CA Application Performance Management

Java 代理实施指南

版本 9.5

CA Technologies 产品引用

本文档涉及以下 CA Technologies 产品和功能：

- 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)

联系技术支持

要获取在线技术帮助以及办公地址、主要服务时间和电话号码的完整列表，请联系技术支持：http://www.ca.com/worldwide。

目录

第 1 章： Java 代理简介

此部分包含以下主题：

关于 Introscope 和 Java 代理

CA Introscope® 是一个企业应用程序性能管理解决方案。您可以利用它全天候监控生产环境中复杂的 Web 应用程序，在客户受到影响之前发现问题，并以协作方式迅速解决这些问题。该解决方案体系结构的关键部分是低开销代理。

此代理是 Introscope 的一个数据收集组件，它收集在执行事务时应用和计算环境的相关详细性能信息。Java 代理从 Java 虚拟机 (JVM) 上运行的应用程序和资源收集这些信息，然后将这些信息发送到企业管理器以供进一步处理。

Java 代理将*探测器*插入到组件的字节码中，这些组件构成了应用程序使用的 JVM。将探测器插入字节码是*检测*过程的一部分，该过程可监控应用程序。

检测应用程序还需要在 ProbeBuilder 指令 (PBD) 文件中定义的跟踪器。PBD 文件中的说明或指令可标识要监控的应用程序组件。跟踪器可标识代理应从将探测器作为应用程序运行的 JVM 中收集的度量标准。您可以通过根据您的环境更改 PBD 文件来控制监控对象。

安装 Java 代理时，将部署多个默认 PBD 文件以启用环境的默认监控。您可以修改默认监控，以实现所需可见性与性能之间的平衡。检测完应用程序之后，Java 代理将收集您感兴趣的数据，并将这些数据报告给企业管理器。然后，企业管理器会处理和存储用于实时和历史报告的数据。这样，您便可以查看和使用通过 Introscope Workstation 收集的数据来创建报警或采取响应操作。

对于应用程序管理，关键活动包括：

- 部署代理以监控应用程序服务器的性能和可用性。
- 测试、调整和优化对应用程序组件的监控。
- 自定义代理配置文件以根据需要控制代理操作。
- 创建特定于应用程序或代理的度量标准组、显示板、报警和操作。
- 调查、分类并诊断应用程序问题。

规划 Java 代理部署

在规划部署时，目标是在代理开销与应用程序性能的可见性之间实现合理平衡。低开销代理允许在生产环境中实时监控所有事务。保持较低的开销可帮助改善关键应用程序和服务器资源的性能和可用性。但是，当出现问题时，保持较低的开销无法提供足够的信息来诊断这些问题。因此，通常要在整个应用程序生命周期期间部署和调整代理配置。此外，开发或测试应用程序时需要监控更多组件，然后在应用程序投入到生产后减少监控的组件。

安装和评估默认功能

部署代理的第一步是安装和评估默认代理配置。默认代理配置展示了对应用程序和计算环境的多个常见组件的数据收集。默认代理配置还包括几项已启用的功能，以及其他不太常用的已禁用功能。

您的目标是评估默认情况下提供的数据收集的深度和广度，并熟悉 Introscope 以及如何监控应用程序。在熟悉默认情况下代理提供的性能度量标准之后，您可以根据需要自定义代理，以便收集数据。

在评估默认性能时请切记，代理收集的度量标准越多，就会消耗越多的系统资源。代理收集的度量标准越少，您所了解的潜在问题也越少。当您优化代理配置时，请尝试打破数据收集的深度与可接受的性能水平之间的平衡。适当的检测级别通常取决于部署代理的位置。例如，在测试环境中进行监控的代理通常配置为收集大量度量标准。但是，生产服务器上的代理通常配置为提供基本信息。

确定配置要求

在部署代理之前，请确定您的数据收集要求。此信息允许您定制代理的数据收集行为，并通过代理的备用配置来评估对开销的影响。

Introscope 可在应用程序的整个生命周期中使用。例如，从开发到测试、负载验证、置备，直到生产。在生命周期的每个阶段，监控目标、环境约束和服务水平期望都会因时而异。要解决这些差异，您可以将代理配置为针对所监控的环境类型而采用不同的行为方式。

您的目标是在性能详细信息的可见性与资源开销之间确定相应的权衡。此外，请根据所监控环境的合理成本考虑最佳的可见性级别。

在预生产应用程序环境（如开发环境）中，通常应配置较高级别的数据收集，以便在应用程序性能方面提供更深的可见性。在生产或大容量事务环境中，通常应减少报告的度量标准以控制代理开销。根据您的要求，还可以将代理属性配置为控制特定的代理行为。例如，跟踪收集的最大度量标准数目以及对旧度量标准的删除。

针对您的环境，确定适当的可见性级别和可接受的性能开销，以便可以根据您的要求配置代理。

使用适当的配置属性定义基准代理配置文件

在确定要监控的应用程序环境类型后，可以创建“候选”代理配置。可使用代理配置文件 (IntroscopeAgent.profile) 中的属性控制大多数代理操作。例如，IntroscopeAgent.profile 文件可定义代理使用的 ProbeBuilder 指令文件和 ProbeBuilder 列表文件。然后，代理配置文件中列出的文件可以控制代理收集的特定度量标准。IntroscopeAgent.profile 文件还提供了用于启用或禁用特定功能或者调整操作（如轮询时间间隔）的属性。

根据您的配置和环境，可以在代理配置文件中调整属性，以便评估每项更改的影响。例如，您可以从未启用 ChangeDetector 的默认代理配置文件开始。然后，可以在添加其他任何功能之前，在配置文件中启用 ChangeDetector 属性，并在更改后评估代理性能。

评估代理性能开销

在评估代理配置时，请确定收集的度量标准是否在应用程序性能和可用性方面提供了足够的可见性。此外，还要确定度量标准的量是否会对操作环境带来无法承受的负载。代理无法报告比识别和定位性能和可用性问题所需更多的度量标准。

通过了解应用程序的性能特点，您可以有效地评估代理性能。例如，可以在实施默认监控之前和之后测试应用程序的负载，以验证影响。

要以受控方式扩展数据收集，请在实施更改之前和之后验证代理操作和开销。例如，一次只能为一个应用程序添加监控支持，这样您可以逐一评估每个附加程序的性能。

验证和部署代理配置

在验证候选代理配置是否提供所需的可见性后，在整个环境中部署该配置。

通过将以下配置项安装到目标环境，可以部署经过验证的配置：

- IntroscopeAgent.profile 文件
- 修改过或自定义的 PBD 文件

部署 Java 代理

部署代理包含以下高级步骤：

1. 在目标计算机上安装代理。
2. 配置应用程序服务器 (see page 32)以检测应用程序。
3. 为应用程序服务器配置启动脚本 (see page 33)或 Java 命令以包括代理以及代理配置文件位置。
4. 配置代理 (see page 55)与企业管理器的连接。
5. 重新启动应用程序服务器以检测应用程序并开始数据收集。
6. 配置 IntroscopeAgent.profile (see page 63)（如果要更改在安装期间设置的任何属性，或者修改数据收集或其他代理操作）。

第 2 章： 安装和配置 Java 代理

此部分包含以下主题：

安装代理之前

在安装代理之前，请执行以下步骤：

1. 检查 Java 代理部署过程。
2. 确认具有支持的应用程序服务器版本。

   注意：计划安装代理的服务器本地必须有一个受支持的 JVM 版本。有关应用程序服务器和 JVM 要求，请参阅*《兼容性指南》*。
3. 确认您有支持的 JVM 版本。如果无法使用支持的版本，请使用先前版本：
   - 将 8.x Java 代理用于 Java 1.4.x。
   - 将 7.2 Java 代理用于 Java 1.3.x。
4. 确认已安装 Introscope 企业管理器和 Workstation 组件。请记下与代理连接的企业管理器的主机名和端口号。

   可以使用 Ping 或 Telnet 验证代理和企业管理器之间的连通性。

注意：有关安装企业管理器、Workstation 和 WebView 组件的信息，请参阅*《CA APM 安装和升级指南》*。

详细信息：

规划 Java 代理部署 (p. 18)
部署 Java 代理 (p. 20)

选择安装 Java 代理的方法

可以使用以下任一方法安装 Java 代理：

- 使用图形用户界面 (GUI) 或基于文本的控制台以交互方式安装
- 使用编辑的响应文件，在没有交互提示的情况下以无人值守方式安装
- 通过解压缩并配置各个应用程序服务器的文件来手工安装

在大多数情况下，如果要在计算机上本地安装文件，则可以使用交互式安装程序。如果以交互方式安装，图形界面或基于文本的界面所显示的提示相同。但是，选择 GUI 安装程序还是基于文本的安装程序取决于运行安装程序的计算机的操作系统。例如，大多数 UNIX 环境支持 GUI 或基于文本的控制台安装程序，但默认启动控制台安装程序。

如果要在计算机上远程安装文件或部署一组预配置安装说明，您可以使用编辑过的响应文件以无人值守方式运行安装程序。

如果不想以交互方式或无人值守方式运行安装程序，您可以使用安装存档手工安装和配置代理。通过手工安装，可以从特定于应用程序服务器以及特定于操作系统的存档解压缩特定的代理文件集，然后手工配置部署选项。CA Technologies 提供的这些存档可更快地将 Java 代理部署到多个系统上，这些系统共享相同类型的应用程序服务器和操作系统。

有关以交互方式、无人值守方式或手工方式执行安装的更多信息，请参阅相应的部分。

以交互方式安装 Java 代理

通过选择适用于操作系统的 Java 代理安装程序、启动安装程序并响应提示，您可以通过交互方式安装 Java 代理。如果使用 GUI 安装程序，可以通过使用下拉菜单和选中复选框做出选择。在基于文本的控制台安装程序中，通过输入文本做出选择。

请执行以下步骤：

1. 选择适合您的操作系统的安装存档。例如：
   - IntroscopeAgentInstaller<*版本*>windows.zip，用于在 Microsoft Windows 上安装。
   - IntroscopeAgentInstaller<*版本*>unix.tar，用于在支持的 UNIX 或 Linux 操作系统上安装。
   - IntroscopeAgentInstaller<*版本*>zOS.tar，用于在 IBM z/OS 上安装。
   - IntroscopeAgentInstaller<*版本*>os400.zip，用于在 IBM OS/400（IBM i 操作系统）上安装。

2. 使用适合您操作系统的命令来解压缩安装存档文件。例如，在 UNIX 或 Linux 上，可使用以下命令：
   `tar -xvf IntroscopeAgentInstaller<版本>unix.tar`
3. 按照提示开始安装。
4. 指定应用程序服务器根目录的位置。

   如果不想指定应用程序服务器根目录，则接受默认响应（在 Windows 上为 C:\，在 UNIX 上为 /）。
5. 从有效的应用程序服务器列表中选择应用程序服务器类型。

   选择哪个应用程序服务器可确定提供哪些其他可用的监控选项。
6. 指定 Java 代理的根安装目录。大多数情况下，使用应用程序服务器根目录。

   安装程序会在根安装目录内创建 wily 目录，即 *<Agent_Home>* 目录。
7. 指定是要创建代理配置文件，还是使用现有代理配置文件。

   如果要创建代理配置文件，系统会提示您选择：

   - 典型/完全检测
   - 代理名称和进程名称
   - 企业管理器主机和端口号

   如果选择现有配置文件，系统将提示您指定文件位置。请指定该文件的完全限定路径。

   如果需要，可以在安装之后通过编辑 IntroscopeAgent.profile 文件来更改指定的设置。
8. 指定用于启动应用程序服务器的 Java 可执行文件的路径。
9. 指定检测应用程序的 ProbeBuilding 方法。在大多数情况下，使用 JVM AutoProbe。
10. 指定是否启用 ChangeDetector 代理扩展。

    如果启用 ChangeDetector，系统会提示您对 ChangeDetector 代理进行命名。

    如果不启用 ChangeDetector，则 ChangeDetector 文件将安装在 *<Agent_Home>*/examples 目录下。您可以稍后通过将这些文件复制到 *<Agent_Home>*/core/ext 目录并修改 IntroscopeAgent.profile 文件来启用 ChangeDetector。

    注意：有关 ChangeDetector 的详细信息，请参阅 *《CA APM ChangeDetector 指南》*。

11. 选择要安装的其他监控选件。例如，如果要监控 Web 服务和其他 SOA 环境组件，请选择 CA APM for SOA。

    如果在安装期间不启用其他监控选项，则这些文件将安装在 *<代理主目录>*/examples 目录下，并且可以稍后启用。

12. 指定放置 .zip 或 .tar 文件的“选取文件夹”的位置，这些文件包含要与代理一起安装的附加程序。

    将在指定位置找到的 .zip 或 .tar 文件解压缩到 *<Agent_Home>* 目录中。

13. 检查选择项，继续安装代理，然后完成安装。

注意：安装代理不会停止或启动应用程序服务器，也不会配置应用程序服务器启动脚本。请在安装之后手工执行这些任务。具体步骤取决于所监控的应用程序服务器的类型。

以无人值守方式安装 Java 代理

在无人值守模式下，从命令行调用代理安装程序，并指定包含安装说明的响应文件。随后安装会在后台运行，不会显示任何关于其进度的信息。由于此安装方式使您无需用户交互即可安装代理，因此最常用于在远程计算机上安装代理或使用相同配置安装多个代理。

如果您之前安装了一个 Java 代理，则可以使用自动生成的响应文件安装更多代理。或者，您可以对安装中提供的示例响应文件进行手工编辑或使用文本编辑器创建自己的响应文件。

关于自动生成的响应文件

只要您以交互方式或无人值守方式运行 Java 代理安装程序，该安装程序就会创建一个记录所选安装选项的响应文件。自动生成的响应文件保存在 *<Agent_Home>/install* 目录中。文件名指明了安装程序创建响应文件的日期和时间，格式如下：

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

例如，如果安装完成于 2005 年 4 月 30 日早上 7:10:05，则自动生成的响应文件的名称将为：

```
autogenerated.responsefile.2005.4.30.7.10.05
```

您可以使用这个自动生成的响应文件进行具有相同设置的后续无人值守安装，或进行编辑以使用新设置。

关于示例响应文件

如果先前没有安装 Java 代理，或者想要使用默认安装选项而不是先前选择的安装选项，您可以编辑 Java 代理安装程序附带的默认示例响应文件。默认示例响应文件位于 *<Agent_Home>*\install 目录中，名为：

`SampleResponseFile.Agent.txt`

示例响应文件提供了大多数属性的默认设置，但在使用该文件进行无人值守安装之前，必须手工对其进行编辑。

配置响应文件属性并安装代理

无论您是使用自动生成的响应文件、默认示例响应文件，还是创建自定义响应文件，都必须在以无人值守模式调用 Java 代理安装程序之前相应地配置文件中的属性。您在响应文件中设置的属性等同于以交互方式运行安装程序时所选择的操作。

注意：有关设置任何属性的其他详细信息，请参阅 *<Agent_Home>*/install/SampleResponse.Agent.txt 文件中的注释。

请执行以下步骤：

1. 在文本编辑器中打开响应文件。
2. 设置相应的属性值。要配置的属性为：

   USER_INSTALL_DIR=*<root_installation_directory>*

   指定代理的安装目录。大多数情况下，应使用应用程序服务器根目录。

   silentInstallChosenFeatures=Agent

   请指定要安装的组件。

   appServer=Default

   指定要监控的应用程序服务器的类型。有效值是 Default、JBoss、Tomcat、WebLogic、WebSphere、Sun、Oracle 或 Interstage。该值区分大小写。

   该设置控制与代理一起安装的 ProbeBuilder 指令 (PBD)，以及其他哪些监控选件是有效的。

   （可选）appServerHome=

   指定应用程序服务器的主目录。如果已将 USER_INSTALL_DIR 属性设置为应用程序服务器根目录，则不需要该属性。

   （可选）appServerJavaExecutable=

   指定用于启动应用程序服务器的 Java 可执行文件的路径。

instrumentationLevel=Typical

指定您想使用完全检测还是典型检测。该值区分大小写。

agentName=Default Agent

指定应显示在 Workstation 中的代理名称。

processName=Default Process

指定应显示在 Workstation 中的进程名称。

emHost=localhost

指定运行企业管理器的计算机的主机名，默认情况下代理应连接到该企业管理器。

emPort=5001

指定代理用于连接到其企业管理器的端口号。

（可选）alternateAgentProfile=

指定现有代理配置文件的绝对路径。

（可选）pickupFolder=

指定“选取文件夹”的绝对路径，该文件夹包含要与代理一起安装的任何附加程序的 .zip 或 .tar 文件。

（可选）changeDetectorEnable=false

指定是否启用 ChangeDetector 代理扩展。如果将该属性设置为 false，将安装 ChangeDetector 文件但不启用。您可以稍后启用它。

（可选）changeDetectorAgentID=

指定 ChangeDetector 代理扩展的名称。如果启用 ChangeDetector，请取消注释并设置该属性。

（可选）shouldEnable*

指定要启用的其他 CA APM 监控解决方案。默认情况下，可选 CA APM 监控解决方案的所有属性均设置为 false。可启用的选项取决于应用程序服务器的类型。

3. 保存响应文件，然后关闭文本编辑器。

4. 通过指定安装程序可执行文件的路径和响应文件的绝对路径，以无人值守模式调用安装程序：
   ```
   <path_to_installer> -f <absolute_path_to_response-file>
   ```
   用于调用安装程序的命令取决于运行该命令的操作系统。

   例如，在 Windows 上，命令格式如下所示：
   ```
   IntroscopeAgent<version>windows.exe -f C:\temp\myResponseFile.txt
   ```
   在 Linux 或 UNIX 上，命令格式如下所示：
   ```
   ./IntroscopeAgent<version>unix.bin -f /
   ```
   请选择适合您的操作系统的命令格式。
5. 检查 *<Agent_Home>/install/Introscope_Agent_<version>_InstallLog.log 文件*以验证代理是否已成功安装。

使用安装存档手工安装

可以将代理文件放置在系统上，而无需以交互方式运行 Java 代理安装程序或配置响应文件。您可以通过特定于应用程序服务器的存档安装代理。

安装存档包含运行代理安装程序时安装的所有文件。在将存档复制到计算机之后，可以解压缩存档内容，并配置代理配置文件以识别要连接的企业管理器及其他属性。使用这些文件通过批处理作业部署多个代理，或将这些文件保留为默认代理文件集的一个存档。

要从存档手工安装 Java 代理，请确认您计划安装的计算机具有 35 MB 可用磁盘空间。

请执行以下步骤：

1. 选择适用于应用程序服务器和操作系统的安装存档。
2. 使用适用于您操作系统的命令，将存档内容解压缩到 JVM 可以访问的位置。例如，在 UNIX 或 Linux 上，可使用以下命令：
   ```
   tar -xvf
   IntroscopeAgentFilesOnly-NoInstaller<version><app_server>.<os>.tar
   ```
3. 在文本编辑器中打开 *<Agent_Home>*/core/config/IntroscopeAgent.profile 文件，并配置与企业管理器的连接。

   有关设置用于与企业管理器进行通信的属性的详细信息，请参阅配置与企业管理器的连接 (see page 55)。
4. 配置其他任何属性，然后保存并关闭 IntroscopeAgent.profile 文件。
5. 使用 Java 代理启动文件和代理配置文件的位置配置应用程序服务器。
6. 重新启动应用程序服务器。

关于 Java 代理目录结构

安装代理时，会在根安装目录中创建以下目录结构：

wily

此目录为 *<Agent_Home>* 目录，其中包含用于启动代理的 Agent.jar。

由于 *<Agent_Home>* 目录是子目录，它们会提供用于启用 Java 代理的各种功能的库和扩展文件。

核心

- config

  包含用于控制代理操作、度量标准数据收集和检测过程的 IntroscopeAgent.profile、ProbeBuilder 指令文件 (.pbd) 以及 ProbeBuilder 列表文件 (.pbl)。IntroscopeAgent.profile 中定义的特定属性以及代理配置文件中部署和引用的 .pbd 与 .pbl 文件取决于您在安装过程中所做的选择。

  在 config 目录内，您可以使用 hotdeploy 子目录来部署新的指令，而无需编辑 IntroscopeAgent.profile 或重新启动应用程序。如果 hotdeploy 目录内的文件包含无效语法或过多度量标准，则检测可能会失败或者应用程序性能可能会降低。

- ext

  包含用于已启用代理扩展或功能的文件。例如，该目录中包含用于应用程序分类视图和 LeakHunter 的文件。

connectors

包含 CreateAutoProbeConnector.jar 实用工具，该实用工具可用于将 AutoProbe 连接器配置为在特定环境中启用检测。

common

包含扩展的配置和属性文件。

examples

包含用于可选代理扩展（例如 CA APM for SOA）的文件夹和文件。如果在安装时未启用扩展，可以稍后使用此目录中的文件来配置扩展。

install

包含用于记录安装过程的日志文件以及用于无提示安装的文件。例如，生成的响应文件和 SampleResponseFile.Agent.txt 文件均位于此目录中。

如果从安装存档手工安装 Java 代理，则不会创建此目录。

logs

存储代理日志文件。

tools

包含以下内容：

- 用于分析 Web 服务器日志文件的 URLGrouper.jar 命令行实用工具。
- 扩展的文件列表。例如，configurePMI.bat、CreateIU.jar、listServers.bat、MergeUtility.jar、NetInterface.jar 以及 setPmiModules.jacl 文件都位于此目录中。
- TagScript 工具文件：TagScript.jar、TagScript.bat 和 TagScript.sh 命令行脚本。

UninstallerData

包含一个子目录，其中包含用于卸载代理及相关资源的可执行文件和关联资源。

如果从安装存档手工安装 Java 代理，则不会创建此目录。

version

包含可选 CA APM 监控解决方案的版本信息。无论您在安装时启用了哪个扩展，都会安装此信息。

配置对综合事务的检测

使用 introscope.agent.synthetic.header.names 参数已完成对综合事务监控的配置设置。

introscope.agent.synthetic.header.names 参数值列出了 HTTP 头参数，这些参数可用于确定监控的 HTTP 请求是否为综合事务的一部分。使用逗号分隔单个参数名。如果此参数是未定义的，或值为空，则检测不到综合事务。如果定义了多个 HTTP 头参数名，则会按指定的顺序对这些参数名进行检查。第一个 HTTP 参数的值可用于定义综合事务。

报告综合事务的节点取决于用于检测每个事务的特定 HTTP 头参数，如下所示：

- 如果参数值是除 *lisaframeid* 或 *x-wtg-info* 信息之外的任何值，则 HTTP 参数值本身将用作节点名称。完成适当的修改以确保使用有效的节点名称。

- 如果参数值是 *lisaframeid*，则合成的节点名称为 CA LISA。
- 如果参数值是 *x-wtg-info*，则假定 HTTP 头参数值包含一系列名称和值对。使用和号符号将对彼此分隔开。等号在每个对之内分隔属性名称和值。*group*、*name*、*ipaddress* 以及 *request_id* 的值使用 | 节点分隔符形成综合事务节点名称。

例如，给定以下参数：

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

以下 *x-wtg-info* 头可导致在节点 SampleGroup|sample|192.168.193.1|start 下报告度量标准：

```
clear
synthetic=true&instance=ewing&name=sample&group=SampleGroup&version=4.1.0&ipa
ddress=192.168.193.1&sequencenumber=1&request_id=start&executiontime=12264550
47
```

未在 *x-wtg-info* HTTP 头参数值中定义的任何属性都具有提供的默认值，如下所示：

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

如果未定义 *introscope.agent.synthetic.header.names*，则忽略以下配置参数。

`introscope.agent.synthetic.node.name=合成用户`

报告识别的综合事务下的节点。此节点位于 *Frontends*|*Apps*|*<WebAppName>* 下，其中 *<WebAppName>* 是 Web 应用程序名称。此值默认为合成用户。

`introscope.agent.non.synthetic.node.name=实际用户`

报告未识别的综合事务下的节点。此节点位于 *Frontends*|*Apps*|*<WebAppName>* 下，其中 *<WebAppName>* 是 Web 应用程序名称。如果未定义，则没有在 *<WebAppName>* 下创建其他节点。

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

其值可用作合成用户名的 HTTP 头参数的名称。合成用户名可用于分隔不同的综合事务。每个合成用户名的节点已创建在 *"合成用户"* 节点之下。如果定义了此配置参数且此名称的 HTTP 标头参数存在，则报告综合事务度量标准。报告事务的节点为 *<Synthetic Users>*|*<Synthetic User>*，其中：

- *introscope.agent.synthetic.node.name* 配置参数将确定 *<Synthetic Users>* 节点名称。
- HTTP 头参数值将确定 *<Synthetic User>* 节点名称。

注意：对这些属性所做的更改将即时生效，无需重新启动托管应用程序。

使用 TagScript 实用工具

CA TagScript 实用工具可以与 HP Vugen 一起使用以指定合成用户信息的提取。

使用 TagScript 实用工具

1. 打开 TagScript 实用工具。

   对于 Windows：

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

   对于 UNIX：

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

   系统会询问您想修改哪个环境的脚本。

2. 单击下列选项之一：
   - 性能测试－适用于 HP Loadrunner 脚本
   - 产品－适用于 HP 业务流程监视器或 Sitescope 脚本
   - 取消标记－反转标记进程
3. 导航到您的 HP Vugen 脚本所在的目录。双击每个 .c 脚本可打开它们。

   HP Vugen 脚本 .c 文件已被全部备份且被修改版替换。
4. 如果 HP Vugen 处于打开状态且执行了实用工具，系统会提示您重新加载修改的脚本。当出现提示时，单击“全是”。
5. 您可以关闭 TagScript 实用工具，或单击“文件选择”对话框的“取消”按钮。您不需要关闭 TagScript 实用工具，许多用户使用 HP Vugen 时不会关闭实用工具。如果已修改脚本，或创建了任何新的脚本，则不关闭实用工具将简化流程。

6. 确认已在以下位置标记您的脚本：
   - HP Vugen 代码的新段落已插入每个脚本的开头。
   - 标记在所有 lr_start_transaction、lr_end_transaction 之前出现，并会在脚本的结尾处出现。
7. （可选）您可以使用一组 blame 堆栈跟踪 HP Loadrunner 性能测试的每个虚拟用户。要跟踪每个用户，请在声明段落的脚本的开头取消注释以下行：

   ```
   web_add_auto_header( “Synthetic_Trace_By_Vuser” ,vuserOverview)
   ```

   注意：如果针对标记脚本的产品取消注释此选项，则将为每个存在点或合成生成器创建一组 blame 堆栈。

Java Autoprobe

如果使用的是通过 Java 1.7 编译和执行的应用程序，请添加 -XX:-UseSplitVerifier，否则代理可能会导致类似于以下内容的字节码验证错误：

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

或

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

如果您是通过 javaagent 或 Xbootclasspath 使用 JVM AutoProbe，此说明适用。

如何检测应用程序

在安装代理之后，必须配置应用程序服务器以检测应用程序。检测应用程序的最常见方法是使用 JVM AutoProbe 和 *–javaagent* 命令行选项。JVM AutoProbe 可以在运行时期间动态检测应用程序。此方法适用于所有 J2EE 应用程序服务器，这些服务器为启动提供挂钩或为 Introscope 提供应用程序服务器类加载器，以查看从文件系统中加载的所有字节码。

大多数 JVM 提供商都支持 *–javaagent* 选项。如果您使用的 JVM 不支持此选项，则必须使用备选方法进行检测。

仅当必须在启动应用程序之前静态地检测字节码时，才需要手工运行 ProbeBuilder。您可以使用 ProbeBuilder 向导或从命令行提示中手工运行 ProbeBuilder。它将检测字节码并输出新命名的、已检测的 jar 或类文件。然后，在启动应用程序之前，此新检测的字节码将放到应用程序的类路径之前（或适当地重命名）。

有关使用 JVM AutoProbe 的替代方法的详细信息，请参阅“附录 B：检测的备选方式”。

配置应用程序服务器以启动 Java 代理

您必须对正在检测的应用程序服务器进行配置以包括代理的主 *.jar* 文件和代理配置文件的路径。在大多数情况下，您可以通过编辑应用程序服务器的启动脚本并重新启动应用程序服务器来完成此操作。应用程序服务器重新启动时，Java 代理会为 JVM 和应用程序环境的默认组件检测发现的类。具体步骤取决于应用程序服务器。

配置 Apache Tomcat 以使用 Java 代理

要配置 Apache Tomcat 以使用 Java 代理，您必须编辑 Tomcat 启动脚本。默认情况下，Tomcat 启动脚本是 *$CATALINA_HOME/bin* 目录中的 *catalina.sh* 或 *catalina.bat*。根据 Web 服务器的要求，您可以使用不同的启动脚本，或者不同的启动脚本位置。

请执行以下步骤：

1. 导航到包含 Tomcat 启动脚本的目录。例如：
   ```
   cd /apache-tomcat-6.0.18/bin
   ```
2. 在文本编辑器中打开 Tomcat 启动脚本。例如：
   ```
   vi catalina.sh
   ```
3. 找到用于设置 Java 选项的命令行并添加以下命令行选项，以指定代理的启动 *.jar* 文件和代理配置文件的路径：
   ```
   -javaagent:<代理 Jar 文件的路径> -Dcom.wily.introscope.agentProfile=<代理配置文件的路径>
   ```
   例如，如果使用 *Agent.jar*，请在用于启动服务器的命令之前添加类似于以下内容的代码：
   ```
   set JAVA_OPTS=%JAVA_OPTS%  -javaagent:c:\apache-root\wily\Agent.jar
   -Dcom.wily.introscope.agentProfile=
   c:\apache-root\wily\core\config\IntroscopeAgent.profile
   ```

4. 保存启动脚本。
5. （可选）通过配置代理配置文件以收集 JMX 度量标准，可报告 Apache Tomcat JMX 度量标准。打开 IntroscopeAgent.profile 并设置以下属性：

   introscope.agent.jmx.enable=true

   注意：如果想要通过结合使用 JMX 远程管理服务器和特定于平台的 MBeanServer 来从控制台中的 Apache Tomcat 查看 JMX 度量标准，应将 com.wily.use.platform.mbeanserver=true 添加到 IntroscopeAgent.profile 中。此配置已相对于先前版本的 Introscope 进行了更改，在先前的版本中，特定于平台的 MBeanServer 的使用是在命令行中设置的。
6. 保存并关闭 IntroscopeAgent.profile。
7. 重新启动 Tomcat 服务器。

