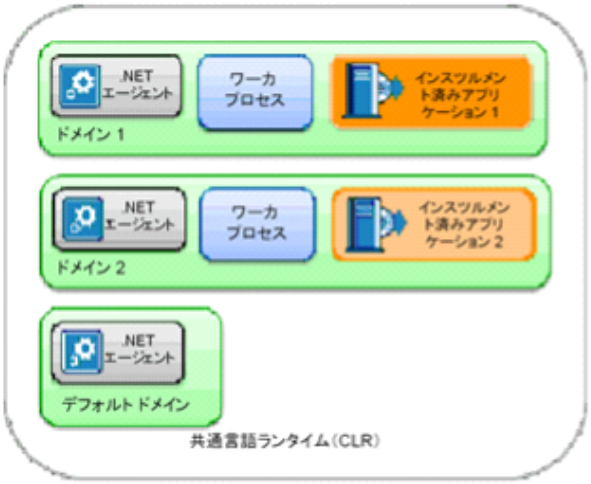
   IIS ワーカ プロセスが実行されている限り、エージェントはメトリックを収集し、それを Enterprise Manager にレポートします。 インスツルメントされたアプリケーションが、ユーザのアクティビティを一定時間感知しない場合、IIS ワーカ プロセスはアプリケーション プロセスを停止します。 IIS がアプリケーション プロセスを停止すると、ユーザ アクティビティが再開するまでは .NET エージェントは停止状態になります。

注: ASP.NET を使用しないスタンドアロン アプリケーションでも、インスツルメントを行うように .NET エージェントで設定されている場合は、インスツルメントすることができます。 スタンドアロン アプリケーションでも手順は同様です。ただし、スタンドアロン アプリケーションは Windows オペレーティング システムによってブートできるため、手順 1 および 2 が必要ありません。

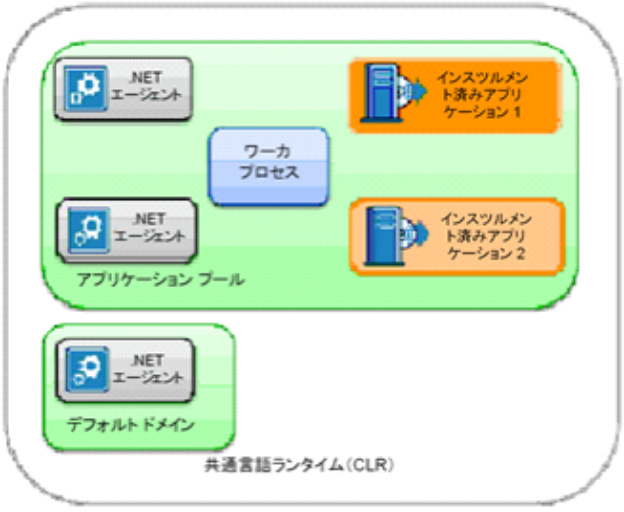
.NET Agent のインスタンス化および IIS ワーカ プロセスについて

.NET Agent は、監視したい管理対象 .NET アプリケーションをホストしている各システムにインストールします。 .NET Agent の起動時点で、CLR のデフォルト ドメインに対してエージェント インスタンスが 1 つ作成されます。 さらに、CLR で実行されるアプリケーションごとに .NET エージェント インスタンスが作成されます。

以下の図は、1 つの .NET Agent が起動された各ドメイン内に IIS ワーカ プロセスが 1 つある管理対象の ASP.NET アプリケーションを示しています。

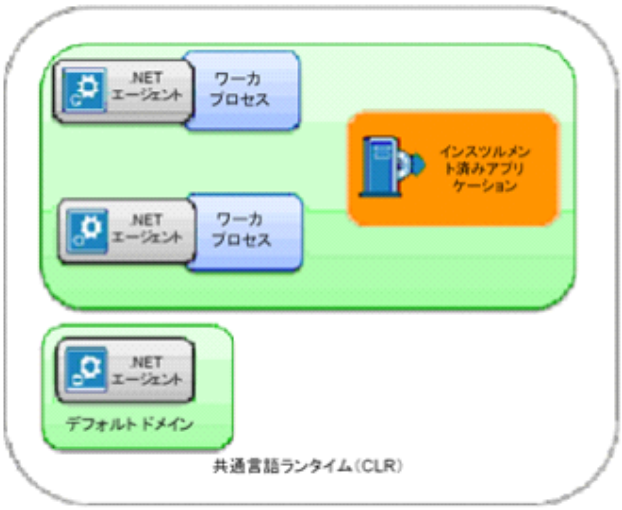


複数の .NET アプリケーションが、1 つのワーカ プロセスを共有する IIS アプリケーション プールとしてグループ化されている場合でも、デフォルト ドメインに .NET エージェント インスタンスが 1 つ作成され、アプリケーション プール内の各アプリケーションにも .NET エージェント インスタンスが 1 つずつ作成されます。以下の図は、このような例を示しています。



スケーラビリティ上の理由から、1 つのアプリケーションに複数のワーカ プロセスを割り当てることができます。 この場合、.NET エージェント インスタンスはデフォルト ドメインに 1 つ作成され、アプリケーションに関連づけられたワーカ プロセスごとにも 1 つずつ作成されます。

注: 以下の図に示す設定は、最も一般的なものです。



ワーカ プロセスが複数あるために複数の .NET Agent が 1 つの管理対象アプリケーションに関連付けられている場合は、そのようなエージェントを 1 つの仮想エージェントとして構成します。 この設定によって、複数の物理 .NET エージェントからのメトリックを集約できます。

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 ドメインの .NET エージェント インスタンスについて

「.NET Agent のインスタンス化」 (P. 20)で説明したように、CLR のデフォルト ドメインには常に 1 つの .NET Agent が作成されます。 デフォルトでは、デフォルト ドメインの .NET Agent は Enterprise Manager に接続せず、Workstation や WebView の Investigator ツリーにノードとして表示されることもありません。 ただし、必要な場合は、デフォルト ドメインの .NET Agent が Enterprise Manager に接続するように構成できます。 詳細については、「デフォルトのドメイン設定」 (P. 225)のエージェント プロパティを参照してください。

企業内での .NET Agent のデプロイ方法について

アプリケーションやそれが実行されるさまざまな環境に対して適切な .NET エージェントの構成を作成する作業は、反復的に発生するプロセスです。

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

.NET エージェントのデプロイ

.NET エージェントをデプロイするには、以下のタスクを実行します。

1. .NET エージェントのインストール
2. .NET エージェントの設定
3. .NET エージェントのプロパティの設定

詳細:

.NET エージェントのディレクトリ構造について (P. 40)

第 2 章: .NET Agent のインストール

このセクションでは、Microsoft インターネット インフォメーション サービス（IIS）サーバ上に.NET エージェントをインストールし、Enterprise Manager と通信を行うエージェントの設定方法を説明します。

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。

このセクションには、以下のトピックが含まれています。

.NET エージェントをインストールする前に

Introscope または .NET エージェントに不慣れな場合は、デプロイ プロセスを確認してください。

以下の手順に従います。

1. 以下のコンポーネントのサポートされているバージョンがインストールされていることを確認します。
   - .NET Framework[.NET Framework]

     注: デフォルトでは、エージェントは 1 つの Framework のみを監視します。エージェント プロパティを設定して、.NET Framework 4 および In-Process、Side-by-Side 実行を使用して監視する対象を管理できます。

   - Windows オペレーティング システムおよび IIS

- Microsoft SQL Server または Oracle データベース
- ODP.NET ドライバ

2. 監視するアプリケーションが IIS ワーカ プロセスを使用して実行されていることを確認します。

   Web アプリケーションがワーカ プロセスを使用していることを確認し、アプリケーションからページをロードして、タスク マネージャを開き以下を検索します。

   - IIS の aspnet_wp.exe
   - IIS の w3wp.exe

   ワーカ プロセスが表示されない場合は、IIS が管理対象コンポーネントを処理できるようにします。使用している .NET のバージョンに対応した .NET Framework ディレクトリに移動し、以下のコマンドを実行します。

   ```
   aspnet_regiis.exe –i
   ```

   例：

   ```
   C:¥WINDOWS¥Microsoft.NET¥Framework¥v2.0.50727¥aspnet_regiis.exe -i
   ```

3. エージェント ファイルをインストールするために十分なディスク空き容量があることを確認します。 CA Technologies では、インストーラ実行ファイルのサイズの 3 倍に相当するディスク領域を確保することをお勧めします。

4. 新しい .NET エージェントをインストールするコンピュータから、旧バージョンの .NET エージェントが削除されていることを確認します。

   .NET エージェントが存在するかどうかを確認するには、コマンド ウィンドウを開き、「*set*」と入力して現在定義されている環境変数のリストを表示します。com.wily.introscope.agentProfile=<*パス*> がリストに表示されていれば、作業を先に進める前にエージェントを削除します。

5. .NET エージェントをインストールするコンピュータに別の監視エージェントや CLR のプロファイラがインストールされていないことを確認します。

6. Introscope Enterprise Manager および Workstation がインストール済みであることを確認します。Enterprise Manager との接続設定に使用する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 の間の接続状況の確認には、*ping* または *telnet* を使用します。

注: .NET エージェントのサポートされているバージョンおよびインストール要件については、「*Compatibility Guide*」を参照してください。

詳細:

企業内での .NET Agent のデプロイ方法について (P. 23)
エージェント プロパティの設定 (P. 63)
.NET エージェントの削除 (P. 60)

.NET 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 .Net 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）インストーラ使用する対話方式（P. 30）
- 対話方式を使用しないサイレント方式（P. 32）
- インストール アーカイブを使用する手動方式（P. 36）

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。 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、コマンド ライン パラメータを指定したスクリプトで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。

重要: 選択した方式にかかわらず、SharePoint ドメインのユーザ アカウントは「サービスとしてログオンする」権限を持つ必要があります。

詳細:

.NET Agent が正しくインストールされているかどうかの確認 (P. 166)

対話モードでの .NET エージェントのインストール

.NET エージェントのインストーラ プログラムを起動し、表示される指示に従って .NET エージェントを対話モードでインストールできます。

以下の手順に従います。

1. 32 ビットまたは 64 ビット オペレーティング システム用のエージェント インストーラの実行可能ファイルをダブルクリックします。たとえば、以下をダブルクリックします。

   `introscope<バージョン>windowsAgent_dotNET_32.exe` ファイル

   初期画面ページが表示されます。［次へ］をクリックして、インストールを開始します。

   注: .exe または .msi 実行可能ファイルを使用してインストールできます。 .msi 実行可能ファイルは、グループ ポリシーによるソフトウェア配信や、スケジュールされたタスクなどのその他の自動配信を可能にするために提供されています。 .exe 実行可能ファイルは追加機能を提供します。 たとえば、.exe 実行可能ファイルを使用する場合は、管理者としてログオンしていないときに、インストーラを右クリックして管理者として実行できます。

2. ［次へ］をクリックしてデフォルトのインストール ディレクトリをそのまま使用するか、または［変更］をクリックして別のディレクトリを選択して、［次へ］をクリックします。

   .NET エージェントのデフォルトのインストール ディレクトリは、以下のとおりです。

   `C:¥Program Files¥CA APM¥Introscope<バージョン>`

   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。

3. エージェントがメトリックをレポートする Enterprise Manager のホスト名とポート番号を入力し、［次へ］をクリックします。

   注: エージェントのデフォルトの Enterprise Manager は *localhost* です。これは、Enterprise Manager とエージェントが同じコンピュータ上にインストールされているテスト環境に適しています。 標準的な実運用環境で、リモートの Enterprise Manager への接続を設定します。 デフォルトの Enterprise Manager ポートは *5001* です。

4. 有効にする監視オプションを選択し、［次へ］をクリックします。

   - ChangeDetector はアプリケーション環境内の変化を追跡し、それを Workstation でレポートします。 ChangeDetector を有効にする場合は、ChangeDetector エージェントの ID を入力します。
     ChangeDetector の有効化および操作の詳細については、「CA APM ChangeDetector ユーザ ガイド」を参照してください。

   - CA APM for SOA は、クライアント側およびサーバ側 Web サービスのための拡張された監視を行います。 デフォルトでは、このオプションが選択されています。

     注: .NET エージェントではデフォルトで Web サービスが監視されます。CA APM for SOA を選択すると、拡張トランザクション追跡、WCF サービスの監視、追加の依存関係メトリックといった追加機能が提供されます。 インストール後の .pbd ファイルの内容は、CA APM for SOA を有効にしたかどうかによって異なります。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。

   - CA APM for Microsoft SharePoint および CA APM Standalone Agent for Microsoft SharePoint（SPMonitor）を使用すると、Microsoft SharePoint Web アプリケーションおよびサービスの監視が可能になります。このオプションは Windows Server 2003 または 2008 でのみ使用できます。詳細については、「*CA APM for Microsoft SharePoint ガイド*」を参照してください。

   どの監視オプションも有効にしない場合は、後で監視オプションを手動で有効にすることができます。

5. 追加のエージェント サービスを自動または手動のどちらで開始するかを選択します。

   CA APM Standalone Agent for Microsoft SharePoint（SPMonitor）を選択した場合は、サービスを実行するために SharePoint ドメインのユーザ アカウントとパスワードを指定します。
   例： *<ドメイン>¥<ユーザ名>*

6. ［Install］をクリックします。

   インストーラにより .NET エージェントのルート インストール ディレクトリが作成され、エージェントが使用するファイルがインストールされます。

7. インストールが完了したら、［完了］をクリックします。
8. エージェントが正常にインストールされていることを確認するには、以下の*いずれか*を実行します。
   - .exe インストーラ：

     IntroscopeDotNetAgentInstall*.exe.log ファイルがインストーラ実行可能ファイルと同じディレクトリ内にあることを確認します。
   - .msi インストーラ：

     MSI*.LOG ファイルが %temp% フォルダ内に作成されていることを確認します。ログ ファイルは Windows Installer 4.0 以降でのみ提供されます。
9. IIS 管理サービスを再起動してインストールを完了します。

注: .NET Agent をインストールすると、システム環境変数 *com.wily.introscope.agentProfile*、*Cor_Enable_Profiling*、および *COR_PROFILER* がオペレーティング システムに追加されます。 システム環境変数が変更されたことを IIS に通知するために、IIS の再起動またはローカル コンピュータの再起動が必要です。

詳細:

.NET エージェントのディレクトリ構造について (P. 40)
Enterprise Manager とエージェントの接続を設定する方法 (P. 42)

サイレント モードでの .NET Agent のインストール

インストーラをサイレント モードで実行するには、コマンド ラインから .NET Agent インストーラを起動し、インストールの指示をコマンド ライン パラメータとして指定します。 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 .NET Agent をインストールする際に使用するのが最も一般的です。 同一の設定内容で複数のエージェントをインストールすることもできます。 コマンドには *.exe* または *.msi* のいずれかを指定できます。

以下の手順に従います。

1. コマンド プロンプト ウィンドウを開きます。
2. 適切なコマンド ライン パラメータを指定して、.exe または .msi インストーラ実行可能ファイルを起動します。 たとえば、64 ビット システムにインストールするには、以下のコマンド ライン パラメータを使用できます。

   .msi インストーラ：

   ```
   IntroscopeDotNetAgentInstall64_9.1.0.0.msi /qn
   [INSTALLDIR="<install_dir>"] [EMHOST=myhost] [EMPORT=5001] [ENABLECD=1
   CDAGENTID=<change_dectector_agent_id>] [ENABLESOA=1] [ENABLESPP=1]
   [INSTALLSPMONITOR=1 IS_NET_API_LOGON_USERNAME=<SP Monitor Domain username>
   IS_NET_API_LOGON_PASSWORD=<SP Monitor Domain password>]
   ```

   .exe インストーラ：

   ```
   IntroscopeDotNetAgentInstall64.exe /s /v"/qn [INSTALLDIR="<install_dir>"]
   [EMHOST=myhost] [EMPORT=5001] [ENABLECD=1
   CDAGENTID=<change_dectector_agent_id>] [ENABLESOA=1] [ENABLESPP=1]
   [INSTALLSPMONITOR=1] [INSTALLSPMONITOR=1 IS_NET_API_LOGON_USERNAME=<SP
   Monitor Domain username> IS_NET_API_LOGON_PASSWORD=<SP Monitor Domain
   password>]"
   ```

   注: サイレント方式でインストールするには、*/s /v /qn* オプションを指定してインストーラを起動します。 その他のパラメータの指定はオプションです。 コマンド ラインでパラメータを指定しない場合、パラメータのデフォルトの設定が使用されます。

   このようなオプションのパラメータは、インストーラを対話モードで実行するときの選択内容と同じものです。

   INSTALLDIR="*<ルート インストール ディレクトリ>*"

   .NET 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。パスに円記号が含まれる場合、パスの先頭に円記号を追加します。例：

   ```
   ¥"C:¥IntroscopeDir"
   ```

   デフォルトのルート インストール ディレクトリは以下のとおりです。

   ```
   C:¥Program Files¥CA APM¥Introscope<バージョン>
   ```

   デフォルト ディレクトリを変更する場合は、このプロパティを変更します。

EMHOST=<*ホスト名*>

エージェントがメトリックをレポートする Enterprise Manager のホスト名を指定します。 エージェントのデフォルト ホスト名は localhost です。

EMPORT=<*ポート番号*>

エージェントがメトリックをレポートする Enterprise Manager のリスニング ポート番号を指定します。 デフォルト ポートは 5001 です。

ENABLECD=0

ChangeDetector を有効にするかどうかを指定します。 このパラメータのデフォルト値は 0 で、ChangeDetector が有効でないことを示しています。

ChangeDetector を有効にする場合は、*ENABLECD* パラメータを 1 に設定します。

ChangeDetector を無効のままにする場合、関連ファイルは *<Agent_Home>*¥examples ディレクトリ内に配置され、後で有効にすることができます。

CDAGENTID=<*agent_name*>

ChangeDetector エージェントの名前を指定します。 このパラメータをコマンド ラインに含めるのは、*ENABLECD* を 1 に設定したときだけです。 ChangeDetector エージェントのデフォルトのエージェント名は *SampleApplicationName* です。

ENABLESOA=0

CA APM for SOA を有効にするかどうかを指定します。 このパラメータのデフォルト値は 0 で、CA APM for SOA が有効でないことを示しています。

CA APM for SOA を有効にする場合は、*ENABLESOA* パラメータを 1 に設定します。

CA APM for SOA を無効のままにする場合、関連ファイルは *<Agent_Home>*¥examples ディレクトリ内に配置され、後で有効にすることができます。

ENABLESPP=0

CA APM for Microsoft SharePoint Portal を有効にするかどうかを指定します。 このパラメータのデフォルト値は 0 で、CA APM for Microsoft SharePoint が有効でないことを示しています。

CA APM for Microsoft SharePoint Portal を有効にする場合は、*ENABLESPP* パラメータを 1 に設定します。Microsoft SharePoint の監視を有効にするのは、オペレーティング システムが Windows Server 2003 または Windows Server 2008 である場合のみです。

CA APM for Microsoft SharePoint Portal を無効のままにする場合、関連ファイルは *<Agent_Home>*¥examples ディレクトリ内に配置され、後で有効にすることができます。

INSTALLSPMONITOR=0

CA APM Standalone Agent for Microsoft SharePoint Portal をインストールするかどうかを指定します。 このパラメータのデフォルト値は 0 で、CA APM Standalone Agent for Microsoft SharePoint をインストールしてはならないことを示します。

CA APM Standalone Agent for Microsoft SharePoint をインストールする場合は、*INSTALLSPMONITOR* パラメータを 1 に設定します。

CA APM Standalone Agent for Microsoft SharePoint をインストールしない場合は、後で CA APM スタンドアロン エージェント インストーラを実行してインストールできます。

たとえば、デフォルト設定を変更するには、64 ビット システムで以下のようなコマンド ラインを指定します。

```
IntroscopeDotNetAgentInstall64.exe /s /v"/qn INSTALLDIR=¥"C:¥IntroscopeAgent¥"
EMHOST=dell-M65.org EMPORT=5001 ENABLECD=1 CDAGENTID=myCDAgent ENABLESOA=1
ENABLESPP=1 INSTALLSPMONITOR=1 IS_NET_API_LOGON_USERNAME=apm-domain¥apmuser
IS_NET_API_LOGON_PASSWORD=apmPassword"
```