自定义数据收集

在 Apache Tomcat 服务器上安装 Java 代理之后，可以修改 PBD 来自定义数据收集。

请执行以下步骤：

1. 导航到 <*代理主目录*>\wily\core\config 目录。
2. 在文本编辑器中打开 tomcat.pbd 文件。
3. 修改您要自定义的部分。
4. 保存并关闭文件。

配置 JBoss 以使用 Java 代理

要将 JBoss 配置为使用 Java 代理，请编辑与您的 JBoss 版本对应的 JBoss 启动脚本：

- 适应于 JBoss 7 及更高版本的 JBoss 启动脚本

  对于独立模式，默认 JBoss 启动脚本是 $JBOSS_HOME/bin 目录中的 standalone.sh 或 standalone.bat； 而对于域模式，默认启动脚本是 $JBOSS_HOME/bin 目录中的 domain.sh 或 domain.bat。
- 适应于 JBoss 6 及更早版本的 JBoss 启动脚本

  默认情况下，JBoss 启动脚本是 $JBOSS_HOME/bin 目录中的 run.sh 或 run.bat。

根据 Web 服务器的要求，您可以使用不同的启动脚本，或者不同的启动脚本位置。

请执行以下步骤：

1. 导航到包含 JBoss 启动脚本的目录，例如：

   cd /jboss.GA/bin

2. 在文本编辑器中打开 JBoss 启动脚本，例如：

   vi run.sh / vi standalone.sh

3. 找到用于设置 Java 选项的命令行。要指定代理的启动 .jar 文件和代理配置文件的路径，请添加以下命令行选项：

   - 对于 JBoss 7 及更高版本

     -Djboss.modules.system.pkgs=com.wily,com.wily.* -javaagent:<代理 Jar 文件的路径> -Dcom.wily.introscope.agentProfile=<*代理配置文件的路径*>

   - 对于 JBoss 6 及更早版本

     -javaagent:<代理 Jar 文件的路径>
     -Dcom.wily.introscope.agentProfile=<*代理配置文件的路径*>

   例如，如果正在使用 Agent.jar，请在用于启动服务器的命令之前添加类似于以下示例的代码：

   - 对于 JBoss 7 及更高版本

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

   - 对于 JBoss 6 及更早版本

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

4. 保存 run.bat 或 standalone.bat 文件。

5. （可选）通过将代理配置文件配置为收集 JMX 度量标准，可报告 JBoss JMX 度量标准。

   a. 在文本编辑器中打开 IntroscopeAgent.profile 并设置 introscope.agent.jmx.enable=true。

      注意：要通过结合使用 JMX 远程管理服务器和特定于平台的 MBeanServer 来从 JConsole 中的 JBoss 查看 JMX 度量标准，应将 com.wily.use.platform.mbeanserver=true 添加到 IntroscopeAgent.profile 中。此配置已相对于先前版本的 Introscope 进行了更改，在先前的版本中，特定于平台的 MBeanServer 的使用是在命令行中设置的。

b. 保存并关闭 introscopeAgent.profile 文件。

c. 对于 JBoss 7 或更高版本，导航到 JBoss 的安装目录 <代理主目录>/wily/common 并将 WebAppSupport.jar 文件移动到 <代理主目录>/wily/core/ext 目录。

d. 对于 JBoss 6 及更早版本：

- 导航到 <代理主目录>/wily/common 目录并将 WebAppSupport.jar 文件移动到 /server/<服务器配置>/lib 目录。
- 导航到 <代理主目录>/wily/deploy 目录并将 introscope-jboss-service.xml 文件移动到 /server/<服务器配置>/deploy 目录。

注意：这些路径假定您使用的是默认配置。否则，将文件移动到等效的 JBoss 安装目录。

JBoss 应用程序服务器日志记录注意事项

使用 JBoss 应用程序服务器时，请考虑以下信息。

症状：

以下行为对于在应用程序服务器上执行日志记录的任何代理来说很常见：

- 对于 JBoss 6 系统，应用程序服务器不会创建 boot.log 文件。
- 对于 JBoss 7 系统，代理不会启动或显示以下错误：

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

解决方案：

使用以下任一方法可以解决这些问题：

- 在代理配置文件中关闭代理日志记录并启动服务器。在服务器启动之后启用日志记录。
- 在文本编辑器中打开 JBoss 启动脚本，并使用以下命令进行更新：

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

JBoss-specific PBD 和 PBL

在 JBoss 应用程序服务器上安装 Java 代理时，您可以使用以下特定于 JBoss 的 PBD 和 PBL 来自定义数据收集：

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

配置 JBoss 7 以进行自动命名

仅当您将 webappsupport.jar 文件复制到应用程序服务器部署文件夹时，自动命名功能才能与 JBoss 7 结合使用。

示例

对于独立服务器，请将 webappsupport.jar 复制到 JBOSS7_HOME/standalone/deployments/ 文件夹。

将 Oracle WebLogic 配置为使用 Java 代理

要将 Oracle WebLogic 配置为使用 Java 代理，请编辑 WebLogic 启动脚本。根据您的要求，您所使用的启动脚本可以特定于 WebLogic 域。默认情况下，WebLogic 启动脚本是 *$WEBLOGIC_HOME*/samples/domain/<*域名*>/bin 目录中的 startWebLogic.sh 或 startWebLogic.cmd。您可以使用其他启动脚本。例如，您可以使用特定于应用程序的启动脚本，或者不同的启动脚本位置。

请执行以下步骤：

1. 导航到包含您要修改的 WebLogic 启动脚本的目录。例如：
   ```
   cd $WL_HOME/samples/domain/wl_server
   ```
2. 在文本编辑器中打开 WebLogic 启动脚本。例如：
   ```
   vi startWebLogic.sh
   ```
3. 找到用于设置 Java 选项的命令行或 Java 命令行并添加以下命令行选项。例如：-javaagent:<*PathToAgentJar*>
   -Dcom.wily.introscope.agentProfile=<*PathToAgentProfile*>
4. 保存并关闭 WebLogic 启动脚本或特定于应用程序的启动脚本。

为 WebLogic 创建启动类

为应用程序服务器或群集创建 WebLogic 启动类，可以使 Java 代理从应用程序服务器中收集其他信息。如果您配置启动类，Java 代理可以自动确定其名称。启动类还可以使 Java 代理报告 JMX 度量标准，这些度量标准用于确定 Workstation 中应用程序的运行状况。

请执行以下步骤：

1. 打开 WebLogic 管理控制台。
2. 在左窗格中，展开“环境”文件夹。
3. 单击“启动和关闭”文件夹。

   将打开“启动”和“关闭”页面。
4. 单击“配置新的启动类”。

   将显示“配置”选项卡。
5. 在“名称”字段中，输入：
   `Introscope 启动类`
6. 在“类别名”字段中，输入：
   `com.wily.introscope.api.weblogic.IntroscopeStartupClass`
7. 单击“创建”。

   将显示“目标和部署”选项卡。
8. 选中要应用此启动类的服务器对应的框。
9. 单击“应用”并选择应用程序部署选项之前的“运行”。
10. 在 <*服务器或应用程序服务器*> 启动类路径中添加 WebAppSupport.jar 的位置。
11. 重新启动应用程序服务器。

在 WebLogic Server 中启用跨进程跟踪

通过事务跟踪会话，您可以跟踪事务中发生的所有操作，包括在具有兼容 JVM 版本的计算机上跨 JVM 边界的事务。

同步事务（例如 EJB 的 servlet）和异步事务都支持跨进程事务跟踪。

为 WebLogic 启用跨进程事务跟踪

1. 如为 WebLogic 创建启动类 (see page 38)中所述创建启动类。
2. 将“-Dweblogic.TracingEnabled=true”添加到用于启动 WebLogic Server 的 java 命令行中。
3. 在文本编辑器中打开 IntroscopeAgent.profile。
4. 找到 introscope.agent.weblogic.crossjvm 属性并将其设置为 true。例如：
   `introscope.agent.weblogic.crossjvm=true`
5. 保存并关闭 IntroscopeAgent.profile。

JMX 度量标准的 Java 代理支持

CA Introscope® 可以收集应用程序服务器或 Java 应用程序作为与 JMX 兼容的 MBean 公开的管理数据，并在调查器度量标准树中显示这些 JMX 数据。WebLogic 提供 JMX 度量标准的以下源：

- RuntimeServiceMBean：每服务器运行时度量标准，包括活动的有效配置
- DomainRuntimeServiceMBean：域范围内的运行时度量标准
- EditServiceMBean：允许用户编辑永久性配置

CA Introscope® 仅轮询 RuntimeServiceMBean 以获取度量标准。RuntimeServiceMBean 支持本地访问（效率问题），并且包含预期相关的大部分数据。但是，CA Introscope® 可以支持内置到 Sun JMX 规范中的任何 MBean。

注意：有关 Sun JMX 规范的详细信息，请参阅 http://java.sun.com/products/JavaManagement/。

为了支持对 JMX 度量标准的收集，在 WebLogic 的 IntroscopeAgent.profile 文件中默认定义并启用了以下关键字：

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

CA Introscope® 在调查器树中的以下节点下显示 JMX 度量标准：

<域>|<主机>|<进程>|<代理>|JMX|

详细信息：

启用 JMX 报告 (p. 181)

如何在 WebLogic 中设置资源度量标准

各种代理可以在 CA Introscope® Workstation 中报告资源度量标准类别。您可以配置 Oracle WebLogic 服务器，以便 Java 代理可以报告资源度量标准。

要为 Oracle WebLogic 设置资源度量标准，请执行以下步骤：

1. 使用《*CA APM for Oracle WebLogic Server 指南*》中的说明安装 CA APM for Oracle WebLogic。
2. 使用《*CA APM for Oracle Databases 指南*》启用“CA APM for Oracle Databases”选项。
3. 按照《*CA APM 配置和管理指南*》中有关配置资源度量标准的说明进行操作。

关于 WebLogic Diagnostic Framework (WLDF)

WebLogic Diagnostic Framework (WLDF) 是一种监控和诊断框架，用于定义和实现在 WebLogic Server 进程内运行并参与标准服务器生命周期的一组服务。使用 WLDF，您可以创建、收集、分析、存档和访问正在运行的服务器以及在其容器内部署的应用程序所生成的诊断数据。通过该数据，可以了解服务器和应用程序的运行时性能，并可在发生故障时隔离和诊断故障。

WLDF 允许用户通过标准接口动态访问服务器数据，并且可以修改在任意给定时间访问的数据量，而不必关闭并重新启动服务器。

如何将 WLDF 数据转换为 Introscope 度量标准

在 WLDF 中，信息被组织到一组数据访问器中，每个数据访问器都包含多个列。Introscope 可将此 WLDF 信息转换为 Introscope 特定的度量标准格式，并在调查器中的以下节点下显示：

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

数据访问器中的信息通过域名以及一个或多个键/值对定义。Introscope 将数据访问器的列转换为键和值信息，这些信息按 Introscope 生成的度量标准的字母顺序进行显示。下面的示例显示所使用的语法：

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

例如，下表显示 HTTPAccessLog 数据访问器中 BYTECOUNT 列的信息：

域名称	键/值对	度量标准名称
WebLogic	Name=HTTPAccessLog, Type=WLDFDataAccessRuntime=Accessor, WLDFRuntime=WLDFRuntime	BYTECOUNT

上表中的数据访问器信息将转换为以下 Introscope 度量标准：

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

请注意，在 Introscope 度量标准中，键/值对按字母顺序显示。

启用 WLDF 报告

默认情况下，在 Introscope 中不启用 WLDF 报告。您可以修改代理配置文件以启用 WLDF 度量标准报告。

启用 WLDF 报告

1. 如果托管应用程序正在运行，将其关闭。
2. 配置 WebLogic 启动类。
3. 在文本编辑器中打开 *IntroscopeAgent.profile*。
4. 找到并设置以下属性：
   ```
   introscope.agent.wldf.enable=true
   ```
5. 保存并关闭 *IntroscopeAgent.profile*。

将具有 JRockit JVM 的 WebLogic 配置为使用 Java 代理

适用于：具有 JRockit JVM 1.5 或更高版本的 WebLogic 9.0 或更高版本

注意：有关支持的 WebLogic 版本的信息，请参阅《*兼容性指南*》。如果使用的是较旧版本的 WebLogic 或 JRockit JVM，则可以创建并运行 AutoProbe 连接器文件 (see page 327)以检测应用程序。

如果使用的是具有 JRockit JVM 的 WebLogic，请使用以下命令行选项启动 JVM：

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

配置具有 JRockit JVM 的 WebLogic 以查看套接字度量标准

适用于：具有 JRockit JVM 1.5.0 的 WebLogic 9.0

症状：

由于第三方兼容性问题，查看套接字客户端的度量标准时可能会遇到问题。

解决方案：

使用下列“托管套接字”选项打开套接字客户端的度量标准。

请执行以下步骤：

1. 在文本编辑器中打开 toggles_typical.pbl 或 toggles_full.pbl 文件。
2. 启用“托管套接字”选项以跟踪套接字度量标准。例如：

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

3. 保存您的更改。

   将启用“托管套接字”选项。

将 IBM WebSphere 配置为使用 Java 代理

要将 IBM WebSphere 配置为使用 Java 代理，请编辑 WebSphere 启动脚本。根据您的要求，您所使用的启动脚本可以特定于应用程序服务器节点。

请执行以下步骤：

1. 导航到 $WEBSPHERE_HOME/profiles/<*应用程序服务器名称*>/config/cells/<*单元名称*>/nodes/<*节点名称*>/servers/server1 目录。

2. 编辑 server.xml 文件。此文件是 WebSphere 默认启动脚本和位置。

   注意：根据应用程序服务器的要求，您可以使用不同的启动脚本。例如，您可以使用特定于应用程序的启动脚本，或者不同的启动脚本位置。此外，不同操作系统上的不同 WebSphere 组合或者使用不同的 JVM 供应商或 JVM 版本时可能会有特殊要求。

3. 保存并关闭文件。

注意：有关详细信息，请参阅与您的 WebSphere 应用程序服务器环境最相近的章节。

在 UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 上配置 WebSphere Application Server 6.1

适用于：UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 上的 WebSphere Application Server 6.1

动态检测功能需要类重新定义支持。在 IBM JDK 版本 5 上运行时，使用类重新定义对性能有着显著的影响。想要使用动态检测的 CA Introscope® 和 IBM JDK 版本 5 客户应注意此性能开销。在部署此配置时，CA Technologies 建议仅在 QA 环境中使用动态检测功能。

注意：有关此性能开销的信息，请参阅 IBM 提供的《*Java Diagnostics Guide*》。

当您使用 IBM JVM 1.5 运行 WebSphere 6.1 时，请使用 Java 代理 .jar 文件和 Java 代理配置文件的备用版本。这些名为 AgentNoRedef.jar 和 IntroscopeAgent.NoRedef.profile 的文件位于 <*代理主目录*>/core/config 目录中。

注意：如果使用的是 AllAppServer 代理分发，则备用配置文件的名称为 IntroscopeAgent.websphere.NoRedef.profile。

由于使用先前的文件和语法，将不再为以下各项报告度量标准：

- 系统类
- NIO（套接字和数据报）
- SSL

以下功能也受到影响：

- 套接字检测对 9.0 版之前的 CA Introscope® 使用 ManagedSocket 样式。
- 远程动态检测已禁用。
- 重新启动已检测的 JVM 之后才能应用对 PBD 文件所做的更改。
- 深度继承和层次结构支持检测已禁用。

使用先前的文件和语法时，代理将通过以下操作报告类重新定义状态：

- 在调查器中的代理节点下添加名为“*已启用代理类重新定义*”的度量标准。度量标准值为 true 或 false。
- 将日志消息写入代理日志文件。
  - 如果启用了类重新定义，将在 WARN 级别写入日志消息，内容为：

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

  - 如果禁用了类重新定义，将在 INFO 级别写入日志消息，内容为：

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

- 向标准错误中写入消息（仅当启用类重新定义时），内容为：

  ```
  Warning: Introscope agent has been configured to support class
  redefinition. IBM JVMs version 1.5 and higher are known to incur
  significant overhead with redefinition enabled. To avoid this overhead
  please use AgentNoRedef.jar instead of Agent.jar.
  ```

如果您使用非 IBM JVM 或 1.5 版之外的 IBM JVM，将不输出先前的度量标准和消息。

您可以将 WebSphere 应用程序服务器配置为使用代理。

请执行以下步骤：

1. 打开 WebSphere 管理员控制台。
2. 导航到“应用程序服务器”> 您的服务器 >“服务器基础架构”>“Java 和进程管理”>“进程定义”>“Java 虚拟机”。
3. 将“常规 JVM 参数”字段设置如下：

   ```
   -javaagent:<代理主目录>/AgentNoRedef.jar
   -Dcom.wily.introscope.agentProfile=<代理主目
   录>/core/config/IntroscopeAgent.NoRedef.profile
   ```

   如果在同一台计算机上同时存在已检测的应用程序和未检测的应用程序，请在常规 JVM 参数中包含 -Xshareclasses:none 设置。此设置可避免在 AIX 上发生错误。

   注意：如果多个 WebSphere 版本使用同一代理目录，需要使用唯一的目录。

在 UNIX、Windows、Sun、HP、其他 JVM 1.5 上配置 WebSphere Application Server 6.1

适用于：UNIX、Windows、Sun、HP、所有其他 JVM 1.5 上的 WebSphere Application Server 6.1

注意：如果使用非 IBM JVM 或 1.5 版之外的 IBM JVM，则不会输出所有度量标准和消息。

您可以将 WebSphere 应用程序服务器配置为使用代理。

以下示例指示在 WebSphere 6.1 上安装 Java 代理时要用于特定 JVM 的 Java 参数和 .jar 文件：

请执行以下步骤：

1. 打开 WebSphere 管理员控制台。
2. 导航到“应用程序服务器”>您的服务器>“服务器基础架构”>“Java 和进程管理”>“进程定义”>“Java 虚拟机”。
3. 将“常规 JVM 参数”字段设置如下：

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

4. 重新启动 WebSphere 应用程序服务器。

在 UNIX、Windows、OS/400、JVM 1.5 上配置 WebSphere Application Server 7.0

适用于：WebSphere Application Server 7.0—UNIX、Windows、OS/400、JVM 1.5 或更高版本

您可以将 WebSphere Application Server 7.0 配置为使用代理。如果您正在监控 WebSphere 7.0，请先确认您已安装内部版本 7.0.0.8 或更高版本，然后再将服务器配置为启动 Java 代理。

请执行以下步骤：

1. 打开 WebSphere 管理员控制台。
2. 导航到“应用程序服务器”>您的服务器>“服务器基础架构”>“Java 和进程管理”>“进程定义”>“Java 虚拟机”。
3. 将“常规 JVM 参数”字段设置如下：

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

4. 重新启动 WebSphere 应用程序服务器。

配置 z/OS 上的 WebSphere Application Server 7.0 - JVM 1.5

适用于：z/OS 上的 WebSphere Application Server 7.0 - JVM 1.5 或更高版本

您可以将 WebSphere Application Server 7.0 配置为使用代理。在将服务器配置为启动 Java 代理之前，请确认您安装了内部版本 7.0.0.8 或更高版本。

请执行以下步骤：

1. 打开 WebSphere 管理员控制台。
2. 导航到“应用程序服务器”> 您的服务器 >“服务器基础架构”>“Java 和进程管理”>“进程定义”>“Java 虚拟机”。
3. 将“常规 JVM 参数”字段设置如下：

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

4. 重新启动 WebSphere 应用程序服务器。

修改 Java2 安全策略

为使 AutoProbe 在已启用 Java2 安全性的 WebSphere 环境中正确运行，可能需要向 Java2 安全策略中添加权限。

向 Java2 安全策略中添加权限

1. 在文本编辑器中打开文件 *$WebSphere home/properties/server.policy*。
2. 向该文件中添加以下权限：

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

   注意：为了提高用户可读性，这里使用了换行符，在向 *server.policy* 文件中添加权限时无需使用换行符。

3. 保存并关闭文件。

在 WebSphere 中配置自定义服务

您可以在 WebSphere 应用程序服务器中创建或修改自定义服务。通过自定义服务，Java 代理可以从应用程序服务器收集其他信息。如果您配置自定义服务，Java 代理可以自动确定其名称。自定义服务还使 Java 代理可以报告 JMX 和性能监控基础架构 (PMI) 度量标准。“应用程序概览”选项卡上的 Introscope Workstation 使用这些度量标准来确定应用程序运行状况。自定义服务可以对用于在 CA Introscope® 中访问 JMX 度量标准的用户凭据进行加密。

注意：要获取已添加到 WebSphere 应用程序服务器中的 SIBus 相关度量标准或新 PMI 模块，需要禁用现有的自定义服务，然后再创建一个自定义服务。

请执行以下步骤：

1. 打开 WebSphere 管理员控制台。
2. 选择您要配置的服务器，然后导航到“服务器基础架构”>“管理”>“自定义服务”。
3. 修改您所需的自定义服务或创建一个自定义服务。
4. 填写“配置”页上的下列字段，然后单击“确定”。

   在服务器启动时启用服务

   指定在服务器启动时启动服务。

   外部配置 URL

   指定配置属性文件的位置。对于 JMX 度量标准配置，可使用 jmxconfig.properties 文件，例如：<*Agent_Home*>/wily/common/jmxconfig.properties。

   类名称

   指定自定义服务类的名称，例如：

   com.wily.introscope.api.websphere.IntroscopeCustomService

   com.wily.powerpack.websphere.agent.PPCustomService

   显示名称

   指定在 CA Introscope® 中显示的名称，例如：Introscope 自定义服务。

   类路径

   指定属性文件的完全限定路径名，例如：

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

5. 配置用于访问 JMX 度量标准的用户凭据：
   a. 导航到 <*代理主目录*>/wily/common，然后在文本编辑器中打开 jmxconfig.properties 文件。
   b. 根据属性说明取消注释并设置值。
   c. 保存并关闭文件。
6. 重新启动应用程序服务器。

   服务器启动时，自定义服务也启动，用户凭据被加密。此后，每当服务器启动时，它都将使用加密后的密码。

例如：配置 jmxconfig.properties 文件以进行密码加密

下列文本显示为了进行密码加密设置的 jmxconfig.properties 的示例。

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

详细信息：

将 WebSphere 和 WebLogic 配置为使用 JMX 报告 (p. 185)

启用 WebSphere PMI 度量标准收集

CA Introscope® 可以通过使用 WebSphere 提供的 PMI 接口提取 WebSphere 性能监控基础架构 (PMI) 度量标准，从而提供 WebSphere 性能数据。

必须先在 WebSphere 中启用 PMI 数据收集，然后 CA Introscope® 才能使用这些数据。在 WebSphere 中，默认情况下，所有性能监控设置均处于关闭状态。

请执行以下步骤：

1. 在 WebSphere 中为 CA Introscope® 配置自定义服务。
2. 在 WebSphere 中启用 PMI 数据收集。
3. 在 IntroscopeAgent.profile 中启用 PMI 数据报告。

在 WebSphere 中启用 PMI 数据收集后，PMI 数据可显示为 CA Introscope® 度量标准。您可以根据需要筛选要在 CA Introscope® 中显示的度量标准类别。

注意：对于 z/OS 上的 WebSphere，CA Technologies 建议您使用 CA APM for WebSphere z/OS。CA APM for WebSphere z/OS 可提供特定于 WebSphere 的 PBD 和度量标准。这些 PBD 和度量标准使用低开销的跟踪器技术来检索特定于 WebSphere 的度量标准。适用于 WebSphere z/OS 的 CA APM 不要求您在 WebSphere for z/OS 中启用 PMI。您也可以在 WebSphere for z/OS 中启用 PMI 报告，但该方法会占用较多的系统资源。

在 Introscope 中配置 WebSphere PMI 度量标准报告

在 WebSphere 中打开性能监控设置后，必须在 Introscope 中启用 PMI 数据收集，并启用要报告的度量标准类别。

在 Introscope 中配置 PMI 度量标准报告

1. 关闭托管应用程序。
2. 在文本编辑器中打开 *IntroscopeAgent.profile*。
3. 在“WebSphere PMI Configurations”部分下，找到属性 *introscope.agent.pmi.enable*，并确认该属性已设置为 *true*。
4. 找到高级 PMI 度量标准类别的以下属性：

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

   默认情况下，会将四个类别（*threadPool*、*servletSessions*、*connectionPool* 和 *j2c*）设置为 *true*。您可以通过将相应的属性设置为 *true* 来启用其他度量标准类别，或者注释掉类别以减少报告的度量标准。

5. 保存更改并关闭文件。
6. 重新启动托管应用程序。

在 Introscope 中启用 PMI 收集后，将在调查器树中的 WebSpherePMI 节点下显示可用的 PMI 度量标准。

为 WebSphere 配置资源度量标准视图数据报告

各种代理可以在 CA Introscope® Workstation 中报告资源度量标准类别。在 WebSphere 中启用 PMI 数据收集后，您可以将应用程序服务器配置为报告资源度量标准视图数据。

请执行以下步骤：

1. 登录到 WebSphere 控制台。
2. 选择“监控和优化性能监控基础架构”。
3. 选择要用于报告的服务器（如 server1）。
4. 单击“配置”选项卡，然后选择“自定义”选项。
5. 单击“自定义”链接。

   将显示选项的配置树。
6. 在树上选择“JDBC 连接池”，然后在右侧窗格上启用“等待线程计数”。
7. 在树上选择“线程池”，然后在右侧窗格上启用“活动计数”。
8. 保存配置并重新启动应用程序服务器。

   应用程序服务器将配置为报告资源度量标准。

注意：有关资源度量标准视图数据的详细信息，请参阅*《CA APM 配置和管理指南》*。

在 WebSphere 中启用跨进程跟踪

通过事务跟踪会话，您可以跟踪事务中发生的所有操作，包括在具有兼容 JVM 版本的计算机上跨 JVM 边界的事务。同步事务（例如 EJB 的 servlet）和异步事务都支持跨进程事务跟踪。

在 WebSphere 中启用跨进程事务跟踪

1. 如在 WebSphere 6.1 中配置自定义服务 (see page 47)中所述创建自定义服务。
2. 打开工作区服务。

   从管理页面的“服务器”>“应用程序服务器”中，依次单击“server1”、“业务流程服务”和“工作区服务”，然后选中“在服务器启动时启用服务”框。
3. 在文本编辑器中打开 *IntroscopeAgent.profile*。
4. 找到 *introscope.agent.websphere.crossjvm* 属性并将其设置为 *true*。例如：

   ```
   introscope.agent.websphere.crossjvm=true
   ```
5. 保存并关闭 *IntroscopeAgent.profile*。

在用于 z/OS 的 WebSphere 中进行日志记录的注意事项

在 WebSphere z/OS 环境中进行日志记录时，需要注意一些事项。有关 Java 代理日志记录选项的详细信息，请参阅 Java 代理监控和日志记录 (see page 127)。

将日志输出标记为 EBCDIC

用于 z/OS 的 WebSphere 已将其默认编码由 EBCDIC CP1047 更改为 ASCII ISO8859-1。由于 z/OS 通常是 EBCDIC 计算机，因此必须对 Java 代理或 AutoProbe 写入的任何日志记录数据进行标记，以使用 EBCDIC 而不是 ASCII 作为最终的输出流。

将数据标记为 EBCDIC 而不是 ASCII

1. 打开位于 *<Agent_Home>\wily\core\config* 目录中的 *IntroscopeAgent.profile*。
2. 将以下属性添加到 *IntroscopeAgent.profile* 中：

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

通过日志记录工具消除启动计时问题

对于 WebSphere for z/OS，您可以使用属性通过 CA Introscope® 日志记录工具消除可能出现的任何启动计时窗口显示。

请执行以下步骤：

1. 打开位于 <*代理主目录*>\wily\core\config 目录中的 IntroscopeAgent.profile。
2. 将该属性添加到 IntroscopeAgent.profile 中：
   `introscope.agent.logger.delay=100000`

   该值以毫秒为单位，所以此示例中的默认延迟是 100 秒。
3. 保存 IntroscopeAgent.profile。

配置 Oracle Application Server 以使用 Java 代理

Oracle Application Server (OAS) 使用一个配置文件管理每个应用程序，因此也包括由 Oracle 控制台管理的每个 JVM。该文件通常命名为 *opmn.xml*。

修改 Oracle Application Server 以使用 Java 代理

1. 关闭应用程序服务器并对 *opmn.xml* 进行备份。
2. 在 *opmn.xml* 文件中查找要检测的应用程序所对应的部分。此时，您很可能要检测多个应用程序。
3. 在选定应用程序的 *<category id="start-parameters">* 下，找到名为 *<data id="java-options"* 的部分。
4. 在 *java-options* 行的末尾、最后的 *"/>* 之前使用适用于您环境的路径插入以下内容：
   `-javaagent:<代理 Jar 文件的路径> -Dcom.wily.introscope.agentProfile=<代理配置文件的路径>`

   例如，一个应用程序的完整条目应该在一行上：

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

配置 GlassFish 以使用 Java 代理

您可以配置 Sun GlassFish 开源应用程序服务器项目以使用 Java 代理。

注意：有关受支持的 Sun GlassFish 版本，请参阅 *《兼容性指南》*。

请执行以下步骤：

1. 在 GlassFish 中，导航到以下位置：
   /appserver-install-dir/domains/domain1/config/
2. 在文本编辑器中打开 domain.xml 文件。
3. 将 Agent.jar 文件和 IntroscopeAgent.profile 的完整路径作为 JVM 选项添加到 domain.xml 文件的 java-config 元素中。例如：

   ```
   <java-config ..........>
   <jvm-options>-javaagent:/sw/wily/Agent.jar</jvm-options>
   <jvm-options>-Dcom.wily.introscope.agentProfile=/sw/wily/core/config/Intr
   oscopeAgent.profile</jvm-options>
   .....>
   ```
4. 在文本编辑器中打开配置属性文件并添加 wily 属性。

   例如，对于 Glassfish 3.2，属性文件为 <glassfish_home>/glassfish/config/osgi.properties。

   要编辑的属性为：

   ```
   eclipselink.bootdelegation=oracle.sql, oracle.sql.*, com.wily.*
   org.osgi.framework.bootdelegation=sun.*,com.sun.*,com.wily.*
   ```
5. 将 Webappsupport.jar 文件从 <Agent_Home>/wily 根目录复制到 <Agent_Home>/wily/ext 目录。
6. 启动应用程序服务器。

将 SAP Netweaver 配置为使用 Java 代理

您可以为 SAP NetWeaver 配置 JVM AutoProbe 以使用 Java 代理。

注意：有关支持的 SAP NetWeaver 版本，请参阅《*兼容性指南*》。

请执行以下步骤：

1. 启动 SAP Configtool (configtool.bat)。
2. 选择要修改的实例。
3. 在右窗格中，选择“VM 参数”>“系统”。
4. 创建参数（不提供 -D）。例如：
   ```
   name: com.wily.introscope.agentProfile
   value: <Agent_Home>/core/config/IntroscopeAgent.profile
   ```

5. 单击“其他”选项卡并创建参数，例如：
   ```
   name: -javaagent:<Agent_Home>\Agent.jar
   value: <leave empty>
   ```
6. 保存您的更改。
7. 重新启动 SAP 服务器。
8. 要确认使用 Configtool 所做的更改已实施，请打开以下文件：
   ```
   <drive>:\usr\sap\...\j2ee\cluster\instance.properties
   ```
9. 找到以 ID<*服务器 id*>.JavaParameters 开头的行，并确认其中包含您输入的信息。

配置与企业管理器之间的连接

要报告度量标准，代理必须连接到企业管理器。默认通信设置可以使代理使用端口 5001 连接到本地企业管理器。但是，代理和企业管理器通常不会位于同一系统上。您可以在安装代理时修改默认设置，或者在安装代理之后通过修改 *IntroscopeAgent.profile* 来修改默认设置。

根据您的要求，您可以将代理与企业管理器之间的通信配置为使用以下连接：

- 直接套接字连接
- HTTP 隧道连接，或通过代理服务器的 HTTP 隧道
- HTTP over SSL（安全套接字层）连接 (HTTPS)
- 安全套接字层 (SSL) 连接

使用直接套接字连接来连接到企业管理器

代理连接到企业管理器的最常用方法是通过直接套接字连接。CA Technologies 建议使用直接套接字连接来连接到企业管理器（如果可以）。

配置从代理到企业管理器的直接套接字连接

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 找到 *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 属性，并指定在默认情况下代理应连接到的企业管理器的主机名或 IP 地址。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=sfcollect01
   ```
   如果您将一个群集与多个企业管理器一起使用，请确保指定一个收集器企业管理器。

3. 将 *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 属性设置为企业管理器侦听端口。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001
   ```
4. 将 *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 属性设置为用于连接到企业管理器的套接字工厂。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.DefaultSocketFactory
   ```
5. （可选）指定当与主企业管理器的连接断开时代理可以连接到的一个或多个备份企业管理器。
6. 保存并关闭 *IntroscopeAgent.profile* 文件。

使用 HTTP 隧道连接到企业管理器

如果与企业管理器的直接套接字连接不可行，您可以配置代理通过 HTTP 连接到企业管理器。通过此配置，通信可以通过仅允许 HTTP 通信的防火墙。

注意：相比于直接套接字连接，HTTP 隧道会在应用程序服务器和企业管理器上产生更多的 CPU 和内存开销。

为代理配置 HTTP 隧道

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 将 *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 属性设置为在默认情况下代理应连接到的企业管理器的主机名或 IP 地址。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT=webhost
   ```
3. 将 *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 属性设置为企业管理器的嵌入式 Web 服务器的 HTTP 侦听端口。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8081
   ```
   此属性的值应与在企业管理器的 *<EM_Home>/config/IntroscopeEnterpriseManager.properties* 文件中为 *introscope.enterprisemanager.webserver.port* 属性指定的值相匹配。默认情况下，该端口为 8081。
4. 将 *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 属性设置为 HTTP 隧道套接字工厂。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory
   ```
5. 保存并关闭 *IntroscopeAgent.profile* 文件。

为 HTTP 隧道配置代理服务器

您可以配置 HTTP 隧道代理，以便通过代理服务器连接到企业管理器。对于代理在防火墙之后运行并且防火墙仅允许通过代理服务器路由出站 HTTP 通信的转发代理服务器而言，此项配置很有必要。

只有将代理配置为通过 HTTP 进行隧道操作时，这些代理服务器配置属性才适用。代理服务器配置适用于代理上任何已配置的 HTTP 隧道连接，而不是仅适用于单个连接。在多个企业管理器之间（其中到每个企业管理器的连接均通过 HTTP 进行）配置故障转移时，此配置是需要考虑的一个重要因素。

重要信息！启用代理的 HTTP 隧道需要 HTTP/1.1。此外，代理服务器必须支持 HTTP Post。

重要信息！如果代理服务器不可访问，则代理绕过该代理服务器，并直接连接到企业管理器。如果代理服务器可访问，但是其身份验证失败，则代理会继续重试通过代理服务器连接到企业管理器。

为 HTTP 隧道配置代理服务器

1. 在文本编辑器中打开 IntroscopeAgent.profile。
2. 将 *introscope.agent.enterprisemanager.transport.http.proxy.host* 属性设置为代理服务器的主机名或 IP 地址。
3. 将 *introscope.agent.enterprisemanager.transport.http.proxy.port* 属性设置为代理服务器的端口号。
4. （可选）如果代理服务器需要用户凭据进行身份验证，请设置以下属性：

   ```
   introscope.agent.enterprisemanager.transport.http.proxy.username=<user_name>
   introscope.agent.enterprisemanager.transport.http.proxy.password=<user_password>
   ```
5. 保存并关闭 IntroscopeAgent.profile。

使用 HTTP 隧道连接到企业管理器

通过在 *IntroscopeAgent.profile* 文件中配置属性，代理可以使用 HTTP over SSL（安全套接字层）连接到企业管理器。

配置通过 HTTPS 进行的连接

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 将 *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 属性设置为目标企业管理器的主机名或 IP 地址。

3. 将 *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 属性设置为企业管理器的嵌入式 Web 服务器的 HTTPS 侦听端口。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444
   ```
4. 将 *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 属性设置为使用 HTTPS 隧道套接字工厂。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory
   ```
5. 保存并关闭 *IntroscopeAgent.profile*。

通过 SSL 连接到企业管理器

如果使用直接安全套接字层 (SSL) 连接，您可以配置代理以使用 SSL 而不使用 HTTP 隧道与企业管理器通信。

配置代理以使用 SSL 连接

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 将代理配置为使用 SSL 套接字工厂连接到企业管理器的 SSL 侦听端口。
3. 将 *introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT* 属性设置为目标企业管理器的主机名或 IP 地址。
4. 将 *introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT* 属性设置为企业管理器的 SSL 侦听端口。
5. 将 *introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT* 属性设置为 SSL 套接字工厂。例如，将该属性设置为：
   ```
   com.wily.isengard.postofficehub.link.net.SSLSocketFactory
   ```
6. 如果代理需要对企业管理器进行身份验证，请取消注释 *introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT* 属性，并将其设置为包含企业管理器证书的 truststore 的位置。如果不指定 truststore，代理将信任所有证书。您可以指定绝对路径或相对于代理配置文件的相对路径。例如：
   ```
   introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/certs
   ```
7. 根据需要设置 *introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT* 属性以指定 truststore 密码。

8. 如果企业管理器需要对客户端进行身份验证，请将 *introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT* 属性设置为包含代理证书的 keystore 的位置。您可以指定绝对路径或相对于代理配置文件的相对路径。在 Windows 上，反斜杠必须进行转义。例如：

   ```
   introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=C:\\keystore
   ```

9. 根据需要将 *introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT* 属性设置为 keystore 密码。

10. 将 *introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT* 属性设置为以逗号分隔的允许的密码套件列表。

    如果不指定此属性的值，则使用默认密码套件。

11. 保存并关闭 *IntroscopeAgent.profile*。

配置代理负载平衡

在工作负荷是代理报告的主要度量标准的群集中，可以通过配置 MOM 代理负载平衡来优化总体群集容量。

注意：有关 MOM 代理负载平衡的详细信息，请参阅《*CA APM 配置和管理指南*》。

升级多种代理类型

某些环境会在许多不同的应用程序服务器上分布数千个代理。例如，某个环境可能有 8000 个代理，其中有 3000 个代理分布在 WebLogic 上，2000 个代理分布在 WebSphere 上，还有 3000 个代理分布在 JBoss 上。为了便于跨多个应用程序服务器升级代理，您可以使用代理超集包。代理超集包是特定于操作系统的软件包，其中包含适用于除 SAP NetWeaver 之外的所有受支持的应用程序服务器的文件。

可用的代理超集包如下：

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

注意：在这些文件名中，*<version>* 表示 Java 代理的当前版本号。

每个包均包含以下文件：

- 所有特定于应用程序服务器的 PBD 和 PBL
- 所有应用程序服务器代理配置文件，文件名中嵌入了应用程序服务器名称。例如：
  - IntroscopeAgent.weblogic.profile
  - IntroscopeAgent.websphere.profile

  注意：不包括默认的 IntroscopeAgent.profile，有关详细信息，请参阅步骤 3。
- 适用于此操作系统类型的所有代理 JAR 文件和平台监视器

请执行以下步骤：

1. 选择适合目标操作系统的超集包。
2. 将选定的代理包提取到应用程序服务器的主目录中。

   注意：可以忽略 *<Agent_Home>*/core/config 目录中涉及您未使用的应用程序服务器的额外 PBD 和 PBL。
3. 如果您*尚未*配置 IntroscopeAgent.profile，请执行以下步骤：
   a. 选择适当的 IntroscopeAgent.<*application_server_name*>.profile。
   b. 将其重命名为 IntroscopeAgent.profile。
   c. 配置该文件，以使其可用于您的环境。
4. 如果您已配置 IntroscopeAgent.profile，请执行以下步骤：
   a. 在编辑器中打开相应的 IntroscopeAgent.<*application_server_name*>.profile。
   b. 查找您要使用的新属性。
   c. 将这些属性传输到现有的 IntroscopeAgent.profile 中。

详细信息：

使用安装存档手工安装 (p. 27)

卸载 Java 代理

卸载 Java 代理需要您知道每个被监控应用程序的 Java 代理的安装位置。

如果您使用 Java 代理安装程序安装 Java 代理，可使用卸载程序来删除安装的代理文件。启动卸载程序并按照屏幕上的说明进行操作。

从任何支持的 JVM 中卸载 Java 代理

1. 停止应用程序服务器。

   注意：在运行卸载程序之前，必须关闭所监控的 JVM。

2. 打开应用程序服务器启动脚本，并从 Java 命令行中删除 Java 代理开关。根据应用程序服务器，启动选项包括以下内容：

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

3. 重新启动应用程序服务器。
4. 运行位于 *<Agent_Home>*/UnstallerData 目录中的 *Uninstall_Introscope_Agent* 程序。
5. 手工删除 *<Agent_Home>* 目录。

从 z/OS 卸载 Java 代理

从 z/OS 卸载 Java 代理的建议方法是使用 *rm -rf* 命令删除 *<Agent_Home>* 目录。这样做很有必要，因为存在第三方错误，使得卸载程序可执行文件无法在 z/OS 中正常运行。

重要信息！务必在关闭受监控的 JVM 后再删除文件。

对于 z/OS 中的活动 Introscope 安装，务必要使 *UninstallerData* 文件夹保持原样，这一点非常重要。如果删除了 *UninstallerData* 文件夹，您将无法升级到 Introscope 的将来版本。请不要删除 *UninstallerData* 文件夹，除非您已决定卸载整个实例。

第 3 章： 配置代理属性

可使用 *IntroscopeAgent.profile* 文件中的属性配置大多数代理操作。本节介绍了要设置的最常见的代理属性。可能还有其他属性适用于您的环境。不同版本的代理可能会提供不同的属性供您设置，也可能具有不同的默认值。

此部分包含以下主题：

如何修改与企业管理器的通信

要报告度量标准，代理必须连接到企业管理器。在安装代理之后，您可以通过修改 *IntroscopeAgent.profile* 来修改使用的通信通道。

根据您的要求，您可以将代理与企业管理器之间的通信配置为使用以下连接：

- 直接套接字连接
- HTTP 隧道连接，或通过代理服务器的 HTTP 隧道
- HTTP over SSL（安全套接字层）连接 (HTTPS)
- 安全套接字层 (SSL) 连接

代理连接到企业管理器的最常用方法是通过直接套接字连接。CA Technologies 建议使用直接套接字连接来连接到企业管理器（如果可以）。如果您想使用其他类型的连接，请参阅相应的章节。

如何配置备份企业管理器和故障切换属性

安装代理时，可以指定代理在默认情况下连接到的企业管理器的主机名和端口号。如有需要，还可以指定一个或多个备份企业管理器。如果代理中断了与其主企业管理器的连接，可以尝试连接到备份企业管理器之一。

要使代理能够连接到备份企业管理器，请在代理配置文件中指定企业管理器的通信属性。如果主企业管理器不可用，代理会尝试连接到允许连接列表上的下一个企业管理器。如果代理无法与其第一个备份企业管理器连接，则它会尝试列表中的下一个企业管理器。此进程会按顺序一直尝试连接列表中的每个企业管理器，直到连接成功。如果代理无法连接到任何企业管理器，则它会等待 10 秒，然后再重试。

请执行以下步骤：

1. 在文本编辑器中打开 *Introscope.Agent.profile* 文件。
2. 通过将以下属性添加到每个备份企业管理器的代理配置文件，指定一个或多个备用企业管理器通信通道：