IS_NET_API_LOGON_USERNAME=*<ドメイン>¥<ユーザ名>*

Microsoft SharePoint 監視ドメイン ユーザ名を指定します。 このパラメータは、*INSTALLMONITOR* パラメータが 0 に設定されている場合に指定されます。 このパラメータのデフォルト値は null です。

IS_NET_API_LOGON_PASSWORD=*<パスワード>*

Microsoft SharePoint 監視用のパスワードを指定します。 このパラメータのデフォルト値は null です。

3. エージェントが正常にインストールされていることを確認するには、以下のようにしてログ ファイルを表示します。
   - *.exe* インストーラ：

     インストーラ実行可能ファイルと同じディレクトリ内の IntroscopeDotNetAgentInstall*.exe.log ファイルを参照します。
   - *.msi* インストーラ：

     %temp% フォルダ内の MSI*.LOG ファイルを参照します。ログ ファイルは Windows Installer 4.0 以降でのみ提供されます。
4. （オプション）追加のコンピュータ上でスケジュールされたタスクとして、またはリモートから、適切な設定でインストーラを実行するスクリプトを作成します。
5. IIS 管理サービスを再起動してインストールを完了します。

注: .NET Agent をインストールすると、システム環境変数 *com.wily.introscope.agentProfile*、*Cor_Enable_Profiling*、および *COR_PROFILER* がオペレーティング システムに追加されます。 システム環境変数が変更されたことを IIS に通知するために、IIS の再起動またはローカル コンピュータの再起動が必要です。

インストール アーカイブを使用した .NET Agent の手動インストール

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 インストーラと同じファイルが含まれます。 アーカイブをコンピュータにコピーしてから展開し、エージェントを設定します。 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。

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、以下に示すインストール アーカイブのうち、ご使用のオペレーティング システムに適したものを *1* つ選んでダウンロードします。

   - DotNetAgentFiles-NoInstaller.x64.<APM バージョン番号>.zip
   - DotNetAgentFiles-NoInstaller.x86.<APM バージョン番号>.zip

   例：

   ```
   DotNetAgentFiles-NoInstaller.x64.9.1.0.0.zip
   ```

2. インストール アーカイブの中身を、選択したディレクトリに展開します。

3. .NET Agent 用の *<Agent_Home>*¥wily ディレクトリを設定します。

   注: .NET Agent インストーラ プログラムを実行するとき、デフォルトのルート インストール ディレクトリは C:¥Program Files¥CA APM¥Introscope<*バージョン*> です。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。

4. 以下のように wily.Agent.dll ファイルを Global Assembly Cache に登録します。

   a. 管理者としてコマンド プロンプトを開きます。

   b. *wily¥bin* ディレクトリに移動します。 例：

   ```
   explorer <Agent_Home>¥wily¥bin
   ```

   c. *assembly* ディレクトリに移動します。 例：

   ```
   explorer C:¥WINDOWS¥assembly
   ```

   d. *<Agent_Home>*wily¥bin ディレクトリの wily.Agent.dll を assembly ディレクトリにコピーします。

5. 以下のように wily.NativeProfiler ファイルを登録します。
   a. コマンド ラインから、wily¥bin ディレクトリに移動し、以下の実行可能ファイルを起動します。
      ```
      C:¥WINDOWS¥SysWOW64¥regsvr32.exe
      <Agent_Home>¥wily¥bin¥x86¥wily.NativeProfiler.dll
      ```
6. IntroscopeAgent.profile ファイルをテキスト エディタで開き、以下のように Enterprise Manager 接続を設定します。
   a. *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* の行を探し、エージェント用の Enterprise Manager のホスト名を指定します。 例：
      ```
      introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=sfcollect01
      ```
   b. 必要に応じて、Enterprise Manager との通信のための、その他のプロパティを設定します。
7. 以下のようにコマンド ラインから PerfMon コレクタ サービスを開始します。
   ```
   sc create PerfMonCollectorAgent binPath=
   "<Agent_home>¥bin¥PerfMonCollectorAgent.exe start= auto DisplayName= "CA APM
   PerfMon Collector Service"
   ```
8. ユーザが *<Agent_Home>* ディレクトリにアクセスする権限があることを確認するには、コマンド ラインからの *wilypermissions* ユーティリティを実行します。 例：
   ```
   wilypermissions.exe c:¥WilyAgent9065¥wily w3wp.exe
   ```
9. 以下の .NET エージェント環境変数を設定します。
   - com.wily.introscope.agentProfile=*<Agent_Home>*¥wily¥IntroscopeAgent.profile
   - Cor_Enable_Profiling=0x1
   - COR_PROFILER={5F048FC6-251C-4684-8CCA-76047B02AC98}
10. インストールを完了するには、IIS 管理サービスを再起動します。

詳細:

エージェント ディレクトリに対するデフォルトのユーザ権限の変更 (P. 42)
Enterprise Manager とエージェントの接続を設定する方法 (P. 42)

中国語または日本語での .NET エージェントのインストール

.msi インストーラ実行可能ファイルを実行するときは、言語の値を指定できます。

注: この情報は、32 ビットおよび 64 ビット Windows にのみ適用されます。

以下の手順に従います。

1. コマンド プロンプト ウィンドウを開きます。
2. 以下のように、言語の値を指定して .msi インストーラ実行可能ファイルを起動します。
   - 日本語
     ```
     IntroscopeDotNetAgentInstall64_<リリース>.msi TRANSFORMS=1041.mst
     ProductLanguage=1041
     ```
   - 中国語：
     ```
     IntroscopeDotNetAgentInstall64_<リリース>.msi TRANSFORMS=2052.mst
     ProductLanguage=2052
     ```
3. インストールを完了するには、アプリケーション サーバを再起動します。

   インストーラにより .NET エージェントのルート インストール ディレクトリが作成され、エージェント ファイルがインストールされます。 .MSI*.LOG ファイルが %temp% フォルダ内に作成されます。

.NET エージェントのディレクトリ構造について

.NET Agent をインストールすると、以下の *<Agent_Home>* 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、またどのプロパティがデプロイされエージェントから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。

<Agent_Home> ディレクトリでは、サブディレクトリで .NET 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。

bin

.NET エージェントの主要なライブラリが含まれます。

dynamic

動的インスツルメンテーション用の PBD 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、拡張機能を設定および有効にすることができます。

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、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および LeakHunter 用のファイルが含まれます。

hotdeploy

このディレクトリはデフォルトでは空です。このディレクトリに PBD ファイルを配置すると、IntroscopeAgent.profile を編集したり、アプリケーションを再起動したりせずに、新しいディレクティブをデプロイすることができます。ただし、このディレクトリ内にファイルを配置する場合は注意が必要です。カスタム PBD 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。

logs

.NET エージェントのログ ファイルを格納します。

readme

このディレクトリはインストール時に拡張機能を有効にした場合にのみ作成されます。ディレクトリが存在する場合は、有効にされた拡張機能の Readme が含まれています。

tools

ネットワーク インターフェース統合ユーティリティおよび Tagscript ツール ファイル（TagScript.jar、TagScript.bat、TagScript.sh コマンド ライン スクリプト）が含まれます。

version

有効にするかどうかに関係なく、オプションのエージェント拡張機能のバージョン情報が含まれます。

エージェント ディレクトリに対するユーザ権限

デフォルトでは、以下のように <Agent_Home> ディレクトリ上のすべてのユーザに対して読み取りアクセス権限が付与されます。

- *<Agent_Home>*¥wily に対する読み取り権限
- *<Agent_Home>*¥bin および *<Agent_Home>*¥ext に対する読み取り権限と実行権限
- *<Agent_Home>*¥logs および *<Agent_Home>*¥dynamic に対する読み取り権限と書き込み権限

このディレクトリへのアクセスを制限する場合は、デフォルトの権限を変更して、IIS ワーカ プロセスを実行するユーザ アカウントだけが *<Agent_Home>* ディレクトリにアクセスできるようにします。

詳細:

エージェント ディレクトリのユーザ権限の確認 (P. 53)

エージェント ディレクトリに対するデフォルトのユーザ権限の変更

<Agent_Home> ディレクトリへのデフォルトのユーザ権限を変更するには、コマンド ラインから *wilypermissions* ユーティリティを実行します。

ユーザを指定しないと、ユーティリティによってアプリケーションのデフォルトの IIS ユーザが検出され、そのユーザに権限が与えられます。

以下の手順に従います。

1. <Agent_Home> ディレクトリに移動します。
2. コマンド ラインから *wilypermissions.exe* を以下のように実行します。
   `wilypermissions.exe <Agent_Home> [プロセス名]`

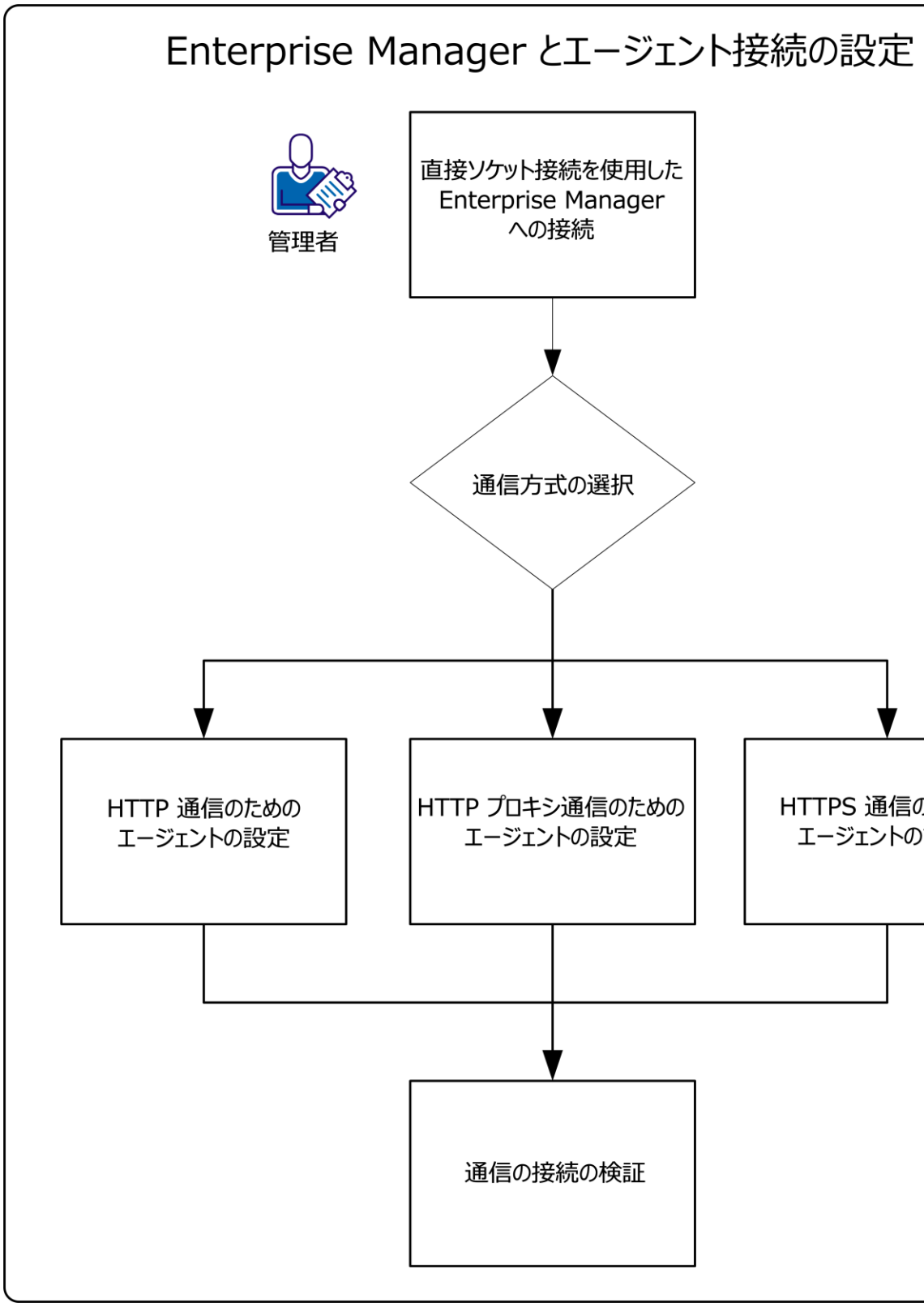
   ユーティリティが実行され、ユーザに *<Agent_Home>* ディレクトリに対する権限とパフォーマンス監視カウンタへのアクセス権が付与されます。

   注: プロセス名にはファイル拡張子を含める必要があります。 プロセス名を指定しないと、IIS ワーカ プロセス名がデフォルトになります。

Enterprise Manager とエージェントの接続を設定する方法

CA APM のデプロイされたシステムでは、エージェントが Web アプリケーションおよび環境のメトリックを収集し、それを Enterprise Manager に渡します。 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 CA APM 通信設定を使用すると、エージェントはローカルの Enterprise Manager に接続できます。 管理者は、Enterprise Manager とエージェントが別のシステム上に存在している場合でも接続できるように、デフォルト設定を変更できます。 通信方式には、HTTP、HTTP プロキシ、または HTTPS を使用するように設定できます。

以下の図は、管理者が Enterprise Manager とエージェントの接続を設定する方法を示しています。


Enterprise Manager とエージェント接続の設定
管理者
直接ソケット接続を使用した Enterprise Manager への接続
通信方式の選択
HTTP 通信のための エージェントの設定
HTTP プロキシ通信のための エージェントの設定
HTTPS 通信の エージェントの
通信の接続の検証

Enterprise Manager とエージェントの接続を設定するには、以下の手順に従います。

1. 直接ソケット接続を使用した Enterprise Manager への接続 (P. 45)。
2. 通信方式の選択 (P. 46)。
   - HTTP 通信のためのエージェントの設定 (P. 46)。
   - HTTP プロキシ通信のためのエージェントの設定 (P. 47)。
   - HTTPS 通信のためのエージェントの設定 (P. 48)。
3. 通信接続の確認 (P. 51)。

直接ソケット接続を使用した Enterprise Manager への接続

エージェントは、直接ソケット接続を使用して Enterprise Manager に接続できます。 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。通信プロパティを設定して、Enterprise Manager への接続に直接ソケット接続を使用することができます。

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。
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 のホスト名を指定します。 例：
   `introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=sfcollect01`
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`

   注: ほとんどの場合、このプロパティのデフォルト設定を使用できます。別のソケット ファクトリを使用する場合は、このプロパティを変更します。

5. IntroscopeAgent.profile ファイルを保存して閉じます。
6. IIS Admin Service を再起動します。

   設定が完了しました。

通信方式の選択

エージェントをインストールした後に、デフォルトのエージェント設定を変更して、以下のいずれかの通信方式を使用することができます。

- HTTP （HyperText Transport Protocol）
- HTTP プロキシ
- HTTPS （Hypertext Transport Protocol Secure）

Enterprise Manager とエージェントの接続を設定するには、方式を選択し、その方式に対応する以下のいずれかのタスクを実行します。

- HTTP 通信のためのエージェントの設定 (P. 46)。
- HTTP プロキシ通信のためのエージェントの設定 (P. 47)。
- HTTPS 通信のためのエージェントの設定 (P. 48)。

HTTP 通信のためのエージェントの設定

HTTP トンネリングにより、あるネットワークのデータをほかのネットワークの接続を使用して安全に送信できます。 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介して通信できるように、エージェントの接続を設定できます。

注: Enterprise Manager 上では、HTTP トンネリングがデフォルトで有効になっています。

以下の手順に従います。

1. *<Agent_Home>*/wily ディレクトリに移動します。
2. テキスト エディタで IntroscopeAgent.profile を開き、以下のプロパティを設定します。

   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=Enterprise Manager のホスト名または IP アドレス
   ```

3. 以下のプロパティを、Enterprise Manager の組み込み 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
   ```

   この値を Enterprise Manager の <EM_Home>/config/IntroscopeEnterpriseManager.properties ファイルの introscope.enterprisemanager.webserver.port プロパティの値と一致させます。 デフォルトでは、このポートは 8081 です。

4. 以下のプロパティを、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. ファイルを保存して閉じます。
6. アプリケーションを再起動します。

   Enterprise Manager Web サーバ ポートへの接続は、Enterprise Manager に接続するためにエージェントが HTTP を使用することを示します。

HTTP プロキシ通信のためのエージェントの設定

HTTP 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。この設定は、フォワード プロキシ サーバが送信 HTTP トラフィックのみを許可し、エージェントがファイアウォールの内部で実行されている場合に必要です。 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。 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場合は、このセットアップを検討してください。

重要: プロキシ サーバは HTTP Post を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。

以下の手順に従います。

1. <*Agent_Home*>/wily 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. 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.enterprisemanager.transport.tcp.host.DEFAULT=プロキシ サーバのホスト名または IP アドレス
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=Enterprise Manager Web サーバ ポート
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory
   ```

4. 以下のプロパティのコメント化を解除し、これらのプロパティをプロキシのホストとポートに設定します。

   ```
   introscope.agent.enterprisemanager.transport.http.proxy.host=ホスト名
   introscope.agent.enterprisemanager.transport.http.proxy.port=ポート番号
   ```

5. 以下のプロパティのコメント化を解除し、これらのプロパティをプロキシのユーザ名とパスワードに設定します。

   ```
   introscope.agent.enterprisemanager.transport.http.proxy.username=ユーザ名
   introscope.agent.enterprisemanager.transport.http.proxy.password=パスワード
   ```

6. ファイルを保存して閉じます。
7. アプリケーションを再起動します。

HTTPS 通信のためのエージェントの設定

エージェントは、HTTPS （HTTP over Secure Sockets Layer）を使用して Enterprise Manager に接続できます。 HTTPS は、ユーザからのページ要求および Web サーバから返されるページを暗号化および復号化します。SSL の使用によって、Web サイトのオンライン トランザクションが保護されます。

以下の手順に従います。

1. <*Agent_Home*>/wily 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. 以下のプロパティのコメント化を解除し、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。

   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=Enterprise Manager のホスト名または IP アドレス
   ```

4. 以下のプロパティのコメント化を解除し、Enterprise Manager の組み込み 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
   ```

5. 以下のプロパティのコメント化を解除し、HTTP トンネル ソケット ファクトリを使用するように設定します。 例：

   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory
   ```

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

7. エージェントおよび Enterprise Manager が相互に通信するための共通の暗号スイートを持っていることを確認します。

   注: エージェント側では、ホスト マシンにインストールされた暗号が Enterprise Manager との通信に使用されます。 ホスト マシンのグループ ポリシー設定で設定されている優先度は、暗号の優先度を制御します。

8. 存在するユーザを wily ディレクトリの pfx ファイルに追加し、それらのユーザに適切な権限 (P. 53) を付与します。

9. IntroscopeAgent.profile ファイルをテキスト エディタで開きます。

   a. 以下のプロパティのコメント化を解除し、pfx ファイルの場所とパスワードを設定します。

      ```
      introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=<pfx ファイルへのパス>
      introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT=<pfx のパスワード>
      ```

   b. ファイルを保存して閉じます。

信頼されたルート証明機関ディレクトリへの信頼できる証明書のインストール

クライアント グループと Enterprise Manager の間で SSL （Secure Socket Layer）接続を使用する場合、エージェントは検証用の信頼できる証明書を提示する必要があります。 証明書のインポート ウィザードを使用して、SSL 接続に使用する適切な証明書ファイルへのパスをインストールおよび指定します。

以下の手順に従います。

1. アプリケーション サーバ上で、［コンソール ルート］-［証明書（ローカル コンピュータ）］に移動します。
2. Trusted Root Certification Authorities フォルダを右クリックし、［すべてのタスク］-［インポート］を選択します。

   証明書のインポート ウィザードが表示されます。
3. ［次へ］をクリックします。
4. ウィザードに従って、以下のフィールドに入力します。

   インポートする証明書ファイル

   インポートするファイル（たとえば、<*DVD*>¥wilyHome¥agentCer）を指定します。

   証明書をすべて次のストアに配置する

   指定したストアに証明書をすべて配置します。

   証明書ストア

   証明書ストアを指定します（例： Trusted Root Certification Authorities）。
5. ［次へ］をクリックし、［完了］をクリックします。

   証明書がインポートされます。

個人ディレクトリへの信頼できる証明書のインストール

秘密鍵を使用してセキュリティを維持するために、個人ディレクトリに信頼できる証明書をインストールします。

以下の手順に従います。

1. クライアント上で、［コンソール ルート］-［証明書（ローカル コンピュータ）］に移動します。
2. ［個人］フォルダを右クリックし、［すべてのタスク］-［インポート］を選択します。