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

   将 *NAME* 替换为新企业管理器通道的标识符。在创建通道时，请不要使用 DEFAULT 或现有通道的名称。例如，要创建两个备份企业管理器：

   ```
   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. 找到 *introscope.agent.enterprisemanager.connectionorder* 属性，并将其设置为主企业管理器和备份企业管理器标识符的逗号分隔列表。标识符的列出顺序定义了它们的连接顺序。例如：

   ```
   introscope.agent.enterprisemanager.connectionorder=DEFAULT,BackupEM1,Back
   upEM2
   ```

4. 找到 *introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds* 属性，并指定代理尝试连接其主企业管理器的频率。默认时间间隔为 120 秒（2 分钟）。例如：

   ```
   introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120
   ```

5. 保存更改并关闭 *IntroscopeAgent.profile* 文件。
6. 重新启动应用程序。

如何启用并使用其他 GC 度量标准

垃圾回收和内存管理对应用程序性能有着重大影响。默认情况下，可以使用基本垃圾回收堆度量标准。此外，还可以启用其他可选度量标准，以提供有关垃圾回收处理和内存池使用情况的其他详细信息。当启用其他度量标准时，这些度量标准显示在调查器中的“GC 监视器”节点下。GC 监视器度量标准报告的信息可帮助您优化内存池分配和垃圾回收处理。因此，当开发或测试应用程序，或者研究应用程序性能问题时，通常会启用这些度量标准。在大多数情况下，这些度量标准不会用于生产环境中的实时应用程序管理，并且默认处于禁用状态。

如果要启用 GC 监视器度量标准，可以在文本编辑器中打开代理配置文件，并将 *introscope.agent.gcmonitor.enable* 属性设置为 true。在启用该属性之后，您可以查看有关正在监控的 JVM 的垃圾回收器和内存池的详细信息。有关度量标准的详细信息，请参阅 *《CA APM Workstation 用户指南》*。

如何启用并配置线程转储

线程转储可以提供有关代理 JVM 中正在发生的事情的有用详细信息。在与度量标准浏览器树的每个代理节点相关联的“线程转储”选项卡中都提供线程转储功能。

有关收集和分析线程转储的信息，请参阅 *《CA APM Workstation 用户指南》*。设置 Thread_Dump 权限可允许用户查看“线程转储”选项卡并使用所有功能。有关更多信息，请参阅 *《CA APM 安全指南》*。

启用线程转储需要使用 *IntroscopeAgent.profile* 和 *IntroscopeEnterpriseManager.properties* 属性。默认情况下，“线程转储”选项卡及其功能处于启用状态。但是，如果将其中一个或两个线程转储启用属性设置为 false，则用户将无法看到“线程转储”选项卡。

如果在 MOM 上启用或禁用线程转储，则该配置将应用于群集中的所有收集器。因此，如果在 MOM 上禁用线程转储，将同时在所有收集器上禁用线程转储。

启用线程转储

1. 打开 *<Agent_Home>*/wily/core/config 目录中的 *IntroscopeAgent.profile* 文件并设置此属性：
   ```
   introscope.agent.threaddump.enable=true
   ```
2. 保存并关闭 *IntroscopeAgent.profile*。

3. 打开位于 *<EM_Home>*/config 目录中的 *IntroscopeEnterpriseManager.properties* 文件并设置此属性：
   ```
   introscope.enterprisemanager.threaddump.enable=true
   ```
4. 保存并关闭 *IntroscopeEnterpriseManager.properties*。

为使 CA Introscope® 用户能够查看“死锁计数”度量标准，请配置 *IntroscopeAgent.profile*。您可以执行其他配置以显示代理“线程”节点度量标准。

启用“死锁计数”度量标准收集

1. 打开 *<Agent_Home>*/wily/core/config 目录中的 *IntroscopeAgent.profile* 文件。
2. 将此属性设置为 true 可启用“死锁计数”度量标准收集。
   ```
   introscope.agent.threaddump.deadlockpoller.enable=true
   ```
3. （可选）设置 PBL 的完整版本以在代理“线程”节点中显示度量标准。
   - 在属性 introscope.autoprobe.directivesFile 中指定 PBL 文件的名称。

     例如，要为 WebLogic 服务器使用标准 PBL 的完整版本，请将该属性设置为：
     ```
     introscope.autoprobe.directivesFile=weblogic-full.pbl
     ```
     用户可以在“*AgentName*”|“线程”下看到活动线程计数和线程组等度量标准。
4. 保存并关闭 *IntroscopeAgent.profile*。

使用 *IntroscopeAgent.profile* 和 *IntroscopeEnterpriseManager.properties* 属性配置线程转储。

配置线程转储

1. 打开位于 *<EM_Home>*/config 目录的 *IntroscopeEnterpriseManager.properties* 文件。
2. （可选）设置此属性，以将线程转储文件保存到企业管理器中的特定目录。例如，TestThreadDumps。
   ```
   introscope.enterprisemanager.threaddump.storage.dir=TestThreadDumps
   ```
3. （可选）设置此属性，以清除指定天数以前的线程转储文件。例如，清除 30 天以前的文件。
   ```
   introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days=30
   ```
4. （可选）设置此属性，以在指定天数以后清除线程转储文件。例如，每两天清除一次文件。
   ```
   introscope.enterprisemanager.threaddump.storage.clean.disk.freq.days=2
   ```

5. （可选）设置此属性，以限制可以存储在企业管理器上的线程转储文件的最大数量。例如，5000 个文件。

   ```
   introscope.enterprisemanager.threaddump.storage.max.disk.usage=5000
   ```

   注意：如果：

   * 存储的线程转储文件数超过在 introscope.enterprisemanager.threaddump.storage.max.disk.usage 属性中设置的限制

   且

   * 没有 introscope.enterprisemanager.threaddump.storage.clean.disk.olderthan.days 属性中设置的天数之前的文件

   则企业管理器不会存储任何线程转储文件。

6. 保存并关闭 *IntroscopeEnterpriseManager.properties*。
7. 重新启动企业管理器。

如果某个企业管理器关闭，您可以将线程转储文件复制到其他企业管理器，以便用户可以查看线程转储数据。

重要信息！在线程转储目录中添加或删除文件后，请重新启动企业管理器。CA Technologies 建议不要将线程转储文件从一个企业管理器移动到其他企业管理器。

将线程转储文件从一个企业管理器复制到其他企业管理器

1. 导航到包含线程转储文件的企业管理器 (EM1) 上的 <*EM_Home*>/threaddumps 目录。
2. 复制线程转储文件。
3. 将文件粘贴到用户要查看线程转储的企业管理器 (EM2) 上的 <*EM_Home*>/threaddumps 目录中。
4. 重新启动企业管理器 EM1 和 EM2。
5. 根据需要，在 EM2 上建立代理连接，然后启用并配置线程转储。

   EM2 用户可以选择代理节点，然后在“线程转储”选项卡中单击“加载先前”按钮。将显示从 EM1 移动的线程转储列表。

如何将代理配置为收集分布统计信息度量标准

您可以将代理配置为收集通过 BlamePointTracers 分析的响应时间分布情况，以创建平均响应时间度量标准。分布统计信息度量标准可提供详细说明，可解释具体操作的变化方式。有关监控的响应时间值的其他统计分布数据可通过 getExtendedMetricData Web 服务获取。

可将 CA APM 代理配置为收集特定操作的响应时间信息。这些响应时间存储在分布统计度量标准中，它与选定操作的平均响应时间度量标准成对。

请执行以下步骤：

1. 打开 <*Agent_Home*>/wily/core/config 目录中的 *IntroscopeAgent.profile* 文件。
2. 通过取消注释并编辑 introscope.agent.distribution.statistics.components.pattern 来指定要为其创建成对的分布统计信息度量标准的平均响应时间度量标准。例如，使用以下表达式：

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

   将生成 LoginServlet 响应时间的分布统计信息。
3. （可选）按照以下准则创建表达式：
   a. 在度量标准节点分隔符前加上反斜杠，因为竖线和句点在正则表达式中具有特殊含义。
   b. 在反斜杠前再加上一条反斜杠，因为反斜杠在代理配置文件中具有特殊含义。

      正则表达式在代理日志中显示在特殊字符之后。例如：

      ```
      Servlets\|Login Servlet:.*
      ```
   c. 将摘要级别与各个度量标准正则表达式进行匹配。如果度量标准不匹配，则汇总或不创建摘要级别的分布统计信息。

      以下示例显示匹配的表达式：

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

      ```
      Servlets.*
      ```
4. 保存并关闭 IntroscopeAgent.profile 文件。
5. 重新启动代理。

分布统计信息度量标准的示例

根据您设置配置属性的方式，可以针对以下示例收集分布统计信息：

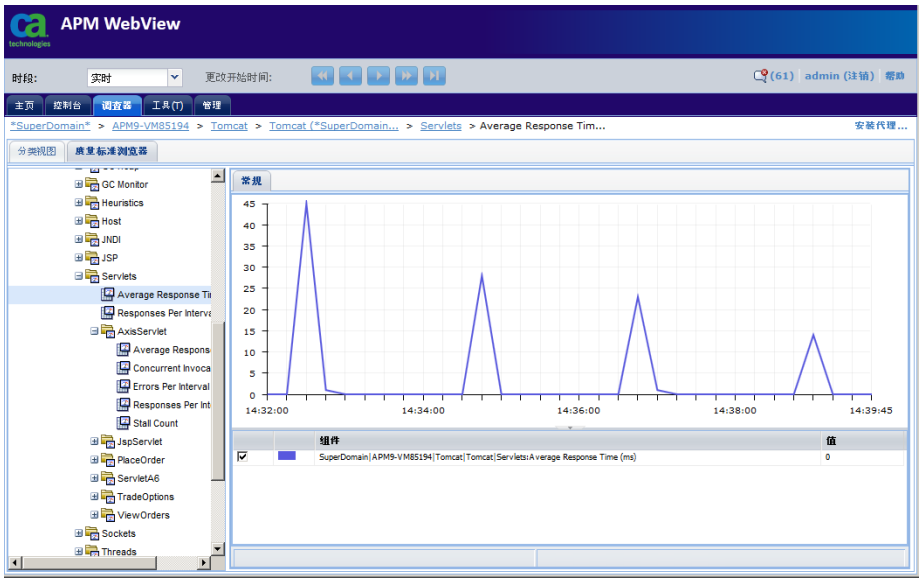
- 单个级别和摘要级别的 Servlet 和 JSP 分布度量标准的示例 (see page 69)
- Servlet 和 JSP 分布度量标准的示例 (see page 70)

单个级别和摘要级别的 Servlet 和 JSP 分布度量标准的示例

要收集 Servlet 和 JSP 单个级别和摘要级别的分布统计信息，请使用以下模式：

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

下图显示的调查器收集 Java 代理节点 servlet 级别和摘要级别的分布统计信息度量标准：

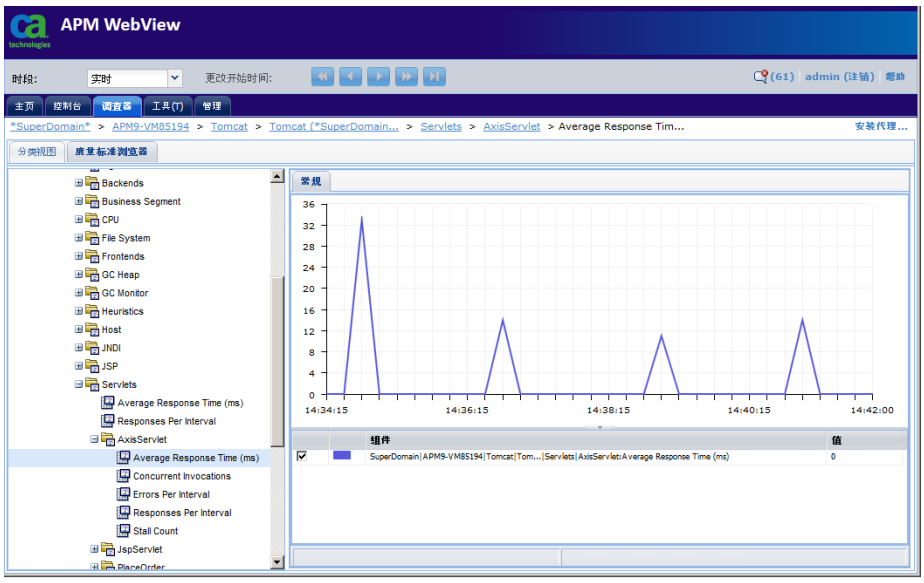


Servlet 和 JSP 分布度量标准的示例

要收集 Servlet 和 JSP 级别（但不收集 Servlet 或 JSP 摘要级别）的分布统计信息，请使用以下模式：

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

下图显示的调查器收集 Java 代理节点 servlet 级别（但不收集摘要级别）的分布统计信息度量标准：



第 4 章： AutoProbe 和 ProbeBuilding 选项

此部分包含以下主题：

AutoProbe 和 ProbeBuilding 概述

Java 代理将探测器插入您要监控的应用程序的字节码。ProbeBuilding 是使用 ProbeBuilder 指令 (PBD) 和 ProbeBuilder 列表 (PBL) 决定要插入应用程序的探测器的过程。Java 代理随附的默认 PBD 和 PBL 文件提供了基本级别的度量标准收集。在这些文件中修改默认设置，以便专为您的环境更好地调整度量标准收集。

配置 PBD 和 PBL，以便为您要在应用程序和环境中收集的度量标准插入探测器。然后，可以使用这些文件通过 JVM AutoProbe 自动检测应用程序或通过 ProbeBuilder 手工检测应用程序。使用 JVM AutoProbe 检测应用程序。但是，JVM AutoProbe 的配置取决于应用程序环境。

注意：有关支持的 JVM 版本，请参阅《*兼容性指南*》。

要在 JVM 中检测应用程序，请使用以下方法之一：

- JVM AutoProbe，具有 -javaagent 属性。CA Technologies 强烈建议使用 JVM AutoProbe 检测 JVM 中的应用程序。
- 手工 ProbeBuilding 是一种高级检测技术。使用该方法之前，请联系 CA Support。

重要信息！在检测应用程序时，只能使用一种检测方法。

确认您已经安装和配置 Java 代理 (see page 21)。

不支持的检测方法

Java 代理不支持“适用于应用程序服务器的 AutoProbe”检测方法。您可以使用“适用于应用程序服务器的 AutoProbe”来检测使用低版本 JVM（1.4 及更低版本）以及低版本 Java 代理的应用程序。CA Technologies 建议您为使用 JVM 1.5 或更高版本的应用程序使用 JVM AutoProbe。

注意：有关支持的 JVM 版本，请参阅《*兼容性指南*》。除非《*兼容性指南*》另有说明，否则支持的 JVM 为应用程序服务器版本随附的 JVM。

配置 ProbeBuilding

ProbeBuilding 技术执行检测过程。在 ProbeBuilder 指令 (PBD) 文件中定义的探测器用于标识代理在运行时从 Web 应用程序和虚拟机中收集的度量标准。

默认情况下，AutoProbe 使用随 Java 代理提供的典型 PBD。这样可以收集适当数量的度量标准。通过以下选项，您可以自定义度量标准收集级别并配置 ProbeBuilding 行为。

- 完整或典型跟踪选项 (see page 72)
- 动态 ProbeBuilding (see page 73)
- ProbeBuilding 类层次结构 (see page 76)
- 在字节码中删除行号 (see page 79)

完整或典型跟踪选项

在 Introscope 中，ProbeBuilder 列表 (PBL) 文件控制检测过程中使用哪些跟踪器组。*introscope.autoprobe.directivesFile* 属性指定一个或多个 PBL 文件。

Introscope 为每个默认 PBL 提供两个版本－*完整*版本和*典型*版本，完整版本启用的跟踪器组集合比典型版本启用的大，因此所生成的度量标准报告更为详细，典型版本启用的跟踪器组集合较小。这将导致度量标准报告不那么详细，但可以降低开销。默认情况下，*introscope.autoprobe.directivesFile* 指定默认 PBL 文件的典型版本。

在完整和典型之间更改跟踪级别

1. 停止托管应用程序。
2. 在文本编辑器中打开 *IntroscopeAgent.profile*。
3. 在以下属性中指定要使用的 PBL 文件的名称：*introscope.autoprobe.directivesFile*。

   例如，要为 WebLogic 服务器使用标准 PBL 的完整版本，请将该属性设置为：

   ```
   introscope.autoprobe.directivesFile=weblogic-full.pbl
   ```
4. 重新启动托管应用程序。

动态 ProbeBuilding

CA Introscope® 使用动态 ProbeBuilding 来实现新的和更改的 PBD，而无需重新启动托管应用程序或代理。动态 ProbeBuilding 对于更正 PBD 很有用，也可用于在分类期间更改数据收集级别，而无需中断应用程序服务。

重要信息！ 动态 ProbeBuilding 仅适用于 Java 1.5 或更高版本。动态 ProbeBuilding 依赖于 Java 1.5 功能和 -javaagent 命令。

注意： Workstation 允许您通过事务跟踪查看器执行动态检测。有关详细信息，请参阅 *《CA APM Workstation 用户指南》*。

动态 ProbeBuilding 会导致 CA Introscope® 定期查找新的 PBD 和更改的 PBD。为了最大程度地降低开销，CA Introscope® 可选择性地重新检测受修改后的 PBD 影响的类。为了提高性能，可将动态代理重新检测的范围限制为仅重新加载在编辑 PBD 时检测已更改的那些类。

在编辑 PBD 或将 PBD 添加到 hotdeploy 目录时，只会重新检测用户指令（如为类添加或删除指令，或者切换跟踪器组）。

重要信息！ 仅支持对使用跟踪器组的指令进行更改，例如，支持对具有 IfFlagged 开关的任何指令（如 TraceAllMethods）进行更改。但是，CA Introscope® 仅提供任何具有跟踪器组或标志的即用型指令。不支持对跳过或转换进行的更改。

不重新检测以下指令：

- 系统指令（如添加跟踪器或更改新的跟踪器映射）。
- 在跳过指令中指定的数组、接口和类以及任何转换。

通过配置重新检测过程，您可以：

- 排除由特定类加载器加载的所有类。
- 将范围限制为特定的类包。

注意： 默认情况下不启用动态 ProbeBuilding。

如果某个类在重新检测后不再报告某一度量标准的数据，该度量标准仍会显示在调查器中。重新检测现有度量标准的类后，这些度量标准不会从调查器窗口中消失。

重要信息！由于 Java 1.5 中存在限制，因此无法访问某些类字节，并且会产生以下影响：

- 无法选取对 j2ee.pbd 文件的修改，可以继续使用原名称发布度量标准。
- 代理日志中可能会显示一些异常。
- 要避免该问题，请在修改 j2ee.pbd 文件后重新启动应用程序服务器。

在配置动态 ProbeBuilding 时，CA Technologies 建议您基于跟踪器组进行更改。

示例：控制跟踪器组 XYZ 的检测级别

此示例演示了您如何可以控制跟踪器组的检测级别。

请执行以下步骤：

1. 创建两个跟踪器组：
   - XYZTracing－常规跟踪选项
   - XYZTracingLite－跟踪的组件较少
2. 在两者之间进行切换：关闭 XYZTracing 并打开 XYZTracingLite。
3. 查看动态 ProbeBuilding 对环境性能的影响。
4. 相应地调整跟踪组。

   调整会影响作为每个跟踪器组的一部分进行跟踪的所有类。

配置动态 ProbeBuilding

要配置动态 ProbeBuilding，请编辑 IntroscopeAgent.profile。

请执行以下步骤：

1. 导航到 <*代理主目录*>/wily/core/config 目录。
2. 在文本编辑器中打开 IntroscopeAgent.profile。
3. 确认属性 introscope.autoprobe.enable 已设置为 true。
4. 取消注释并设置以下属性：
   - introscope.autoprobe.dynamicinstrument.enabled=true

     该属性启用动态 ProbeBuilding。此属性将在您重新启动托管应用程序后生效。

- introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1

  检查 PBD 更改的轮询时间间隔（以分钟为单位）。默认时间间隔设置为一分钟。此属性将在您重新启动托管应用程序后生效。

- introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1

  根据观察，一些类加载器实现会返回较大的类文件。此行为是为了防止出现内存错误。此属性将在您重新启动托管应用程序后生效。

- introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10

  一次重新定义过多的类可能会占用大量 CPU。当 PBD 中的更改触发多个类的重新定义时，该属性会尝试以适当的速率对该过程进行批处理。

5. 将更改保存到 IntroscopeAgent.profile 并关闭文件。
6. 重新启动托管应用程序（如果适用）。

动态检测影响 IBM JDK 的性能

适用于：CA Introscope® 与 IBM JDK 版本 5

注意：在将 CA Introscope® 与 IBM JDK 版本 6 结合使用时，使用类重新定义没有性能开销。

症状：

动态检测需要支持类重新定义。使用类重新定义可能会大大影响性能。IBM 在《Java Diagnostics Guide》中提供了有关此性能开销的技术信息。想要利用动态检测的 CA Introscope® 和 IBM JDK 客户应记住此性能开销。

解决方案：

CA Technologies 建议仅在生产环境中使用动态检测功能。

详细信息：

在 UNIX、Windows、OS/400、z/OS、IBM JVM 1.5 上配置 WebSphere Application Server 6.1 (p. 43)
套接字度量标准 (p. 128)

动态 ProbeBuilding 与动态检测

动态 ProbeBuilding 与动态检测的不同：

- 动态 ProbeBuilding (see page 73) 基于您对 PBD 文件所做的手工更改以及在 IntroscopeAgent.profile 中进行的手工配置。如果您更新或更改 PBD 文件并将其保存在正确位置，动态 ProbeBuilding 将实现更改。动态 ProbeBuilding 要求您配置 IntroscopeAgent.profile，并更改要更新的 PBD 文件。所有更改都是永久的（直到您再次手工更新或更改文件）。
- 动态检测在 Workstation 事务跟踪查看器中进行。将自动对您使用该界面选择的检测进行更改，并且这种更改通常只是临时性的。动态地检测方法是指在运行时期间插入检测。您可以在事务跟踪会话期间动态检测一个、多个或所有方法。然后，可以查看新检测的方法返回的度量标准。通过此方法您可以动态调整应用程序性能。动态检测不需要对 IntroscopeAgent.profile 进行任何更改。如果决定使检测更改变为永久性更改，将为您创建一个 PBD 文件并将其保存在正确的位置。

  注意：有关如何在 Workstation 事务跟踪查看器中使用动态检测的详细信息，请参阅《*CA APM Workstation 用户指南*》。

  重要信息！ CA Introscope® 将动态检测缓存存储在应用程序服务器主机计算机上。为了使动态检测正常运行，应用程序服务器用于访问企业管理器的用户帐户必须对应用程序服务器主机计算机上的 <*代理主目录*> 和 <*代理主目录*>/logs 目录拥有写入访问权限。

ProbeBuilding 类层次结构

对于 1.5 版之前的 JVM，CA Introscope® 不会自动检测类层次结构中级别较深的类。CA Introscope® 仅检测明确扩展被探测类的类。

在使用支持的 JVM 的情况下，您可以将 CA Introscope® 配置为检测被探测类的多级子类。此产品可相应更新关联的内部指令中的跟踪器组并对这些类进行动态检测。指令更改将写入日志文件中。

注意：有关支持的 JVM 的信息，请参阅《*兼容性指南*》。

如果希望手工更新 PBD，可以禁用指令更新并使用日志文件确定相应的更新。

详细信息：

检测和继承 (p. 111)

启用多级子类的检测

请按照下列步骤将 Introscope 配置为动态更新内部指令。

启用多级子类的检测

1. 确认已按动态 ProbeBuilding (see page 73) 中所述启用动态检测。
2. 打开 *IntroscopeAgent.profile*。
3. 要启用多级子类的检测，请取消注释以下属性设置：
   `introscope.autoprobe.hierarchysupport.enabled=true`
4. 保存 *IntroscopeAgent.profile*。

对多重继承、接口和抽象方法的支持

Java 代理支持按接口和继承进行检测。此功能可扩展为动态检测。

Java 代理支持使用 API getMethodCalls 通过调用子类来检测方法。使用 getMethodCalls 时，您可以通过它提供的以下信息更好地了解继承方法或接口方法的检测结果：

- 定义方法的类是否为接口。
- 受可能的方法检测影响的类的数目。这是子类的数目或实现类的数目。
- 方法是否在特定的堆栈跟踪内调用。

Java 代理提供一个用于检测接口和抽象方法的跟踪器，语法如下：

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

在以下情况下，该跟踪器将检测实现接口或扩展超类的任何类的方法：

- 如果方法已在接口中定义。
- 如果方法是超类中的抽象方法。

重要信息！使用此跟踪器可能会严重影响系统性能。在系统启动和检测过程中测试该跟踪器的影响，然后再部署到较大的代理配置中。

详细信息：

创建自定义跟踪器 (p. 102)

为未检测的子类配置定期轮询

如果启用了多级子类检测，Introscope 将在应用程序启动时检查是否有未检测的子类。

配置 Introscope 以轮询未检测的子类

1. 打开 *IntroscopeAgent.profile*。
2. 取消注释以下属性设置：
   `introscope.autoprobe.hierarchysupport.runOnceOnly=false`
3. 要更改 Introscope 轮询未检测子类的默认频率值 5，请取消注释以下属性并将其设置为所需的轮询频率：
   `introscope.autoprobe.hierarchysupport.pollIntervalMinutes`
4. 也可以通过取消注释以下属性并将其设置为所需限值来限制 Introscope 轮询未检测子类的次数：
   `introscope.autoprobe.hierarchysupport.executionCount`

   该属性的默认值为 3 分钟。
5. 保存 *IntroscopeAgent.profile*。

禁用指令更新

如果启用了多级子类检测，则当 Introscope 检测到未检测子类时，默认情况下，它会适当更新内部指令以确保对这些类进行检测。如果您希望手动更新 PBD，则可通过在 *IntroscopeAgent.profile* 中取消注释以下属性来禁用内部指令更新：

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

控制指令日志记录

如果启用了多级子类检测，则必须在 *IntroscopeAgent.profile* 中取消注释以下属性，以创建多级子类检测日志。配置这些属性后，将在 *<Agent_Home>*/wily 目录（默认情况下）或自定义位置（如果进行了指定）创建名为 pbdupdate.log 的日志文件。多级检测详细信息将写入代理日志。

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

您必须重新启动托管应用程序，对这些属性所做的更改才能生效。

在字节码中删除行号

在检测应用程序字节码时，默认情况下会保留字节码行号。在使用调试程序或获取堆栈跟踪信息时，保留字节码行号信息很有帮助。

可以通过在 Java 命令行中添加系统属性来禁用该功能。如果禁用该功能，则会在 AutoProbe 或 ProbeBuilder 检测应用程序代码时删除所有行号。

在使用 AutoProbe 或 ProbeBuilder 时删除字节码中的行号

- 在 Java 命令行中使用 **-D** 选项定义以下系统属性：

  com.wily.probebuilder.removeLineNumbers=true

第 5 章： ProbeBuilder 指令

本节介绍如何创建和修改 ProbeBuilder 指令。

此部分包含以下主题：

ProbeBuilder 指令概览

ProbeBuilder 指令 (PBD) 文件可告知 Introscope ProbeBuilder 如何通过添加计时器和计数器等探测器来检测应用程序。PBD 文件控制代理向 Introscope 企业管理器报告哪些度量标准。

注意：所有度量标准都使用系统时钟所设置的时间来计算。如果在事务期间重置了系统时钟，则针对该事务报告的用时可能会具有误导性。

Introscope 包括一组默认 PBD 文件。您也可以创建自定义 Introscope PBD 文件以跟踪任何类或方法，从而获取有关应用程序的特定信息。

以下两种文件用于指定 ProbeBuilder 指令：

- **ProbeBuilder 指令 (PBD) 文件**

  ProbeBuilder 指令 (PBD) 文件包含 ProbeBuilder 用于检测应用程序的指令。此文件确定代理向企业管理器报告哪些度量标准。

- **ProbeBuilder 列表 (PBL) 文件**

  ProbeBuilder 列表 (PBL) 文件包含一个列有多个 PBD 文件名的列表。不同的 PBL 文件可能指示相同的 PBD 文件。

重要信息！PBD 和 PBL 仅支持 ASCII 字符。在 PBD 或 PBL 中放置其他字符（如 Unicode 字符）可能导致 AutoProbe 出现问题。

如果您安装 Java 代理时正在使用 Introscope AutoProbe，将包括您的特定应用程序服务器的相关 PBD 和 PBL 文件。这些文件位于 *<Agent_Home>*\wily\core\config 目录中。

详细信息：

默认 PBD 文件 (p. 83)
默认 PBL 文件 (p. 86)
创建自定义跟踪器 (p. 102)

默认 PBD 跟踪的组件

默认 Introscope PBD 文件会实现对下列 Java 组件的跟踪：

- Oracle JDBC
- JSP 标记库
- JSP IO 标记库
- JSP DB 标记库
- Struts
- Servlet
- JavaServer Faces (JSF)
- JavaServer Pages (JSP)
- Enterprise JavaBeans (EJB)
- Java 数据库连接 (JDBC)
- Network Sockets
- 远程方法调用 (RMI)
- 可扩展标记语言 (XML)
- Java 事务 API (JTA)
- Java Naming and Directory Interface (JNDI)
- Java Message Service (JMS)
- 公共对象请求代理体系结构 (CORBA)
- 用户数据报协议 (UDP)
- 文件系统
- 线程
- 系统日志
- 引发并捕获异常（在默认情况下关闭）

有时候，在 ProbeBuilder 指令 (PBD) 文件中选择过多的 Java 类进行监控会导致 Java 代理在启动时出现错误或发生“挂起”。如果出现这种情况，请使用 *AutoProbe.log* 文件来标识导致 Java 代理挂起的类，并向 PBD 文件中添加一条跳过指令，从而跳过可能引起该问题的类。有关向自定义 PBD 文件中添加跳过指令的详细信息，请参阅跳过指令 (see page 73)。

默认 PBD 文件

Java 代理包括以下默认 PBD 文件：

appmap.pbd

为应用程序分类视图检测提供跟踪器指令。

appmap-ejb.pbd

为应用程序分类视图 EJB 检测提供跟踪器指令。

appmap-soa.pbd

为支持 SPM 的 Java SOAP 堆栈的应用程序分类视图 SOA 检测提供跟踪器指令。

注意：有关详细信息，请参阅 *《CA APM for SOA 实施指南》*。

bizrecording.pbd

提供用于设置代理业务记录的跟踪器定义和指令。

biz-trx-http.pbd

为以业务为中心的 HTTP 检测提供跟踪器指令。

di.pbd

提供指令以使 ProbeBuilder 不检测 Apache Derby 实现类。

errors.pbd

通过指定哪些代码级别的事件构成严重错误来配置 Error Detector。默认情况下，仅将前端和后端错误视为严重错误。也就是说，只有明显的面向用户的错误页面或指示后端系统（ADO.NET、Messaging 等）出现问题的错误才构成严重错误。

j2ee.pbd

为公共 Java Enterprise Edition 组件提供跟踪器组。使用 toggles-full.pbd 或 toggles-typical.pbd 打开特定跟踪。

java2.pbd

为公共 Java 2 组件提供跟踪器组。使用 *toggles-full.pbd* 或 *toggles-typical.pbd 打开*特定跟踪。

jsf.pbd

为 Java Server Face (JSF) 组件提供跟踪器组。

jsf-toggles-full.pbd

提供打开/关闭开关（以 *TurnOn* 指令的形式），用于执行 *jsf.pbd* 中提供的跟踪。大多数跟踪器组都已打开。

jsf-toggles-typical.pbd

提供打开/关闭开关（以 *TurnOn* 指令的形式），用于执行 *jsf.pbd* 中提供的跟踪。

jvm.pbd

提供的指令可实现对各种 Java 虚拟机的支持。您可以将此属性与 Introscope 默认文件结合使用。

leakhunter.pbd

为 CA APM LeakHunter（一种泄漏检测实用工具）提供检测设置。通常，您无需修改此文件的内容。

lisa.pbd

为 CA LISA 检测提供跟踪器指令，以便于 CA APM 集成。

oraclejdbc.pbd

为 Oracle JDBC 组件提供跟踪器组。注释或取消注释 *TurnOn* 指令，以改变跟踪的 Oracle JDBC 组件集。

ServletHeaderDecorator.pbd

启用 Servlet Header Decorator，它是与 CA CEM 集成的一部分。

smwebagentext.pbd

为 SiteMinder Web Agent Introscope 插件提供跟踪器。

soaagent.pbd

为属于 CA SOA Security Manager（Web 服务器和应用程序服务器的 SOA 代理）一部分的 TransactionMinder 代理提供跟踪器。

spm-correlation.pbd

提供用于跨组件控制事务跟踪的关联的指令。当您使用 CA APM for SOA 时，启用跨进程事务跟踪需要此文件。

struts.pbd

提供用于监控 Apache struts 的指令。可将此属性与 Introscope 默认文件结合使用。

summary-metrics-6.1.pbd

提供在 7.0 之前的 Introscope 实例中进行 JSP 跟踪、Servlet 跟踪和 EJB 跟踪所需的指令。

taglibs.pbd

提供的指令用于监控应作为 JSP 标记库、Jakarta I/O 库和 DGTags 标记库进行跟踪的类。

TIBCO pbd

Java 代理在安装时提供了若干与通过 SOA 扩展监控 TIBCO 有关的 PBD。

注意：有关详细信息，请参阅 *《CA APM for SOA 实施指南》*。

toggles-full.pbd

提供打开/关闭开关（以 *TurnOn* 指令的形式），用于执行其他指令文件中提供的跟踪。大多数跟踪器组都已打开。

toggles-typical.pbd

提供打开/关闭开关（以 *TurnOn* 指令的形式），用于执行其他指令文件中提供的跟踪。只有一小部分跟踪器组已打开。

webMethods pbd

Java 代理在安装时提供了若干与通过 CA APM for webMethods Broker 监控 webMethods 有关的 PBD。

注意：有关详细信息，请参阅 *《CA APM for SOA 实施指南》*。

WebSphere MQ PBD

Java 代理在安装时提供了若干与通过 CA APM for IBM WebSphere MQ 监控 WebSphere MQ 连接器和消息系统有关的 PBD。

注意：有关详细信息，请参阅*《CA APM for IBM WebSphere MQ 指南》*。

Java 代理还会安装特定于应用程序服务器的 PBD，这些 PBD 会因监控的应用程序服务器而异。

详细信息：

默认跟踪器组和 Toggles 文件 (p. 86)
打开或关闭跟踪器组 (p. 95)

先前版本中的默认 PBD 文件

默认情况下，代理使用当前版本中的 PBD 和 PBL 文件。但是，产品在 <*代理主目录*>/wily/examples/legacy 目录中提供了先前版本中的 PBD 和 PBL 文件。该目录中的每个文件名都带有 -legacy 后缀，例如，default-full-legacy.pbl。

默认 PBL 文件

每个代理都有两个可用的 PBL 文件：

default-full.pbl（默认）

该文件引用其中大多数跟踪器组已打开的 PBD 文件。默认情况下，Introscope 使用此组文件来演示 Introscope 的完整功能。

default-typical.pbl

打开引用的 PBD 文件中的跟踪器组的子集。典型组包含通用设置，您可以针对特定环境自定义该组文件。

Java 代理还会安装特定于应用程序服务器的 PBL，这些 PBL 会因监控的应用程序服务器而异。

默认跟踪器组和 Toggles 文件

跟踪器组是在 PBD 文件中进行定义的。它们会报告有关一组类的信息。在 PBD 文件中，跟踪器组信息由术语 *“标志”* 来表示。例如，TraceOneMethodIfFlagged 或 SetFlag 定义了跟踪器组信息。

跟踪器组包括一组跟踪器，可应用于一组类。例如，有些跟踪器组会报告所有 RMI 类的响应时间和速率。

您可以通过打开或关闭某些跟踪器组来优化系统上的度量标准收集。此方法会影响开销使用情况，是增加还是减少则取决于您如何配置跟踪器组。

跟踪器组在 toggles-full.pbd 和 toggles-typical.pbd 文件中进行修改，并由 default-full.pbl 和 default-typical.pbl 文件引用。下表列出了默认跟踪器组及其默认配置：

AgentInitialization

代理初始化配置

完全：打开

典型：打开

ApacheStandardSessionTracing

HTTP 会话配置

完全：打开

典型：打开

AuthenticationTracing

身份验证配置

完全：打开

典型：打开

CorbaTracing

CORBA 方法调用

完全：打开

典型：打开

DBCPTracing

DBCP 配置

完全：打开

典型：打开

DBCPv55Tracing

DBCP 配置

完全：打开

典型：打开

EJB2StubTracing

EJB 2.0 配置

完全：打开

典型：打开

EJB3StubTracing

EJB 3.0 配置

完全：打开

典型：打开

EntityBean3Tracing

实体 EJB 3.0 方法调用

完全：打开

典型：打开

EntityBeanTracing

实体 EJB 方法调用

完全：打开

典型：打开

HTTPServletTracing

HTTP Servlet 服务响应

完全：打开

典型：打开

如果您使用的是应用程序服务器 AutoProbe，请打开此跟踪器组：HTTPAppServerAutoProbeServletTracing。

InstanceCounts

通过跟踪器组标识的对象类型实例的计数。

完全：打开

典型：打开

在使用此跟踪器组标识类之前，将不跟踪任何内容。

J2eeConnectorTracing

J2EE 连接器信息

完全：打开

典型：打开

JavaMailTransportTracing

邮件发送次数

完全：打开

典型：打开

JDBCQueryTracing

JDBC 查询

完全：打开

典型：打开

JDBCUpdateTracing

JDBC 更新

完全：打开

典型：打开

JMSConsumerTracing

JMS 消息处理时间

完全：打开

典型：打开

JMSListenerTracing

JMS 消息处理时间

完全：打开

典型：打开

JMSPublisherTracing

JMS 消息广播次数

完全：打开

典型：打开

JMSSenderTracing

JMS 消息广播次数

完全：打开

典型：打开

JSPTracing

JSP 服务响应

完全：打开

典型：打开

MessageDrivenBean3Tracing

消息驱动的 EJB 3.0 方法调用

完全：打开

典型：打开

MessageDrivenBeanTracing

消息驱动的 EJB 方法调用

完全：打开

典型：打开

NIOSocketTracing

为“NIO|通道|套接字”节点下的每个套接字连接生成一组度量标准，更多度量标准位于“后端”节点下。

完全：打开

典型：打开

NIOSocketSummaryTracing

生成一组涵盖所有 NIO 套接字连接的度量标准

完全：打开

典型：打开

这些度量标准包括通过代理属性从 NIOSocketTracing 度量标准中排除的连接。还包括始终从 NIOSocketTracing 度量标准中排除的一些内部 NIO 套接字。

NIOSelectorTracing

防止某内部 JVM 使用 NIO 通道被计入 NIO 通道度量标准中。

用户不能控制此选项。

NIODatagramTracing

为每个数据报“连接”生成一组度量标准。

完全：打开

典型：打开

有关详细信息，请参阅第 269 页中的 NIODatagramTracing 度量标准。

NIODatagramSummaryTracing

生成一组用于度量所有 NIO 数据报活动的度量标准。

完全：打开

典型：打开

这些度量标准包括通过代理属性从 NIODatagramTracing 度量标准中排除的连接。还包括始终从 NIODatagramTracing 度量标准中排除的一些内部 NIO 套接字。

PersistentSessionTracing

HTTP 会话配置

完全：打开

典型：打开

RMIClientTracing

RMI 客户端方法调用

完全：打开

典型：打开

RMIServerTracing

RMI 服务器方法调用

完全：打开

典型：打开

ServerInfoTracing

服务器信息配置

完全：打开

典型：打开

SessionBean3Tracing

会话 EJB 3.0 方法调用

完全：打开

典型：打开

SessionBeanTracing

会话 EJB 方法调用

完全：打开

典型：打开

SocketTracing

网络套接字带宽和 SSL 跟踪

完全：打开

典型：打开

StrutsTracing

Struts 框架中的操作执行时间

完全：打开

典型：打开

SuperpagesSessionTracing

HTTP 会话配置

完全：打开

典型：打开

ThreadPoolTracing

线程池配置

完全：打开

典型：打开

UDPTracing

用户数据报协议 (UDP) 套接字带宽

完全：打开

典型：打开

UnformattedSessionTracing

HTTP 会话配置

完全：打开

典型：打开

EJB3MethodLevelTracing

方法级别的 EJB 3.0 活动

完全：打开

典型：关闭

EJBMethodLevelTracing

方法级别的 EJB 活动

完全：打开

典型：关闭

FileSystemTracing

写入和读取的文件系统字节数

完全：打开

典型：关闭

JAXMListenerTracing

JAXM 消息发送

完全：打开

典型：关闭

JNDITracing

JNDI 查找时间

完全：打开

典型：关闭

JSPDBTagsTagLibraryTracing

用于从 SQL 数据库进行读写的 Jakarta DB 标记自定义标记库

完全：打开

典型：关闭

JSPIOTagLibraryTracing

用于各种输入和输出任务的 Jakarta IO 自定义标记库

完全：打开

典型：关闭

JTACommitTracing

使用 JTA 的提交次数

完全：打开

典型：关闭

ThreadTracing

按类划分的活动线程数

完全：打开

典型：关闭

XMLSAXTracing

分析 XML 文档所用的时间

完全：打开

典型：关闭

XSLTTracing

XML 转换时间

完全：打开

典型：关闭

CatchException

异常配置

完全：关闭

典型：关闭

FormattedSessionTracing

HTTP 会话配置

完全：关闭

典型：关闭

HTTPAppServerAutoProbeServletTracing

HTTP Servlet 配置

完全：关闭

典型：关闭

HTTPSessionTracing

HTTP 会话配置

完全：关闭

典型：关闭

JSPTagLibraryTracing

自定义 JSP 标记的处理时间

完全：关闭

典型：关闭

ManagedSocketTracing

网络配置

完全：关闭

典型：关闭

ThrowException

异常配置

完全：关闭

典型：关闭

通常，不应编辑默认的 *toggles* PBD 文件。但是，您可以通过打开或关闭某些跟踪器组来优化度量标准收集。可在 *toggles* 文件中通过以下方式修改跟踪器组：

- 打开/关闭跟踪器组以节约系统开销
- 向跟踪器组中添加类

仅当跟踪器组打开（取消注释）并使用关键字 *TurnOn* 激活后，才会报告信息。

注意：Java 代理支持 EJB（2.0 或更高版本）检测。要定制您的度量标准收集，请打开或关闭与 EJB 关联的跟踪器组。应用程序分类视图中对 EJB 的支持只扩展到会话和实体 Bean。不支持消息驱动的 Bean。

设置切换以收集其他度量标准信息

打开以下切换时，将在所有 API 中为已启用的 CA Technologies 提供的跟踪器组收集其他度量标准。必须将这些切换添加到完全或典型切换文件中，以更改配置。

DefaultStalledMethod 跟踪

停顿方法跟踪

完全：打开

典型：打开

DefaultConcurrentInvocationTracing

并发调用信息

完全：打开

典型：关闭

DefaultRateMetrics

调用速率度量标准

完全：关闭

典型：关闭

打开或关闭跟踪器组

您可以通过打开或关闭某些跟踪器组来优化系统上的度量标准收集。

打开跟踪器组

1. 根据 AutoProbe 或 Java 代理使用的文件类型是 *<appserver>-full.pbl* 还是 *<appserver>-typical.pbl*，查找 toggles-full.pbd 或 toggles-typical.pbd 文件。这些文件位于 *<appserver home>*wily/core/config 目录或 *<Introscope_Home>*/core/config/systempbd 目录中。
2. 找到要打开的跟踪器组，并通过删除行首的井号将该行取消注释。以下示例中的指令已打开，将对所有 HTTP Servlet 进行跟踪。

   ```
   TurnOn: HTTPServletTracing
   ```

   注意：跟踪器组中任何取消注释（打开）的指令都会导致跟踪器组被使用。

关闭跟踪器组

- 通过在行首放置井号来注释跟踪器组，如以下示例中所示：

  ```
  #TurnOn: HTTPServletTracing
  ```

向跟踪器组中添加类

您可以通过将特定类添加到现有跟踪器组来打开对该类的跟踪。要将某个类标识为跟踪器组的一部分，请使用其中一个标识关键字。

例如，将类 *com.myCo.ejbentity.myEJB1* 添加到跟踪器组 *EntityBeanTracing* 中：

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

标识关键字为：

- IdentifyInheritedAs
- IdentifyClassAs
- IdentifyCorbaAs

EJB 子类跟踪

默认情况下，与实体和会话 EJB 相关的指令只为直接、显式实现实体、会话或消息驱动的 EJB 接口的 EJB 添加探测器。

通常，应用程序的 EJB 是直接、显式实现实体或会话 EJB 接口的类的子类。默认情况下，Introscope 不跟踪这些子类。

对于 Introscope 要跟踪的 EJB 子类，必须将其添加到相应的跟踪器组。为此，请添加引用要跟踪的 EJB 子类的直接父类的条目。

从这些模型中，用要检测的 EJB 的直接父级的完全限定类名替换 *<entity.bean.ancestor.class>* 或 *<session.bean.ancestor.class>*。

对于实体 EJB：

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

对于会话 EJB：

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

下面的示例基于该类层次结构：

```
mySessionEJB 实现 javax.ejb.SessionBean
    mySessionEJBsubclass1 扩展 mySessionEJB
```

mySessionEJBsubclass1a 扩展 mySessionEJBsubclass1

mySessionEJBsubclass1b 扩展 mySessionEJBsubclass1

```
    mySessionEJBsubclass2 扩展 mySessionEJB
```

跟踪器组 *SessionBeanTracing* 会导致跟踪 *mySessionEJB*：

以下跟踪器跟踪 *mySessionEJBsubclass1* 和 *mySessionEJBsubclass2*。

IdentifyInheritedAs: mySessionEJB SessionBeanTracing

以下跟踪器跟踪 *mySessionEJBsubclass1a* 和 *mySessionEJBsubclass1b*。

IdentifyInheritedAs: mySessionEJBsubclass1 SessionBeanTracing

注意：此示例不使用包。如果您的代码在包中，则需通过类名来包含包名。

EJB 3.0 注释

通过下面的指令可将包含给定类级别注释的任何类划分到跟踪器组中。此指令支持 EJB 3.0。符合 3.0 规范的 EJB 不会显式实现任何已知接口，而是通过注释完全启用。为便于识别 EJB 3.0 类，请使用以下指令：

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

要使用此指令，请为新指令创建指令类和指令分析器类。然后，必须增加一个匹配器类，以通过检查字节码来确定类是否包含给定注释。

注意：此指令不支持方法级别的注释。

应用程序分类视图的 EJB 支持

CA Introscope® 支持 EJB（2.0 或更高版本）会话和实体 Bean 的即用型跟踪，尤其适合在 Workstation 应用程序分类视图中使用。由于此配置会影响代理的启动时间，因此 CA Technologies 建议只在测试环境中使用该即用型功能。

如果将此功能部署到您的生产环境，则针对特定内容配置 EJB 跟踪器。即用型功能可能过于宽泛。

使用以下指令可指示 ProbeBuilder 来标记继承自或者实现超类或接口的类：

```
IdentifyDeepInheritedAs
```

为了获得 EJB 2.0 应用程序分类视图支持，应在 j2ee.pbd 文件中包含以下指令：

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

通过这些指令，ProbeBuilder 可以标识应用程序分类视图使用的客户端的 EJB Stub 和服务器端的 Bean。

为了获得 EJB 3.0 应用程序分类视图支持，应在 j2ee.pbd 文件中包含以下指令：

```
IdentifyInheritedAnnotatedClassAs
```

该指令可匹配直接或通过超级接口实现接口的所有类。

在应用程序分类视图上下文中，在 j2ee.pbd 内设置了以下附加指令：

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

EJB 命名

您可以对调用的后端、常规前端和处理 EJB 的被监控组件进行命名。通过名称格式化程序，可为 EJB（2.0 或更高版本）客户端 Stub 和 Bean 实现配置合适的名称。

EjbNameFormatter 类可定义与 EJB 相关的度量标准名称、应用程序分类视图应用程序名称或节点名称。您可以使用以下占位符：

- 对于 EJB 客户端 Stub：*{classname}*、*{interface}* 和 *{method}*
- 对于 EJB Bean：*{classname}*、*{bean}*、*{interface}* 和 *{method}*

默认情况下，使用以下度量标准名称：

- EJB Bean 前端：*EJB | {interface}*
- EJB 客户端 Stub 后端：*EJB | {interface}*
- EJB Bean 的应用程序分类视图所有者名称：*{interface}*
- EJB 客户端 Stub 的应用程序分类视图节点名称：*客户端 {interface}*
- EJB Bean 的应用程序分类视图节点名称：*服务器 {interface}*

这些名称是默认的 EJB 名称格式化程序。它们用于 j2ee.pbd 和 appmap-ejb.pbd 文件中。您可以使用相同的名称格式化程序，但使用不同的度量标准名称。例如，可以修改现有的跟踪器指令以使用更合适的名称，但保留相同的标志：

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

注意：在 EJB 2.0 Bean 的 setContext() 方法中设置 EJB 上下文跟踪器。此跟踪器是用于 EJB 2.0 Bean 名称格式化程序的内部 CA Introscope® 跟踪器，它使得名称格式化程序可以正常运行。

将 IntroscopeAgent.profile、PBL 和 PBD 结合使用

首次安装 Java 代理时，需要将检测级别设置为完全或典型。此设置引用 ProbeBuilder 列表 (PBL) 文件 *default-typical.pbl* 和 *default-full.pbl*（有关详细信息，请参阅默认 PBL 文件 (see page 86)）。

应用 ProbeBuilder 指令

应用 PBD 的方式取决于您选择使用的方法。CA Technologies 建议您使用 JVM AutoProbe 来实现 PBD。也可以使用 ProbeBuilder 向导或命令行 ProbeBuilder 来实现 PBD。

使用 JVM AutoProbe

准备好实现 PBD 文件后，请将其添加到 *hotdeploy* 目录中。AutoProbe 会在包含 *IntroscopeAgent.profile* 文件的目录（默认情况下是 *<Agent_Home>*/wily/core/config 目录）和 *<Agent_Home>*/wily/core/config/hotdeploy 目录中查找 PBD 文件。AutoProbe 将相对于这些目录来解析文件名。如果移动了 *wily* 目录的位置，请务必将文件路径映射到正确的目录。

使用 AutoProbe 实现 PBD

1. 将修改过的标准 PBD 或 PBL 保存到 *<Agent_Home>*/wily/core/config 目录。
2. 将自定义 PBD 复制到 *<Agent_Home>*/wily/core/config/hotdeploy 目录中。将实现添加到此目录的任何 PBD，而不必在 *IntroscopeAgent.profile* 中更新或修改 *introscope.autoprobe.directivesFile* 属性。

   注意：如果已启用动态检测，将从文件夹中实时选取 *hotdeploy* 目录中的 PBD。无需重新启动。有关动态检测的详细信息，请参阅动态 ProbeBuilding (see page 73)。
3. 保存 *IntroscopeAgent.profile*。
4. 重新启动应用程序。

使用 ProbeBuilder 向导或命令行 ProbeBuilder

准备好实现 PBD 文件后，请将其添加到 *hotdeploy* 目录中。命令行 ProbeBuilder 将在 ProbeBuilder 的运行目录以及 *<Agent_Home>*/wily/core/config/hotdeploy 目录中查找自定义指令文件。命令行 ProbeBuilder 将相对于这些目录来解析文件名。

使用 ProbeBuilder 向导或命令行 ProbeBuilder 实现 ProbeBuilder 指令的步骤与使用 JVM AutoProbe 时的步骤相同。有关详细信息，请参阅使用 JVM AutoProbe (see page 99)。

使用新的和更改的 PBD 进行检测

为使新指令或更改的指令生效，必须使用最新的 PBD 检测应用程序。此过程因您所使用的 ProbeBuilding 方法而异。

通过 -javaagent 使用 JVM AutoProbe 的 JVM 1.5 系统

您可以配置动态检测，使得更改的 PBD 无需重新启动应用程序或 Java 代理即可生效。这使您在执行 PBD 更正或分类驱动检测时不会中断应用程序服务。有关详细信息，请参阅动态 ProbeBuilding (see page 73)。

新的和更改的 ProbeBuilder 文件

适用于：1.5 版之前的 JVM 或使用 –Xbootclasspath 进行的安装

下次应用程序服务器加载应用程序类时，新的和更改的 ProbeBuilder 指令文件或 ProbeBuilder 列表文件将生效。

如果添加或更改指令时您的托管应用程序未运行，则在下次启动应用程序时会使用更新的指令对应用程序进行检测。

如果您的托管应用程序正在运行，则必须加载或重新加载托管应用程序类。

引起类重新加载的方式取决于您使用的应用程序服务器。大多数应用程序服务器都需要重新启动。

使用 ProbeBuilder 向导

使用 ProbeBuilder 向导

1. “自定义指令”屏幕将列出您在使用 ProbeBuilder 向导或命令行 ProbeBuilder (see page 100) 中所述的 *hotdeploy* 目录中放置的 PBD 文件。
2. 选择要使用的自定义指令文件。

使用命令行 ProbeBuilder

重要信息！ CA Technologies 建议使用命令行 ProbeBuilder 作为 Introscope 启用最新 PBD 的最后选项。

使用命令行 ProbeBuilder

1. 停止托管应用程序。
2. 运行命令行 ProbeBuilder 或 ProbeBuilder 向导，在命令行中提供自定义 PBD 和 PBL 文件。
3. 配置应用程序以使用新文件。
4. 启动托管应用程序。
5. 如果企业管理器和 Workstation 尚未运行，请将其启动。

创建自定义跟踪器

您可以通过创建自定义 PBD 文件来进一步改善度量标准的收集。创建自定义指令（通过创建跟踪器来跟踪特定于应用程序的度量）需要使用特定的语法和关键字。要编写自定义跟踪器，您必须定义：

- 指令类型（通常指示要跟踪的类或方法的数量）
- 要跟踪的特定类或方法
- 类或方法中要跟踪的信息类型（例如，时间、速率或计数）
- 要在其下显示该信息的完全限定度量标准名称（包括资源路径）

自定义 PBD 存储在 *<Agent_Home>*/wily/core/config/hotdeploy 目录中。将实现添加到此目录的任何 PBD，而不必在 *IntroscopeAgent.profile* 中更新或修改 *introscope.autoprobe.directivesFile* 属性。如果已启用动态检测，将从文件夹中实时选取 *hotdeploy* 目录中的 PBD。无需重新启动。有关动态检测的详细信息，请参阅动态 ProbeBuilding (see page 73)。

创建了自定义 PBD 后，Introscope 会将其视为即用型 PBD。您可以在创建的度量标准中设置报警，将报警保存到 SmartStor，或者在 Introscope Workstation 中创建自定义控制板时使用报警。

注意：请务必谨慎选择要跟踪的方法，因为跟踪的方法越多，意味着系统的开销越大。

将自定义 BlamePointTracer 跟踪器用于常用度量标准

BlamePointTracer 是最常用的跟踪器。此跟踪器为关联的方法或类生成五个单独的度量标准：

- 平均响应时间 (毫秒)
- 并发调用
- 每个时间间隔的错误
- 每个时间间隔的响应数
- 停顿计数

下面是 *BlamePointTracer* 的一个示例。已为类 *petshop.catalog.Catalog* 中名为 *search* 的方法设置了 *BlamePointTracer*。 *“PetShop|目录|搜索”* 是在 Introscope 调查器中将显示 BlamePoint 度量标准的节点的名称。

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

跟踪器语法中使用的指令名称和参数

除用于将跟踪器关联到组或启用/禁用组的简单关键字外，PBD 文件还包含跟踪器定义。为使 Introscope 能够识别并处理跟踪器，必须在构建自定义跟踪器时使用特定的语法。跟踪器由指令和有关要跟踪的方法或类的信息组成，格式如下：

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

其中，*[arguments]* 是一个列表，且特定于指令。

注意：根据所使用的指令，仅需要这些参数中的一部分。

<directive>

最常用的指令是以下跟踪指令：

TraceOneMethodOfClass

跟踪指定类中的指定方法。

TraceAllMethodsOfClass

跟踪指定类中的所有方法。

TraceOneMethodIfInherits

跟踪指定类的所有直接子类实现或指定接口的所有直接接口实现中的一个方法。

TraceAllMethodsIfInherits

跟踪指定类的所有直接子类实现或指定接口的所有直接接口实现中的所有方法。

TraceOneMethodIfFlagged

如果指定的类包含在已使用 *TurnOn* 关键字启用的跟踪器组中，则跟踪一个方法。

TraceAllMethodsIfFlagged

如果指定的类包含在已使用 *TurnOn* 关键字启用的跟踪器组中，则跟踪所有方法。

注意：只能跟踪已实施的具体方法，而且只有此类方法会在运行时报告度量标准数据。在自定义跟踪器中指定抽象方法将导致无法报告度量标准数据。

跟踪指令的预期语法通常包括以下参数：

<Tracer-Group>

与跟踪器相关联的组。

<class>

要跟踪的完全限定类或接口名称。完全限定类包括类的完整程序集名称以及类名称，例如：

[MyAssembly]com.mycompany.myassembly.MyClass

程序集名称必须括在方括号 [] 内。

<method>

方法名称（例如，*MyMethod*）

或者

具有返回类型和参数的完整方法签名（例如 *myMethod;[mscorlib]System.Void([mscorlib] System.Int32)*。有关方法签名的详细信息，请参阅区分签名 (see page 107)。）

<Tracer-name>

指定要使用的跟踪器类型。例如，*BlamePointTracer*。有关跟踪器名称的说明，请参阅下面的跟踪器名称表。

<metric-name>

控制所收集的数据如何在 Introscope Workstation 中显示。

以下示例介绍了三种在度量标准树中的不同级别上指定度量标准名称和位置的方法。

metric-name—度量标准直接显示在代理节点内。

resource:metric-name—度量标准显示在代理节点下的一个资源（文件夹）内。

resource|sub-resource|sub-sub-resource:metric-name—度量标准显示在代理节点下的多个资源（文件夹）级别中。使用管道字符（|）分隔资源。

常用的跟踪器名称和示例

以下列表描述了最常用的跟踪器的名称及其跟踪内容。

BlamePointTracer

提供标准度量标准组，包括平均响应时间、每个时间间隔的计数、并发、停顿和问题组件的错误。

ConcurrentInvocationCounter

报告方法已启动但未完成的次数。将在调查器树的跟踪器 *<metric-name>* 中指定的度量标准名称下报告该结果。此跟踪器的一个使用示例是对同时进行的数据库查询数目进行计数。

DumpStackTraceTracer

针对堆栈跟踪所应用到的方法，将堆栈跟踪转储到被检测应用程序的标准错误中。Dump Stack Tracer 所抛出的异常堆栈跟踪不是真正意义上的异常－它是一种输出方法堆栈跟踪的机制。

此功能对于确定方法的调用路径非常有用。

重要信息！此功能会大大增加系统开销。强烈建议仅在诊断上下文（此时容许系统开销急剧增加）中使用此跟踪器。

MethodCPUTimer

在方法执行期间花费的平均 CPU 时间（以毫秒为单位），在度量标准树中的 *<metricname>* 下报告该时间。

注意：此跟踪器需要受支持平台上的平台监视器。

MethodTimer

平均方法执行时间（毫秒），会在度量标准树中的跟踪器 *<metric-name>* 中指定的度量标准名称下报告该时间。

PerIntervalCounter

每个时间间隔内的调用数。此时间间隔将基于数据使用方（例如，调查器中的“视图”窗格）的查看时间段进行更改。将在调查器树中的跟踪器 *<metric-name>* 中指定的度量标准名称下报告该时间间隔。

在自定义跟踪器定义中使用引号

自定义跟踪器可包含带有空格的度量标准名称。在自定义度量标准名称中使用空格时，CA Technologies 建议在所有度量标准名称两侧加上引号 ("")。

重要信息！请不要在类名称两侧加引号。否则会引起自定义跟踪器出现故障。例如：

正确

```
IdentifyClassAs: MyClass MyTracers
```

不正确

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

如果创建包含类名称的度量标准名称，则必须在整个度量标准名称两侧加上引号。允许在度量标准名称中使用空格，度量标准名称中的所有空格都必须包含在引号内。例如，度量标准名称 "{classname}|Test One Node" 应按如下所示表示：

正确

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

不正确

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

重要信息！ Introscope 将不监控类文件名无效的类。例如，在下面的类文件名中：

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

_[B: 会导致类文件名无效。左方括号（[）不能作为 Java 类文件名的一部分。当 Introscope 遇到这类具有无效类名的类时，它将无法检测这些类，并会在代理日志中以错误消息的形式报告这些类。

以下部分是方法跟踪器示例。在以下示例中，在度量标准名称两侧使用了引号 ("")。CA Technologies 强烈建议在创建自定义度量标准名称时在所有度量标准名称两侧加上引号。

平均值跟踪器示例

此跟踪器将跟踪指定方法的平均执行时间（毫秒）。

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

比率跟踪器示例

此跟踪器将对每秒内的方法调用次数进行计数，并在指定的度量标准名称下报告此比率。

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

每个时间间隔内的计数器跟踪器示例

该方法跟踪器将对每个时间间隔内的方法调用次数进行计数，并在指定的度量标准名称下报告每个时间间隔内的计数。

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

时间间隔由企业管理器中的监控逻辑（例如，图表频率）决定。

调查器中的预览窗格默认采用 15 秒的时间间隔。

计数器跟踪器示例

此跟踪器将对调用方法的总次数进行计数。

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

组合的计数器跟踪器示例

这些跟踪器会将增量跟踪器和减量跟踪器组合起来，以保持计数的连续性。

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

高级单度量标准跟踪器

指令和跟踪器将跟踪方法、类和类组。单度量标准跟踪器将报告特定方法的特定度量标准，这是 Introscope 可跟踪的最小单位。可通过以下多种方式创建单度量标准跟踪器：通过方法签名、通过替换关键字或者通过使用度量标准名称参数。

区别签名

可基于方法签名将跟踪器应用到方法。

要使用特定签名跟踪某个方法的单个实例，请将签名附加到使用内部方法描述符格式指定的方法名称（包含返回类型）。

例如，*myMethod(Ljava/lang/String;)V* 跟踪带有字符串参数和 void 返回类型的方法的实例。

有关此格式的完整信息，请参阅 *Sun Java Virtual Machine Specification*（“Sun Java 虚拟机规范”）

基于关键字的度量标准名称替换

基于关键字的替换允许在运行时将值替换为度量标准名称。

会在运行时将跟踪器中的度量标准名称中代表实际值的参数替换为度量标准名称。此功能可与任何指令一起使用。

{method}

所跟踪方法的名称

{classname}

所跟踪类的运行时类名称

{packagename}

所跟踪的类的运行时包名称

{packageandclassname}

所跟踪的类的运行时包和类名称

注意：如果 Introscope 处理的类没有程序包，它会将 {packagename} 替换为字符串“<UnnamedPackage>”。

基于关键字的替换：示例 1

如果 PBD 文件中跟踪器的度量标准名称为：

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

且将跟踪器应用于具有运行时类 myClass（在包 myPackage 中）的方法 myMethod，将得到以下度量标准名称：

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

基于关键字的替换：示例 2

如果将 .pbd 文件中具有度量标准名称

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

的跟踪器应用于同一方法，将得到以下度量标准名称

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

注意： 在程序包与类之间使用的是“.”，而不是第一个示例中使用的“|”。

基于度量标准名称的参数

您可以按以下格式创建一个单方法跟踪器，该跟踪器将使用 *TraceOneMethodWithParametersOfClass* 关键字基于传递给方法的参数创建度量标准名称：

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

可在度量标准名称中使用参数。这可通过在度量标准名称中使用占位符字符串替换参数值来完成。要使用的占位符字符串是“{#}”，其中 # 是要替代的参数的索引。索引从零开始计数。可按任何顺序使用任何数目的参数替换项。在替换为度量标准名称之前，所有参数都将转换为字符串。应谨慎使用除字符串之外的对象参数，因为它们是使用 toString() 方法进行转换的。

重要信息！如果您不清楚参数将转换成什么字符串，请不要在度量标准名称中使用该参数。

基于度量标准名称的示例

Web 站点使用名为 order 的类，其中方法名称为 process。该方法使用以下参数来表示不同类型的顺序：book 或 music。

您可以创建如下跟踪器：

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

此跟踪器将生成如下度量标准：

Order

BookOrder

- 平均响应时间 (毫秒)

MusicOrder

- 平均响应时间 (毫秒)

您还可以使用 TraceOneMethodWithParametersIfInherits 关键字。

跳过指令

AutoProbe 或 ProbeBuilder 可使用跳过指令跳过某些包、类或方法。默认情况下，AutoProbe 或 ProbeBuilder 会跳过 Java 代理以及基本 Java 类和程序包。

对象实例的计数

InstanceCounts 跟踪器组会对与其相关联的特定对象类型的实例进行计数（有关使用标准的 IdentifyClassAs 和 IdentifyInheritedAs 指令将对象类型与 InstanceCounts 跟踪器组相关联的信息，请参阅向跟踪器组中添加类 (see page 96)）。任何在代码中显式分配的实例均会计入其中。子类型也会计算在内。通过不同机制（如反序列化或克隆）创建的对象可能不会计算在内。使用此跟踪器组进行的跟踪可能增大对性能（和内存）的影响，这完全取决于进行计数的实例数量。

注意：CA Technologies 测试表明，实践中的实例数量必须非常大，才会有明显影响。

开启 InstrumentPoint 指令

有两种类型的指令使用关键字 *InstrumentPoint* 进行标识：一是跟踪异常的指令，二是在应用程序启动时（而不是在第一个探测器运行时）引起代理初始化的指令。

异常

下面的指令用于开启对发生丢弃或捕获的异常的跟踪。这些指令会降低性能，因此默认情况下不开启。要开启其中任何一个指令，请取消注释对应的行：

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

代理初始化

代理初始化检测点指令不会造成额外开销，因此无论在完全还是典型 PBD 设置中，默认情况下都会开启该指令。

```
InstrumentPoint: AgentInitialization
```

如果使用多个 ProbeBuilder 指令文件，则任何文件中开启的任何设置（如跟踪器组、跳过内容、InstrumentPoint、自定义方法跟踪器）都会生效。

组合自定义跟踪器

您可以使用影响同一度量标准的多个跟踪器，实际上就是将这些跟踪器组合在一起。这是增量器和减量器最常使用的方法。

该示例创建了一个名为 *Total Purchases* 的度量标准。使用类 *cart* 以及方法 buyBook 和 buyCD，创建以下跟踪器：

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

当有人购买商品时，此指令将增加 *Total Purchases* 度量标准。

检测和继承

适用于：1.5 版之前的 JVM

CA Introscope® 不会自动检测 1.5 版之前的 JVM 的类层次结构中较深级别的类。在所加载的被探测类的子类深度超过一级时，不会自动检测新的方法和覆盖的方法。如果类将探测到的接口明确指定为已实现，则会对这些类进行检测，即使这些类间接实现了该接口亦是如此。

例如，假定在类层次结构中，ClassB 扩展了 ClassA，ClassC 扩展了 ClassB，如下所示：

```
Interface/ClassA
    ClassB
        ClassC
```

当您检测 ClassA 时，也会检测 ClassB，因为它明确扩展了 ClassA。但是，CA Introscope® 不会检测 ClassC，因为 ClassC 未明确扩展 ClassA。要检测 ClassC，请明确识别 ClassC。

在 1.5 版之前的 Java 环境中，为了确保对子类进行检测，请按照 EJB 子类跟踪 (see page 96)说明进行操作。

如果使用 JVM 1.5，您可以配置 CA Introscope®，对被探测类 (see page 76) 的多级子类进行检测。

Java 注释

创建自定义度量标准时，CA Introscope® 允许使用 Java 1.6 注释。

- **注意：**有关 Java 注释的详细信息，请参阅 Java 开发人员网站上的文档。

使用 IdentifyAnnotatedClassAs 将类置于跟踪器组中，然后使用 TraceAllMethodsIfFlagged 指令检测该类中的方法。例如：

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

在该示例中，com.test.MyAnnotation 是注释名称。在创建自己的注释时，请在代码中使用术语。包含注释名称的类将被标识。

使用 Blame 跟踪器标记 Blame 点

CA Introscope® 的 Blame 技术应用于托管 Java 应用程序，使您可以在应用程序层（应用程序的前端和后端）查看度量标准。该功能称为 Boundary Blame，允许用户将问题归类到应用程序的前端或后端。此信息也用于在 Workstation 应用程序分类视图中标记应用程序的边缘。

有关 CA Introscope® 如何确定前端和后端，以及用于配置 URL 组以控制前端度量标准聚集方式的选项的信息，请参阅配置 Boundary Blame (see page 153)。

以下部分介绍了如何使用跟踪器在应用程序中明确标记前端和后端。

Blame 跟踪器

Introscope 提供了用于捕获前端和后端度量标准的跟踪器：FrontendMarker 和 BackendMarker。这两个跟踪器分别用于明确地标记前端和后端。

您可以使用 FrontendMarker 和 BackendMarker 检测自己的代码（例如访问后端的代码），以便 Introscope 为调查器树中的自定义组件捕获和提供度量标准。

如果未使用 FrontendMarker 跟踪器（或其子类 HttpServletTracer 和 PageInfoTracer）检测任何组件，则不会生成任何前端度量标准，也不会将任何组件标记为事务的前端。

当用 FrontendMarker 跟踪器（或其子类）检测一个事务中的多个组件时，只有第一个指定的组件将生成前端度量标准。

注意：在使用前端跟踪器时，前端跟踪器中给定的应用程序的名称必须与为应用程序分类视图跟踪器给定的名称相匹配，注意两者均区分大小写。例如，如果您将前端跟踪器命名为 **AppOne**，而应用程序分类视图跟踪器将此跟踪器引用为 **APPONE**，则 Workstation 应用程序分类视图将不会正确显示有关 AppOne 的信息。

要防止将特定类标记为前端，可以指定 PBD 参数 **is.frontend.unless**。有关 PBD 指令 *is.frontend.unless* 的信息，请参阅自定义 FrontendMarker 指令 (see page 114)。

如果没有配置 BackendMarker，Introscope 将推断一个后端，也就是说，如果没有明确标记后端，打开客户端套接字的任何组件将作为默认后端。

使用 BackendMarker 执行以下操作很有用：

- 为 Introscope 检测为后端的项目指定所需名称。
- 标记 Introscope 不检测的自定义 Java 套接字。
- 对于通过 Java 本机接口 (JNI) 调用的本机套接字，将 Java/JNI 桥接方法标识为后端。

FrontendMarker 和 BackendMarker 都是 BlamePointTracer 的实例。BlamePointTracer 提供被指责组件的诸多度量标准，如平均响应时间、每间隔计数、并发数、停顿和错误。BlamePointTracer 可以应用于粒度更高的 Blame 堆栈的中间组件。

深度嵌套的前端事务所产生的高代理 CPU 开销

Servlet 在 Introscope 的配置下显示为前端。典型事务以 Servlet 开头，它可能调用 EJB，而 EJB 将调用后端。Servlet 能以嵌套的方式调用其他 Servlet，Introscope 将这视为嵌套的前端。在大多数情况下，这并不会增加代理的 CPU 开销。

但是，具有嵌套的前端级别的深度事务（例如 40 级深）可能导致很高的 CPU 开销。例如，如果 servlet 在某个事务中重复调用自身（连续的重复调用）或调用其他多个 Servlet，您可能会发现代理的 CPU 开销有所增加。如果开销不可接受，请联系 CA Support。

自定义 FrontendMarker 指令

通过 PBD 参数 is.frontend.unless 可以将一些类不标记为前端组件。FrontendMarker（或其子类，如 HttpServletTracer）可检测这些类。应将该参数设置为一个以逗号分隔的绝对类名称列表。如果第一个组件是常规分发程序，此参数将非常有用。此分发程序可将请求转发给更加具体的组件（处理所收到的请求类型）。因此，第二个组件将是更好的前端标记。默认值为空列表。PBD 参数不是动态的。如果此参数的值发生更改，则需要重新启动已检测的应用程序服务器。

重要信息！使用逗号而不是空格来分隔类名称。使用空格将导致 SetTracerParameter 指令无效。

在参数列表中指定的、由跟踪器（此参数所应用到的跟踪器）检测的任何类：

- 不会指定为前端。
- 不会在调查器的“前端”节点中生成度量标准。

例如，要防止类 NotAFrontend 和 AnotherNonFrontend 在包 com.ABCCorp 中被视为前端（这些类使用名为 MyFrontendTracer 的 FrontendMarker 检测），您应使用以下 PDB 指令：

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

标准 PBD 中的 Blame 跟踪器

Introscope 提供的两个标准 PBD（*j2ee.pbd* 和 *sqlagent.pbd*）可实现 Boundary Blame 跟踪。

- *j2ee.pbd* 中的 *HttpServletTracer* 是 FrontendMarker 的一个实例。
- *sqlagent.pbd* 中的 *SQLBackendTracer* 是 BackendMarker 的一个实例。

Introscope 的早期版本中使用的以下 Blame 跟踪器仍然存在，但通常不用在 Introscope PBD 中：

- BlamedMethodTimer
- BlamedMethodRateTracer
- BlamedMethodTraceIncrementor
- BlamedMethodTraceDecrementor

Boundary Blame 和 Oracle 后端

在当前的 Introscope 版本中，不会根据套接字连接检测 Oracle 数据库 — 必须提供 SQL 代理，Introscope 才能自动检测 Oracle 后端。

要允许 Introscope 在缺少 SQL 代理的情况下检测 Oracle 后端，请对 oraclejdbc.pbd 进行以下修改：

在 oraclejdbc.pbd 的以下部分中：

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

如以下示例所示，注释掉相应内容：

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

注意：有关详细信息，请参阅 http://ca.com/support 上的知识库文章 *《Disabling Database Name Formatting in 7.1 (KB 1240)》*（禁用 7.1 中的数据库名称格式化 (KB 1240)）。

第 6 章： Java 代理命名

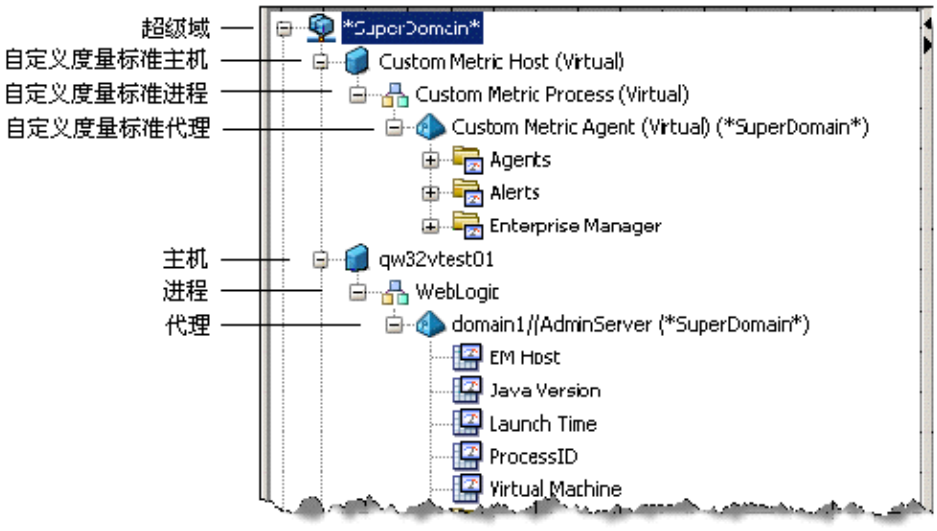
本节包含关于代理命名、相关的环境与部署注意事项以及自动命名代理的选项的信息。

此部分包含以下主题：

了解 Java 代理名称

无论您是显式指定名称、配置一个方法自动指定名称还是启动 Java 代理监控的被检测应用程序，在 Introscope 环境中运行的每个 Java 代理都会有一个名称。Java 代理名称是 Introscope Workstation 和调查器中的许多视图的核心，也是将监控逻辑与目标应用程序相关联的过程的关键。

当代理向企业管理器报告度量标准时，会在调查器树中为该代理创建节点。在 Workstation 中配置管理逻辑（例如显示板、报警和操作）时，代理名称将作为正则表达式的一个组成部分。这些正则表达式用于确定管理逻辑所适用的应用程序。下面的“调查器”树显示名为 *domain1//Adminserver* 的代理在主机 *qw32vtest01* 上的 WebLogic 进程下运行。



代理如何确定其名称

Java 代理按以下顺序来确定名称：

1. 如果 Java 代理使用第一种方法确定名称，则会接受该名称并连接到企业管理器。
2. 如果 Java 代理未使用第一种方法确定名称，则会尝试第二种方法，依此类推。
3. 如果 Java 代理未使用任何方法确定名称，则会命名为“UnnamedAgent”。

方法 1：在 Java 系统属性中指定代理名称

代理名称通过在命令行中使用 Java 系统属性来定义。使用此方法将替代其他任何代理命名方法。请参阅使用 Java 系统属性指定代理名称 (see page 120)。

方法 2：在 IntroscopeAgent.profile 中的系统属性键中指定代理名称

代理名称获取自 *IntroscopeAgent.profile* 中的一个属性中指定的 Java 系统属性。请参阅使用系统属性键指定代理名称 (see page 121)。

方法 3：从应用程序服务器自动获取代理名称

如果您使用某些版本的 WebLogic 或 WebSphere，可以使用自动代理命名功能从应用程序服务器自动获取代理名称。您可以配置延迟时间，在代理连接到企业管理器之前为其留出确定名称所必要的时间。请参阅从应用程序服务器获取代理名称 (see page 121)。

方法 4：在代理配置文件中显式指定代理名称

代理名称在 *IntroscopeAgent.profile* 的属性 *introscope.agent.agentName* 中进行定义。这是早期 Introscope 版本中命名代理的标准方法。当每个应用程序都有一个代理配置文件时可使用此选项。

方法 5：将代理名称确定为“未知代理”

如果代理使用上述任何方法均无法确定名称，则会将自己命名为“UnnamedAgent”（未知代理）。

Introscope 如何解决代理命名冲突

由主机名称、进程名称和代理名称构成的完全限定的代理名称通常在 Introscope 环境中对每个代理都是唯一的。代理名称相同的代理很可能主机名称和进程名称不同，因此它们通常有一个唯一的 *“完全限定”* 的代理名称。多个代理只有驻留在同一主机上、监控同一进程并且具有相同代理名称时，它们才会具有相同的完全限定代理名称。

如果一个代理尝试连接的企业管理器已连接了一个具有相同完全限定代理名称的代理，企业管理器会在新连接的代理的名称中附加一个唯一标识符。该标识符由一个百分号 (%) 字符和一个数字组成。此机制可以确保在连接期间能对多个使用相同完全限定名称进行连接的代理进行唯一标识。企业管理器将通过在代理名称中附加“%1”来重命名第一个要连接的重复代理。

例如，假定两个具有以下完全限定代理名称的代理：