   証明書のインポート ウィザードが表示されます。
3. ［次へ］をクリックします。

4. プロンプトで、以下のフィールドに入力します。

   ファイル名

   パスとインポートするファイル（たとえば、<*DVD*>¥wilyHome¥agentCer）を指定します。

   証明書をすべて次のストアに配置する

   1 つのストアにすべての証明書を配置します。

   証明書ストア

   証明書を配置するストアを指定します（例：［個人］）。

5. ［次へ］をクリックし、［完了］をクリックします。

   証明書がインポートされます。

通信接続の確認

Enterprise Manager およびエージェントを設定した後、通信接続を確認します。たとえば、正常なポート 8081 接続により、HTTP 通信を確認します。

以下の手順に従います。

1. <*Agent_Home*>/wily/logs ディレクトリに移動します。
2. エージェントのログを開き、以下のメッセージを探します。

   ```
   [DEBUG] [IntroscopeAgent.HttpOutgoingConnection] Established client connection:
   host:ホスト名, port: ポート番号
   ```

   注: *ホスト名*は、エージェントが接続するコンピュータの名前で、*ポート番号*はエージェントおよび Enterprise Manager が接続するポートです。

   たとえば、ポート 8444 に接続した場合、ログには以下のメッセージが含まれます。

   ```
   [DEBUG] [IntroscopeAgent.HttpOutgoingConnection] Established client connection:
   host:ホスト名, port: 8444
   ```

IIS ワーカ プロセスのユーザ権限の確認

通常、.NET Web アプリケーションは IIS ワーカ プロセス内で実行されます。デフォルトでは、IIS ワーカ プロセス名は *w3wp.exe* または *aspnet_wp.exe* です。 ワーカ プロセスを実行するデフォルト アカウントは NETWORK SERVICE です。 .NET エージェントのインストール時に、インストーラは、.NET *<Agent_Home>*（*wily*）ディレクトリ、*bin*、*ext*、*log* サブディレクトリ、およびパフォーマンス監視カウンタにアクセスする適切な権限が設定されたルート インストール ディレクトリを作成します。

ただしアプリケーション プール レベルでは、ワーカ プロセスを実行する他のアカウントを設定できます。 IIS ワーカ プロセスを実行するためにほかのユーザ アカウントを設定した場合は、以下の手順に従います。

- アプリケーションを実行するユーザ アカウントを決定します
- アプリケーションを起動するために使用されるすべてアカウントに、*<Agent_Home>* ディレクトリにアクセスする権限があることを確認します
- パフォーマンス監視コレクション エージェントを実行するアカウントのユーザ権限を、パフォーマンス監視カウンタへのアクセスが許可されるように設定します

アプリケーションを実行するユーザの決定

企業によっては、*NETWORK SERVICE* 以外のアカウントが IIS ワーカ プロセスを実行します。 ほかのアカウント下で実行されるアプリケーションを監視できるようにするには、監視対象となるすべてのアプリケーションにまたがって、IIS ワーカ プロセスに関連付けるユーザ名を決定する必要があります。 識別する各ユーザ名について、ユーザに *<Agent_Home>* ディレクトリおよびパフォーマンス監視カウンタにアクセスする権限があることを確認する必要があります。

以下の手順に従います。

1. アプリケーションが実行していることを確認します。
2. タスク バーで右クリックし、タスク マネージャを選択して Windows タスク マネージャを開きます。
3. ［プロセス］ タブをクリックします。
4. ［イメージ名］列で、*w3wp.exe* または *aspnet_wp.exe* プロセスのエントリをスクロールして探します。
5. ［ユーザ名］列で、プロセスを実行しているユーザ アカウントを確認します。

   たとえば、w3wp.exe ワーカ プロセスのデフォルト ユーザは NETWORK SERVICE です。別のユーザ アカウントを使用して IIS ワーカ プロセスを実行している場合は、アカウント名をメモして、そのアカウントに適切な権限があることを確認してください。

エージェント ディレクトリのユーザ権限の確認

IIS ワーカ プロセスを実行するユーザ アカウントには、<Agent_Home> ディレクトリに対する適切なユーザ権限が必要です。また、.NET エージェントが正しく機能するように、パフォーマンス監視カウンタへのアクセス権を持っている必要があります。 デフォルトでは、*<Agent_Home>* フォルダ上のすべてのユーザに対して読み取りアクセス権限が付与されます。 したがって、このディレクトリへのアクセスを制限する場合は、デフォルトの権限を変更して、IIS ワーカ プロセスを実行するユーザ アカウントだけが *<Agent_Home>* ディレクトリにアクセスできるようにする必要があります。

以下の手順に従います。

1. Windows エクスプローラで、.NET エージェントのルート フォルダを右クリックし、［プロパティ］を選択します。

   フォルダの［プロパティ］ウィンドウが表示されます。
2. ［セキュリティ］タブをクリックします。
3. ［追加］をクリックします。

   ［ユーザまたはグループの選択］ダイアログ ボックスが表示されます。
4. ［*選択するオブジェクト名を入力してください*］に、ユーザ名のすべてまたは一部を入力します。

5. ［名前の確認］をクリックし、一致するユーザ名を 1 つ以上検索します。
6. 適切なユーザ アカウントを選択し、［OK］をクリックします。

   ［アクセス許可エントリ］ダイアログ ボックスが表示されます。
7. ［フル コントロール］で、ユーザの［*許可*］のチェックがオンであることを確認します。

   フル コントロールを有効にすると、ユーザには変更、読み取りおよび実行、フォルダ内容の一覧表示、読み取りおよび書き込みの権限が与えられます。
8. ［詳細設定］をクリックします。

   ［セキュリティの詳細設定］ダイアログ ボックスが表示されます。
9. ［子オブジェクトすべてのアクセス許可エントリを、ここに表示されているエントリで子オブジェクトに適用するもので置換する］をオンにして、子ディレクトリに権限が反映されるようにし、［OK］をクリックします。

パフォーマンス監視メトリックのユーザ権限の設定

Introscope Investigator でパフォーマンス監視 メトリックを参照するには、IIS ワーカ プロセスを実行するユーザ アカウントに適切な権限が必要です。wilypermissions ユーティリティを使用している場合は、これらの権限は自動的に定義されます。wilypermissions ユーティリティを使用していない場合は、IIS ワーカ プロセスを実行するユーザ アカウント用の権限を手動で設定する必要があります。

以下の手順に従います。

1. ［スタート］-［設定］-［コントロール パネル］-［管理ツール］-［ローカル セキュリティ ポリシー］-［ローカル ポリシー］-［ユーザー権利の割り当て］を選択します。
2. ［単一プロセスのプロファイル］を右クリックして、［プロパティ］を選択します。
3. ［ユーザーまたはグループの追加］をクリックします。
4. IIS ワーカ プロセスを実行するユーザ アカウントをユーザのリストに追加し、［OK］をクリックします。

5. ［システム パフォーマンスのプロファイル］を右クリックして、［プロパティ］を選択します。
6. ［ユーザーまたはグループの追加］をクリックします。
7. IIS ワーカ プロセスを実行するユーザ アカウントをユーザのリストに追加し、［OK］をクリックします。

詳細:

エージェント ディレクトリに対するデフォルトのユーザ権限の変更 (P. 42)

インスツルメンテーションのカスタマイズ

.NET Agent はデフォルトで、NativeProfiler を使用して、1 つ以上の IIS ワーカ プロセスで実行されるすべてのアプリケーションおよびアプリケーション プールをインスツルメントします。 デフォルトのインスツルメンテーションは .NET 環境および Web アプリケーションに対する広範で詳細な監視を提供していますが、要件に合わせて監視対象を調整できます。

デフォルト インスツルメンテーションの変更

デフォルトでは、NativeProfiler はすべての IIS Web アプリケーションおよびアプリケーション プールに対してインスツルメントを行い、IIS の外部で作動するスタンドアロン アプリケーションにはインスツルメントを行いません。 使用中の環境がより複雑になると、デフォルト インスツルメントを変更する必要がある場合があります。 たとえば、以下が必要になる場合があります。

- デプロイされたデフォルトの PBL または PBD を変更する。
- 非 IIS アプリケーションにインスツルメントされるように .NET エージェントを構成する。
- 特定のプロセスまたはアプリケーションがインスツルメントされないようにする。
- 特定のアプリケーション プールにインスツルメントを制限する。

これらの変更を行うには、.NET エージェント プロファイルを設定する必要があります。

デプロイするデフォルトの ProbeBuilder ディレクティブの指定

エージェント プロファイルに含まれる *introscope.autoprobe.directivesFile* プロパティは、デプロイする必要のある ProbeBuilder ディレクティブ（*.pbd*）ファイルを指定します。個々の ProbeBuilder ディレクティブ ファイルは、コードに挿入される特定のプローブ（タイマやカウンタなど）を制御します。これらのディレクティブ ファイルは、デプロイ対象の *.pbd* ファイル セットを定義する ProbeBuilder リスト（*.pbl*）ファイルでグループ化できます。エージェントのインストール時に、*introscope.autoprobe.directivesFile* は、*default-full.pbl* にリストされているファイルをアプリケーションのインスツルメントに使用するように設定されます。*default-full.pbl* は、すべての .NET コンポーネントにわたってフル インスツルメンテーションを実施する *.pbd* ファイルを参照します。別の *.pbl* を使用するための *introscope.autoprobe.directivesFile* プロパティの変更、特定の *.pbd* ファイルの登録、またはデプロイする *.pbl* に登録される *.pbd ファイル*のリストの変更は、すべて行うことができます。

デプロイする ProbeBuilder ディレクティブを変更するための最も一般的な方法は、*default-typical.pbl* ファイルが使用されるように *introscope.autoprobe.directivesFile* プロパティを変更することです。*default-full.pbl* は、監視対象のコンポーネントのサブセットをインスツルメントする *.pbd* ファイルを参照します。

フル インスツルメンテーションから標準インスツルメンテーションに変更する方法

1. IIS 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. *introscope.autoprobe.directivesFile* プロパティを探します。
4. *default-full.pbl* を *default-typical.pbl* に変更します。例：
   `introscope.autoprobe.directivesFile=default-typical.pbl,hotdeploy`
5. ファイルを保存して閉じます。
6. IIS を再起動します。

ProbeBuilder ディレクティブの操作および .NET Agent によって監視されるデフォルト コンポーネントの詳細については、「デフォルト データ収集のカスタマイズ」を参照してください。ProbeBuilder ディレクティブ ファイルで使用される構文および監視のカスタマイズの詳細については、「ProbeBuilder ディレクティブの操作 (P. 97)」を参照してください。

IIS の外部で実行されるプロセスおよびアプリケーションのインスツルメント

IIS の外部で実行されるアプリケーションを監視するには、.NET エージェント プロファイルを変更して、監視するアプリケーションを含めます。プロファイルを変更する前に、含めるアプリケーションの実行可能ファイルの正確な名前を確認してください。

IIS の外部で実行されるプロセスおよびアプリケーションのインスツルメント方法

1. IIS を停止します。
2. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
3. 「*Restricted Instrumentation*」セクションに移動します。
4. アプリケーション名を以下のプロパティに追加します。
   ```
   introscope.agent.dotnet.monitorApplications
   ```
   デフォルトでは、*w3wp.exe* および *aspnet_wp.exe* がすでにこのプロパティにリストされています。 カンマで区切って、ほかのアプリケーションをリストに追加できます。 例：
   ```
   introscope.agent.dotnet.monitorApplications=w3wp.exe,aspnet_wp.exe,RandomApp.exe,testapp.exe,readloop.exe
   ```
   重要: プロパティ リストでは、大文字と小文字が区別されます。 相対パスおよびワイルドカードはサポートされていません。フル パスの指定はサポートされていません。アプリケーション名のみを使用します。
5. 以下のプロパティを *IntroscopeAgent.profile* に追加し、false に設定します。
   ```
   introscope.agent.dotnet.runInRestrictedMode=false
   ```
6. ファイルを保存して閉じます。
7. IIS を再起動します。

アプリケーションの監視を無効にするには、*introscope.agent.dotnet.monitorApplications* プロパティのリストからアプリケーションを削除します。削除されたアプリケーションの CLR プロファイラはアクティブなままです。 NativeProfiler はオフにされます。また、そのアプリケーションと関連付けられた .NET エージェントの Enterprise Manager への接続は切断され、メトリックはレポートされません。

特定のアプリケーション プールへのインスツルメント

デフォルトでは、すべてのアプリケーション プールがインスツルメントされます。ただし、オーバーヘッドを減少させたい、または最もクリティカルなリソースのみに集中して監視を行いたい場合などは、インスツルメントするアプリケーションを限定することができます。 監視するアプリケーションを限定するには、エージェント プロファイルでインスツルメント対象のアプリケーション プールを特定する必要があります。

特定のアプリケーション プールにインスツルメントする方法

1. IIS を停止します。
2. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
3. 「*Restricted Instrumentation*」セクションに移動します。
4. 以下のプロパティのコメント化を解除します。
   ```
   introscope.agent.dotnet.monitorAppPools=
   ```
5. インスツルメントするアプリケーション プールをカンマ区切りのリストでプロパティに追加します。 例：
   ```
   introscope.agent.dotnet.monitorAppPools="NULL","DefaultAppPool","AppPool1","AppPool2"
   ```
   注: IIS 5 は、アプリケーション プールなしで実行できます。 IIS 5 で実行中のアプリケーションをインスツルメントする場合は、「Null」値を使用します。
6. ファイルを保存して閉じます。
7. IIS を再起動します。

アプリケーションのアイドル時間の設定

Introscope のデプロイで、監視対象のアプリケーションを実行する各システムに .NET Agent をインストールします。インストール後、.NET エージェントは Microsoft インターネット インフォメーション サービス（IIS）によって制御されます。 インスツルメントされたアプリケーションがユーザのアクティビティを検出しなくなると、IIS はアプリケーション プロセスを停止します。アクティビティがないために IIS がアプリケーション プロセスを停止した場合、Introscope Investigator 内の.NET エージェントのノードは利用不可になります。

アプリケーションが利用不可になるまでのアイドル時間の総量は、インターネット インフォメーション サービス（IIS）マネージャを使用して設定することができます。アイドル時間を設定することで、アプリケーションのメトリックが Introscope Investigator 内で利用可能な時間を制御することができます。

以下の手順に従います。

1. ［スタート］-［すべてのプログラム］-［管理ツール］-［インターネット インフォメーション サービス（IIS）マネージャ］に移動します。
2. 設定するアプリケーションを右クリックして、［プロパティ］を選択します。
3. ［仮想ディレクトリ］タブをクリックします。
4. タブのアプリケーション設定部分で［構成］をクリックします。
5. ［オプション］タブをクリックします。
6. ［セッションの状態を有効にする］オプションを確認します。
7. アイドル時間を分数で指定し、［OK］をクリックして［アプリケーションの構成］ダイアログ ボックスを閉じます。
8. ［OK］をクリックして、［プロパティ］ダイアログ ボックスを閉じます。

   設定が完了しました。

.NET エージェントの削除

ご使用のオペレーティング システム用の .NET エージェントを削除するには、以下の*いずれか*の方法を使用します。

- 対話方式によるアンインストール (P. 60)
- サイレント方式によるアンインストール (P. 61)
- 手動アンインストール (P. 62)

注: ご使用のオペレーティング システム用のアンインストール プログラムを使用してください。 たとえば、.exe インストーラ プログラムを使用する場合は、対応する .exe アンインストーラ プログラムを使用してエージェントを削除します。 それ以外の場合、プロセスはエージェントを削除できません。

対話モードでの .NET エージェントの削除

.NET エージェントは、インスツルメントされたアプリケーションの実行中は常にアクティブです。エージェント DLL ファイルを削除または変更する前に、インスツルメント済みのアプリケーションがすべて停止されていることを確認してください。 その後、［プログラムの追加と削除］コントロール パネルを使用してローカルでエージェントファイルを削除するか、アンインストール プログラムを使用してサイレントで削除できます。

以下の手順に従います。

1. IIS サービスを停止します。 この操作は、インスツルメント済みのアプリケーションをすべて停止します。
2. ［スタート］-［設定］-［コントロール パネル］-［プログラムの追加と削除］を選択します。
3. 現在インストールされているプログラムのリストから、以下の*いずれか*を選択します。
   - CA APM .NET Agent<*リリース*> （32 ビット）
   - CA APM .NET Agent<*リリース*> （64 ビット）
4. ［削除］をクリックします。

   エージェントを削除するかどうかを確認するプロンプトが表示されます。

5. ［はい］をクリックしてエージェントを削除します。

   .NET エージェントの DLL が登録解除され、関連する環境変数が削除されます。また、エージェント ファイルが削除されます。

6. ［プログラムの追加と削除］を閉じます。
7. IIS 管理サービを再起動するか、コンピュータを再起動してください。
8. エージェントのルート インストール ディレクトリを選択し、右クリックして［削除］を選択します。

   .NET Agent が削除されます。

サイレント モードでの .NET エージェントの削除

.NET エージェントおよび関連ファイルをサイレント モードで削除することができます。

以下の手順に従います。

1. すべてのインスツルメント済みアプリケーションを停止するために IIS サービスを停止します。
2. コマンド プロンプトで、以下の*いずれか*のコマンドを実行します。

   ```
   IntroscopeDotNetAgentInstall*.exe /s /x /v"qn"
   IntroscopeDotNetAgentInstall*.msi /x /qn
   ```

   .NET Agent が削除されます。

手動モードでの .NET Agent の削除

.NET Agent および関連付けられたファイルを手動で削除することができます。

注: .NET Agent は、インスツルメントされたアプリケーションの実行中は常にアクティブです。エージェント DLL ファイルを削除または変更する前には、インスツルメント済みのアプリケーションがすべて停止されていることを確認してください。

以下の手順に従います。

1. C:¥WINDOWS¥assembly ディレクトリに移動します。
2. wily.Agent.dll を右クリックし、メニューから［アンインストール］を選択します。

   エージェント ファイルが削除されます。
3. 管理者としてコマンド プロンプトを開きます。
   a. 以下のようにして NativeProfiler ファイルを削除します。
      - C:¥WINDOWS¥system32¥regsvr32.exe /u <*Agent_Home*>¥bin ¥wily.NativeProfiler.dll
      - C:¥WINDOWS¥SysWOW64¥regsvr32.exe /u <*Agent_Home*>¥bin¥x86 ¥wily.NativeProfiler.dll

   b. 以下のコマンドを実行して PerfMon Collector サービスを削除します。

      ```
      sc delete PerfMonCollectorAgent
      ```

   c. 以下の .NET Agent 環境変数を以下のように設定します。
      - com.wily.introscope.agentProfile=<*Agent_Home*>¥IntroscopeAgent.profile
      - Cor_Enable_Profiling=0x1
      - COR_PROFILER={5F048FC6-251C-4684-8CCA-76047B02AC98}
4. IIS を再起動してアンインストール プロセスを完了します。