```
hostPA|processNIM|PodAgent
```

先后连接到企业管理器。企业管理器将重命名第二个代理：

```
PodAgent%1
```

如果其他使用相同完全限定名称的代理进行连接，将依次被重命名为 *PodAgent%2*、*PodAgent%3*、*PodAgent%4* 等，其中百分号字符后面的数字是序列中的下一个数字。

当重命名的代理断开连接时，之前为其分配的后缀可以重复使用。例如，如果在 *PodAgent* 保持连接时 *PodAgent%1* 断开连接，则连接的下一个具有完全限定名称 *hostPA*|*processNIM*|*PodAgent* 的代理将重命名为 *PodAgent%1*。

重用后缀标识符使企业管理器可以在不同连接中为某个特定代理名称指定同一后缀。但是，在后续的连接中，也可以将给定代理重命名为其他名称。代理名称因连接而异会在查询历史数据时出现问题，因此最好配置一个命名策略，避免企业管理器重命名代理。

群集应用程序的代理命名注意事项

如果您运行同一应用程序的多个实例，Introscope 将尝试在代理名称中附加字符和随机数字，以解析相同的代理名称，*包括*自定义度量标准代理。但是，CA Technologies 建议您告知 Introscope 如何解析命名。

用于解析相同代理命名的选项有：

- 启用克隆代理命名（如在群集环境中启用克隆代理命名 (see page 124)中所述）,以告知 Introscope 所涉及代理是克隆代理。
- 自己定义唯一的代理名称，并为每个代理创建单独的代理配置文件（如为应用程序实例配置唯一名称 (see page 125)中所述。）
- 让 Introscope 使用自己的命名方案唯一地命名每个代理（如 Introscope 如何解决代理命名冲突 (see page 119)中所述）。

使用 Java 系统属性指定代理名称

使用 Java 系统属性指定代理名称

- 在 Java 命令行中，使用以下属性提供希望的名称：
  `-Dcom.wily.introscope.agent.agentName=`

使用系统属性键指定代理名称

这是代理查找其名称所使用的第二种方法。如果您想根据部署中现有的 Java 系统属性的值来命名代理，请使用此方法。

使用系统属性键指定代理名称

1. 打开 *IntroscopeAgent.profile*。
2. 在“Agent Name”部分下，指定将在此属性中提供代理名称的 Java 系统属性：
   `introscope.agent.agentNameSystemPropertyKey`
   注意：如果此处指定的 Java 系统属性不存在，将忽略此属性。
3. 重新启动应用程序服务器。

从应用程序服务器获取代理名称

您可以配置代理，使其自动从应用程序服务器提取应用程序服务器实例名称，并使用该信息对自己命名。这样便无需在单独的代理配置文件中配置各个代理名称。如果应用程序服务器环境发生变化，代理也可以对自己进行重命名。这样，您便能够跨多个可能混合使用应用程序服务器平台的环境部署一个代理配置文件。

支持代理命名的应用程序服务器

Introscope 与以下受支持的应用程序服务器版本一起使用时可支持自动代理命名：

- JBoss
- WebLogic 9.x
- 分布式 WebSphere 6.1.x
- WebLogic 10.0
- 分布式 WebSphere 7.0.x
- WebLogic 10.3

Introscope Workstation 中显示的应用程序服务器的名称由一个 Java J2EE API 决定。由于所有应用程序服务器实现 API 的方式不尽相同，因此有时会导致应用程序服务器的名称在 Workstation 中的显示有所不同。多个应用程序服务器的名称在 Workstation 中可能采用不同格式，甚至相同的应用程序服务器名称也可能因为版本不同而采用不同格式。

自动代理命名

自动代理命名功能启用时，代理将启动并在应用程序服务器中查找名称信息。代理会一直等待，直至获得代理名称，然后再尝试连接到企业管理器。

代理找到命名信息时，Introscope 会编辑此信息，使代理名称符合代理命名规则。

受支持的应用程序服务器上的代理名称由若干块信息组成，这些信息因应用程序服务器而异。

- 对于 JBoss，代理名称基于启动服务器时指定的配置名称。
- 对于 WebLogic，代理名称由以下部分组成：

  域（数据中心）+ 群集 + 实例 (WLS）
- 对于 WebSphere，代理名称由以下部分组成：

  单元（域）+ 过程（WAS 的实例）

获取信息时，段是由正斜杠分隔的，例如：

```
medrec/MyCluster/MedRecServer
```

段名称中的任何正斜杠都将转换为下划线。例如，如果域命名为 Petstore/West，则该名称将转换为 Petstore_West。

注意：Introscope 根据以下规则编辑用于构造代理名称的信息：

- 竖线、冒号或百分号符号等字符替换为下划线
- 以除字母之外的任何字符开头的名称之前附加字母“A”
- 空名称替换为“UnnamedAgent”，以便与“UnknownAgent”情况相区分

启用自动代理命名

1. 在 *IntroscopeAgent.profile* 中，将 *introscope.agent.agentAutoNamingEnabled* 设置为 *true*。
2. 进行以下特定于应用程序服务器的更改：
   - 对于 WebLogic，创建 Introscope 启动类。请参阅为 WebLogic 配置启动类 (see page 38)。
   - 对于 WebSphere，创建 Introscope 自定义服务。请参阅在 WebSphere 中配置自定义服务 (see page 47)。
   - 对于 JBoss，创建 XML 文件。请参阅配置 JBoss (see page 34)。

自动代理命名和重命名的代理

使用自动代理命名功能时，代理总是尝试获取特定于应用程序服务器的最新代理名称。代理会定期检查新名称。

如果应用程序服务器配置的更改导致代理名称更改，代理将自动对自己重命名。在调查器树中，代理显示为断开连接。断开连接的代理仍保留在调查器树中，并在卸载时间段过去后自动卸载，也可以手动将其卸载。

重命名的代理将重新连接到企业管理器并出现在调查器树中。代理会记录这些更改。

有关为企业管理器连接延迟和重命名检查间隔时间配置自动代理命名属性的信息，请参阅高级自动代理命名选项 (see page 123)。

高级自动代理命名选项

您可以更改一些属性来控制环境的自动代理命名。

初始企业管理器连接延迟

使用自动代理命名功能时，代理会先等待一段可配置的时间，然后再连接到企业管理器并尝试查找代理名称信息。默认延迟为 120 秒。

更改延迟值

1. 打开 *IntroscopeAgent.profile*。
2. 在“**Agent Name**”部分下，在属性 *introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds* 中配置需要的延迟。
3. 重新启动应用程序服务器。

代理重命名检查时间间隔

使用自动代理命名功能时，代理会定期检查来自应用程序服务器的命名信息是否已更改。默认时间间隔为 10 分钟。

更改此时间间隔

1. 打开 *IntroscopeAgent.profile*。
2. 在“**Agent Name**”部分下，在 *introscope.agent.agentAutoRenamingIntervalInMinutes* 属性中配置所需的时间间隔。
3. 重新启动应用程序服务器。

关闭代理的日志文件自动命名功能

默认情况下，当自动找到代理名称时（无论是通过 Java 系统属性还是应用程序服务器提供的信息），与该代理相关联的日志文件便会使用相同信息自动命名。但是，您可以关闭此自动日志命名功能，继续使用 *IntroscopeAgent.profile* 中指定的代理日志名称。

关闭代理日志文件自动命名

1. 打开 *IntroscopeAgent.profile*。
2. 将属性 *introscope.agent.disableLogFileAutoNaming* 的值设置为 *true*。
3. 保存 *IntroscopeAgent.profile*。
4. 重新启动应用程序服务器。

在群集环境中启用克隆代理命名

如果存在两个代理使用相同名称并监控相同主机和进程，并且用户未对这两个代理进行唯一命名，则名称后将附加一个数字。通过克隆代理命名功能，您可以将代理与群集应用程序中的特定应用程序实例相关联。

在以下情况下，表明您正在运行克隆代理：

- 您正在运行的代理与其他一个或多个代理共享主机、进程或 Java 代理名称，或者
- 您正在运行两个或多个使用相同代理配置文件的代理。

启用克隆代理命名

1. 停止托管应用程序和 Java 代理。
2. 打开 *IntroscopeAgent.profile* 并将以下属性设置为 true：
   `introscope.agent.clonedAgent=true`
3. 保存 *IntroscopeAgent.profile*。
4. 重新启动托管应用程序和 Java 代理。

克隆代理命名方案

在 Java 代理克隆属性开启的情况下，如果您有四个名为 *AgentX* 的 Java 代理，企业管理器会将这些代理命名为 *AgentX-1*、*AgentX-2*、*AgentX-3* 和 *AgentX-4*。如果 *AgentX-1* 断开连接后重新连接，它仍将使用 *AgentX-1* 作为自己的名称。使用这种命名方式，数据库中的 Java 代理名称永远不会超出最初克隆的 Java 代理的数量。

为应用程序实例配置唯一名称

如果您在同一台计算机上监控某一应用程序的多个实例，可以显式配置唯一的代理名称。

配置唯一的代理名称

1. 为每个应用程序创建单独的代理配置文件。
2. 在代理配置文件中对每个代理进行唯一命名。
3. 指定每个应用程序应使用哪个代理配置文件。

应用程序分类视图和代理名称

Workstation 中的应用程序分类视图在定义应用程序的前端和后端时，使用代理名称作为几个唯一标识符的一部分，并用其在 Introscope 数据库中存储有关应用程序组件的信息。如果代理名称更改，应用程序分类视图的某些方面也可能更改。例如，在代理的初始注册期间，企业管理器可能会向代理名称附加 *%<序列号>*（例如 *MyAgent%1*），为重复的代理名称指定一个唯一名称。如果应用程序分类视图的某些部分正好依赖重复的代理名称提供信息，则该分类视图的某些方面可能会发生变化。

尽管这不会影响到代理或企业管理器的正常使用，但可能会减少系统的总容量。因此，CA Technologies 建议用户注意其 Introscope 环境中使用的命名约定。用户可能希望指定特定的代理名称来避免此问题。

第 7 章： Java 代理监控和日志记录

Introscope 在监控您的应用程序的同时，还会监控 Java 代理本身的运行状况和活动。本节介绍关于监控代理运行状况以及 Java 代理的日志记录选项的信息。

此部分包含以下主题：

配置代理连接度量标准

默认情况下，Introscope 会生成有关连接到企业管理器的代理的连接状态的度量标准，您可以对度量标准进行监控。您可以监控代理连接度量标准，以确定代理与企业管理器之间的当前连接状态。

代理连接度量标准显示在企业管理器进程（自定义度量标准主机）下的 Workstation 调查器中：

自定义度量标准主机（虚拟）\ 自定义度量标准进程（虚拟）\ 自定义度量标准代理（虚拟）（*超级域*）\ 代理 \ <主机名> \ <代理进程名称> \ <代理名称> \ 连接状态

连接度量标准具有以下值：

- 0－未提供有关代理的任何数据
- 1－代理已连接
- 2－代理报告迟缓
- 3－代理已断开连接

断开连接的代理也会生成“需要关注”事件。如同其他事件一样，用户可以使用历史查询界面来查询代理断开连接事件。代理断开连接事件是在 Workstation 控制台的“概览”选项卡中用于评估应用程序运行状况的数据的一部分。

代理与企业管理器断开连接后，Introscope 会继续生成断开连接状态度量标准，直到代理超时。代理超时后，将不再生成连接度量标准，也不再向企业管理器报告连接度量标准。

请执行以下步骤：

1. 在安装有企业管理器的计算机上，打开位于 *<Introscope 主目录>/config* 目录中的 *IntroscopeEnterpriseManager.properties* 文件。
2. 修改以下属性：
   introscope.enterprisemanager.agentconnection.metrics.agentTimeoutInMinutes

   时间增量以分钟为单位。
3. 保存 *IntroscopeEnterpriseManager.properties*

注意：有关企业管理器属性的信息，请参阅《*CA APM 配置和管理指南*》。

套接字度量标准

默认情况下，代理中启用了套接字和安全套接字层 (SSL) 度量标准收集。

注意：使用 *Agent.NoRedef.jar* 的 JVM 不报告套接字度量标准。有关详细信息，请参阅用于 WebSphere 6.1 的 AutoProbe (see page 43)。

限制套接字和 SSL 度量标准收集

套接字和 SSL 度量标准收集功能默认情况下是启用的，但是您可以限制某些度量标准收集，以便定位到更为相关的信息或适当缩减开销。

限制套接字和 SSL 度量标准收集

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 在“Agent I/O Socket Metrics”部分中，编辑属性值，以包含需要度量标准的主机或端口的列表：

   如果参数值中包含无效的主机或端口，代理日志中将写入一条警告消息，并且该无效值将被忽略。如果因此该列表不包含任何条目，将不应用任何限制。

   - *introscope.agent.io.socket.client.hosts*
     主机的逗号分隔列表；将只生成指定主机的“客户端”套接字度量标准。可以通过名称或 IP 地址的文本表示形式（以 IPv4 或 IPv6 形式）来指定主机。

     注意：重复的主机名将被丢弃。如果多个主机名映射到一个单个 IP，则仅会保留其中一个名称。但是，该属性将匹配具有任何同义名称集的客户端连接。

- *introscope.agent.io.socket.client.ports*
  逗号分隔的端口号列表；将只生成指定端口的“客户端”套接字度量标准。

  注意：重复的端口将被丢弃。

- *introscope.agent.io.socket.server.ports*
  逗号分隔的端口号列表；将只生成指定端口的“服务器”套接字度量标准。

上述属性是动态属性，您无需重新启动应用程序便可使更改生效。

3. 保存 *IntroscopeAgent.profile*。

微调套接字和 SSL 度量标准收集

在可以限制套接字和 SSL 度量标准收集的同时（如限制套接字和 SSL 度量标准收集 (see page 128)中所述），您还可以通过开启和关闭特定的跟踪器组来进一步细化度量标准的收集。这有助于在定位所需信息的同时降低开销成本。

微调套接字和 SSL 度量标准收集

1. 打开位于 *<Agent_Home>*/wily/core/config 目录中的 *java2.pbd* 文件。
2. 在 *java2.pbd* 的“I/O Socket Tracer Group”或“Network Tracer Group”部分中，确定您要开启或关闭的跟踪器组，并对其进行注释或取消注释。例如，要隐藏输入带宽的度量标准，请注释掉以下内容：

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

   注意：不得注释掉名称以 *MappingTracer* 结尾的跟踪器。
3. 保存 *java2.pbd* 文件。

应用程序分类视图中的 SSL、NIO 和套接字跟踪

Workstation 调查器中的应用程序分类视图可以显示已检测的客户端套接字连接。代理会记录有关事务中使用的外部系统的详细信息，并将此信息发送给企业管理器。然后，此信息将以图形方式显示在应用程序分类视图中。

位于 *<代理主目录>*/wily/core/config 目录的 appmap.pbd 中的跟踪器扩展了现有的 SSL、NIO 和套接字检测。通过这些跟踪器，代理可以向应用程序分类视图发送更多组件信息。默认情况下，这些跟踪器处于启用状态。可以通过在 appmap.pbd 的“跟踪套接字”和“跟踪 NIO 套接字”部分中注释掉特定跟踪器来加以禁用。

在应用程序分类视图中更改组件名称

应用程序分类视图中显示的组件名称包括目标主机和端口的标识，显示方式与套接字客户端度量标准的名称相同。组件名称中的主机标识可以配置为主机名或主机 IP 地址。默认为主机名。您可以更改显示的组件名称。

请执行以下步骤：

1. 打开位于 *<Agent_Home>*/wily/core/config 目录中的 appmap.pbd。
2. 在“跟踪套接字”和“跟踪 NIO 套接字”部分中，选择您要修改的跟踪器。
3. 将相关跟踪器中的 *{hostname}* 更改为 *{hostip}*。

   例如，原始跟踪器使用默认的 *{hostname}*：

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

   要显示主机 IP，请改用 *{hostip}*：

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

4. 保存 appmap.pbd 文件。

关闭套接字和 SSL 度量标准收集

如果不需要收集套接字和 SSL 度量标准，则可将其完全关闭。

关闭套接字和 SSL 度量标准收集

1. 打开 *toggles-full.pbd* 或 *toggles-typical.pbd* 文件（打开您的部署中正在使用的文件）。
2. 注释掉（在行首放置井号 #）*SocketTracing*：

   ```
   #TurnOn: SocketTracing
   ```

3. 保存已修改的文件。

有关在 *toggles-full.pbd* 和 *toggles-typical.pbd* 文件中开启和关闭跟踪器组的详细信息，请参阅默认跟踪器组和切换文件 (see page 86)以及打开或关闭跟踪器组 (see page 95)。

向后兼容性

适用于：9.0 版本之前的跟踪器

Java 代理套接字跟踪器比 9.0 版本之前的跟踪器更便于配置。但是，您可以还原回之前的跟踪器（并禁用套接字跟踪功能和配置选项）。

如果您使用的是 9.0 版本之前的跟踪器，您可以配置 Java 代理，以便执行以下操作：

- 收集套接字度量标准和之前的跟踪器 (see page 131)。
- 收集输入和输出带宽度量标准 (p. 131)。

收集套接字度量标准

适用于：9.0 版本之前的跟踪器

要使用跟踪器收集套接字度量标准，请配置 Java 代理。

请执行以下步骤：

1. 打开 toggles-full.pbd 或 toggles-typical.pbd 文件（打开部署中正在使用的文件）。
2. 注释掉（在行首放置井号 #）SocketTracing：
   ```
   #TurnOn: SocketTracing
   ```
3. 取消注释 ManagedSocketTracing：
   ```
   TurnOn: ManagedSocketTracing
   ```
4. 保存该文件。

收集输入和输出带宽度量标准

适用于：9.0 版本之前的套接字跟踪器

如果使用的是套接字跟踪器，并且需要输入和输出带宽度量标准，请配置 Java 代理。

请执行以下步骤：

1. 打开 IntroscopeAgent.profile。
2. 找到“Agent Socket Rate Metrics”部分，并将以下属性更改为 true：
   ```
   introscope.agent.sockets.reportRateMetrics=true
   ```
   注意：仅当启用 ManagedSocketTracing 并禁用 SocketTracing 时才起作用。
3. 保存 IntroscopeAgent.profile。

配置日志记录选项

在应用程序服务器上安装 Java 代理时，在服务器启动后，将在以下位置创建一个日志目录：*<Agent_Home>*/wily/logs。应用程序服务器进程必须对 *<Agent_Home>* 目录拥有完全的读取/写入/执行权限。为此，请将 Java 代理安装在运行应用程序服务器进程的用户的同一操作系统上。或者以不同用户的身份安装 Java 代理，然后使用 *chmod* 命令授予必要的权限。

Java 代理可以选择以详细模式运行。详细模式会记录有关操作以及代理与环境交互的更高级别的详细信息。此信息对于解决环境或代理功能的问题很有用。

Introscope 使用这些功能的 Log4J 功能。如果您要使用其他 Log4J 功能，请参考 Log4J 文档。

以详细模式运行代理

以详细模式运行代理会向代理日志中记录更高级别的信息。

以详细模式运行代理

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 修改此属性，将现有的 *INFO* 替换为 *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel*：

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverityLevel, console, logfile
   ```

3. 保存 *IntroscopeAgent.profile*。

   注意：对此属性所做的更改应在一分钟之内生效，不需要重新启动托管应用程序。

将代理输出重定向到文件

在详细模式中用于控制代理日志记录的属性还控制着代理日志的输出位置和此日志文件的位置（有关详细信息，请参阅以详细模式运行代理 (see page 132)）。

将代理输出重定向到文件

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。

2. 查找属性：*log4j.logger.IntroscopeAgent*

   该属性的选项为：

   - *console*：将日志文件中的信息发送给控制台
   - *logfile*：将日志文件中的信息发送给日志文件。如果选择此选项，则将使用 *log4j.appender.logfile.File* 属性配置日志文件的位置。请参阅更改代理日志文件的名称或位置 (see page 133)。

   例如，如果您希望代理以详细模式只向日志文件报告，应将此属性设置为：

   ```
   log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverity
   Level,logfile
   ```

   如果您希望代理同时向日志文件和控制台报告，则应在属性中同时包括 *logfile* 和 *console*。

   注意：默认情况下，会将代理日志 *IntroscopeAgent.log* 写入 *<Agent_Home>*/wily/logs 目录。如果您配置了代理自动命名选项，代理日志文件也会自动命名，如代理日志文件和自动代理命名 (see page 134)中所述。

3. 保存 *IntroscopeAgent.profile*。

更改代理日志文件的名称或位置

您也可以通过修改属性来更改日志文件的位置和名称。

更改日志文件的名称或位置

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 找到 *log4j.appender.logfile.File* 属性。

   如果 *log4j.logger.IntroscopeAgent* 属性中指定的是 *logfile*，则使用 *log4j.appender.logfile.File* 属性配置日志文件的位置。有关详细信息，请参阅将代理输出重定向到文件 (see page 132)中的第 2 步。

   注意：系统属性（Java 命令行 -D 选项）可作为文件名的一部分。例如，如果 Java 命令以 *-Dmy.property=Server1* 开头，则 *log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* 将扩展为 *log4j.appender.logfile.File=../../logs/Introscope-Server1.log*。
3. 使用新位置和文件的完全限定路径设置日志文件的位置和名称。例如：

   ```
   log4j.appender.logfile.File=C:/Logs/AgentLog1.log
   ```
4. 保存 *IntroscopeAgent.profile*。

代理日志文件和自动代理命名

如果您使用自动代理命名功能，则默认将使用命名代理所用的相同信息来命令与代理相关的日志文件。

自动代理命名功能通过以下方式影响日志文件：

- 如果日志文件的原名不以 *.log* 结束，则添加句点和 *log*。
- 所有不是字母或数字的字符都将替换为下划线
- 如果使用了高级 Log4J 功能，则代理日志文件自动命名功能可能不起作用。

下面的示例说明了如何命名代理日志文件。这些示例使用 *DOM1//ACME42* 作为代理名称，其中 *DOM1* 是 WebLogic 域，*ACME42* 是代理的实例。

创建代理日志文件时（默认名为 *AutoProbe.log*），如果代理名称尚不可用，则会在文件名中加入一个时间戳：

```
AutoProbe.20040416-175024.log
```

一旦代理名称可用，日志文件将使用代理的自动名称重命名。

```
AutoProbe.DOM1_ACME42.log
```

您可以禁用自动日志命名功能—有关详细信息，请参阅高级自动代理命名选项 (see page 123)。

按日期或大小向上滚动日志

您可以根据大小或日期向上滚动日志，保留指定天数的信息并清除其余信息。

向上滚动日志文件

1. 打开 *IntroscopeAgent.profile*，并找到“Logging Configuration”部分。
2. 修改以下属性：

```
log4j.logger.IntroscopeAgent
log4j.appender.logfile.File
log4j.appender.console.layout
log4j.appender.console.layout.ConversionPattern
log4j.appender.logfile
log4j.appender.logfile.MaxFileSize
log4j.appender.logfile.MaxBackupIndex
```

注意：您必须重新启动托管应用程序，对该属性所做的更改才能生效。

3. 保存 *IntroscopeAgent.profile*。

   例如，下面的配置将最多保留三个备份/滚动的日志，每个日志最大为 2 kb：

```
log4j.logger.IntroscopeAgent=VERBOSE#com.wily.util.feedback.Log4JSeverity
Level, console, logfile
log4j.appender.logfile.File=logs/IntroscopeAgent.log
log4j.appender.console.layout=com.wily.org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z}
[%-3p] [%c] %m%n
log4j.appender.logfile=com.wily.introscope.agent.AutoNamingRollingFileApp
ender
log4j.appender.logfile.MaxFileSize=2KB
log4j.appender.logfile.MaxBackupIndex=3
```

管理 ProbeBuilder 日志

ProbeBuilder 会记录自己发现的所有类、检测的所有类以及没有为在检测过程中添加的探测器和使用的 PBD 添加检测的所有类。此外，它还会记录由于跳过而未检测的类。

命令行 ProbeBuilder 和 ProbeBuilder 向导日志的名称与位置

命令行 ProbeBuilder 和 ProbeBuilder 向导日志文件的位置是由您使用 ProbeBuilder 向导或命令行 ProbeBuilder 指定 Java 类所在的位置决定的。对于目录，日志文件位于目标目录内。对于文件，日志文件位于目标文件旁边。

ProbeBuilder 日志文件命名为：

```
<original-directory-or-original-file>.probebuilder.log
```

<original-directory> 或 *<original-file>* 是您使用 ProbeBuilder 向导或命令行 ProbeBuilder 指定的 Java 类的位置。

只有最新的日志得以保留，所有之前的日志文件将被覆盖。

AutoProbe 日志名称和位置

AutoProbe 将始终尝试记录它所做的更改。默认情况下，AutoProbe 日志文件命名为 *AutoProbe.log*。

更改 AutoProbe 日志的名称或位置

1. 在文本编辑器中打开 *IntroscopeAgent.profile*。
2. 找到 *introscope.autoprobe.logfile* 属性，使用完全限定的文件路径修改日志名称和位置。非绝对名称将相对于 *IntroscopeAgent.profile* 文件的位置进行解析。

   注意：在加载某个类路径上的资源中的代理配置文件时，AutoProbe 无法写入 AutoProbe 日志文件，因为 *IntroscopeAgent.profile* 文件位于资源内部。

   您必须重新启动托管应用程序，对此属性所做的更改才能生效。
3. 保存 *IntroscopeAgent.profile*。

第 8 章： 配置 LeakHunter 和 ErrorDetector

本节介绍如何启用和配置 LeakHunter 和 ErrorDetector。

此部分包含以下主题：

LeakHunter

Introscope LeakHunter 是一个附加组件，旨在通过观测随着时间推移而逐渐增大的集合实例来帮助查找潜在的内存泄漏源－也就是说，观测存储在集合中的对象数是否随着时间的推移而增长。

短期（数分钟或数小时）运行的程序中发生内存泄漏可能不会引起大问题。但是，对于每天运行 24 小时的应用程序（例如网站）来说，很小的内存泄漏也可能很快升级为大问题。

Introscope LeakHunter 旨在通过打开、发现集合类，然后在收集信息之后关闭来跟踪值得注意的内存泄漏的相关信息。这种使用 LeakHunter 的方法只会产生很小的临时开销。

LeakHunter 的工作原理

启用 LeakHunter 的同时，还应定义 LeakHunter 寻找新的潜在泄漏的超时时间。如果使用 AutoProbe，只需重新启动托管应用程序。如果使用 ProbeBuilder 向导或命令行 ProbeBuilder，则必须使用 leakhunter.pbd（除了先前使用的任何 PBD 文件之外）重新检测应用程序。

如果 LeakHunter 发现了随着时间推移而逐渐增长的集合，会执行以下操作：

- 通过唯一的 ID 识别集合
- 将有关集合的信息作为度量标准数据报告给企业管理器
- 将有关集合的信息报告到代理计算机上的日志文件
- 继续跟踪和报告该集合的数据

如果 LeakHunter 发现集合不再泄漏，它会将此事报告给企业管理器和日志文件，但会继续跟踪和报告该集合的数据。

LeakHunter 会继续寻找潜在泄漏并监控已识别的潜在泄漏，直到超时期满。超时期满之后，LeakHunter 会停止在新分配的集合中寻找潜在泄漏，并仅继续检查已识别为潜在泄漏的集合。这大大地减少了 LeakHunter 的开销，可以腾出更多开销来监控其他潜在泄漏。LeakHunter 会继续监控已识别的潜在泄漏，直到关闭托管应用程序。

要查找内存泄漏源，可以浏览 Introscope 调查器中的度量标准数据或查看日志文件。

LeakHunter 在 Java 中跟踪的内容

Introscope LeakHunter 跟踪 Java 实现中的以下收集：

- java.util.Collection 实现：
  - java.util.ArrayList
  - java.util.LinkedList
  - java.util.TreeSet
  - java.util.HashSet
  - java.util.LinkedHashSet
  - java.util.Vector
  - java.util.Stack

- java.util.Map 的实现：
  - java.util.Map
  - java.util.SortedMap
  - java.util.HashMap
  - java.util.TreeMap
  - java.util.IdentityHashMap
  - java.util.Hashtable
  - java.util.Properties
  - java.util.LinkedHashMap

LeakHunter 不跟踪哪些内容

Introscope LeakHunter 不跟踪：

- 不是由集合引起的泄漏。
- 自定义集合实施或引用数量逐渐增长的其他数据结构。
- 未检测的泄漏集合。

除上述内容之外，LeakHunter 也不跟踪 Java 实现中的以下内容：

- 为 LeakHunter 在 Java 中跟踪的内容 (see page 138)中所述的收集创建的任何子类。但是，可以更新 ProbeBuilder 指令 (PBD) 文件以获取此信息。有关详细信息，请参阅 *leakhunter.pbd* 文件。

注意：如果正在使用应用程序服务器 AutoProbe，LeakHunter 不会自动跟踪由应用程序服务器分配的收集。要跟踪这些收集，必须静态检测应用程序服务器，或使用 JVM AutoProbe。

系统和版本要求

Introscope LeakHunter 与 Java 代理具有相同的系统要求。

默认情况下，LeakHunter 在安装之后不会启用。您必须启用 LeakHunter 以使用其功能。

启用和禁用 LeakHunter

LeakHunter 作为代理扩展运行，因此不需要更新任何类路径。默认情况下，LeakHunter 在安装之后不会启用。您必须启用 LeakHunter 以使用其功能。

启用 LeakHunter

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 在 *“LeakHunter 配置”* 标题下，找到 introscope.agent.leakhunter.enable 属性，并输入 *true* 值。
3. 保存代理配置文件。
4. 打开 **.typical.pbl* 或 **.full.pbl*（打开 *IntroscopeAgent.profile* 属性 *introscope.autoprobe.directivesFile* 中列出的文件），并取消注释 *leakhunter.pbd*。

   注意： 使用 ProbeBuilder 向导时，请将 *leakhunter.pbd* 文件复制到 *<Agent_Home>\wily\core\config\hotdeploy* 目录。
5. 重新启动应用程序。

重要信息！ 默认情况下，可从 *<Agent_Home>*\wily\core\ext 目录中找到和引用代理扩展，如 LeakHunter。但是，您可以在代理配置文件中更改代理扩展目录的位置。如果您更改了 *\ext* 目录的位置，请务必同时移动 *\ext* 目录的内容。

禁用 LeakHunter

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 在 *“LeakHunter 配置”* 标题下，找到 introscope.agent.leakhunter.enable 属性，并输入 *false* 值。
3. 保存代理配置文件。
4. 重新启动应用程序。

配置 LeakHunter 属性

LeakHunter 配置属性位于代理配置文件 *IntroscopeAgent.profile* 中。

配置 LeakHunter

1. 打开代理配置文件 *IntroscopeAgent.profile*。

2. 根据需要配置以下 LeakHunter 属性：

introscope.agent.leakhunter.logfile.location

指定 LeakHunter.log 文件的位置。文件名相对于 *<IntroscopeAgent.profile>* 目录。如果该属性已注释掉或保留空白，则不会写入任何日志文件。

默认值为 *logs/LeakHunter.log*。

introscope.agent.leakhunter.logfile.append

指定在应用程序重新启动时替换日志文件（值为 false）还是附加现有的日志文件（值为 true）。

默认值为 false。

introscope.agent.leakhunter.leakSensitivity

指定检测内存泄漏的敏感级别。较高的泄漏敏感设置将导致报告较多的潜在泄漏，较低的敏感设置则导致报告较少的潜在泄漏。

属性值必须是 1-10 的整数。

默认敏感级别为 5。

introscope.agent.leakhunter.timeoutInMinutes

指定 LeakHunter 查找潜在的新泄漏过程的时间（分钟）。该属性值必须是非负整数。值为零表示没有超时。

默认值为 120 分钟。

introscope.agent.leakhunter.collectAllocationStackTraces

指定是否收集分配堆栈跟踪信息。启用该选项可能会导致系统 CPU 使用率和内存较高。对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

默认值为 false。

introscope.agent.leakhunter.ignore.*n*

指定希望 LeakHunter 忽略的特定收集。

对于常规集合，请使用包含常规类型限定的语法，例如：system.Collections.Generic.List`1

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

这些属性（其中 *n*=0-4）的默认值为：

```
introscope.agent.leakhunter.ignore.0=org.apache.taglibs.standard.lang
.jstl.*
introscope.agent.leakhunter.ignore.1=com.bea.medrec.entities.RecordEJ
B_xwcp6o__WebLogic_CMP_RDBMS
introscope.agent.leakhunter.ignore.2=net.sf.hibernate.collection.*
introscope.agent.leakhunter.ignore.3=org.jnp.interfaces.FastNamingPro
perties
introscope.agent.leakhunter.ignore.4=java.util.SubList
```

3. 保存对代理配置文件所做的更改。

重要信息！ *IntroscopeAgent.profile* 包含用于控制哪些包被 LeakHunter 忽略的属性。这些属性在默认情况下处于启用状态。如果注释掉这些属性，代理日志中将报告异常。

忽略会导致性能下降的集合

如果收集的 *size()* 方法的执行时间与收集中的对象数成正比，则此类收集会降低性能。换言之，如果收集的 *size()* 方法需要的执行时间越来越长（例如，对于要获取列表大小的、实现不当的 LinkedList，我们需要遍历列表的每个元素并进行计数），这将对应用程序性能产生负面影响。

如配置 LeakHunter 属性中所述，应使用 IntroscopeAgent.profile 中的忽略属性来忽略这种集合。

运行 LeakHunter

LeakHunter 作为代理扩展运行。

运行 LeakHunter

- 在启用 LeakHunter 之后使用 JVM AutoProbe，重新启动应用程序。
- 使用命令行 ProbeBuilder：使用 *leakhunter.pbd*（以及先前使用的任何 .pbd 文件）重新检测托管应用程序。重新启动托管应用程序。
- 使用 ProbeBuilder 向导：运行 ProbeBuilder 向导并从自定义 pbd 列表中选择 *leakhunter.pbd*。实现新 .jar 文件以供使用后，重新启动托管应用程序。

使用集合 ID 标识潜在泄漏

LeakHunter 使用唯一的集合 ID 来标识每个潜在泄漏，该 ID 可用于将调查器树中的度量标准数据与日志文件中的数据关联，并在各应用程序之间提供稳定的名称。

唯一的集合 ID 具有特定的语法，语法基于以下类型之一：

<method>-<4 digit hash code>#<unique number>

<field>-<4 digit hash code>#<unique number>

- *<method>*：分配收集的方法名称
- *<field>*：收集所分配到的字段名称
- *<4 digit hash code>*：包含方法或字段名称的类全名的哈希代码
- *#<unique number>*：附加到具有相同方法和哈希代码的潜在泄漏的一个数字，用于在代理运行过程中确保使用唯一的收集 ID。

潜在泄漏的集合 ID 在应用程序的运行过程中是稳定的。

下面是一些集合 ID 示例：

```
theLookupTable-6314#1
getLoginID-1234#1
getLoginID-1234#2
getLoginID-1234#3
verifyCart-5678#1
verifyCart-0012#1
```

LeakHunter 日志文件

LeakHunter 日志文件包含有关由托管应用程序中的 Introscope LeakHunter 识别的潜在泄漏的信息。每个条目均包含有关潜在泄漏的信息。LeakHunter 日志文件包括有关四个方案的条目：

- 当首次识别到潜在泄漏时－请参阅首次识别潜在泄漏日志条目 (see page 144)
- 当已识别的泄漏已停止泄漏时－请参阅已识别的潜在泄漏停止泄漏日志条目 (see page 145)
- 当之前识别的泄漏再次开始泄漏时－请参阅已识别的潜在泄漏再次泄漏日志条目
- 当发生 LeakHunter 超时时－请参阅 LeakHunter 超时日志条目

首次识别潜在泄漏日志条目

此类型的 LeakHunter 日志条目包含有关某个潜在泄漏首次被识别时的信息：

- 当前时间戳（写入日志的时间）
- 集合 ID
- 集合的类
- 集合的分配方法
- 收集的分配时间
- 集合的分配堆栈跟踪
- 收集所分配到的字段名称
- 集合的当前大小

注意：LeakHunter 日志文件中记录的已泄漏收集的当前大小不会动态更新。日志文件标识了首次识别泄漏时的泄漏大小。要查看有关已泄露连接大小的最新信息，请单击 Introscope Workstation 中的“泄漏”选项卡。

示例：检测潜在泄漏时的日志文件条目

以下示例说明了首次识别潜在泄漏时 Java 日志文件中的条目：

```
09-5-2 09:55:06 PDT
Potential leak identified
Assigned ID: testInst-2604#1
Collection Class: java.util.Vector
Allocation Method: sonOfLH_test.testInst()
Allocation Timestamp: 09-5-2 09:54:21 PDT
Allocation Stack Trace:
未知
Field Name(s):
sonOfLH_test.v3
sonOfLH_test.v4
sonOfLH_test.v5
Current Size: 44
```

已识别的潜在泄漏停止泄漏日志条目

此类型的 LeakHunter 日志条目包含有关已停止泄漏的潜在泄漏的信息：

- 当前时间戳（写入日志的时间）
- 集合 ID
- 集合的类
- 集合的当前大小

示例：潜在泄漏停止泄漏时的日志文件条目

以下示例说明了潜在泄漏显示已停止泄漏时 Java 日志文件中的条目：

```
4/27/10 1:18:12 PM PDT
Potential leak no longer appears to be leaking
Assigned ID: createNewInstance-2815#3
Collection Class: java.util.HashMap
Current Size: 70
```

已识别的潜在泄漏又开始泄漏日志条目

此类型的条目包含有关再次开始泄漏的潜在泄漏的信息：

- 当前时间戳（写入日志的时间）
- 集合 ID
- 集合的类
- 集合的当前大小

示例：潜在泄漏继续泄漏时的日志文件条目

以下示例说明潜在泄漏显示又开始泄漏时 Java 日志文件中的条目：

```
4/27/10 1:21:42 PM PDT
Potential leak appears to be leaking again
Assigned ID: createNewInstance-2815#3
Collection Class: java.util.HashMap
Current Size: 79
```

LeakHunter 超时日志条目

这种类型的 LeakHunter 日志条目包含将继续跟踪的潜在泄漏数目。

示例：发生超时后的日志文件条目

```
LeakHunter timeout occurred at 4/27/10 1:32:12 PM PDT
LeakHunter will only continue to track the 3 potential leaks
```

使用 LeakHunter

有关如何使用 LeakHunter 的详细信息，请参阅*《CA APM Workstation 用户指南》*。

ErrorDetector

使用 Introscope ErrorDector，应用程序支持人员可以检测和诊断严重错误的原因，这些错误会防止用户完成 Web 事务。此类应用程序可用性问题通常会导致向用户发出错误消息，如 "404 Not Found" 等，但是错误消息缺少 IT 人员确定问题根本原因所需的具体信息。使用 Introscope ErrorDetector，当应用程序运行中发生这些严重错误时，您可以监控错误、确定错误的频率和本质，并将有关根本原因的具体信息发送给开发者。

ErrorDetector 是唯一可通过启用严重应用程序错误的根本原因分析，来帮助确保卓越用户体验并提高事务完整性的应用程序管理解决方案。

通过 Introscope ErrorDetector，IT 团队可以：

- 确定异常事务的频率
- 确定所记录的异常是否影响用户
- 查看错误在事务路径中发生的精确位置
- 获取再现、诊断和消除严重错误所需的信息

Introscope ErrorDetector 与 Introscope 集成，使您可以在 Introscope Workstation 中监控错误。当确实发生应用程序错误时，您也可以使用实时错误查看器来检查每个错误的详细信息。

错误类型

CA Technologies 已定义一组条件，基于 J2EE 规范中包含的信息来描述“严重”错误。ErrorDetector 将错误和异常均视为错误。最常见的错误类型是抛出的异常。

常见错误包括：

- HTTP 错误（404、500 等）

  注意：有时，HTTP 404 错误源自 Web 服务器，而不是应用程序服务器。如果发生该情况，ErrorDetector 无法通过代理检测 Web 服务器错误。

- SQL 语句错误
- 网络连接错误（超时错误）
- 后端错误（例如，无法通过 JMS 发送消息、无法将消息写入消息队列）

CA Technologies 所认为的重要错误可能与您认为的重要错误不同。如果您认为 ErrorDetector 跟踪的某些错误不重要，可以选择忽略。如果想要跟踪其他错误，可以使用错误跟踪器来创建新指令来跟踪这些错误。

ErrorDetector 的工作原理

Introscope 在安装代理时附带了一个名为 *errors.pbd* 的 ProbeBuilder 指令 (PBD) 文件。此 PBD 中的跟踪器可捕获严重错误。

安装代理时会自动安装 ErrorDetector。安装 ErrorDetector 之后，将 Introscope 配置为使用 *errors.pbd*，并启用 ErrorDetector 功能。如果正在使用 ProbeBuilder 向导或命令行 ProbeBuilder，则必须使用 *errors.pbd*（除了先前使用的任何 PBD 文件之外）重新检测应用程序。

代理根据 *errors.pbd* 文件中的定义收集错误信息。

在 Workstation 中，您可以查看：

- 调查器中的错误度量标准数据
- 实时错误查看器中的实时错误
- 错误快照中的错误详细信息，其中显示了有关错误如何发生的组件级别信息

ErrorDetector 集成了事务跟踪器，使您能够确切地了解事务路径上下文中为什么会发生严重错误以及这些错误是如何发生的。另外，所有错误和事务都一直存储在事务事件数据库中，使您能够通过分析历史数据发现其中的趋势。

Introscope 将事务定义为服务的调用和处理。在 Web 应用程序上下文中，事务是发送自 Web 浏览器的 URL 的调用和处理。在 Web 服务上下文中，事务是 SOAP 消息的调用和处理。

在 Java 代理中启用 ErrorDetector

应在 *IntroscopeAgent.profile* 中将属性 *introscope.agent.errorsnapshots.enable* 设置为 true，从而使 Java 代理能够捕获错误数据。默认情况下，会启用 Java 代理来捕获错误数据。

启用/禁用 Java 代理中的 ErrorDetector 数据捕获

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 确认 *introscope.agent.errorsnapshots.enable* 属性设置为 *true*。

   注意：要禁用 ErrorDetector，请将此属性设置为 *false*。
3. 如果使用 ProbeBuilder 向导，请将 *errors.pbd* 文件复制到 *<EM_Home>*/config/custompbd 目录，并重新检测应用程序。
4. 保存代理配置文件。

配置 ErrorDetector 选项

您可以配置 ErrorDetector 来限制代理发送到企业管理器的最大错误数，或指定要忽略的错误。

启用 ErrorDetector 可在捕获错误数据的同时不带来过多开销。即用限制设置为每 15 秒 10 个错误。如果您想在此时间段内捕获更多错误，可以增大该限制，但需要准备好承担更多开销。

更改 ErrorDetector 限制（可选）

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 为 *introscope.agent.errorsnapshots.throttle* 属性输入一个新值。
3. 保存代理配置文件。

   注意：对该属性所做的更改可立即生效，不需要重新启动托管应用程序。

您可以将代理配置为忽略不想跟踪的错误。您指定用来标记错误的信息可以是精确的错误消息，也可以是消息的任何部分加上星号通配符。

忽略特定错误（可选）

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 对于 *introscope.agent.errorsnapshots.ignore* 属性，将值定义为任何您认为会识别该错误类型的信息。

   例如，以下忽略属性会忽略在它内部找到的任何带有短语 "IOException" 的错误：

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   ```

3. 要忽略其他错误，请按顺序添加其他忽略属性。例如，要忽略两种错误类型，属性应如下所示：

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code *500*
   ```

4. 保存对代理配置文件所做的更改。

   注意：对该属性所做的更改可立即生效，不需要重新启动托管应用程序。

高级错误数据捕获

虽然 ErrorDetector 在默认情况下可捕获许多常见的错误类型，但是 CA Technologies 还提供了一些选项，可自定义错误检测机制以符合您的需求。

您可以使用以下与错误相关的跟踪器创建 ProbeBuilder 指令 (PBD) 来捕获错误。

- **ExceptionErrorReporter** 可报告标准异常。
- **ThisErrorReporter** 将当前对象报告为错误。
- **HTTPErrorCodeReporter** 可捕获 HTTP 错误代码和关联的错误消息。
- **MethodCalledErrorReporter** 报告是否调用了特定的方法。

您应将使用这些跟踪器创建的任何新指令放置在 *<代理主目录>*/wily 目录下的 *errors.pbd* 文件中。

定义新错误类型

可使用 PBD 为 ErrorDetector 定义错误。还有几个特殊的跟踪器，用于确认错误是否存在并捕获（或构建）错误消息。通过将这些跟踪器放在正确的位置，可以让 ErrorDetector 了解应用程序或基础架构中的新错误。

ExceptionErrorReporter

ExceptionErrorReporter 跟踪器可用于检查从已检测的方法抛出的异常。如果抛出了异常，此跟踪器会将异常当作错误，并从异常中获取错误消息。这是对错误的最常见定义。

为了捕获错误消息，ExceptionErrorReporter 跟踪器必须与 "...WithParameters" 指令配合使用。例如：

TraceOneMethodWithParametersOfClass: com.bank.CustomerAccount getBalance ExceptionErrorReporter "CustomerAccount:Errors Per Interval"

此指令指定从 *CustomerAccount* 上的 *getBalance()* 方法抛出的任何异常均视为错误。

请注意，您必须使用“...WithParameters”指令增加“每个时间间隔的错误”度量标准，但对于任一方法，您只需指定“...WithParameters”指令一次，该方法上的参数即可对所有追踪器可用。例如，您可以指定：

TraceOneMethodWithParametersOfClass: com.myClass myMethod BlamePointTracer

此指令使 *com.myClass myMethod* 方法的参数可用于其他追踪器，包括 *ExceptionErrorReporter* 追踪器。

MethodCalledErrorReporter

MethodCalledErrorReporter 跟踪器作用于方法，调用方法即表示已发生错误。例如：

TraceOneMethodOfClass: com.bank.CheckingAccount cancelCheck MethodCalledErrorReporter "CustomerAccount:Canceled Checks Per Interval"

此指令指定只要调用了 cancelCheck() 方法（不管出于任何原因），就意味着发生了错误。错误消息仅指出了所调用的类和方法。

如果您不知道哪些方法会抛出异常或错误，请尝试使用 ThisErrorReporter 跟踪器。

ThisErrorReporter

ThisErrorReporter 跟踪器与 *MethodCalledErrorReporter* 类似，但是它通过在已检测的对象上调用 *toString()* 来构建错误消息。该跟踪器对于指出异常类的构建器非常有用。例如：

```
TraceOneMethodWithParametersOfClass: ezfids.util.exception.EasyFidsException
[set the init variable for your book] ThisErrorReporter
"Exceptions|{packageandclassname}:Errors Per Interval"
```

注意：为了捕获错误消息，*ThisErrorReporter* 跟踪器必须与“...WithParameters”指令配合使用。

指令指定只要调用了 *InvalidPINException* 的构建器（“init”或“.ctor”），就构成错误。错误消息是通过对 *InvalidPINException* 调用 *toString()* 来确定的，InvalidPINException 通常会返回应用程序开发者指定的错误消息。

如果您有一个基于自己的异常类型的自定义错误管理系统，此跟踪器将非常有用。

注意：Introscope 无法检测 *java.** 软件包中的任何代码，因此将此跟踪器置于 *java.lang.Exception* 或 *java.sql.SQLException* 将不起作用。

HTTPErrorCodeReporter

HTTPErrorCodeTracer 跟踪器报告 Servlet 和 JSP 中的错误代码。这是一个按时间间隔的计数器，用于计算以下事件的数目：

- HTTP 响应代码 400 或更高代码。
- *javax.servlet.http.HttpServletResponse* 的 *sendError* 或 *setStatus* 子类调用，针对 Java 环境中 400 或更高代码。

有关使用情况示例，请参阅 *errors.pbd*。

谨慎使用错误跟踪器指令

请合理使用前面几节中介绍的跟踪器。最佳做法是使与错误跟踪相关联的开销仅报告最严重的问题，如后端系统中出现的无法恢复的问题。

默认的 *errors.pbd* 旨在报告严重错误，同时最大程度地减少开销。过度使用错误跟踪（例如，将 ExceptionErrorReporter 应用于每个受监控的方法）会导致大量“误报”。例如，如果用户在数字字段中输入“加利福尼亚”，可能导致 *NumberFormatException*，而您并不希望将其报告为严重问题。

使用 ErrorDetector

有关如何使用 ErrorDetector 的详细信息，请参阅《*CA APM Workstation 用户指南*》。

第 9 章： 配置 Boundary Blame

本节介绍了默认的 Java 代理 Blame 报告行为和相关的配置选项。

此部分包含以下主题：

了解 Boundary Blame

Blame 技术在托管 Java 应用程序中使用，使您能够在应用程序的前端和后端查看度量标准。该功能称为 Boundary Blame，允许用户将问题分类。

Introscope 检测后端的方式因应用程序而异。对于数据库活动，Introscope 使用 SQL 代理检测后端。如果 SQL 代理不可用，Introscope 会检测后端活动，因为诸如客户端/服务器数据库、JMS 服务器和 LDAP 服务器等后端是通过套接字访问的。如果您具有 Oracle 后端且没有使用 Introscope SQL 代理，请参阅 Boundary Blame 和 Oracle 后端 (see page 115)。

有关如何在 Introscope 调查器中显示 Boundary Blame 的信息，请参阅《*CA APM Workstation 用户指南*》。

使用 URL 组

您可以使用 URL 组监控浏览器响应其路径前缀始于您定义的字符串的一系列请求的时间。路径前缀是 URL 中与主机名相同的部分。例如，在以下 URL 中：

```
http://burger1.com/testwar/burgerServlet?ViewItem&category=
11776&item=5550662630&rd=1
```

路径前缀为：

```
/testwar
```

您可以为能够从 URL 路径前缀派生出的任何有用的请求类别定义 URL 组。例如，根据应用程序 URL 的形式，您可以为应用程序支持的每个客户、为每个主要应用程序或者为子应用程序定义 URL 组。这样您就可以监控已提交服务级别上下文中的性能，或者监控应用程序的关键业务部分。

示例：如何定义 URL 组属性

以下示例是 Java 代理配置文件的摘录，其中显示了 URL 组是如何定义的：

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
introscope.agent.urlgroup.group.alpha.pathprefix=/testWar
introscope.agent.urlgroup.group.alpha.format=foo {host} bar {protocol} baz
{port} quux {query_param:foo} red {path_substring:2:5} yellow
{path_delimited:/:0:1} green {path_delimited:/:1:4} blue {path_substring:0:0}
introscope.agent.urlgroup.group.beta.pathprefix=/nofilterWar
introscope.agent.urlgroup.group.beta.format=nofilter foo {host} bar {protocol}
baz {port} quux {query_param:foo} red {path_substring:2:5} yellow
{path_delimited:/:0:1} green {path_delimited:/:1:4} blue {path_substring:0:0}
introscope.agent.urlgroup.group.gamma.pathprefix=/examplesWebApp
introscope.agent.urlgroup.group.gamma.format=Examples Web App
```

定义 URL 组的键

属性 *introscope.agent.urlgroup.keys* 定义了所有 URL 组的 ID 或键的列表。URL 组的键可在声明 URL 组属性的其他属性定义中引用。以下示例定义了三个 URL 组的键：

```
introscope.agent.urlgroup.keys=alpha,beta,gamma
```

如果您定义了多个 URL 组并因此使某些 URL 属于多个组，则属性中 URL 组的键列表的排列顺序就便得非常重要。成员范围较小的 URL 组应位于成员范围较大的 URL 组之前。例如，如果带有键 **alpha** 的 IP 组具有路径前缀 */examplesWebApp*，带有键 **delta** 的 URL 组具有路径前缀 */examplesWebApp/cleverones*，则 *delta* 在 *keys* 参数中应位于 *alpha* 之前。

定义每个 URL 组的成员资格

属性 *introscope.agent.urlgroup.group.***groupKey**.*pathprefix* 指定了 URL 路径前缀所匹配的模式，定义了哪些请求落在 URL 组的范围之内。

示例：将组键映射到 URL 路径前缀

该属性定义将 URL 路径部分以 */testWar* 开头的所有请求分配给其键为 *alpha* 的 URL 组：

```
introscope.agent.urlgroup.group.alpha.pathprefix=/testWar
```

与指定的 *pathprefix* 相匹配的请求包括：

```
http://burger1.com/testWar/burgerServlet?ViewItem&category=
11776&item=5550662630&rd=1
http://burger1.com/testWar/burgerServlet?Command=Order&item=5550662630
```

示例：按应用程序路径创建 URL 组

如果公司提供了呼叫中心服务，可通过为每个应用程序功能设置一个 URL 组来监控功能区域的响应时间。如果客户通过以下 URL 访问服务：
*http://genesystems/us/*application_function/

其中，*application_function* 对应于应用程序（如 *OrderEntry*、*AcctService* 和 *Support*），每个 URL 组的 *pathprefix* 属性用于指定相应的 *application_function*。

注意：您可以在 *pathprefix* 属性中使用星号 (*) 作为通配符。

定义 URL 组的名称

属性 *introscope.agent.urlgroup.group.groupKey.format* 确定了键为 *groupKey* 的 URL 组的响应时间度量标准在 Introscope Workstation 中出现在哪些名称下。

通常情况下，*format* 属性用于分配文本字符串作为 URL 的名称。在以下示例中，键为 *alpha* 的 URL 组度量标准以 *Alpha Group* 名称显示在 Workstation 中：

introscope.agent.urlgroup.group.alpha.format=Alpha Group

URL 组的高级命名技术（可选）

您可以从请求元素（如服务器端口、协议）或从请求 URL 的子字符串派生 URL 组名称。如果通过检查请求能够轻易地区分应用程序模块，这将非常有用。本节介绍了 *format* 属性的高级表单。

将主机用作 URL 组名称

要将 URL 组度量标准组织到反映与请求相关联的 HTTP 服务器主机名的名称下，请按如下所示定义 *format* 参数：

introscope.agent.urlgroup.group.alpha.format={host}

当 *format={host}* 时，这些请求的统计信息将分别显示在 *us.mybank.com* 和 *uk.mybank.com* 度量标准名称下：

https://us.mybank.com/mifi/loanApp.....

https://uk.mybank.com/mifi/loanApp.....

将协议用作 URL 组名称

要将 URL 组统计信息组织到反映与请求相关联的协议的名称下，请按如下所示定义 *format* 参数：

introscope.agent.urlgroup.group.alpha.format={protocol}

当 *format={protocol}* 时，统计信息在调查器中会分组到请求 URL 的协议部分所对应的度量标准名称下。例如，这些请求的统计信息将显示在 *https* 度量标准名称下：

https://us.mybank.com/cgi-bin/mifi/scripts……

https://uk.mybank.com/cgi-bin/mifi/scripts……

将端口用作 URL 组名称

要将 URL 组统计信息组织到反映与请求相关联的端口的名称下，请按如下所示定义 *format* 参数：

introscope.agent.urlgroup.group.alpha.format={port}

当 *format={port}* 时，统计信息会分组到请求 URL 的端口部分所对应的名称下。例如，这些请求的统计信息将显示在 *9001* 名称下。

https://us.mybank.com:9001/cgi-bin/mifi/scripts……

https://uk.mybank.com:9001/cgi-bin/mifi/scripts……

将参数用作 URL 组名称

要在调查器中将 URL 组统计信息组织到反映与请求相关联的参数值的度量标准名称下，请按如下所示定义 *format* 参数：

```
introscope.agent.urlgroup.group.alpha.format={query_param:param}
```

其中 *format={query_param:*param*}* 统计信息在调查器中分组在与指定参数值相对应的度量标准名称下。没有参数的请求将列在 <empty> 下。例如，给定以下参数定义：

```
introscope.agent.urlgroup.group.alpha.format=
{query_param:category}
```

这些请求的统计信息将显示在度量标准名称“734”下

```
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=3772&tc=photo
http://ubuy.com/ws/shoppingServlet?ViewItem&category=734
&item=8574&tc=photo
```

将请求路径的子字符串用作 URL 组名称

要将 URL 组统计信息组织到反映请求 URL 的路径部分子字符串的名称下，请按如下所示定义 *format* 参数：

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:m:n}
```

其中，*m* 是第一个字符的索引，*n* 是大于最后一个字符的索引的索引。字符串选择操作的运行如 *java.lang.String.substring()* 方法一样。例如，给定以下设置：

```
introscope.agent.urlgroup.group.alpha.format=
{path_substring:0:3}
```

以下请求的统计信息将显示在度量标准节点 */ht* 下：

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

将请求路径的分隔部分用作 URL 组名称

要将 URL 组统计信息组织到反映请求 URL 路径的字符分隔部分的名称下，请按如下所示定义 *format* 参数：

```
introscope.agent.urlgroup.group.alpha.format=
{path_delimited:delim_char:m:n}
```

其中，*delim_char* 是用于分隔路径中各区段的字符，*m* 是要选择的第一个区段的索引，*n* 是大于要选择的最后区段的索引的索引。例如，给定以下设置：

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/:2:4}
```

以下形式的请求的统计信息：

```
http://www.buyitall.com/userid,sessionid/pageid
```

将显示在度量标准名称 */pageid* 下

请注意：

- 在本示例中，分隔符正斜杠 (/) 算作一个区段
- 区段计数从 0 开始

在 URL 示例中由斜杠字符分隔的区段有：

0=/、1=*userid,sessionid*、2=/ 和 3=*pageid*

您可以根据需要指定多个分隔符。例如，给定以下设置：

```
introscope.agent.urlgroup.group.alpha.format={path_delimited:/,:3:4}
```

上面显示的表单请求的统计信息将显示在度量标准名称 *sessionid* 下，其中在 URL 示例中由斜杠和逗号分隔的区段为：

0=/、1=*userid*、2=,、3=*sessionid*、4=/ 和 5=*pageid*

为 URL 组使用多种命名方法

您可以在单个 *format* 字符串中结合使用多种命名方法，如下所示：

```
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
```

运行 URLGrouper

URLGrouper 是一个命令行实用工具，用于分析通用格式的 Web 服务器日志文件。使用 URLGrouper，您可以着手定义自己的 URL 组。

注意：您可以使用 URLGRouper 实用工具分析 Web 服务器日志文件。URLGrouper 基于 Web 服务器日志文件的内容输出潜在 URL 组的一组属性设置。星号 (*) 可作为通配符用于 URLGrouper。

运行 URLGrouper

1. 打开一个命令 shell。
2. 输入以下命令

   ```
   java -jar urlgrouper.jar logfile
   ```

   其中 *logfile* 是 Web 服务器日志文件的完整路径。
3. URL 组集合的属性定义输出到 STDOUT。
4. 要配置建议的 URL 组，请将 URL Grouper 生成的属性语句复制到 *IntroscopeAgent.profile* 中。

使用 Blame 跟踪器

您可以使用跟踪器在应用程序中显式地标记前端和后端。有关详细信息，请参阅使用 Blame 跟踪器标记 Blame 点 (see page 112)。

第 10 章： 配置事务跟踪选项

本节介绍了有关默认的事务跟踪行为和相关配置选项的信息。

此部分包含以下主题：

事务跟踪新模式

CA APM Introscope 9.1 引入了新的代理跟踪体系结构和新的跟踪器。此事务跟踪新模式将替换之前的跟踪器在早期版本中使用的事务 blame 堆栈。此新模式实施可优化计算和处理，从而提高代理性能。