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。

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,BackupEM
   2
   ```

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。

特定のエージェント プロファイルをアプリケーションに使用する方法

デフォルトでは、Web アプリケーションを監視する .NET エージェントは、アプリケーションの仮想ディレクトリまたはコンテキスト パスに基づいて自動的に名前が割り当てられます。 Web ベースのアプリケーション以外を監視するエージェントは、アプリケーション ドメイン名に基づいて名前が割り当てられます。可能な場合は、常に自動ネーミング機能を使用することを推奨します。 ただし、必要な場合は明示的にエージェントへの名前を割り当てることができます。 たとえば、.NET Agent のインスタンスが複数ある場合などは、インスタンスを別々に管理し、手動で名前を割り当てる必要があるでしょう。

手動でエージェント名を割り当てるには、以下の手順に従います。

- 各エージェント インスタンスおよびアプリケーションに個別のエージェント プロファイルを作成します。
- 各プロファイルでプロセス名およびエージェント名のプロパティを定義します。
- 特定のアプリケーションおよびエージェント インスタンス用のエージェント プロファイルの場所を設定します。

アプリケーションごとの個別プロファイルの作成

特定の .NET アプリケーションを監視するエージェント インスタンスに明示的に名前を割り当てる場合は、アプリケーションごとに *IntroscopeAgent.profile* を作成する必要があります。 各エージェント インスタンスおよびアプリケーションで個別のプロファイルを使用すると、プロセス名およびエージェント名を管理し、必要に応じて他のプロパティをカスタマイズすることもできます。

以下の手順に従います。

1. <Agent_Home> ディレクトリの *IntroscopeAgent.profile* ファイルを開きます。
2. コピーしたプロファイルを新しいディレクトリにペーストします。
3. （オプション）エージェントのインスタンス プロファイルを識別するためにプロファイル名を変更します。 例： IntroscopeAgentCalc1.profile

エージェント名の定義

カスタムのプロセス名およびエージェント名を使用すると、特定のアプリケーションを監視するエージェント インスタンスを特定するのに役だちます。 また必要に応じて、インスタンスの他のプロパティも変更できます。

以下の手順に従います。

1. エージェント インスタンスおよびアプリケーション用に作成した IntroscopeAgent.profile をテキスト エディタで開きます。 たとえば、C:¥NetApps¥wily ディレクトリで IntroscopeAgentCalc1.profile という名前のカスタム プロファイルを作成した場合は、テキスト エディタでそのファイルを開きます。
2. Custom Process Name セクションまで移動します。

3. introscope.agent.customProcessName プロパティを使用して、カスタム プロセスの表示名を指定します。 例：
   ```
   introscope.agent.customProcessName=ArcadeCaclcProcess
   ```
4. introscope.agent.AutoNamingEnabled プロパティを使用して、エージェントの自動ネーミングを無効にします。 例：
   ```
   introscope.agent.agentAutoNamingEnabled=false
   ```
5. introscope.agent.agentName プロパティを使用して、カスタム エージェントの表示名を指定します。 例：
   ```
   introscope.agent.agentName=ArcadeCalcAgent
   ```
6. 更新したカスタム プロファイルを保存し閉じます。

特定のアプリケーションおよびエージェント インスタンス用のエージェント プロファイルの場所の設定

デフォルトでは、IntroscopeAgent.profile ファイルは *<Agent_Home>*¥wily ディレクトリにインストールされます。 たとえば、デフォルトの場所は C:¥Program Files¥CA APM¥Introscope<バージョン>¥wily などです。特定のアプリケーションをモニタするエージェント インスタンス用の個別のプロファイルを作成する場合は、そのアプリケーションおよびエージェント インスタンスのプロファイルの場所を指定する設定ファイルを作成する必要があります。

以下の手順に従います。

1. *<Agent_Home>¥Sample.exe.config* ファイルをテキスト エディタで開きます。
2. Sample.exe.config ファイルからサンプルの <configSections> および <com.wily.introscope.agent> セクションをすべてコピーします。
3. .NET アプリケーション用の web.config ファイルを開くか、または IIS アプリケーション以外の実行ファイル用のスタンドアロン構成ファイルを開きます。

   アプリケーション実行ファイル用の設定ファイルを作成する詳細については、「アプリケーション固有の設定 (P. 285)」を参照してください。
4. <env.parameters> セクションで、エージェント プロファイルの場所を変更します。 例：
   ```
   <env.parameters>
      <add key="com.wily.introscope.agentProfile"
   value="C:¥¥NETApps¥¥wily¥¥IntroscopeAgentCalc1.profile" />
   </env.parameters>
   ```

5. web.config またはスタンドアロン構成ファイルを保存します。
6. 管理対象アプリケーションを再起動します。

パフォーマンス監視データを収集してカスタマイズする方法

デフォルトでは、.NET エージェントは個別の Windows サービス、パフォーマンス監視コレクション エージェントをデプロイし、すべての Windows パフォーマンス監視オブジェクト、カウンタ、およびインスタンスからメトリックを収集します。 この Windows サービスは、IIS サーバ上で実行されるすべてのエージェント インスタンスおよびプロセスについて、これらの情報をレポートします。 インストール時に、このサービスを自動的に開始するか手動で開始するかを指定します。 インストール後は、コントロール パネルの［サービス］を使用してサービスを管理できます。 たとえば、コントロール パネルの［サービス］を使用して、スタートアップの種類を変更したり、パフォーマンス監視カウンタの収集を一時停止して再開したりできます。

エージェント プロファイル内のプロパティの変更によって、パフォーマンス監視コレクション エージェントが収集するデータをカスタマイズできます。 たとえば、以下のことが可能になります。

- 正規表現を使用して、収集された特定のメトリックをフィルタする。
- 返されるパフォーマンス監視メトリックの総数を制御する。
- パフォーマンス監視カウンタを確認する頻度を制御する。
- パフォーマンス監視コレクション エージェントが新しいパフォーマンス監視オブジェクトをチェックする頻度を制御する。
- パフォーマンス監視コレクション エージェントが新しいパフォーマンス監視オブジェクトを確認することを禁止する。

注: パフォーマンス監視コレクション エージェントを実行するアカウントが、Investigator で提供されるデータ用のパフォーマンス監視カウンタにアクセスする権限を持っている必要があります。 デフォルトでは、パフォーマンス監視コレクション エージェント サービスは、パフォーマンス監視カウンタへのアクセス権を持つローカル システム アカウントを使用して実行されます。 ただし、コントロール パネルの［サービス］を使用して、サービスが実行されるさまざまなユーザ アカウントおよびパスワードを識別できます。適切な権限設定の詳細については、「IIS ワーカ プロセスのユーザ権限の設定 (P. 52)」を参照してください。

正規表現を使用したメトリック コレクションのフィルタリング

.NET エージェントのプロパティ *perfmon.metric.filterPatter* は、エージェントが読み取るパフォーマンス監視カウンタを指定します。 デフォルトの設定は、以下のとおりです。

```
introscope.agent.perfmon.metric.filterPattern=|Processor|*|*,|.NET Data
Provider*|*|*,|.NET CLR*|{osprocessname}|*,|.NET CLR
Data|*|*,|Process|{osprocessname}|*,|ASP.NET|*
```

フィルタは、 *|Object|Instance|Counter*、インスタンスがないときは *|Object|Counter* の形式です。 ここでのパラメータは以下のとおりです。

- *Object* は、Memory、Processor、または Process などのパフォーマンスの監視対象のカテゴリを指定します。
- *Instance* は、指定したオブジェクトの特定のインスタンスを指定します。 オブジェクトには、Memory のようにインスタンスがないものもあります。
- *Counter* は、収集対象の *Object|Instance* に固有のタイプのメトリックを指定します。 たとえば、.NET CLR Memory のパフォーマンス監視オブジェクトには、# Bytes in all heaps、Gen 0 heap size、# GC handles、% time in GC などのカウンタがあります。

デフォルト フィルタには *{osprocessname}* プレースホルダも含まれます。 Investigator で、*{osprocessname}* プレースホルダは、監視対象のスタンドアロン アプリケーションのインスタンス、または IIS ワーカ プロセスのアプリケーション プール名（例： *w3wp(BusinessServiceAppPool)*）を識別するために置換されます。

重要: フィルタ |*|* を使用すると、すべてのカウンタを instance-less として列挙するようパフォーマンス監視に指示することになります。その結果、カウンタが破損するおそれがあります。

introscope.agent.perfmon.metric.filterPattern プロパティの値を変更して、.NET エージェントが収集するパフォーマンス監視データを調整することができます。 たとえば、デフォルトのフィルタを変更して、レポートされるデータを増減させることができます。 また、アプリケーションに対してカスタムのパフォーマンス監視カウンタを定義している場合は、それらを追加することができます。

パフォーマンス監視メトリックの一部は、将来使用するために Microsoft によって予約されています。 これらのメトリックはパフォーマンス監視では「NotDisplayed」のタグが付いて表示されます。 これらのメトリックを Introscope Investigator で表示すると、プレースホルダ タグが表示されます。

メトリックの総数に関する制限の設定

デフォルトでは、パフォーマンス監視コレクション エージェントは、利用可能なすべてのパフォーマンス監視オブジェクト、インスタンス、およびカウンタのパフォーマンス監視データを収集します。 レポートされるパフォーマンス監視メトリックの総数に上限を設定することで、データの全体的な量を制限することができます。 パフォーマンス監視メトリックの最大数を設定すると、開始対象サーバでのパフォーマンス監視コレクション エージェントのオーバーヘッドを低減しやすくなります。

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。
2. *perfmon.metric.limit* プロパティを探し、各間隔で許可するパフォーマンス監視メトリックの最大数に設定します。 例：
   `introscope.agent.perfmon.metric.limit=100`
3. *IntroscopeAgent.profile* ファイルを保存して閉じます。

パフォーマンス監視データの収集頻度の制御

デフォルトでは、パフォーマンス監視コレクション エージェントは、すべてのパフォーマンス監視オブジェクト、インスタンス、およびカウンタのメトリック値を 15 秒おきにチェックします。 このポーリング間隔により、以前に検出されたパフォーマンス監視オブジェクトの現在のデータがレポートされます。 また、パフォーマンス監視コレクション エージェントは、データの収集元となる新しいパフォーマンス監視オブジェクトを定期的にチェックします。 デフォルトでは、この参照間隔は 10 分で、オーバーヘッドを発生させずに、利用可能なオブジェクトを検出し、古いオブジェクトを削除します。 これらの間隔のいずれか、または両方を、エージェント プロファイルで変更できます。

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。
2. *introscope.agent.perfmon.metric.pollIntervalInSeconds* プロパティを探し、パフォーマンス監視コレクション エージェントのポーリング間隔を設定します。 設定した値により、パフォーマンス監視コレクション エージェントがメトリック値をチェックする頻度が制御されます。 例：

   ```
   introscope.agent.perfmon.metric.pollingIntervalInSeconds=20
   ```
3. *introscope.agent.perfmon.category.browseIntervalInSeconds* プロパティを探し、パフォーマンス監視コレクション エージェントの参照間隔を設定します。 設定した値により、パフォーマンス監視コレクション エージェントが新しいまたは古いパフォーマンス監視オブジェクトをチェックする頻度が制御されます。 例：

   ```
   introscope.agent.perfmon.category.browseIntervalInSeconds=900
   ```
4. *IntroscopeAgent.profile* ファイルを保存して閉じます。

パフォーマンス監視オブジェクトの参照の禁止

パフォーマンス監視コレクション エージェントは、パフォーマンス監視オブジェクトに対して定期的にクエリを実行し、すべての利用可能なサービスがカウントされていることを確認します。 このアクションにより、エージェントは、必要に応じて新しいカウンタを検出するか、古いカウンタを除去することができます。 この機能は、デフォルトで有効になっており、クエリ間隔はデフォルトでは 600 秒（10 分）に設定されています。

システムのオーバーヘッドを軽減したい場合は、パフォーマンス監視コレクション エージェントが新しいまたは古いカウンタをチェックすることを禁止できます。

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。
2. *introscope.agent.perfmon.category.browseEnabled* プロパティを探し、それを *false* に設定します。 例：
   `introscope.agent.perfmon.category.browseEnabled=false`
3. *IntroscopeAgent.profile* ファイルを保存して閉じます。

パフォーマンス監視データの収集の開始

.NET Agent は、外部サービスである CA APM PerfMon コレクタ サービスをデプロイして、パフォーマンス監視メトリックを収集およびレポートします。このサービスでは、.NET Agent ごとにメトリックを収集する代わりに、コンピュータ レベルでメトリックを収集できます。 このインスタンスの .NET Agent のみが PerfMon システム リソースを使用します。

注: .NET Agent をアプリケーション サーバにデプロイした後で、このサービスを実行します。

以下の手順に従います。

1. 管理者として Windows Management Console にログインします。
2. CA APM PerfMon コレクタ サービスに移動します。
3. サービスを右クリックし、［開始］をクリックします。

   データの収集が開始されます。

パフォーマンス監視データの収集の停止

すべてのパフォーマンス監視カウンタの収集から、.NET アプリケーションの管理に必要のないパフォーマンス監視データの収集を停止できます。

以下の手順に従います。

1. コントロール パネルの［サービス］を開きます。
2. サービス名のリストでパフォーマンス監視コレクタ サービスを探します。
3. パフォーマンス監視コレクタ サービスを選択し、右クリックして［プロパティ］を選択します。
4. ［停止］をクリックします。
5. スタートアップの種類として、［手動］または［無効］を選択します。
6. ［OK］をクリックします。

起動時間の制御方法

IIS を使用する多くの組織では、各アプリケーション ドメインの .NET アプリケーション プールをリサイクルするために定期的に IIS サービスを再起動します。 IIS の再起動時は常に、各アプリケーション プール内でアプリケーションをインスツルメントするために .NET エージェントも呼び出されます。 この初期起動時間は、監視対象のアプリケーションとクラスの数、エージェント プロファイルの設定、およびカスタマイズした PDB が存在するかどうかによって異なります。

NativeProfiler を使用してインスツルメントされたエージェントのデフォルト設定では、エージェントとアプリケーション サーバが適切な時間で開始できるようになっています。 起動時のパフォーマンスを改善するために、いくつかのオプションの手順を実行できます。

.NET エージェントの起動時間を改善するには、以下のタスクを実行します。

- エージェント プロファイル内の introscope.nativeprofiler.directivematching.cache.max.size プロパティの値を変更します。

  デフォルトでは、監視対象のクラスが含まれていることが以前に検出されたディレクティブ グループのメモリ内キャッシュがエージェントによって作成されます。 ユーザが IIS を開始すると、エージェントは以前に検出したクラスのキャッシュを作成します。 アプリケーション コードが新しいクラスを監視するため、キャッシュは徐々に増加します。 デフォルトでは、メモリ内キャッシュは 最大で 5000 のクラス名を格納します。 キャッシュ サイズがこの制限に達すると、エージェントはキャッシュが一杯になったことを示すエントリを NativeProfiler ログ ファイルに記録します。

  IntroscopeAgent.profile ファイル内の introscope.nativeprofiler.directivematching.cache.max.size プロパティを使用してキャッシュのサイズを増減させることができます。 キャッシュが 5000 を超えるクラス名を格納している場合、値を増加させることで起動時間が改善する場合があります。ただし、値を増加させると、エージェントのメモリ オーバーヘッドが増加します。プロパティ値を減少させると、エージェントのメモリ オーバーヘッドは減少します。監視するクラスが 5000 に満たない場合や、多数のディレクティブ グループの監視を停止している場合は、値を減少させる方が適切である可能性があります。

- クラスの識別方法、およびカスタム PBD ファイル内のディレクティブを確認します。

  同じグループ内のクラスに対して IdentifyInheritedAs ディレクティブを使用すると、エージェントが継承階層を最適に使用できるようになります。

In-Process、Side-by-Side 実行を有効にする方法

.NET Framework 4 では、同じプロセス内で異なるバージョンの .NET Framework を使用するアプリケーションを実行できます。古いコンポーネントは古い .NET Framework バージョンを引き続き使用し、新しいコンポーネントは新しい .NET Framework バージョンを使用します。

デフォルトでは、.NET エージェントは、ホスト プロセス内で最初にロードされるアプリケーションが使用する .NET Framework をインスツルメントします。たとえば、起動する最初のアプリケーションが .NET Framework 2.0 を使用する場合、.NET Framework 2.0 のコンポーネントのみがデフォルトでインスツルメントされます。

com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs プロパティを使用すると、.NET エージェントは同一プロセス内で 1 つ以上の .NET Framework バージョンのコンポーネントをインスツルメントできます。

com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs プロパティを使用すると、監視対象となる .NET Framework のインスタンスを指定できます。たとえば、このプロパティに .NET Framework 4 のみを監視するように設定すると、.NET Framework 4 上で実行されるコンポーネントのみがメトリックをレポートします。
com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs プロパティを使用すると、.NET Framework 4 と .NET Framework 2 の両方を監視して、In-Process、Side-by-Side 実行を使用したメトリックのレポートを行うことができます。

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。
2. *com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs* プロパティを見つけます。
3. 監視する .NET Framework のバージョンを指定するには、このプロパティを以下の例のように（カンマ区切りで）設定します。
   `com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs=v2,v4`
4. ファイルを保存して閉じます。
5. 管理対象アプリケーションを再起動します。

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。

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。

   ```
   ASP¥¥.NET¥¥|login_aspx:.*
   ```

   login.aspx 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：

      ```
      "ASP¥.NET¥|login_aspx:.*"
      ```

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。

「ASP¥¥.NET(¥¥|.*):.*」は、ASP.NET サマリ レベルおよびすべての個別の ASP ページに一致します。

「ASP¥¥.NET.*」は、「ASP.NET」で始まるメトリック パスがほかに存在しない場合は機能します。

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。

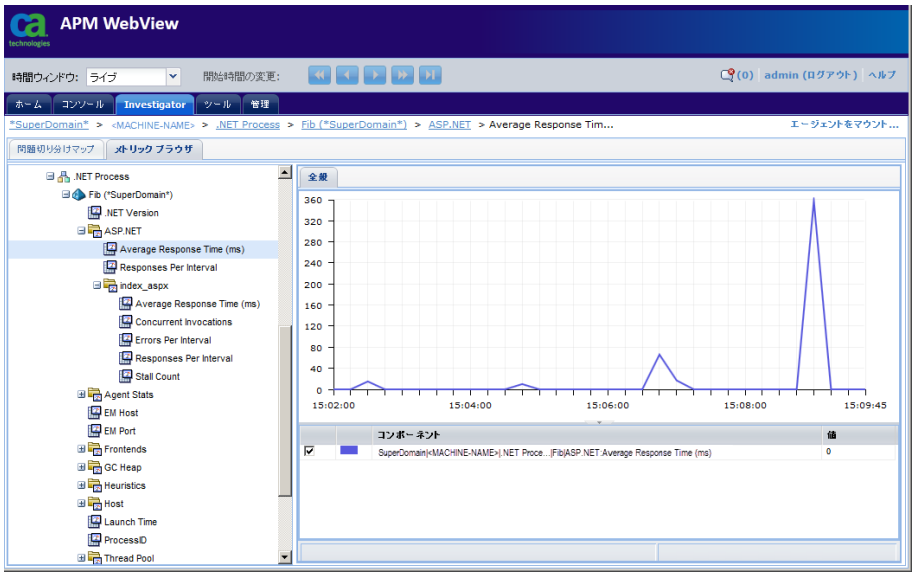
- ASP ページおよびサマリ レベルの分布メトリック（P. 78）
- ASP ページ レベルの分布メトリック（P. 79）

ASP ページおよびサマリ レベルの分布メトリックの例

以下のパターンでは、ASP ページおよびサマリ レベルの分布メトリックを収集します。

```
introscope.agent.distribution.statistics.components.pattern=ASP¥¥.NET(¥¥|.*):.*
```

以下の図は、.NET エージェント ノードの ASP ページおよびサマリ 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。

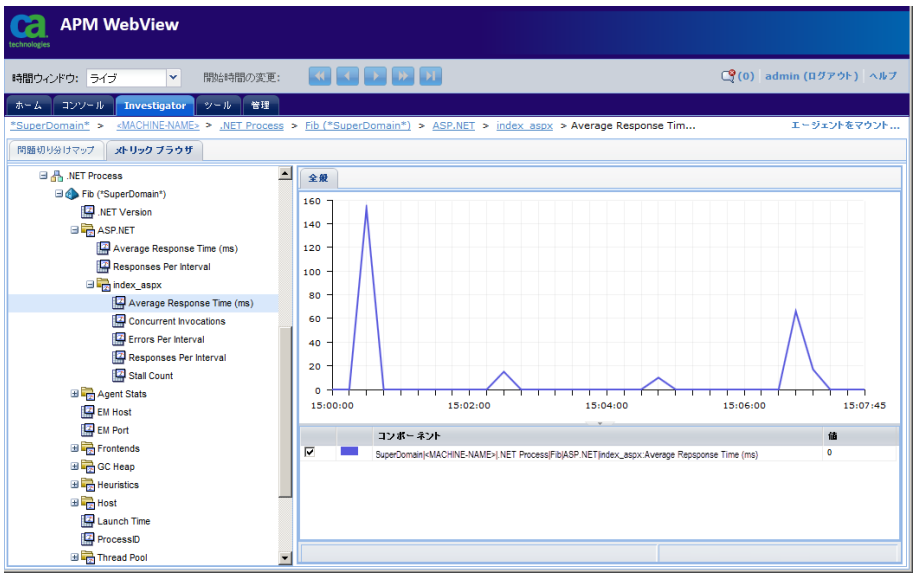


ASP ページの分布メトリックの例

以下のパターンでは、ASP ページの分布メトリックのみを収集します。

```
introscope.agent.distribution.statistics.components.pattern=ASP¥¥,NET¥¥|.*
```

以下の図は、.NET エージェント ノードの ASP ページ 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。



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。

第 4 章: デフォルト データ収集のカスタマイズ

エージェントをインストールするときは、いくつかのデフォルトの ProbeBuilder ディレクティブ（*.pbd*）ファイルと、.NET Framework および ASP.NET アプリケーションの多数の一般的なコンポーネントで有効な監視機能も一緒にインストールされます。 このセクションでは、デフォルトで提供される監視、およびカスタム *.pbd* ファイルを作成せずにデフォルトの監視を変更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。

デフォルトの ProbeBuilder ファイルについて

ProbeBuilder ディレクティブ（*.pbd*）ファイルは、挿入するプローブのタイプと、プローブを配置するコード内の場所を指定します。 タイマやカウンタなどのプローブは、エージェントが Introscope Enterprise Manager にレポートするメトリックを制御します。 カスタマイズなしで、エージェントが初期状態でメトリックを収集できるようにするために、Introscope には、事前に定義された ProbeBuilder ファイルのデフォルト セットが含まれます。

- *ProbeBuilder ディレクティブ（PBD）ファイル：*プローブを挿入し、アプリケーションのメトリックを取得するための、具体的な指示が含まれます。
- *ProbeBuilder リスト（PBL）ファイル：*すべてまたは一部のコンポーネントを監視するために一緒にデプロイする、具体的な PBD ファイルのリストが含まれます。

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。

デフォルトの ProbeBuilder ファイルでは、.NET 環境の最も一般的なコンポーネントの監視が可能です。 デフォルトのファイル セット内の設定を細かく調整することで、ご使用の環境に合わせて、一般的なコンポーネントの監視をカスタマイズできます。

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 .NET コンポーネントを追跡します。

- .NET ディレクトリ サービス
- .NET Messaging
- .NET Remoting
- ADO.NET
- ASP.NET
- エンタープライズ サービス
- ネットワーク ソケット
- SMTP メール
- Web サービス

デフォルトの ProbeBuilder ディレクティブ（PBD）ファイル

以下の表では、.NET エージェントと共にインストールされるデフォルトの PBD ファイルについて説明します。

PBD ファイル名	説明
appmap.pbd	このファイルは、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-soa.pbd	このファイルは、インストール時に CA APM for SOA を有効にしたかどうかに応じて、.NET Framework クラス ライブラリまたは SOA スタックのための、アプリケーション問題切り分けマップのトレーサ ディレクティブを提供します。
appmap-soa.spm.pbd	このファイルは、.NET Framework でサポートされる SOAP スタック用のアプリケーション問題切り分けマップのトレーサ ディレクティブを提供します。 このファイルは、インストール時に CA APM for SOA を有効にしない場合にのみインストールされます。 インストール時に CA APM for SOA を有効にする場合、このファイルは *appmap-soa.pbd* という名前に変更され、デフォルトのエージェントである *appmap-soa.pbd* は *appmap-soa.core.pbd* という名前に変更されます。

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のファイルです。
biz-trx-http.pbd	このファイルは、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。
dotnet.pbd	このファイルは、.NET Framework クラス ライブラリをサポートするディレクティブを提供します。
errors.pbd	このファイルで、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、ErrorDetector 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) の問題を示すエラーのみです。
httpheaderdecorator.pbd	CA CEM との統合ソリューションの一部である HTTP ヘッダ デコレータの有効化に使用されるファイルです。
leakhunter.pbd	Introscope LeakHunter のインスツルメンテーション設定ファイルです。 通常、このファイルの内容を変更する必要はありません。
sharepoint-full.pbd	Microsoft SharePoint データの追跡について (*TurnOn* ディレクティブの形式で) オン/オフを切り替えるファイルです。 ほとんどのトレーサ グループがオンになります。
sharepoint-typical.pbd	Microsoft SharePoint データの追跡について (*TurnOn* ディレクティブの形式で) オン/オフを切り替えるファイルです。 トレーサ グループのごく一部のみがオンになります。
skips.pbd	再入可能に問題のある一部のアセンブリ、クラスおよびメソッドをスキップするように NativeProfiler に指示するファイルです。
spm-correlation.pbd	このファイルは、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。 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。
sqlagent.pbd	このファイルは、SQL エージェントの設定ファイルです。 このファイルを使用して、ADO.NET ベンダー ライブラリ (.dll) をインスツルメントします。通常、このファイルを編集する必要はありません。
toggles-full.pbd	このファイルは、他のディレクティブ ファイル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 スイッチを提供します。 ほとんどのトレーサ グループがオンになります。

toggles-typical.pbd	他のディレクティブ 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/オフを切り替えるファイルです。トレーサ グループのごく一部のみがオンになります。
webservices.pbd	このファイルは、.NET Web サービスの監視をサポートするディレクティブを提供します。 このファイルの内容は、インストール時に CA APM for SOA を有効にしたかどうかにより異なります。有効にされているとき、.NET Web サービスおよび SOAP スタックが監視されます。有効にされていないときは .NET Web サービスのみが監視されます。

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. 88)
トレーサ グループのオンまたはオフ (P. 90)

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 ProbeBuilder リスト（PBL）ファイル

各エージェントで利用できる .pbl ファイルのセットは、以下の 2 つです。

- **default-full.pbl**：デフォルトのファイルです。 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。

トレーサ グループは、PBD ファイルに含まれており、PBL ファイルで参照されます。 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「flag」という用語で記述されます。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。

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、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、すべてのシステム メッセージング 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。このことは、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。 *toggles-full.pbd* ファイルでは、すべてのデフォルト トレーサ グループに対して追跡がオンになっており、検出されたすべての .NET コンポーネントの追跡が有効になります。

toggles-typical.pbd ファイルでは、トレーサ グループのサブセットに対して追跡がオフになっています。デフォルトでは、*toggles-typical.pbd* ファイル内の以下のデフォルト トレーサ グループに対して追跡がオフになっています。

- ネットワークの設定： SocketTracing
- トランザクション ユーティリティの追跡： ContextUtilTracing
- ランタイム Remoting の追跡： RemotingWebServiceTracing
- システム Web メールの追跡： WebMailTracing

ほとんどの場合、デフォルトの *toggles-full.pbd* および *toggles-typical.pbd* ファイルは編集せずに使用できます。 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。たとえば、*toggles-typical.pbd* ファイルを使用している場合、ソケット メトリックの追跡を追加するには、*Network Configuration* セクションを探し、*TurnOn* ステートメントをコメント化解除して *SocketTracing* を有効にします。

```
TurnOn: SocketTracing
```

同様に、特定のトレーサ グループの追跡を停止するには、*TurnOn* ステートメントをコメント化してシステム オーバーヘッドを低減します。 たとえば、*toggles-full.pbd* ファイルを使用しているものの、SQL エージェント オペレーションの追跡には関心がない場合、そのようなオペレーションの追跡を中止するには、*SQL Agent Tracing* セクションを探し、SQL エージェント トレーサ グループに対して *TurnOn* ステートメントをコメント化します。

```
#TurnOn: SQLAgentCommands
#TurnOn: SQLAgentDataReaders
#TurnOn: SQLAgentTransactions
#TurnOn: SQLAgentConnections
```

また、既存のトレーサ グループにクラスを追加することで追跡をカスタマイズすることもできます。

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。

以下の手順に従います。

1. *default-full.pbl* または *default-typical.pbl* で使用しているファイル タイプに応じて、*toggles-full.pbd* または *toggles-typical.pbd* を開きます。 これらのファイルは、*<Agent_Home>* ディレクトリ内にあります。
2. オンまたはオフにするトレーサ グループを見つけます。
3. 行の先頭にシャープ記号（#）を追加または削除することで、その行をコメント化またはコメント化を解除できます。 以下の例は、トレーサ グループをオンまたはオフにしたディレクティブを示しています。

   ```
   TurnOn: SocketTracing
   ```

   このトレーサ グループはオンになっています。行のコメント化が解除されています。

   ```
   #TurnOn: SocketTracing
   ```

   このトレーサ グループはオフになっています。行がコメント化されています。
4. *toggles-full.pbd* または *toggles-typical.pbd* を保存します。

   設定が完了しました。

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 メトリックの無効化

デフォルトでは、.NET Agent は個々のソケットの入出力帯域幅メトリックを収集するように設定されています。 ソケット別の帯域幅を追跡するメトリックによって、オーバーヘッドが高くなる可能性があります。ソケット レベル ネットワーク メトリックの収集が多くのプロセッサまたは I/O 時間を消費する場合は、ソケット メトリックの収集を完全にオフにすることができます。

ソケット メトリックのレポートをオフにする方法

1. *IntroscopeAgent.profile* ファイルをテキスト エディタで開きます。
2. Agent Socket Rate Metrics セクションを探します。
3. *introscope.agent.sockets.reportRateMetrics* プロパティを *false* に設定します。 例：
   ```
   introscope.agent.sockets.reportRateMetrics=false
   ```
4. *IntroscopeAgent.profile* ファイルを保存して閉じます。

.NET エージェントのログ オプションの構成

以下のセクションでは、.NET エージェントを冗長モードで実行する方法、およびエージェントのログ ファイルのオプションを設定する方法を説明します。 これらの機能について、Introscope の .NET エージェントでは、Log4net の機能が使用されます。 ほかの Log4net 機能を使用する場合は、http://logging.apache.org/log4net/release/features.html にある Log4net のドキュメントを参照してください。

冗長モードでの .NET Agent の実行

.NET Agent を冗長モードで実行すると、ログ ファイルに詳細な情報が記録されます。 これはデバッグ時に役立ちます。

.NET Agent を冗長モードで実行する方法

1. .NET Agent を停止します。
2. *logging.config.xml* ファイルを開きます。
3. *level value* 属性を **VERBOSE** に変更します。 デフォルトは *INFO* です。
   ```
   <root>
   <level value="VERBOSE" />
   <appender-ref ref="logfile" />
   <appender-ref ref="console" />
   </root>
   ```
4. *logging.config.xml* ファイルを保存して、.NET Agent を再起動します。

.NET Agent のログ ファイルの場所の変更

ログ ファイルはデフォルトで、*<Agent_Home>¥logs* ディレクトリに書き込まれます。通常は、*C:¥Program Files¥CA Wily¥Introscope<version>¥wily¥logs* です。ここで、*<version>* はインストールしてある Introscope のバージョンです。ログ ファイル使用上の便宜に応じて、.NET Agent ログ ファイルの場所を変更できます。

.NET Agent ログ ファイルの場所を変更する方法

1. .NET Agent を停止します。
2. *logging.config.xml* ファイルを開きます。
3. **file value** 属性を、ログ ファイルを置きたい場所に変更します。たとえば、以下のとおりです。
   ```
   <file value="c:¥introscope_logs¥IntroscopeAgent.log" />
   ```
4. *logging.config.xml* ファイルを保存します。
5. .NET Agent を再起動します。

エージェントの名前を自分で設定した場合は、名付けたエージェントのログが、デフォルトの場所または指定した新しい場所の *logs* ディレクトリに書き込まれます。

.NET Agent のログ ファイルおよびエージェントの自動ネーミング

デフォルトで、.NET Agent には自動的に名前が付けられます。.NET Agent の名前が自動的に付けられると、そのエージェントに関連するログ 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。 エージェントによって生成されるログ ファイルには、インスツルメンテーション プロセス中に使用された PBD および挿入されたプローブに関する情報が記録されます。 自動名前付けを使用する場合、デフォルトでは、作成されるログ ファイルには、タイム スタンプを使用した名前が最初に付けられます。 例：

```
AutoProbe20060928-175024.log
```

エージェント名を利用できるようになると、ログ ファイルの名前はエージェント名を含んだものに変更されます。 たとえば、エージェント名が *MyDomain//MyAgent* である場合、*MyDomain* がドメイン、*MyAgent* がインスタンスです。

```
AutoProbeMyDomain_MyStuff.log
```

ログ ファイルに実際の名前ではなく、タイム スタンプ名が付けられている場合は、エージェント名が付けられる前にプロセスがタイムアウトした可能性があります。 また、高度な Log4Net 機能を使用すると、自動名前付け機能は動作しません。

注: .NET Agent 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、NativeProfiler は自身のログ ファイルに書き込むことができなくなります。

ログ ファイルの自動名前付けを無効にする場合は、エージェント プロファイル内の introscope.agent.disableLogFileAutoNaming (P. 190) プロパティを true に設定します。

デフォルトドメインのログ

デフォルト ドメインは Enterprise Manager に接続せず、メトリックもレポートしません。また、それ自体ではアプリケーションを実行することもありません。 ただし、デフォルト ドメイン内にある .NET Agent では、デフォルト ドメインにホストされているすべてのアプリケーション ドメインのバイト コードのインスツルメンテーションがすべて処理されるため、ここではログ ファイルが生成されます。 生成されるログ ファイルの 1 つである *AutoProbe.DefaultDomain.log* には、デフォルト ドメインで行われるバイト コードのインスツルメンテーションに関する情報が含まれています。 バイト コードのインスツルメンテーションはすべてデフォルト ドメインで行われるため、これらのログ ファイルにはバイト コードのインスツルメンテーションに関する重要な情報が含まれています。

デフォルト ドメインでは、.NET Agent の *IntroscopeAgent.DefaultDomain.log* ファイルも生成されます。

第 5 章: ProbeBuilder ディレクティブの操作

エージェントはデフォルトで、.NET アプリケーションの多数のコンポーネントを監視します。 ただし、Introscope を最大限に活用するため、少なくともアプリケーション固有のクラスおよびメソッドのいくつかはインスツルメントするのが一般的です。 このセクションでは、ProbeBuilder ディレクティブ キーワードの操作およびカスタム PBD ファイルの作成について簡単に説明します。

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 にほかの文字（Unicode 文字など）を挿入すると、問題が発生する可能性があります。

このセクションには、以下のトピックが含まれています。

既存のトレーサ 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 グループの一部として識別するには、Identify キーワードのいずれかを使用します。

たとえば、クラス *System.EnterpriseServices. ServicedComponent* をトレーサ グループ *ServicedComponentTracing* に追加するには、以下のように記述します。

```
IdentifyClassAs:
System.EnterpriseServices.ServicedComponent ServicedComponentTracing
```

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 コレクションをさらに細かく設定することができます。 アプリケーション固有の情報を追跡するトレーサ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、速度、数など）
- この情報を示すメトリックの完全修飾名（リソース パスを含む）

作成したカスタム .pbd は、デフォルトの .pbd と同様に扱われます。 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。

セクション トピック

一般的なカスタム追跡の例

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* が、「petshop.catalog.Catalog」というクラスの「search」と呼ばれるメソッドの追跡に設定されているとします。Introscope Investigator で BlamePoint メトリックが表示されるノードの名前は「MSPetShop|Catalog|search」です。この場合、以下の記述となります。

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

トレーサ構文

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。 追跡ディレクティブで使用される引数は、*<トレーサ グループ>*、*<クラス>*、*<メソッド>*、*<トレーサ名>*、および *<メトリック名>* 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>

カスタム追跡に利用できる主なディレクティブは、以下の 6 つです。

- **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。

注: 実装された具象メソッドのみが、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。

- **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かどうか、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かどうか、すべてのメソッドを追跡します。

<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
```

注: 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（例、*MyMethod*）

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（例： *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. 105）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。 トレーサ名および追跡対象項目の説明については、「トレーサ名」 (P. 101)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。

トレーサ名

以下の表は、トレーサ名およびその追跡対象項目を示しています。

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

このトレーサは、スタック トレースを、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 トレースは、本当の例外ではありません。これは、メソッドのスタック トレース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。

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 メソッド トレーサの例

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: My.Name.Space.MyClass MyTracers

誤

IdentifyClassAs: "My.Name.Space.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、「{classname}|Test One Node」というメトリック名は以下のように記述します。

正

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

誤

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

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: petshop.catalog.Catalog search BlamedMethodTimer
"MSPetShop|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: petshop.catalog.Catalog search BlamedMethodRateTracer
"MSPetShop|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: petshop.catalog.Catalog search PerIntervalCounter
"MSPetShop|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。

Introscope Investigator のプレビュー ペインでは、デフォルトの間隔は 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: petshop.cart.ShoppingCart placeOrder
BlamedMethodTraceIncrementor "MSPetShop|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と結合しています。

```
TraceOneMethodOfClass: petshop.account.LoginComponent login MethodTraceIncrementor
"MSPetShop|Account:Logged In Users"
TraceOneMethodOfClass: petshop.account.LogoutComponent logout
MethodTraceDecrementor "MSPetShop |Account:Logged In Users"
```

高度なカスタム トレーサの作成

以下のセクションでは、単一メトリック トレーサ、Skip、およびカスタム トレーサの結合など、高度なカスタム トレーサの作成について説明します。

セクション トピック

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;[mscorlib]System.Void([mscorlib]System.Int32)* は、*int* 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。

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。 以下の表は、パラメータおよびそれらの実行時の代入の一覧です。

パラメータ	実行時の代入
{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
{namespacename}	追跡されるクラスの実行時ネームスペース名
{namespaceandclassname}	追跡されるクラスの実行時ネームスペース名およびクラス名
{assemblyname}	追跡されるアセンブリの名前
{fullclassname}	アセンブリ名を含む完全なクラス名をレポートする

注: Introscope では、*ネームスペース*を持たないクラスの処理では、*{namespacename}* が「*<Unnamed Namespace>*」という文字列に置き換えられます。

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。

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

また、トレーサが、*myNamespace* 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。

```
"myNamespace|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。

```
"{namespaceandclassname}|{method}:Response Time (ms)"
```

これが例 1 と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。

```
"myNamespace.myClass|myMethod:Response Time(ms)"
```

注: この例では、 *namespace* と *class* の間に、例 1 の | （パイプ記号）ではなく、.（ピリオド）が 使用されています。

メトリック名ベースのパラメータ

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。 このメトリック名は、以下の形式を使用した、TraceOneMethodWithParametersofClass キーワードを使用してメソッドに渡されるパラメータに基づいています。

```
TraceOneMethodWithParametersOfClass: <クラス> <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。 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ます。 プレースホルダには文字列「{#}」を使用します。# 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: パラメータがどのような文字列に変換されるかがはっきりしない場合は、そのパラメータをメトリック名で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;
[mscorlib]System.Void([mscorlib]System.Int32) 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

スキップ ディレクティブを使用することで、特定のパッケージ、クラス、またはメソッドをインスツルメントしないようにすることができます。スキップ ディレクティブを使用すると、ProbeBuilder ではネームスペース、クラス、またはアセンブリがスキップされます。

さらに、条件に一致する範囲を小さくすることにより、インスツルメントされるパッケージ、クラス、またはメソッドの量を制限することもできます。 この調整は、*IdentifyAllClassAs* または *TraceAllMethodsIfFlagged* ディレクティブをより範囲の狭いものに置き換えることで実行できます。

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。

以下の例は、「*Logged-in Users*」という名前のメトリックを作成します。クラス「*user*」、メソッド「*login*」および「*logout*」を使用して、以下のトレーサを作成します。

```
TraceOneMethodOfClass user login MethodTraceIncrementor "Logged-in Users"
TraceOneMethodOfClass user logout MethodTraceDecrementor "Logged-in Users"
```

これは、任意のユーザがログインしたときにメトリック「*Logged-in Users*」を増分し、ログアウトしたときに「*Logged-in Users*」を減分します。

特定のトレーサに関する注意事項

以下の識別子とトレーサは、.NET 環境に特有の動作をします。

- *IdentifyAnnotatedClassAs:* <attribute-class-name> <Tracer-group>

  指定の属性クラスの注釈が付けられたクラスをすべて指定のトレーサグループに関連付けます。

  一部のクラスには、属性クラスの注釈を付けてクラスに追加機能を持たせることができます。 以下の例では、*System.EnterpriseServices.Transaction* という属性クラスがクラス *ServicedComponent* に付加されています。

```
[Transaction]
Public class ServicedComponent {
}
```

.pbd ファイルでは以下のように記述します。

```
IdentifyAnnotatedClassAs: System.EnterpriseServices.Transaction MyTracerGroup
```

これにより、*[Transaction]* の注釈があるクラスがすべて、*ServicedComponent* も含め、識別されます。

注: この識別子を使用した場合、Introscope .NET Agent では継承された属性は追跡されませんが、ベース クラスに適用された属性は追跡されます。

- *TraceAnnotatedMethodsIfFlagged*: <Tracer-group> <attribute-class-name> <Tracer-name> <metric-name>

  指定のトレーサ グループに関連付けられているクラスについて、指定のクラスにより注釈を付けられているメソッドをすべて追跡します。

明示的なインターフェースの実装

.NET Agent では、.pbd ファイルによる明示的なインターフェースの実装が使用されます。 あるクラスのメソッドを追跡する場合に、そのメソッド名が、他のクラスで使用される他のメソッドと同じ場合は、追跡するメソッドとそのメソッドが所属するインターフェースを明示的に指定する必要があります。 たとえば、*InterfaceA* と *InterfaceB* の両方に *MethodX* というメソッドがある場合、*InterfaceA* の *MethodX* を呼び出すには、*InterfaceA.MethodX* のようにインターフェースとメソッドの両方を指定する必要があります。

以下に、明示的なインターフェースの実装を使用して、あるクラスのメソッドを追跡する例を示します。

```
SetFlag: customInterfaceTracing
TurnOn: customInterfaceTracing
IdentifyInheritedAs: EdgeCaseInterface customInterfaceTracing
TraceOneMethodIfFlagged: customInterfaceTracing EdgeCaseInterface.method2
BlamePointTracer "Interface|{namespaceandclassname}|{method}"
```

インスツルメントおよび継承

Introscope では、クラス階層の下位レベルのクラスは自動的にはインスツルメントされません。 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* を識別する必要があります。

ProbeBuilder ディレクティブの適用

ProbeBuilder ディレクティブ ファイルを実装する準備が整った段階で、新しいファイルを実装する方法として以下の 3 つがあります。

- hotdeploy ディレクトリの使用（P. 110）
- <Agent_Home>/wily ディレクトリの使用（P. 111）
- カスタムの場所とアクセス権（P. 112）

hotdeploy ディレクトリの使用

hotdeploy ディレクトリを使用すると、Introscope の管理者は、*IntroscopeAgent.profile* を編集することなく、また場合によってはアプリケーションを再起動することもなく、新しいディレクティブをより迅速かつ簡単にデプロイできます。 この機能を使用する場合は注意が必要です。カスタム PBD に無効な構文が含まれていたり、非常に多くのメトリックを収集するよう設定されていたりすると、即座に影響を及ぼします。PBD が無効である場合は NativeProfiler のシャットオフを引き起こす可能性があります。また、非常に多くのメトリックを収集する PBD はアプリケーションのパフォーマンスに影響を与えます。 これを解決するため、以下のことを行うようお勧めします。

- 実運用環境に導入する前に、すべてのディレクティブを QA 環境およびパフォーマンス環境でテストおよび検証する。
- PBD をデプロイするため、サーバ環境の変更管理プロセスが更新されており、新しいオプションが反映されていることを確認する。

新しい PBD が hotdeploy ディレクトリに格納されると、.NET Agent によって自動的にこの新しい PBD がデプロイされます。ただし、アプリケーションが再起動されるまでは、すでに実行されているクラスおよびアプリケーションに新しい PBD または変更された PBD の影響が及ぶことはありません。 新しい PBD がこのディレクトリに格納されていれば、IntroscopeAgent.profile を編集して新しい PBD または変更された PBD を指定する必要はありません。

hotdeploy ディレクトリを使用して .pbd を適用する方法

- カスタムまたは変更したファイル（.pbd および .pbl）を *<Agent_Home>¥wily¥hotdeploy* ディレクトリにコピーします。

<Agent_Home>/wily ディレクトリの使用

新しいまたは変更した PBD と PBL をデプロイするには、該当するファイルを、*introscope.autoprobe.directivesFile* プロパティ内に追加し、*IntroscopeAgent.profile* ファイルと同じディレクトリまたは *IntroscopeAgent.profile* ファイルの場所を起点とする相対ディレクトリに配置する必要があります。

その他のディレクトリにファイルを配置する場合は、*introscope.autoprobe.directivesFile* プロパティを設定するときにファイルのフル パスを指定する必要があります。

新しいまたは変更された .pbd および .pbl ファイルをデプロイする方法

1. カスタムまたは変更したファイル（.pbd および .pbl）を *<Agent_Home>* ディレクトリにコピーします。
2. *IntroscopeAgent.profile* ファイルの *introscope.autoprobe.directivesFile* プロパティを更新して、新しいディレクティブ ファイルの名前をカンマで区切って追加します。

   たとえば、カスタムの *petstore.pbd* ファイルをプロパティに追加します。

   ```
   introscope.autoprobe.directivesFile=default-full.pbl,petstore.pbd,hotdeploy
   ```
3. *IntroscopeAgent.profile* ファイルを保存して閉じます。
4. アプリケーションまたは IIS サービスを再起動します。

注: 既存の .pbl または .pbd ファイルによって制御される監視を無効にする場合を除き、プロパティで定義された既存の *.pbl* または *.pbd* ファイルは削除しないでください。

カスタムの場所とアクセス権

上記で説明した *hotdeploy* ディレクトリまたは *wily* ディレクトリを使用する方法に加えて、これらのディレクトリとは異なる任意の場所を選択して PBD を置くこともできます。

.pbd ファイルをカスタムの場所に置く場合は、*IntrosocopeAgent.profile* で .pbd ファイルの場所を指定する必要があります。 たとえば、*leakhunter.pbd* を *C:* ドライブのカスタムの場所に配置した場合は、*introscope.autoprobe.directivesFile* プロパティを以下のように更新します。

```
introscope.autoprobe.directivesFile=default-full.pbl,C:¥¥sw¥¥leakhunter.pbd
```

.pbd をカスタムの場所に置く場合は、IIS プロセスを開始するユーザに、その場所（上記の例では *C:¥¥sw*）に対する適切な権限が付与されている必要があります。 IIS プロセスを開始するユーザに、この場所に対する権限が付与されていない場合、デフォルト ドメインのログにエラー メッセージが記録され、カスタムの場所にある .pbd は有効になりません。

重要: PBD を hotdeploy ディレクトリに配置することを強くお勧めします。

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

Introscope Workstation からトランザクション追跡をするときは、動的インスツルメンテーションを実行するオプションがあります。 動的インスツルメンテーションでは、PBD ファイルを作成したり、エージェント プロファイルを変更したりせずに、実行時にインスツルメントする 1 つ以上のメソッドを選択できます。 動的インスツルメンテーションでは、以下を実行できます。

- トランザクション コンポーネントに属する、1 つ、複数、またはすべての呼び出されたメソッドを表示およびインスツルメントする。
- 一時的にインスツルメントされたメソッドでの追跡を表示する。
- 一時的にインスツルメントされたメソッドで収集されたメトリックを表示する。
- 一時的なインスツルメンテーションを永続化する。

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することを強くお勧めします。

注: .NET オペレーティング環境のサポートは制限されています。詳細については、「CA APM Workstation ユーザ ガイド」の「トランザクション追跡」を参照してください。

詳細:

動的インスツルメンテーション (P. 226)

第 6 章: LeakHunter の構成

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。

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 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。 次に、管理対象アプリケーションを再起動します。

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。

.NET 環境での LeakHunter の追跡対象

以下の .NET 特殊コレクションが追跡されます。

セクション情報

特殊コレクション (P. 117)
特殊インターフェース (P. 117)
ジェネリック コレクション (P. 118)
ジェネリック インターフェース (P. 119)

特殊コレクション

- System.Collections.ArrayList
- System.Collections.BitArray
- System.Collections.CollectionBase
- System.Collections.DictionaryBase
- System.Collections.Hashtable
- System.Collections.Queue
- System.Collections.SortedList
- System.Collections.Stack
- System.Collections.Specialized.HybridDictionary
- System.Collections.Specialized.ListDictionary
- System.Collections.Specialized.NameObjectCollectionBase
- System.Collections.Specialized.NameValueCollection
- System.Collections.Specialized.StringCollection
- System.Collections.Specialized.StringDictionary
- System.Collections.Specialized.OrderedDictionary

特殊インターフェース

- System.Collections.ICollection
- System.Collections.IDictionary
- System.Collections.IList
- System.Collections.Specialized.IOrderedDictionary

LeakHunter では、.NET の System.Collections の IList、ICollection、および IDictionary の実装が追跡されます。

ICollection、IDictionary、および IList の詳細については、以下を参照してください。

- IList の実装：
  http://msdn2.microsoft.com/en-us/library/system.collections.ilist(VS.71).aspx
- ICollection の実装：
  http://msdn2.microsoft.com/en-us/library/system.collections.icollection(VS.71).aspx
- IDictionary の実装：
  http://msdn.microsoft.com/en-us/library/system.collections.idictionary%28VS.71%29.aspx

LeakHunter では、System.Collections.Specialized の IOrderedDictionary のインスタンスも追跡できます。ただし、.NET Framework では IOrderedDictionary は実装されていないため、アプリケーションに実装された IOrderedDictionary のみが追跡されます。

ジェネリック コレクション

- System.Collections.Generic.List
- System.Collections.Generic.SortedList
- System.Collections.Generic.Dictionary
- System.Collections.Generic.SortedDictionary
- System.Collections.Generic.LinkedList
- System.Collections.Generic.Queue
- System.Collections.Generic.Stack
- System.Collections.Generic.SynchronizedKeyedCollection
- System.Collections.Generic.SynchronizedCollection
- System.Collections.Generic.KeyedByTypeCollection
- System.Collections.Generic.HashSet
- System.Collections.ObjectModel.KeyedCollection
- System.Collections.ObjectModel.Collection
- System.Collections.ObjectModel.ObservableCollection

ジェネリック インターフェース

- System.Collections.Generic.IList
- System.Collections.Generic.ICollection
- System.Collections.Generic.IDictionary

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

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 の有効化方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. LeakHunter Configuration の見出し以下で、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. IntroscopeAgent.profile のプロパティ introscope.autoprobe.directivesFile にリストした、*.typical.pbl または *.full.pbl ファイルを開きます。
5. leakhunter.pbd のコメントを解除します。

6. *.typical.pbl または *.full.pbl ファイルを保存し閉じます。
7. アプリケーションを再起動します。

注: 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>¥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。 拡張機能のディレクトリの場所を変更する場合は、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 の無効化方法

1. エージェント プロファイル *IntroscopeAgent.profile* を開きます。
2. LeakHunter Configuration の見出し以下で、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、*<Agent_Home>/wily* ディレクトリにあるエージェント プロファイル *IntroscopeAgent.profile* にあります。

LeakHunter の設定方法

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
   - introscope.agent.leakhunter.logfile.append
   - introscope.agent.leakhunter.leakSensitivity
   - introscope.agent.leakhunter.timeoutInMinutes
   - introscope.agent.leakhunter.collectAllocationStackTraces
   - introscope.agent.leakhunter.ignore.0

   LeakHunter プロパティの詳細については、「LeakHunter プロパティ」(P. 232)を参照してください。

   重要: IntroscopeAgent.profile には、LeakHunter が無視するパッケージを制御するデフォルトのプロパティが含まれています。 通常、これらのプロパティはパフォーマンスの問題を引き起こすコレクションを無視するように設定されています。 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。

そのようなコレクションは、IntroscopeAgent.profile の ignore プロパティを使用して無視するようにしてください。

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 の実行方法

1. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
2. *introscope.agent.leakhunter.enable* プロパティを true に設定し、必要に応じてその他の LeakHunter プロパティも設定します。
3. 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ができます。 コレクション ID は、アプリケーション間で不変的な名前を提供することもできます。

コレクション ID は以下の形式を持ちます。
<メソッド>-<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
- <4 桁のハッシュ コード>: メソッド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。

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 のログ 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格納されます。各エントリに、1 つの潜在的リークに関する情報が記述されます。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. 123）」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. 124）」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（P. 125）」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（P. 125）」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。

.NET のログ ファイルの例: 潜在的なリークが初めて特定されたとき

この例では、インスツルメンテーション メソッドとして AutoProbe を使用している場合を想定しています。

```
10/18/2007 11:54:47 AM
Potential leak identified
Assigned ID: createNewInstance-2975#1
Collection Class:
System.Collections.Generic.LinkedList`1[[com.wily.tools.munger.ContainedObject,
munger, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null]]
Allocation Method:
[munger]com.wily.tools.munger.MungerFactory.createNewInstance[mscorlib,
Version=2.0.0.0,
PublicKeyToken=b77a5c561934e089]System.Object([mscorlib]System.String)
Allocation Timestamp: 10/18/2007 11:54:47 AM
Allocation Stack Trace:
Field Name(s):
    Unknown
Current Size: 10500
```

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

.NET のログ ファイルの例: 潜在的リークが停止したと判断されたとき

```
10/18/2007 11:55:47 AM
Potential leak no longer appears to be leaking
Assigned ID: createNewInstance-2975#1
Collection Class:
System.Collections.Generic.LinkedList`1[[com.wily.tools.munger.ContainedObject,
munger, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null]]
Current Size: 28000
```

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

.NET のログ ファイルの例: 潜在的なリークが再開したと判断されたとき

```
10/18/2007 11:57:47 AM
Potential leak appears to be leaking again
Assigned ID: createNewInstance-2975#1
Collection Class:
System.Collections.Generic.LinkedList`1[[com.wily.tools.munger.ContainedObject,
munger, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null]]
Current Size: 35000
```

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。

ログ ファイルの例: タイムアウトが発生したとき

```
LeakHunter timeout occurred at 4/27/04 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。

第 7 章: ErrorDetector の構成

このセクションには、以下のトピックが含まれています。

ErrorDetector の概要

Introscope ErrorDetector 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。

セクション情報

エラーの種類 (P. 128)
ErrorDetector の仕組み (P. 129)

エラーの種類

CA Technologies では、.NET 仕様に含まれる情報に基づき、深刻なエラーを示す一定の基準を定めています。 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）

  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 による、エージェントを通じた Web サーバ エラーの検出は行われません。
- 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。

Introscope 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
- 新しい Error Snapshot 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 CA Wily のトランザクション 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。

.NET Agent での ErrorDetector の有効化

デフォルトで、.NET Agent はエラー データのキャプチャが有効になっています。 *IntroscopeAgent.profile* でプロパティ *introscope.agent.errorsnapshots.enable* を *false* に設定することにより、エラー データ キャプチャを無効にすることができます。*errors.pbd* ファイルは、デフォルトで .NET Agent の *default-full.pbl* ファイルおよび *default-typical.pbl* ファイルに登録されており、追加の設定は必要ありません。

ErrorDetector オプションの設定

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。

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。 このトレーサーは、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 .ctor
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。

HTTPErrorCodeReporter

HTTPErrorCodeTracer トレーサは、ASP.NET ページのエラー コードをレポートします。 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
- .NET 環境でのコード 400 以上に対する、ProcessRequest の *System.Web.IHttpHandler* 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。

警告

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、「APM Workstation ユーザ ガイド」を参照してください。

第 8 章: Boundary Blame の設定

このセクションには、以下のトピックが含まれています。

Boundary Blame の理解

Introscope の Blame テクノロジを使用すると、管理対象 .NET アプリケーション内の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）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、問題の原因をアプリケーションのフロントエンドとバックエンドに切り分けることができます。

このセクションでは、Introscope の Boundary Blame 機能の構成オプションについて説明します。

Introscope では、SQL エージェントの SQL ステートメント監視機能を使用して、バックエンドが自動的に検出されます。 SQL エージェントを使用できない場合は、ソケット コールがバックエンドとして自動的に検出されます。たとえば、ソケットを通じてアクセスされるクライアント/サーバ データベース、LDAP サーバなどです。

Boundary Blame が Introscope Investigator でどのように表示されるかについては、「Introscope 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 グループは、URL パスのプリフィックスに関して定義するトランザクションの名前付きグループです。 Introscope では、各 URL グループのメトリックが集約され、それらのメトリックが Introscope Investigator で *Frontends|Apps|<アプリケーション名>|URLs* ノードの下に表示されます。

パスのプレフィクスは、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
```

URL グループは、1 つの URL のパス プレフィックスから派生しうる要求を分類する任意の有用なカテゴリについて定義できます。 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パフォーマンス、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の監視ができます。

デフォルトでは、すべての URL が「*デフォルト*」と呼ばれる URL グループに割り当てられます。 これにより、無効な URL が、たとえば 404 エラーを発生させ、一意で 1 回限りのメトリックを作成するようなオーバヘッドを発生しないようにできます。 意味のある URL グループを設定すると、ユーザはサブアプリケーション レベルで性能のモニタを行うことができます。

関連情報

URL グループのプロパティ (P. 136)
サンプル URL グループ (P. 137)
URL グループの定義 (P. 137)
URL グループの高度な名前付け手法（オプション） (P. 139)

URL グループのプロパティ

IntroscopeAgent.profile ファイルで URL グループを定義するには、以下のプロパティを使用します。

- *introscope.agent.urlgroup.***keys**
- *introscope.agent.urlgroup.***group.default.pathprefix**
- *introscope.agent.urlgroup.***group.default.format**

サンプル URL グループ

以下のリストは、エージェント プロファイルの抜粋で、URL グループがどのように定義されるかを示しています。 リストの後のセクションでは、必須プロパティの設定の手順を説明します。

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
introscope.agent.urlgroup.group.alpha.pathprefix=/testWarintroscope.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=/nofilterWarintroscope.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=/examplesWebAppintroscope.agent.urlgroup.group.gamma.format=Examples Web App
```

URL グループの定義

以下のセクションでは、URL グループを設定するプロパティについて説明します。

セクション情報

URL グループのキーの定義 (P. 137)
各 URL グループのメンバシップの定義 (P. 138)
URL グループの名前の定義 (P. 138)

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 すなわち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
```

URL グループのキーを指定する順序は重要です。 詳細については、「URL のグループ化 (P. 282)」を参照してください。

各 URL グループのメンバシップの定義

*introscope.agent.urlgroup.group.*GroupKey.*pathprefix* を使用して、URL のパスプレフィックスの照合先とするパターンを指定して、URL グループに属する要求を定義します。

例 1

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 2

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/
```

ここで、*application_function* は、*OrderEntry*、*AcctService*、*Support* などのアプリケーションに対応し、各 URL グループのパスのプレフィックスのプロパティに、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

introscope.agent.urlgroup.group.GroupKey.format を使用して、キーが *GroupKey* である URL グループの応答時間メトリックが Workstation で表示されるときの表示名を指定します。

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、キーが *alpha* である 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手法（オプション）

必要に応じて、URL グループ名を、サーバ ポートなどの要求エレメント、プロトコル、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。 以下のセクションでは、*format* プロパティの高度な形式について説明します。

セクション情報

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、以下の要求の統計情報は、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} とすると、Investigator に表示される統計情報は、要求 URL のプロトコル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} とすると、統計情報は、要求 URL のポートの部分に対応した名前の下に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 に表示される URL グループの統計情報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} とすると、Investigator に表示される統計情報は、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}
```

以下の要求に対する統計情報は、「734」というメトリック名で表示されます。

```
http://ubuy.com/ws/shopping?ViewItem&category=734&item=3772&tc=photo
http://ubuy.com/ws/shopping?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。 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}
```

以下の要求に対する統計情報は、「*/ht*」というメトリック ノード名で表示されます。

```
http://research.com/htmldocu/WebL-12.html
```

URL グループ名に要求パスの文字区切り部分を使用

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* の下に表示されます。

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。

以下の表は、上記の例のスラッシュ文字で区切られたセグメントを示します。

セグメント インデックス	0	1	2	3
セグメント文字列	/	*userid,sessionid*	/	*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。

以下の表は、上記の例のスラッシュおよびカンマ文字で区切られたセグメントを示します。

セグメント インデックス	0	1	2	3	4	5
セグメント文字列	/	*userid*	,	*sessionid*	/	*pageid*

URL グループに複数の名前付け手法を使用

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
```

Blame トレーサを使用した Blame ポイントのマーク付け

Introscope Blame テクノロジで.NET アプリケーションのパフォーマンスを追跡すると、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、問題の原因をアプリケーションのフロントエンドとバックエンドに切り分けることができます。

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。

フロントエンドとバックエンドのメトリック情報は、アプリケーションに関する情報をアプリケーション問題切り分けマップに表示する際にも使用されます。

セクション トピック

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（FrontendTracer および BackendTracer）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。

FrontendTracer および BackendTracer 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 がカスタム コンポーネントのメトリックをキャプチャしたり、Investigator ツリーに表示することができます。

FrontendTracer が設定されていない場合は、Blame スタックの 1 つ目のコンポーネントがデフォルト フロントエンドになります。 BackendTracer 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。

特定のクラスがフロントエンドとしてマークされないようにするために、PBD パラメータ **is.frontend.unless** を指定することができます。詳細については、「カスタム FrontendMarker ディレクティブ (P. 144)」を参照してください。

BackendTracer を使用して、Introscope がバックエンドとして検出する項目に望ましい名前を割り当てたり、Introscope がインスツルメントしないカスタムのソケットにマークを付けることは有益です。

FrontendTracer および BackendTracer は、平均応答時間、指定間隔あたりのカウント、並行処理、およびストールなどのメトリックを提供する BlamePointTracer のインスタンスです。BlamePointTracer を「中間」コンポーネントに適用すれば、より細かく Blame スタックを調整できます。 ただし、BlamePointTracer には、Introscope Investigator の［間隔ごとのエラー数］メトリックはありません。

標準 PBD での Blame トレーサ

Introscope および .NET Agent で提供される 2 つの標準 PBD （dotnet.pbd および sqlagent.pbd）は、Boundary Blame 追跡を実装します。

- *dotnet.pbd* 内の *PageInfoTracer* は、FrontendMarker のインスタンスです。
- *sqlagent.pbd* 内の *SQLBackendTracer* は、BackendTracer のインスタンスです。

カスタム FrontendMarker ディレクティブ

Introscope 9.0 では、PBD パラメータ **is.frontend.unless** が導入されました。このパラメータを使用して、FrontendMarker（または PageInfoTracer などのサブクラス）によってインスツルメントされた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。パラメータは、絶対クラス名を記載したカンマ区切りのリストとして設定する必要があります。これは、最初のコンポーネントが汎用のディスパッチャであり、受信した要求タイプを処理するために、特定のコンポーネントに要求を転送する場合に便利です。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。そのため、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: クラス名の区切り記号としては、スペースを**使用せず**、カンマのみを使用してください。 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、このクラスはフロントエンドとして指定されません。また、このクラスは Introscope Investigator のフロントエンド ノードの下にメトリックを生成しません。

たとえば、*ASP.index_aspx* クラスおよび *Fib.CalculatorController* クラスがパッケージ *com.ABCCorp* 内で PageInfoTracer という名前の *FrontendMarker* でインスツルメントされたフロントエンドとして扱われないようにするには、以下の PDB ディレクティブを使用します。

```
SetTracerParameter: PageInfoTracer is.frontend.unless
ASP.index_aspx,Fib.CalculatorController
```

第 9 章: Transaction トレーサのオプション

このセクションには、以下のトピックが含まれています。

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

拡張機能名	レガシー標準 pbl または pbd ファイル
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

トランザクション追跡サンプリングの制御

デフォルトでは、Introscope エージェントは、1 時間に 1 回、アプリケーション内の正規化された一意の URL をそれぞれ追跡して、トランザクションの動作をサンプリングします。 このサンプリングによって、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。

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。 この動作を無効にするには、エージェント プロファイル IntroscopeAgent.profile の以下のプロパティのコメント化を解除します。

```
introscope.agent.transactiontracer.sampling.enabled
```

コメント化を解除して 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。

間隔ごとにサンプリングするトランザクションの数と、その間隔を設定するには、エージェント プロファイルの以下のプロパティのコメント化を解除します。

重要: これらの設定は通常、Enterprise Manager で行われます。 エージェント プロファイルで以下のプロパティを構成すると、Enterprise Manager で行われた構成はすべて無効になります。

```
introscope.agent.transactiontracer.sampling.perinterval.count
```

コメント化を解除して、間隔ごとにサンプリングするトランザクションの数を設定します。 デフォルトは 1 です。

```
introscope.agent.transactiontracer.sampling.interval.seconds
```

コメント化を解除して、サンプリング間隔の長さを秒単位で設定します。 デフォルトは 120 秒です。

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（デフォルトで 5,000 コンポーネントに設定）されています。 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。

これによって、大量のトランザクションによってコンポーネント数が予想以上に増えるようなコンポーネントをクランプすることができます。 たとえば、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が該当します。 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みなし、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、CLR 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
```

CountClamp を超える程度のクランプされるコンポーネントが発生する追跡には、アスタリスクのマークが付けられ、詳細な情報を提供するヒントが関連付けられています。 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。

クランプの値が低すぎると、アプリケーションの起動時にパフォーマンスの監視（PerfMon）または LeakHunter の例外が発生する可能性があります。このような場合は、管理対象 .NET アプリケーションを再起動する必要があります。

Transaction トレーサのオプション

指定した条件を満たすトランザクションのみが追跡されるように、Introscope Transaction Tracer を構成できます。 追跡するトランザクションは、ユーザ ID データまたは HTTP 要求、およびセッション プロパティでフィルタできます。

重要: フィルタは、ユーザ ID *または* HTTP 属性のどちらかで行います。同時に両方を使用できません。 両方のタイプのフィルタを設定すると、メトリックが正しくなくなります。

追跡するトランザクションを制御する方法

1. 「フィルタ パラメータの収集の有効化 (P. 150)」での説明に従って、.NET Agent でフィルタ パラメータがレポートされるように設定します。
2. ユーザ ID または HTTP 要求データによりフィルタを設定します。
   - ユーザ ID によるトランザクション追跡のフィルタ (P. 151)
   - HTTP 要求データによるトランザクション追跡のフィルタ

フィルタ パラメータの収集の有効化

.NET Agent は、デフォルトでは追跡するトランザクションの URL のみをレポートします。 トランザクション追跡がシステムのオーバーヘッドに与える影響を最小にするため、個別の HTTP プロパティをレポートすることは制限されています。 フィルタを有効にするには、まずエージェント プロファイルに以下の行を追加して、HTTP プロパティの収集を有効にします。

```
introscope.agent.asp.disableHttpProperties=false
```

これにより、以下のプロパティの収集が有効になります。

- アプリケーション名
- Session ID
- コンテキスト パス
- Server Name
- URL
- コンテキスト パス
- 正規化された URL
- HTTP ヘッダ
- HTTP パラメータ
- HTTP 属性

エージェント プロファイルに以下の行を追加すると、そのほかのプロパティの収集が有効になります。

```
introscope.agent.asp.optionalProperties=true
```

これにより、以下のプロパティの収集が有効になります。

- スキーム
- URL リファラ
- メソッド

ユーザ ID によるトランザクション追跡のフィルタ

.NET エージェントでフロントエンド コンポーネントのユーザ ID を使用してトランザクション追跡がフィルタされるように構成するには、アプリケーションでユーザ ID が特定される方法を確認します。 この情報は、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ます。

Introscope Transaction Tracer は、以下のいずれかの方法でアクセスされるユーザ ID を識別できます。

- HTTP のコンテキスト ID
- HTTP 要求ヘッダ
- URL のユーザ情報
- HTTP セッションの属性

重要: アプリケーションでユーザ ID を特定する方法に適用される設定プロセスのみを実施してください。

セクション情報

コンテキスト ID によるフィルタ

HTTP のコンテキスト ID からユーザ ID にアクセスする場合は、エージェント プロファイルで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=HttpContext.User.Identity.Name
```

URL ユーザによるフィルタ

URL のユーザ情報からユーザ ID にアクセスする場合は、エージェント プロファイルで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=HttpContext.Request.Uri.UserInfo
```

要求ヘッダによるフィルタ

HTTP 要求ヘッダからユーザ ID が決定される場合は、エージェント プロファイルで以下の 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=HttpRequest.Headers.Get
introscope.agent.transactiontracer.userid.key=<application defined key string>
```

セッションの属性によるフィルタ

ユーザ ID が HTTP セッションの属性である場合は、エージェント プロファイルで以下の 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=HttpContext.Session.Contents
introscope.agent.transactiontracer.userid.key=<application defined key string>
```

HTTP 要求データによるトランザクション追跡のフィルタ

以下に示すような HTTP 要求プロパティでトランザクション追跡をフィルタできます。

- 要求ヘッダ
- 要求パラメータ
- セッション属性

複数のパラメータを使用してフィルタできます。たとえば、要求パラメータとセッション属性の両方を使用できます。

重要: ユーザ ID によるフィルタをすでに設定してある場合は、HTTP プロパティによるフィルタは設定しないでください。

以下の手順に従います。

1. IntroscopeAgent.profile ファイルを開きます。
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。
3. 特定の HTTP 要求ヘッダを収集するには、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`
4. HTTP 要求パラメータのデータを収集するには、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`
5. HTTP セッション属性のデータを収集するには、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=attribute1,attribute2`
6. IntroscopeAgent.profile ファイルへの変更を保存します。
7. 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］タブ ビューおよびライブ エラー ビューアで表示できます。または、履歴クエリ ビューアでクエリ「type:errorsnapshot」を実行することで表示できます。ストールしたトランザクションは、ライブ エラー ビューアおよび［エラー］タブにエラーとしてレポートされます。 「エラー検索」条件を使用して追跡した場合、［トランザクション追跡］ウィンドウにストールしたトランザクションはエラーとしてレポートされません。これは正常な動作です。

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、ストールのしきい値を変更したり、.NET エージェントでストールがチェックされる頻度を変更したりするには、以下のプロパティを使用します。

- introscope.agent.stalls.enable： エージェントがストールをチェックし、検出したストールのイベントを作成するかどうかを制御します。
- introscope.agent.stalls.thresholdseconds： 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： 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。

重要: イベントとしてのストールのキャプチャを無効にすると、PBD のストール トレーサのサポートが使用されなくなります。

非識別トランザクションの追跡

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。

非識別トランザクションの追跡を有効にするには、introscopeAgent.profile で以下のプロパティ値を FALSE に設定します。

```
introscope.agent.bizdef.turnOff.nonIdentifying.txn=FALSE
```

第 10 章: Introscope SQL エージェントの設定

このセクションには、以下のトピックが含まれています。

SQL エージェントの概要

Introscope 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。

.NET Agent が .NET Web アプリケーションを監視するのと同様に、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 (?, ?)
```

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/ext/wily.SQLAgent.ext.dll*
- *wily/sqlagent.pbd*

デフォルトで、*wily.SQLAgent.ext.dll* ファイルなどの エージェントの拡張は、*wily/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。 Hibernate のようなテクノロジが使用されているときは、長くて独自の 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。

Hibernate の詳細については、http://www.hibernate.org/ を参照してください。

セクション情報

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_12398139821038192 0912 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. 161）
- カスタム SQL ステートメント ノーマライザ（P. 161）
- 正規表現の SQL ステートメントのノーマライザ
- コマンドラインの SQL ステートメントのノーマライザ

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、ユーザ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 DLL 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 DLL ファイルを作成します。

   SQL ノーマライザ拡張機能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。

DLL 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
   ```
   注: 上記のキーには、一意のノーマライザ名（たとえば normalizer1）が含まれている必要があります。
   ```
   com-wily-Extension-Plugin-testNormalizer1-Entry-Point-Class:
   <Thefully-qualified classname of your implementation of ISQLNormalizer>
   ```
3. 作成した拡張機能ファイルを *<Agent_Home>/wily/ext* ディレクトリに格納します。
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
   ```
   注: 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. 163)」を参照してください。

   注: カスタム ノーマライザ拡張によるエラーがエラー スロットル カウントの数を超えたとき、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。

第 11 章: トラブルシューティングとヒント

このセクションでは、.NET エージェントの作業での一般的な設定の問題、および推奨事項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。

.NET Agent が正しくインストールされているかどうかの確認

場合によっては、特定のコンピュータにおいて .NET Agent がインストールされているのか無効にされているのかがわからなかったり、インストールや設定のエラーに気が付かなかったりすることがあります。.NET Agent がインストールされているかどうかに確信がない場合や、.NET Agent が正しく設定されていることを確認したい場合は、以下の手順に従って環境をチェックできます。

- .NET Agent 環境変数が設定され、正しい値が設定されていることを確認します。

  コマンド ウィンドウを開き、「*set*」と入力して、出力をテキスト ファイルにリダイレクトします。次に、ファイルを参照して、以下の環境変数が設定されていることを確認します。

  ```
  com.wily.introscope.agentProfile=C:¥Program Files¥CA
  Wily¥Introscope9.1.0.0¥wily¥IntroscopeAgent.profile
  Cor_Enable_Profiling=0x1
  COR_PROFILER={5F048FC6-251C-4684-8CCA-76047B02AC98}
  ```

- Windows システム フォルダ内の *assembly* ディレクトリに移動し（*C:¥WINDOWS¥assembly* など）、登録済みのアセンブリ名として *wily.Agent* がリスト表示され、バージョン情報が正しいことを確認します。

  wily.Agent がリスト表示されていない場合は、*wilyregtool.exe* または *gacutil.exe* を使用して、グローバル アセンブリ キャッシュにアセンブリを手動で登録します。

- IIS ワーカ プロセスを実行するユーザ アカウントをチェックし、ワーカ プロセスが実行されるすべてのアカウントが *<Agent_Home>* ディレクトリに対してフル コントロールの権限を持っていることを確認します。

エージェント拡張機能のバージョン情報の修正

.NET エージェントおよび任意のオプションの拡張機能をどのタイミングでインストールするかによって、拡張機能のバージョン番号が .NET エージェントのバージョン番号と異なっていることがあり得ます。通常、バージョン情報が .NET エージェントと拡張機能との間で異なる場合、エージェントではエラー メッセージがログ記録され、拡張機能は正常に動作しません。 バージョン情報を手動で更新することで、この種の問題を修正できます。お使いの環境に応じて、以下のいずれかの操作を行います。

- 個別のアプリケーションが正しいエージェント バージョン番号を使用するように設定します。
- すべてのアプリケーションが正しいエージェント バージョン番号を使用するように一括して設定します。

重要: 選択できるオプションは 1 つのみです。両方のオプションを設定しないでください。

個別のアプリケーションのバージョン情報の設定方法

1. インストールする機能拡張が標準の *.exe* ファイルの場合、*<Extension_Name>.exe.config* という名前のファイルを作成し、このファイルを元の *.exe*.ファイルと同じディレクトリに配置します。
2. インストールする拡張機能が IIS アプリケーションである場合は、*w3wp.exe.config* という名前のファイルを作成し、それを *w3wp.exe* と同じディレクトリに格納します。 これは、デフォルトのドメイン用です。
3. IIS アプリケーションの機能拡張の場合、アプリケーションごとに *web.config* に以下の設定を追加します。

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity name="..." .../>
      <bindingRedirect oldVersion="0.0.0.0 - 65535.65535.65535.65535"
newVersion="<AGENT.VERSION.NUMBER>"/>
   </dependentAssembly>
</assemblyBinding>
```

   assemblyIdentity name を入力し、エージェントのバージョン番号を使用する .NET エージェントのバージョン番号に置き換えます。4 桁で構成されるバージョン情報を使用してください。 たとえば、バージョン **9.0.7** の .NET エージェントをインストールした場合は、以下の記述を追加します。

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity name="wily.Agent"  publicKeyToken="2B41FDFB6CD662A5"/>
      <bindingRedirect oldVersion="9.0.5.0 - 65535.65535.65535.65535"
newVersion="9.0.7.0" />
   </dependentAssembly>
</assemblyBinding>
```

注: 上記のファイルがすでに存在する場合は、*<runtime>* の下に *<assemblyBinding>* ノードを追加します。

グローバルにすべてのアプリケーションを構成する方法

- 通常、アプリケーションの *%runtime install path%¥Config* ディレクトリ内にある *machine.config* ファイルに以下の記述を追加します。

```
assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity name="..." .../>
      <bindingRedirect oldVersion="0.0.0.0 - 65535.65535.65535.65535"
newVersion="<AGENT.VERSION.NUMBER>"/>
   </dependentAssembly>
</assemblyBinding>
```

  assemblyIdentity name を入力し、エージェントのバージョン番号を使用する .NET エージェントのバージョン番号に置き換えます。4 桁で構成されるバージョン情報を使用してください。 たとえば、バージョン **9.0.7** の .NET エージェントをインストールした場合は、以下の記述を追加します。

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity name="wily.Agent"  publicKeyToken="2B41FDFB6CD662A5"/>
      <bindingRedirect oldVersion="9.0.5.0 - 65535.65535.65535.65535"
newVersion="9.0.7.0" />
   </dependentAssembly>
</assemblyBinding>
```

注: このコードスニペットを *machine.config* に追加すると、すべてのアプリケーションにグローバルに変更が適用されます。

hotdeploy ディレクトリの有効/無効の選択

hotdeploy ディレクトリを有効にすると、IntroscopeAgent.profile を編集したり、場合によっては管理対象アプリケーションを再起動することなく新しいディレクティブをデプロイすることができます。 この機能を使用する場合は注意が必要です。 カスタム PBD に無効な構文が含まれていたり、非常に多くのメトリックを収集するよう設定されていたりすると、即座に影響を及ぼします。 PBD が無効である場合は NativeProfiler のシャットオフを引き起こす可能性があります。また、非常に多くのメトリックを収集する PBD はアプリケーションのパフォーマンスに影響を与えます。

これを解決するため、以下のことを行うようお勧めします。

- 実運用環境に導入する前に、すべてのディレクティブを QA 環境およびパフォーマンス環境でテストおよび検証する。
- PBD をデプロイするため、サーバ環境の変更管理プロセスが更新されており、新しいオプションが反映されていることを確認する。

新しい PBD が hotdeploy ディレクトリに配置されると、.NET Agent によって自動的にこの新しい PBD がデプロイされます。 ただし、すでに実行されているクラスおよびアプリケーションは、アプリケーションが再起動されるまでは、新しい PBD または変更された PBD の影響を受けません。

注: hotdeploy ディレクトリには、PBD ファイルのみをデプロイできます。エージェントは、ディレクトリに配置された ProbeBuilder リスト（PBL）はすべて無視します。

hotdeploy ディレクトリの使用を無効にすることで、不正な PBD ファイルが自動的にデプロイされるのを防ぐことができます。

hotdeploy ディレクトリを無効にする方法。

1. hotdeploy ディレクトリに保存されているカスタム PBD をすべて <Agent_Home> ディレクトリに移動します。
2. テキスト エディタで IntroscopeAgent.profile ファイルを開きます。
3. *introscope.autoprobe.directivesFile* プロパティから *hotdeploy* を削除します。
4. 使用するカスタム PBD を *introscope.autoprobe.directivesFile* プロパティに追加します。 例：

   ```
   introscope.autoprobe.directivesFile=default-typical.pbl,custom1.pbd,custom2.pbd,custom3.pbd
   ```
5. *IntroscopeAgent.profile* を保存し、エージェントを再起動します。

付録 A: .NET Agent のプロパティ

このセクションには、以下のトピックが含まれます。

.NET Agent から 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。

エージェント フェールオーバ

以下のプロパティでは、エージェントからプライマリ Enterprise Manager への接続が失われた場合に、エージェントが接続を試みるバックアップ Enterprise Manager、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。

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. 182) 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. 182) 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. 183) 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. 183) 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。

introscope.agent.simpleInstanceCounter.referenceTrackingLimit

SimpleInstanceCounter メトリックは、各クラスに対して作成されたインスタンスの数を追跡します。 インスタンスが大量にあると、.NET Agent で重大なオーバーヘッドが生じる可能性があります。 このプロパティを使用して、各クラスで追跡されるインスタンスの数を制限します。

オプション

整数

デフォルト

25000

例

```
introscope.agent.simpleInstanceCounter.referenceTrackingLimit
=25000
```

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。

エージェント ネーミング

エージェントとログ ファイルの自動名前付けおよびネーミング関連のオペレーションは、制御することができます。

introscope.agent.agentAutoNamingEnabled

エージェントの自動名前付け機能を使用して、サポートされているアプリケーション サーバの .NET Agent 名を取得するかどうかを指定します。

オプション

True または False

デフォルト

True

例

```
introscope.agent.agentAutoNamingEnabled=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。
- アプリケーション サーバが WebLogic の場合は起動クラスが指定されている必要があり、WebSphere の場合はカスタム サービスが指定されている必要があります。
- サポートされているアプリケーション サーバに付属しているエージェント プロファイルでは、True に設定され、コメント化されていません。

introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds

Enterprise Manager に接続する前にネーミング情報を取得するために、エージェントが待機する最大秒数を指定します。

プロパティ設定

遅延秒数を表す正の整数。

デフォルト

デフォルトは 120 秒です。

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

エージェントが、エージェント名が変更されたかどうかを確認する頻度を指定します。 エージェントは、エージェント名が変更されたかどうかを指定した間隔で確認します。

プロパティ設定

間隔の分数を表す正の整数。

デフォルト

デフォルトでは 10 分間隔です。

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.disableLogFileAutoNaming

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かどうかを指定します。 デフォルトでは、エージェントが自動名前付けを使用するように設定されている場合、そのログ ファイルもエージェント名またはタイム スタンプを使用して、自動的に名前が付けられます。 このプロパティを true に設定すると、デフォルトの動作は無効になります。

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。

introscope.agent.agentName

他のすべてのエージェント ネーミング方法が失敗した場合に、エージェントに使用する名前を指定します。 このプロパティの値が無効であるか、またはプロパティがプロファイルから削除されている場合、デフォルトのエージェント名は「Unknown Agent」になります。 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。 ほかのエージェント ネーミング方法が失敗する場合は、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ることができます。

プロパティ設定

他のいずれの方法を用いてもエージェント名を決定できない場合にエージェント名として使用するテキスト文字列。

デフォルト

AgentName

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。

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.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. 195)
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. 203)

エージェント スレッドの優先順位

エージェント スレッドの優先順位は制御することができます。

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

0（低）～ 4（高）の範囲で値は異なります。

デフォルト

デフォルトの優先順位は 2 です。

例

```
#introscope.agent.thread.all.priority=2
```

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. 287)

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。

AutoProbe

エージェントと AutoProbe との連携方法は制御することができます。

重要: 以下のプロパティは必須パラメータです。AutoProbe プロパティが設定されていないか、値が無効である場合、Introscope は機能しません。

introscope.autoprobe.directivesFile

デプロイする ProbeBuilder ディレクティブ（*.pbd*）および ProbeBuilder リスト（*.pbl*）ファイルを指定します。 このプロパティで登録されたファイルによって、インスツルメントされるコンポーネントが決まります。 このプロパティは必須です。

プロパティ設定

単一エントリ、または複数エントリのカンマ区切りリスト
リストには、以下のものの任意の組み合わせを含めることができます。

- ProbeBuilder ディレクティブ（*.pbd*）ファイル名
- ProbeBuilder リスト（*.pbl*）ファイル名
- *hotdeploy* ディレクトリ名（*hotdeploy* ディレクトリに配置される *.pbd* ファイルは、エージェント プロファイルを編集しなくても自動的にロードされます）

登録するファイル名の絶対パスまたは *IntroscopeAgent.profile* ファイルの場所を起点とする相対パスを使用したファイル名を指定できます。

デフォルト

デフォルト エントリは、エージェントをインストールするときの選択内容によって異なります。

例

```
introscope.autoprobe.directivesFile=default-full.pbl,hotdeploy
```

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.enable

このプロパティを false に設定する場合は、以下の条件が存在します。

- AutoProbe が無効
- .NET エージェントが Enterprise Manager に接続しない
- .NET エージェントが Investigator に表示されない
- .NET エージェントがメトリックをレポートしない

プロパティ設定

オプション

true または false

デフォルト

true

例

```
introscope.autoprobe.enable=true
```

注: 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

インスツルメンテーション ログ ファイルの名前と場所。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

インスツルメンテーション ログ ファイルの絶対パスまたは相対パス。

デフォルト

デフォルトの場所は *<Agent_Home>* ディレクトリを起点とする相対パスの *logs¥AutoProbe*.log です。

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、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。

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 に設定すると、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. 76)

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。

プロセスにまたがるトランザクション追跡

後部フィルタによるダウンストリーム追跡の自動収集は有効にすることができます。 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。

デフォルトのドメイン設定

デフォルト ドメインから Enterprise Manager への接続方法は制御することができます。

introscope.agent.dotnet.enableDefaultDomain

デフォルト ドメインに接続されているエージェントが Enterprise Manager に接続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.dotnet.enableDefaultDomain=false
```

注

このプロパティを有効にするには、*IntroscopeAgent.profile* にこれを追加する必要があります。

このプロパティが true に設定されていると、デフォルトのドメインを監視するエージェントも Investigator でレポートされます。

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.agent.remoteagentdynamicinstrumentation.enabled

このプロパティは、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

- 変更を有効にするために、管理対象アプリケーションを再起動します。
- 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。
- 動的インスツルメンテーションを有効にした場合は、同じプロセス内で複数の CLR （In-Process、Side-by-Side 実行など）に対する監視は機能しません、 詳細については、com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs（P. 246）プロパティを参照してください。

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。

フィルタされたパラメータ

フィルタされたパラメータの収集は有効にすることができます。

introscope.agent.asp.disableHttpProperties

.NET Agent は、デフォルトでは追跡するトランザクションの URL のみをレポートします。 トランザクション追跡がシステムのオーバーヘッドに与える影響を最小にするため、個別の HTTP プロパティをレポートすることは制限されています。

フィルタリングを有効にするには、まずこのプロパティを false に設定して、HTTP プロパティのコレクションを有効にする必要があります。

これによってコレクションが有効となるプロパティは、アプリケーション名、セッション ID、コンテキスト パス、サーバ名、URL、正規化 URL、HTTP ヘッダ、HTTP パラメータ、HTTP 属性です。

プロパティ設定

オプション

True または False

デフォルト

False

例

```
#introscope.agent.asp.disableHttpProperties=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。

HTTP ヘッダ デコレータ

CA CEM との統合ソリューションの一部であるサーブレット ヘッダ デコレータは、有効にすることができます。

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。

HTTP ヘッダ デコレータが、各トランザクションに GUID を付与し、HTTP ヘッダ、x-wily-info に GUID を挿入します。これによって、CA CEM と 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。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.agent.decorator.enabled=false
```

LeakHunter の設定

エージェントと LeakHunter との連携方法は制御することができます。

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.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.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.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.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.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.ignore.0

1 つ以上の無視するプロパティを使用して、潜在リークの検出時に LeakHunter に無視させる特定のコレクションを指定します。 汎用コレクションには、たとえば次のような、汎用的な修飾子を含む構文を使用します。 *system.Collections.Generic.List`1* 指定した条件に一致するコレクションは無視されます。

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.0=
```

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。

ログ

プロパティを使用してログ記録を設定できます。

introscope.agent.log.config.path

このプロパティは、Log4Net 設定ファイルを指しています。

プロパティ設定

デフォルト

`logging.config.xml`

例

`introscope.agent.log.config.path=logging.config.xml`

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。

NativeProfiler

IntroscopeAgent.profile 内の以下のプロパティを設定して NativeProfiler 操作を制御できます。 以下のプロパティは、ネイティブ イメージ生成プログラムを使用して作成されたアプリケーションの監視、クラス名のインメモリ キャッシュのサイズ、および NativeProfiler ログ ファイルの場所と内容を制御します。

introscope.nativeprofiler.clrv4.transparency.checks.disabled

.NET 4 CLR では透過的なアセンブリ上でチェックが行われ、プロファイラによってインスツルメントされたコードが無効化される場合があります。 これらの確認が発生しないようにするには、このプロパティの値を "true" に設定します。

プロパティ設定

true または false

デフォルト

true

例

```
#introscope.nativeprofiler.clrv4.transparency.checks.disable=true
```

注: この設定値を有効にするには IIS をリセットする必要があります。

introscope.nativeprofiler.logfile

NativeProfiler が読み取るディレクティブおよびインスツルメントされるメソッドに関する情報を記録するログ ファイルへのパスを設定します。

プロパティ設定

ファイルの場所への絶対パスまたは相対パス。

デフォルト

logs/nativeprofiler.log

例

```
introscope.nativeprofiler.logfile=logs/nativeprofiler.log
```

注

NativeProfiler は、読み込みを行うように設定されている PBD および PBL でアクティブになっているディレクティブに関する情報をログ ファイルに記録します。また、NativeProfiler によってインスツルメントされた特定のメソッドの詳細もログ ファイルに記録されます。

introscope.nativeprofiler.logBytecode

NativeProfiler が、インスツルメント済みのバイトコードをリスト表示するかどうかを指定します。 true に設定された場合は、NativeProfiler ログ ファイルはインスツルメント済みのバイトコードをリスト表示します。 デフォルトでは、このプロパティは false に設定された上でコメント 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

False

例

```
#introscope.nativeprofiler.logBytecode=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.nativeprofiler.logAllMethodsNoticed

このプロパティが有効である場合は、インスツルメントされていないメソッドも含め、NativeProfiler が検知したすべてのメソッドが記録されます。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.nativeprofiler.logAllMethodsNoticed=false
```

注

デフォルトで無効になっています。

introscope.nativeprofiler.directivematching.cache.max.size

エージェントでメモリ内キャッシュに格納するクラス名の最大数を指定します。 デフォルトでは、モニタ対象のクラスが含まれていることが以前に検出されたディレクティブグループのメモリ内キャッシュがエージェントによって作成されます。ユーザが IIS を開始するたびに、エージェントは以前に検出したクラスのキャッシュを作成します。 モニタ対象の新しいクラスがアプリケーション コードによって使用されていくに従い、キャッシュのサイズは増加します。デフォルトでは、メモリ内キャッシュは 最大で 5000 のクラス名を格納します。

キャッシュが 5000 を超えるクラス名を格納する必要がある場合、このプロパティの値を増加させることで起動時間が改善する場合があります。 ただし、値を増加させると、エージェントで必要なメモリのオーバーヘッドが増加します。 プロパティ値を減少させると、エージェントのメモリオーバーヘッドは減少します。 監視するクラスが 5000 に満たない場合は、値を減少させる方が適切である可能性があります。

注: キャッシュにはクラス オブジェクトは格納されません。

デフォルト

デフォルトでは、キャッシュは 最大で 5000 のクラス名を格納可能です。

例

```
introscope.nativeprofiler.directivematching.cache.max.size=5000
```

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.nativeprofiler.generic.agent.trigger.enabled

このプロパティを使用すると、一般的なトリガを使用して .NET エージェントを開始できます。ユーザがこのプロパティを有効にすると、最初のユーザ コードが実行された後、全般的なトリガを使用してこのエージェントが開始されます。 このアクションではデフォルトのドメインのプローブが開始されますが、このプローブは制限可能です。 このプロパティが無効の場合は、エージェントは、IIS の実行中に、メイン メソッドまたはスタートアップ メソッドを使用して開始されます。

プロパティ設定

True または False

デフォルト

False

例

```
introscope.nativeprofiler.generic.agent.trigger.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。
- ProcSxS では、エージェント名は以下のように表示されます。
  <プロセス名>_<アプリケーション ドメイン名>

com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple.clrs

複数バージョンの共通言語ランタイムの監視をエージェントが処理する方法を制御します。 デフォルトでは、エージェントは、最初にロードされた CLR のバージョンに基づいて、.NET Framework の 1 つのバージョンのみを監視します。 このプロパティを使用して、デフォルトの動作を変更し、.NET Framework 2.0 コンポーネントを監視するか、.NET Framework 4 コンポーネントを監視するかを指定します。

プロパティ設定

有効な値は以下のとおりです。

- **None** -- デフォルトの動作を使用することを指定すると同時に、最初にロードされた CLR に基づいて、監視する .NET Framework のバージョンを選択します。
- **V2** -- .NET Framework 2.0 コンポーネントのみを監視することを指定します。 あるいは、CLR のバージョン番号（v2.0.50727 など）を指定することもできます。
- **V4** -- .NET Framework 4 コンポーネントのみを監視することを指定します。 あるいは、CLR のバージョン番号 (v4.0.30319 など) を指定することもできます。
- **デフォルト** -- デフォルト値は None です。 この値は、.NET Framework の 1 つのバージョンのみが監視されることを意味します。最初にロードされるアプリケーションによって、監視される .NET Framework のバージョンが決まります。

注: .NET Framework 2.0 および .NET Framework 4 コンポーネントの両方を同時に監視することはできません。動的インスツルメンテーションが有効な場合、この機能は動作しません。

例

```
com.wily.introscope.nativeprofiler.monitor.inprocsxs.multiple
.clrs=v2
```

注

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

パフォーマンス監視データの収集

パフォーマンス監視カウンタからのデータの収集を制御するプロパティは、設定することができます。

introscope.agent.perfmon.metric.filterPattern

監視するパフォーマンス監視カウンタを指定する正規表現を指定します。

プロパティ設定

パフォーマンス監視カウンタ名に照合させるためのテキスト文字列正規表現。プロセス特有のスレッドメトリックなどの新しいカウンタを追加するには、リストの最後に新しい式を、カンマで区切って追加します。 たとえば次のように追加します。、*...,|Thread|{osprocessname}*|**

デフォルト

デフォルト フィルタでは、プロセッサ、.NET Data Provider、.NET CLR、ASP.NET のパフォーマンス監視カウンタ、オブジェクト、およびインスタンスが監視されます。

例

```
introscope.agent.perfmon.metric.filterPattern=|Processor|*|*,|.NET Data Provider*|*|*,|.NET CLR*|{osprocessname}|*,|.NET CLR Data|*|*,|Process|{osprocessname}|*,|ASP.NET|*
```

注

フィルタ |*|* を使用すると、すべてのカウンタを instance-less として列挙するようパフォーマンス監視に指示することになります。 この設定は一部のカウンタにとって有効ではありません。

introscope.agent.perfmon.metric.limit

各間隔内にレポートできるパフォーマンス監視メトリックの最大数を指定します。

デフォルト

デフォルトは 1,000 メトリックです。

例

```
introscope.agent.perfmon.metric.limit=1000
```

introscope.agent.perfmon.metric.pollIntervalInSeconds

パフォーマンス監視コレクション エージェントが、パフォーマンス監視オブジェクト、カウンタ、およびインスタンスに新しいメトリック値があるかどうかをチェックする頻度を指定します。 デフォルトのポーリング間隔では、15 秒ごとにすべてのメトリック値がチェックされます。

デフォルト

デフォルトの間隔は 15 秒です。

例

```
introscope.agent.perfmon.metric.pollIntervalInSeconds=15
```

introscope.agent.perfmon.category.browseEnabled

新しいパフォーマンス監視カウンタの検出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（有効）

例

```
introscope.agent.perfmon.category.browseEnabled=true
```

introscope.agent.perfmon.category.browseIntervalInSeconds

パフォーマンス監視コレクション エージェントが、検出対象のパフォーマンス監視オブジェクトをチェックする頻度を指定します。

プロパティ設定

間隔の秒数を表す正の整数。

デフォルト

デフォルトの間隔は 600 秒（10 分）です。

例

```
introscope.agent.perfmon.category.browseIntervalInSeconds=600
```

プロセス名

プロセスを設定してプロセス名を定義することができます。

introscope.agent.customProcessName

Enterprise Manager および Workstation で表示されるプロセス名を指定します。 このプロパティのコメント化を解除し、Enterprise Manager および Workstation で表示されるプロセス名をカスタム定義します。

デフォルト

カスタム プロセス名

例

```
#introscope.agent.customProcessName=CustomProcessName
```

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。
- 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.defaultProcessName

カスタム プロセス名が定義されておらず、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、デフォルトのプロセス名が使用されます。

デフォルト

.NET Process

例

```
introscope.agent.defaultProcessName=.NET Process
```

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。

インスツルメンテーション設定の制限

目的のプロセスまたは実行可能ファイルのセットに対するインスツルメンテーションは、設定することができます。

注: インスツルメンテーションの制限の詳細については、「インスツルメンテーションの構成」を参照してください。

introscope.agent.dotnet.monitorApplications

インスツルメントするプロセスおよびアプリケーションを指定します。

オプション

完全修飾パスはサポートされていません。 アプリケーション実行可能ファイルのプロセス イメージ名のみを使用します。名前の大文字と小文字を区別し、Windows のプロセス管理にレポートされている名前と正確に一致させる必要があります。

デフォルト

```
w3wp.exe,aspnet_wp.exe
```

例

```
introscope.agent.dotnet.monitorApplications=w3wp.exe,aspnet_wp.exe,dllhost.exe
```

introscope.agent.dotnet.monitorAppPools

インスツルメント対象のアプリケーション プールを指定します。

オプション

このプロパティのコメント化を解除し、インスツルメントする IIS アプリケーション プールを指定します。

デフォルト

"NULL","DefaultAppPool","AppPool1","AppPool2"

例

```
#introscope.agent.dotnet.monitorAppPools=
"NULL","DefaultAppPool","AppPool1","AppPool2"
```

注

- アプリケーション プール名は引用符で囲む必要があります。 どのアプリケーション プールにも属していないアプリケーションを指定するには、"NULL" を使用します。
- すべてのアプリケーション プールをインスツルメントするためにコメント化したままにするか、またはインスツルメントするアプリケーション プールのみをコメント化解除してリストにします。

「ソケット メトリック」

以下のプロパティは、ソケット メトリックの生成を制御します。

introscope.agent.sockets.reportRateMetrics

個々のソケットの入出力帯域幅レート 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

True

例

```
introscope.agent.sockets.reportRateMetrics=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。

SQL エージェント

以下のプロパティは、SQL エージェントためのプロパティです。

introscope.agent.sqlagent.sql.maxlength

SQL エージェント メトリックについて、Investigator ツリー内に表示される SQL ステートメントのサイズ (バイト単位) を制限します。

デフォルト

デフォルトの制限は 990 バイトです。

例

```
introscope.agent.sqlagent.sql.maxlength=990
```

注

このプロパティはデフォルトで *IntroscopeAgent.profile* 内に表示されません。 値を変更するには、エージェント プロファイルにこのプロパティを追加します。

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. 262）
- introscope.agent.sqlagent.normalizer.regex.keys（P. 258）
- introscope.agent.sqlagent.normalizer.regex.key1.pattern（P. 259）
- introscope.agent.sqlagent.normalizer.regex.key1.replaceAll（P. 260）
- introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat（P. 261）
- introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive（P. 262）

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.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. 256) 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. 256) 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. 256) 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. 256) 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. 256) 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.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. 256) 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.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. 265)=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。

ストール メトリック

以下のセクションでは、ストール メトリックに関連するプロパティを定義します。

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。

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用です。

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.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.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. 269) 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.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.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.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.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。

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.sessioni
d=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.sessioni
d=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。

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. 281) 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
```

URL のグループ化

以下のプロパティにより、フロントエンド メトリック用に URL グループを設定します。

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
```

付録 B: アプリケーション固有の設定

.NET Framework では、アプリケーション固有のパラメータの設定に、*.config* 拡張子を持つオプションの XML 形式ファイルを使用することができます。

構成用のプロパティの追加

ASP.NET アプリケーションの場合、Web.config がアプリケーション固有の設定および構成用のメイン ファイルです。 このファイルは、アプリケーションのルート ディレクトリに保存されます。

その他の.NET 実行ファイルの場合、設定ファイルにはアプリケーションと同じ名前が付けられ、.config 拡張子が追加されます。 このファイルは、アプリケーションの実行可能ファイルと同じディレクトリに保存されます。 たとえば、*testapp.exe* の場合、オプションの設定ファイルは *testapp.exe.config* です。

Introscope 固有の設定を、*.config* ファイルに追加することが可能です。 たとえば、個々のアプリケーションが *IntroscopeAgent.profile* ファイルの自分自身のインスタンスを（異なるアプリケーションが別のエージェントの設定を持つことができるように）参照でき、また Web サービスとプロセスにまたがるトランザクションの相関付けができるように、パラメータを設定できます。

以下の手順に従います。

1. アプリケーションの設定ファイルを開きます。
2. アプリケーションの設定ファイルに、*sectionGroup* および *section* を追加します。 それぞれの名前は以下のように指定します。

```
<configuration>
   <configSections>
      <sectionGroup>
         <sectionGroup name="com.wily.introscope.agent">
         <section
name="env.parameters"type="System.Configuration.NameValueSectionHandler" />
      </sectionGroup>
```

3. *env.parameters* セクションに新しいプロパティを追加します。 例：

```
<com.wily.introscope.agent>
   <env.parameters>
      <add key="com.wily.introscope.agentProfile"
value="e:¥¥junkyard¥¥dotnettest¥¥Agent.profile" />
   </env.parameters>
</com.wily.introscope.agent>
```

例については、*sample.exe.config* を参照してください。

注: .NET Agent のインストールに構成ファイルが必須であるわけではありません。

付録 C: 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. 287)

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。

以下の手順に従います。

1. 以下のディレクトリに移動します。

   <Agent_Home>¥wily¥tools

2. NetInterface.exe を実行します。

   .NET 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. 202)