CA APM Introscope 9.1 仍允许您使用事务 blame 堆栈配置和运行代理。代理的此“传统”模式受到完全支持，但未来版本中的代理增强功能将仅针对新模式实施。当在“传统”模式下运行代理时，以下功能将不可用：

- 版本 9.1 中的代理 CPU 使用量和响应时间优化
- .NET 代理的动态检测
- SQL 代理属性

  ```
  introscope.agent.sqlagent.sql.artonly

  introscope.agent.sqlagent.sql.turnoffmetrics
  ```

重要信息！在新模式下使用 CA APM 代理运行传统跟踪器称为“混合模式”配置。请**不要**在混合模式下运行，因为可能会出现内存消耗和服务中断。

如果运行的是传统模式跟踪器，请在传统模式下运行代理。当客户的传统跟踪器不兼容时，以下消息将写入代理日志中以通知客户。

> “检测到使用传统 API 的代理跟踪器。不建议在代理使用新模式的情况下运行传统的跟踪器。请联系支持人员，他会向您介绍有关如何升级传统跟踪器的文档。临时情况下，请配置代理使用传统模式或使用预配置版本的传统代理包。例如，UNIX 上 Oracle WebLogic 代理的传统软件包为 IntroscopeAgentFiles-Legacy-NoInstallerx.x.x.xweblogic.unix.tar”

恢复到在传统模式下运行代理，直到您运行新模式跟踪器。

如果您使用的是以下开箱即用型扩展，请将您的代理配置为在传统模式下运行：

- 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

将代理配置为使用传统模式事务跟踪

转换为传统模式事务跟踪仅适用于具有新模式的 CA APM 版本。版本 9.1 之前的 CA APM 版本不具有新模式。

提供了两个选项来将代理配置为使用传统模式：

- 部署预配置的传统代理软件包。可在代理的仅限于文件、无安装程序的软件包中获取这些软件包。例如，UNIX 上 Oracle WebLogic 代理的传统软件包为：

  `IntroscopeAgentFiles-Legacy-NoInstaller<版本号>weblogic.unix.tar`

- 针对传统模式手工配置。

请执行以下步骤：

1. 停止受监控的应用程序。
2. 存档并删除 <Agent_Home>/logs 目录中的现有日志文件，从而为新日志做好准备。
3. 备份 <Agent_Home>/core/config 目录中的现有 .pbl 和 .pbd 文件。
4. 备份现有文件 <Agent_Home>/core/config/IntroscopeAgent.profile。
5. 将旧版的 .pbl 和 .pbd 文件从 <Agent_Home>/examples/legacy 目录复制到 <Agent Home>/core/config 目录。
6. 打开 <Agent_Home>/core/config/IntroscopeAgent.profile 并进行以下更改：
   - 添加属性 introscope.agent.configuration.old=true。
   - 更新代理属性 introscope.autoprobe.directivesFile，以便指向已复制的相应传统 .pbl 和 .pbd 文件。例如，使用 spm-legacy.pbl 替换 spm.pbl。
7. 重新启动受监控的应用程序。

要配置特定 CA APM 扩展以便在典型安装中使用传统模式，请参阅下表中列出的 .pbl 和 .pbd 文件。

扩展名称	旧版的典型 pbl 或 pbd 文件
CA APM for Oracle Weblogic Server	ppweblogic-legacy.pbd spm-legacy.pbl
CA APM for Oracle Weblogic Portal	powerpackforweblogicportal-legacy.pbl spm-legacy.pbl
CA APM for Oracle Service Bus	OSB-typical-legacy.pbl spm-legacy.pbl
CA APM for IBM Websphere Portal	powerpackforwebsphereportal.pbl spm-legacy.pbl
CA APM for IBM Websphere MQ	webspheremq-legacy.pbl
CA APM for IBM Websphere Process Server /Business Process Management	wps-legacy.pbd wesb-legacy.pbd spm-legacy.pbl
CA APM for TIBCO BusinessWorks	tibcobw-typical-legacy.pbl spm-legacy.pbl
CA APM for Software AG Webmethods Integration Server	webmethods-legacy.pbl spm-legacy.pbl
CA APM for CA SiteMinder Web Access Manager	smwebagentext.pbd
CA APM for IBM CICS Transaction Gateway	PPCTGTranTrace.pbd PPCTGServer-typical.pbd PPCTGClient-typical.pbd
CA APM for IBM z/OS	zos-full.pbl
CA APM for CA LISA	lisa-typical.pbl
CA APM for SYSVIEW	CTG_ECI_Tracer_For_SYSVIEW-legacy.pbd HTTP_Tracer_For_SYSVIEW-legacy.pbd WS_Tracer_For_SYSVIEW-legacy.pbd

控制自动事务跟踪行为

通过自动事务跟踪，可以对可能有问题的事务类型进行历史分析，而不需要显式运行事务跟踪。Introscope 提供两种类型的自动事务跟踪：

- 事务跟踪采样，默认是启用的，基于 URL 分组
- 可配置的自动跟踪采样，收集跟踪信息时不考虑 URL 分组

事务跟踪组件限定

Introscope 设置了一个组件限定来限制跟踪的大小。默认为 5,000 个组件。到达此限制时，日志中将显示警告，跟踪将停止。

您可以限定超出预期组件计数的组件事务，例如，当 servlet 执行数百个对象交互和后端 SQL 调用时。如果没有限定，事务跟踪器会将其视为一个事务而不停地继续。在某些极端的情况下，如果没有限定，JVM 会在跟踪完成之前耗尽内存。

用于限定事务的属性位于 IntroscopeAgent.profile 文件中：

introscope.agent.transactiontrace.componentCountClamp=*5000*

产生受限定组件的跟踪会使用星号进行标记。有关查看这些跟踪的详细信息，请参阅 *《CA APM Workstation 用户指南》*。

重要信息！如果事务跟踪组件限定大小增加，事务跟踪所需的内存也可能会增加。有关详细信息，请参阅 *《CA APM 配置和管理指南》*。

事务跟踪采样

默认情况下，事务跟踪采样处于启用状态。可以根据需要禁用此行为。有关事务跟踪属性的详细信息，请参阅事务跟踪 (see page 299)。

配置自动跟踪采样时，可指定在指定时间间隔内要跟踪的事务数量。

注意：这些属性位于企业管理器属性文件中。在更改 *sampling.perinterval* 和 *sampling.interval* 属性的默认值之前，请注意更高的采样率可能会增加企业管理器中的负载。企业管理器会将此配置推进到与其连接的所有代理。还可以在代理中配置这些属性。在代理中配置这些属性将覆盖企业管理器为单个代理设置的配置。

要配置自动跟踪采样，请修改以下属性

- *introscope.agent.transactiontracer.sampling.enabled*

  设置为 false 可禁用事务跟踪采样。默认值为 true。

- *introscope.agent.transactiontracer.sampling.perinterval.count*

  指定在指定时间间隔内要跟踪的事务数量。

  事务的默认数量是 1。

- *introscope.agent.transactiontracer.sampling.interval.seconds*

  指定跟踪指定数量的事务所需的时间长度。

  默认时间间隔为每 2 分钟一次。

代理堆大小调整

代理使用 Java 堆内存来存储收集的数据。可以在“GC 堆”概览中查看代理 GC 堆使用情况。

以下信息适用于堆使用情况：

- 如果堆的使用率较高，则安装代理时可能需要增加堆分配。
- 如果受监控应用程序上的事务非常深入或持续时间很长，则事务跟踪采样可能需要更多的堆内存。

注意：有关 GC 堆使用情况的详细信息，请参阅《*CA APM Workstation 用户指南*》。如果正在运行高性能 CA Introscope® 环境，请与 CA 专业服务联系以获得正确的代理 JVM 堆设置。

详细信息：

事务跟踪组件限定 (p. 162)
事务跟踪采样 (p. 162)

跨进程事务跟踪

CA Introscope® Java 代理可跟踪 WebLogic Server 或 WebSphere 上跨 JVM 边界的事务。条件是环境必须包含相同供应商提供的应用程序服务器的兼容版本。同步事务（例如 EJB 的 servlet）和异步事务都支持跨进程事务跟踪。

重要信息！只有 CA Introscope® 9.0 或更高版本的代理才能使用跨进程事务跟踪。

详细信息：

在 WebLogic Server 中启用跨进程跟踪 (p. 38)
在 WebSphere 中启用跨进程跟踪 (p. 51)

扩展事务跟踪数据收集

可以配置 Introscope 事务跟踪器以获取其他信息，包括 Servlet 和 JSP 调用的用户 ID 数据，以及其他事务跟踪数据，例如 HTTP 请求标头、请求参数和会话属性。要捕获此信息，必须在 IntroscopeAgent.profile 中定义条件。

关于用户 ID 数据

要配置 Java 代理以标识 Servlet 和 JSP 调用的用户 ID，必须首先获取有关托管应用程序如何指定用户 ID 的信息。开发托管应用程序的应用程序架构师可能会提供此信息。

Introscope 事务跟踪器可标识按以下方式之一存储用户 ID 的托管应用程序中的用户 ID：

- HttpServletRequest.getRemoteUser()
- HttpServletRequest.getHeader (字符串键)
- HttpSession.getValue (字符串键)，其中返回的对象是表示用户 ID 的字符串，或自身的 toString() 返回到用户 ID 的对象

如果托管应用程序使用这些方法之一存储用户 ID，请参阅配置代理以收集其他事务跟踪数据 (see page 165)来配置 Java 代理设置以收集用户 ID 数据。

关于 servlet 请求数据

使用 Introscope，可以收集与用户可配置的参数匹配的事务跟踪数据。例如，可以指定代理收集包含用户代理 HTTP 请求标头的事务的事务跟踪数据。

Introscope 可记录以下 servlet 请求信息：

- 请求标头
- 请求参数
- 会话属性

要记录托管应用程序的此 servlet 请求信息，请参阅配置代理以收集其他事务跟踪数据 (see page 165)来配置 Java 代理设置以收集此数据。

配置代理以收集其他事务跟踪数据

可以配置 Java 代理以收集其他事务跟踪数据，例如用户 ID、HTTP 请求标头、HTTP 请求参数或 HTTP 会话属性。

请执行以下步骤：

1. 打开代理配置文件 *IntroscopeAgent.profile*。
2. 在“事务跟踪器配置”标题下找到事务跟踪器属性。

收集用户 ID 数据

配置 Java 代理以标识用户 ID

1. 配置与托管应用程序用于存储用户 ID 的方法相对应的属性。

   注意：请确保只有一组属性未注释，否则可能会使用错误的属性。

2. 对于 HttpServletRequest.getRemoteUser()，请将以下属性取消注释：
   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getRemoteUser
   ```
3. 对于 HttpServletRequest.getHeader (字符串键)，请将以下一对属性取消注释并为第二个属性定义键字符串：
   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getHeader
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```
4. 对于 HttpSession.getValue（字符串键），请取消注释以下一对属性并定义第二个属性的键字符串：
   ```
   introscope.agent.transactiontracer.userid.method=HttpServletRequest.getValue
   introscope.agent.transactiontracer.userid.key=<application defined key string>
   ```

收集 servlet 请求数据

记录 servlet 请求信息（如 HTTP 请求标头和参数）

1. 要指定收集其事务跟踪数据的 HTTP 请求标头，请将此属性取消注释，并在以逗号分隔的列表中指定要跟踪的 HTTP 请求标头：
   `#introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent`
2. 要指定用于收集事务跟踪数据的 HTTP 请求参数，请在以逗号分隔的列表中取消注释该属性，并指定要跟踪的 HTTP 请求参数：
   `#introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter1,parameter2`
3. 要指定跟踪其数据的 HTTP 会话属性，请将此属性取消注释，并在以逗号分隔的列表中指定要跟踪的 HTTP 会话属性，例如：
   `#introscope.agent.transactiontracer.parameter.httpsession.attributes=cartID,deptID`
4. 重新启动托管应用程序。

配置组件停顿报告

Application Performance Management 停顿是指在某个定义的时间长度内没有来自已探测组件的响应。默认情况下，APM Introscope 代理可检测出此状况并报告停顿度量标准。

每当代理检查停顿时，会检查方法堆栈上所有最常检测的组件。当组件停顿时，会创建停顿度量标准和停顿快照。还会针对订阅下游监控的每个组件创建停顿度量标准。

首先针对方法堆栈上的最常检测的组件创建停顿度量标准。然后再针对所需时间超过 *introscope.agent.stalls.thresholdseconds* 值的组件创建这些度量标准。

下游订户组件停顿

在调用停顿的下游组件时，订阅下游监控的组件将停顿。

默认情况下，以下组件会订阅下游停顿监控：

- FrontendTracer 探测组件
- BackendTracer 探测组件
- WebServiceBlamepointTracer@Servicelevel 探测组件
- WebServiceBlamepointTracer@Opertationlevel 探测组件

示例

前端组件对调用停顿后端组件的 Web 服务进行调用。

前端--> Web 服务-->后端

将针对后端、Web 服务和前端组件创建停顿度量标准，因为它们在默认情况下都订阅下游监控。

上游继承组件停顿

停顿检查器首先检查组件堆栈的顶级成员，查看其是否已停顿。如果顶级组件未停顿，停顿检查器会查找已停顿的父组件。当父组件停顿时，将针对该组件和堆栈中的每个上游父组件创建停顿度量标准。

如果组件 M1() 调用多个组件 M2()（每个仅需花费几秒种），则可以为 M1() 报告停顿度量标准。

示例：

停顿阈值设置为 15 秒。方法 M1() 循环调用方法 M2() 100 次。M2() 从不停顿，因为每次响应只需 2 秒。但是，M1() 最终将停顿。

禁止将停顿作为事件进行捕获

默认情况下，Introscope 在事务事件数据库中将事务停顿作为事件进行捕获，并从已检测的事件生成停顿度量标准。将为事务中的第一个和最后一个方法生成停顿度量标准。用户可以在 Workstation 历史事件查看器中查看停顿事件以及关联的度量标准。

注意：生成的停顿度量标准始终可用，但是停顿事件只有在安装 Introscope 错误检测器后才可见。停顿作为普通错误进行存储，可在错误类型视图中查看，也可以通过查询“type:errorsnapshot”在历史查询查看器中查看。

要禁止将停顿作为事件进行捕获、更改停顿阈值，或更改代理检查停顿的频率， 请使用以下属性：

- *introscope.agent.stalls.thresholdseconds* 指定响应时间的最小阈值，达到该时间后，事务将被视为停顿。
- *introscope.agent.stalls.resolutionseconds* 指定代理检查停顿的频率。

第 11 章： 配置 Introscope SQL 代理

本节包含配置 Introscope SQL 代理的说明。

此部分包含以下主题：

SQL 代理概览

SQL 代理可将详细的数据库性能数据报告给企业管理器。通过跟踪托管应用程序与数据库之间的交互，SQL 代理可以了解应用程序中各个 SQL 语句的性能。

SQL 代理监控 SQL 语句的方法与 Java 代理监控应用程序的方法相同。SQL 代理是非侵入式的，可使用非常低的开销来监控应用程序或数据库。

为了在单个 SQL 语句级别上提供有意义的性能度量标准，SQL 代理通过剥离特定于事务的数据并将原始 SQL 语句转换为特定于 Introscope 的规范化语句来汇总性能数据。由于规范化语句不包括敏感信息（如信用卡号），因此该过程还可以保护您数据的安全。

例如，SQL 代理将此 SQL 查询：

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

转换为此规范化语句：

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

同样，SQL 代理将此 SQL 更新语句：

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES ('Atwood', 'The Robber Bride')
```

转换为此规范化语句：

```
INSERT INTO BOOKS (AUTHOR, TITLE) VALUES (?, ?)
```

注意：只有引号内的文本 ('xyz') 是规范化的。

规范化语句的度量标准汇集在一起，可以在 Workstation 调查器的 JDBC 节点中进行查看。

SQL 代理文件

SQL 代理包含在所有代理安装中。提供 SQL 代理功能的文件包括：

- wily/core/ext/SQLAgent.jar
- wily/core/config/sqlagent.pbd

注意：默认情况下，诸如 SQLAgent.jar 文件等代理扩展安装在 wily/core/ext 目录中。可以使用代理配置文件中的 introscope.agent.extensions.directory 属性更改代理扩展目录的位置。如果您更改了 */ext* 目录的位置，请务必同时移动 */ext* 目录的内容。

SQL 语句规范化

有些应用程序会生成大量唯一的 SQL 语句。如果正在使用诸如 EJB 3.0 之类的技术，则生成很长的唯一 SQL 语句的可能性会增加。长 SQL 语句会在代理中造成度量标准爆发，从而导致性能下降以及其他系统问题。

编写不当的 SQL 语句如何导致度量标准爆发

一般情况下，SQL 代理度量标准的数目应接近唯一 SQL 语句的数目。如果即使应用程序使用一小组 SQL 语句，SQL 代理仍显示大量且越来越多的唯一 SQL 度量标准，问题可能出在 SQL 语句的编写方式上。

SQL 语句可导致度量标准爆发的情况很常见。例如，每次执行用于包括嵌入的注释、创建临时表或插入值列表的 SQL 语句时，都会无意中创建唯一的度量标准名称。

示例：SQL 语句中的注释

度量标准爆发的一个常见原因是在 SQL 语句中使用注释的方法有误。例如，如果您的 SQL 语句如下：

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

SQL 代理将创建度量标准名称中包含注释的度量标准：

```
"/* John Doe, user ID=?, txn=? */ select * from table..."
```

SQL 语句中嵌入的注释对于数据库管理员查看由谁执行何种查询、哪个执行查询的数据库忽略查询非常有用。然而，SQL 代理在捕获 SQL 语句时无法解析注释字符串。因此，对于每个唯一的用户 ID，SQL 代理会创建唯一的度量标准，这可能导致度量标准爆发。

可以通过将 SQL 注释放在单引号中来避免此问题。例如：

```
"/*' John Doe, user ID=?, txn=? '*/ select * from table..."
```

然后，SQL 代理将创建以下度量标准，其中注释不再产生唯一的度量标准名称：

```
"/* ? */ select * from table..."
```

示例：临时表或自动生成的表名称

度量标准爆发的另一个可能原因是：应用程序在 SQL 语句中引用临时表或自动生成名称的表。例如，如果打开调查器在“后端|{backendName}|SQL|{sqlType}|sql”下查看度量标准，可能会看到与下面类似的度量标准：

```
SELECT * FROM TMP_12398139821038192O912 WHERE ROW_ID = ?
```

此 SQL 语句正在访问将唯一标识符附加到表名的临时表。附加到 *TMP_* 表名的其他数字会在每次执行语句时创建一个唯一的度量标准名称，从而导致度量标准爆发。

示例：可生成值列表或插入值的语句

度量标准爆发的另一个常见原因是：SQL 语句生成值列表或对值进行大量修改。例如，假设系统已警告您可能会发生度量标准爆发，并且您通过调查对以下 SQL 语句进行了检查：

```
#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)
```

在研究代码时，请注意 *"CHANGE CITY FROM CARROLTON, TO CAROLTON, _ "* 会生成一个城市数组。

同样，如果您正在调查可能的度量标准爆发，可能会复查与下面类似的 SQL 语句：

```
CHANGE COUNTRY FROM US TO CA _ CHANGE EMAIL ADDRESS FROM TO BRIGGIN @ COM _ "
```

在研究代码时，会注意到 CHANGE COUNTRY 产生很长的国家/地区列表。此外，国家/地区的引号放置可导致在电子邮件地址中插入 SQL 语句，从而创建可能会引起度量标准爆发的唯一度量标准。

SQL 语句规范化选项

为了解决长 SQL 语句问题，SQL 代理提供了以下规范化程序以供使用：

- 默认 SQL 语句规范化程序 (see page 172)
- 自定义 SQL 语句规范化程序 (see page 173)
- 正则表达式 SQL 语句规范化程序 (see page 174)
- 命令行表达式 SQL 语句规范化程序 (see page 178)

默认 SQL 语句规范化程序

默认情况下，标准的 SQL 语句规范化程序在 SQL 代理中处于启用状态。它对单引号内的文本 ('xyz') 进行规范化。例如，SQL 代理将此 SQL 查询：

```
SELECT * FROM BOOKS WHERE AUTHOR = 'Atwood'
```

转换为此规范化语句：

```
SELECT * FROM BOOKS WHERE AUTHOR = ?
```

规范化语句的度量标准汇集在一起，可以在 Workstation 调查器中进行查看。

自定义 SQL 语句规范化程序

SQL 代理允许您添加扩展来执行自定义规范化。为此，请创建一个 JAR 文件，其中包含由 SQL 代理实施的规范化方案。

应用 SQL 语句规范化程序扩展

1. 创建扩展 JAR 文件。

   注意：SQL 规范化程序扩展文件的入口点类必须实现 *com.wily.introscope.agent.trace.ISqlNormalizer* 接口。

   创建 JAR 扩展文件包括创建清单文件，该清单文件包含下面步骤 2 中详细介绍的 SQL 规范化程序扩展的特定键。然而，要使扩展能够正常运行，还需要其他常规键。这些键是您将要用来构建任意扩展文件的类型。创建的扩展文件与数据库 SQL 语句文本规范化相关，例如 *Backends|{backendName}|SQL|{sqlType}|{actualSQLStatement}* 节点下的度量标准。*{actualSQLStatement}* 由 SQL 规范化程序进行规范化。

2. 将以下键放置在所创建扩展的清单中：

   - *com-wily-Extension-Plugins-List:testNormalizer1*

     注意：此键的值可为任意值。在本例中，*testNormalizer1* 用作示例。无论您为此键指定何值，请在以下键中使用该值。

   - *com-wily-Extension-Plugin-testNormalizer1-Type: sqlnormalizer*
   - *com-wily-Extension-Plugin-testNormalizer1-Version: 1*
   - *com-wily-Extension-Plugin-testNormalizer1-Name: normalizer1*

     应当包含规范化程序的唯一名称，例如 normalizer1。

   - *com-wily-Extension-Plugin-testNormalizer1-Entry-Point-Class: <Thefully-qualified classname of your implementation of ISQLNormalizer>*

3. 将创建的扩展文件放在 *<Agent_Home>/wily/core/ext* 目录中。

4. 在 *IntroscopeAgent.profile* 中，找到以下属性并进行设置：

   ```
   introscope.agent.sqlagent.normalizer.extension
   ```

   从所创建扩展的清单文件中，将属性设置为 *com-wily-Extension-Plugin-{plugin}-Name*。此属性的值不区分大小写。例如：

   ```
   introscope.agent.sqlagent.normalizer.extension=normalizer1
   ```

   重要信息！这是一个热属性。更改扩展名称将导致重新注册扩展。

5. 在 *IntroscopeAgent.profile* 中，您可以选择性地添加以下属性来设置错误限制计数：
   `introscope.agent.sqlagent.normalizer.extension.errorCount`
   有关错误和异常的更多信息，请参阅异常 (see page 174)。
   注意：如果自定义规范化程序扩展抛出的错误超过了错误限制计数，将禁用扩展。
6. 保存 *IntroscopeAgent.profile*。
7. 重新启动应用程序。

异常

如果创建的扩展针对某个查询抛出了异常，默认 SQL 语句规范化程序将为该查询使用默认的规范化方案。发生这种情况时，将记录一条 ERROR 消息，指出扩展抛出了异常，还会记录一条包含堆栈跟踪信息的 DEBUG 消息。但是，抛出五个此类异常之后，默认 SQL 语句规范化程序将禁用所创建的扩展，并停止尝试将创建的扩展用于将来的查询，直到更改了规范化程序。

空字符串

如果创建的扩展针对某个查询返回了空字符串，StatementNormalizer 将为该查询使用默认的规范化方案，并记录一条 INFO 消息，指出扩展返回了空值。但是，返回五个此类空字符串之后，StatementNormalizer 将停止记录消息，但是将尝试继续使用扩展。

正则表达式 SQL 语句规范化程序

SQL 代理附带了一个扩展，该扩展根据可配置的正则表达式 (regex) 对 SQL 语句进行规范化。该文件 (*RegexNormalizerExtension.jar*) 位于 *<Agent_Home>/wily/core/ext* 目录中。

有关如何使用正则表达式 SQL 语句规范化程序的示例，请参阅正则表达式 SQL 语句规范化程序示例 (see page 176)。

应用正则表达式扩展

1. 打开 *IntroscopeAgent.profile*。
2. 找到并设置以下属性：
   - *introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer*
     指定将用于覆盖预配置规范化方案的 SQL 规范化程序扩展的名称。启用正则表达式扩展时，将此属性设置为 RegexSqlNormalizer。

- *introscope.agent.sqlagent.normalizer.regex.keys=key1*

  此属性指定正则表达式组键，这些键将按排列顺序进行评估。要启用正则表达式扩展，此属性是必需的。没有默认值。

- *introscope.agent.sqlagent.normalizer.regex.key1.pattern=A*

  此属性指定用于与 SQL 语句匹配的正则表达式模式。*java.util.Regex package* 类允许的所有有效正则表达式均可以在此处使用。要启用正则表达式扩展，此属性是必需的。没有默认值。

- *introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=B*

  此属性指定替换字符串的格式。*java.util.Regex package* 类允许的所有有效正则表达式均可以在此处使用。要启用正则表达式扩展，此属性是必需的。没有默认值。

- *introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false*

  如果此属性设置为 true，将针对所有正则表达式键组评估 SQL 字符串。实施是连锁的。因此，如果 SQL 字符串与多个键组匹配，则来自 group1 的规范化 SQL 输出将作为输入送入 group2，依此类推。

  如果该属性设置为 false，只要键组与 SQL 字符串匹配，就会返回来自该组的规范化 SQL 输出。MatchFallThrough 属性不会启用或禁用扩展。

  例如，如果您有一个 SQL 字符串，例如：Select * from A where B，您可以设置以下属性：

  ```
  introscope.agent.sqlagent.normalizer.regex.keys=key1,key2
  introscope.agent.sqlagent.normalizer.regex.key1.pattern=A
  introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=X
  introscope.agent.sqlagent.normalizer.regex.key2.pattern=B
  introscope.agent.sqlagent.normalizer.regex.key2.replaceFormat=Y
  ```

如果 *introscope.agent.sqlagent.normalizer.regex.matchFallThrough* 为 *false*，则 SQL 会根据 key1 正则表达式进行规范化。该正则表达式的输出将为 *Select * from X where B*。将返回此 SQL。

如果 *introscope.agent.sqlagent.normalizer.regex.matchFallThrough* 为 *true*，则 SQL 首先针对 key1 正则表达式进行规范化。该正则表达式的输出为 *Select * from X where B*。然后，此输出将被送入 *key2* 正则表达式。*key2* 正则表达式的输出为 *Select * from X where Y*。将返回该 SQL。

注意：要启用正则表达式扩展，此属性不是必需的。

- *introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false*

  此属性指定模式匹配是否区分大小写。默认值为 false。要启用正则表达式扩展，此属性不是必需的。

- *introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false*

  如果此属性设置为 false，它会将 SQL 中首次出现的匹配模式替换为替换字符串。如果此属性设置为 true，它会将 SQL 中出现的所有匹配模式均替换为替换字符串。

  例如，如果您有一个 SQL 语句，例如：Select * from A where A like Z，则可以对属性进行如下设置：

  ```
  introscope.agent.sqlagent.normalizer.regex.key1.pattern=A
  introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=X
  ```

  如果 *introscope.agent.sqlagent.normalizer.regex.key1.replaceAl* 为 *false*，将会生成规范化的 SQL 语句：*Select * from X where A like Z*。

  如果 *introscope.agent.sqlagent.normalizer.regex.key1.replaceAl* 为 *true*，将生成一个规范化 SQL 语句：*Select * from X where* **X** *like Z*。

  默认值为 false。要启用正则表达式扩展，此属性不是必需的。

  注意：如果没有一个正则表达式模式与输入的 SQL 匹配，RegexNormalizer 将返回空字符串。然后，语句规范化程序将使用默认的规范化方案。

3. 保存 *IntroscopeAgent.profile*。

重要信息！上面列出的所有属性均为热属性，也就是说，保存 *IntroscopeAgent.profile* 之后，对这些属性所做的更改会立即生效。

正则表达式 SQL 语句规范化程序示例

下面三个示例可以帮助您了解如何实施正则表达式 SQL 语句规范化程序。

示例 1

这是在进行正则表达式 SQL 语句规范化之前的 SQL 查询：

```
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)
```

这是所需的规范化 SQL 语句：

```
INSERT INTO COMMENTS (COMMENT_ID, ...) VALUES (?, ?, ?, ?, ?, ?,?, CHANGE CITY
FROM ( )
```

这是 *IntroscopeAgent.profile* 文件生成上述规范化 SQL 语句所需的配置：

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=key1,key2
introscope.agent.sqlagent.normalizer.regex.key1.pattern=(INSERT INTO
COMMENTS \\(COMMENT_ID,)(.*)(VALUES.*)''(CHANGE CITY FROM \\().*(\\))
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1 ...) $3$4 $5
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
introscope.agent.sqlagent.normalizer.regex.key2.pattern='[a-zA-Z1-9]+'
introscope.agent.sqlagent.normalizer.regex.key2.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.key2.replaceFormat=?
introscope.agent.sqlagent.normalizer.regex.key2.caseSensitive=false
```

示例 2

这是在进行正则表达式 SQL 语句规范化之前的 SQL 查询：

```
SELECT * FROM TMP_12398139821038192091 2 WHERE ROW_ID =
```

这是所需的规范化 SQL 语句：

```
SELECT * FROM TMP_ WHERE ROW_ID =
```

这是 *IntroscopeAgent.profile* 文件生成上述规范化 SQL 语句所需的配置：

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=key1
introscope.agent.sqlagent.normalizer.regex.key1.pattern=(TMP_)[1-9]*
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
```

示例 3

如果想要对 SQL 语句进行规范化，例如：*Select …. ResID1, CustID1* where *ResID1=.. OR ResID2=.. n times OR CustID1=.. OR n times*，您可以将属性设置如下：

```
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=true
introscope.agent.sqlagent.normalizer.regex.keys=default,def
introscope.agent.sqlagent.normalizer.regex.default.pattern=(ResID)[1-9]
introscope.agent.sqlagent.normalizer.regex.default.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.default.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.default.caseSensitive=true
introscope.agent.sqlagent.normalizer.regex.def.pattern=(CustID)[1-9]
introscope.agent.sqlagent.normalizer.regex.def.replaceAll=true
introscope.agent.sqlagent.normalizer.regex.def.replaceFormat=$1
introscope.agent.sqlagent.normalizer.regex.def.caseSensitive=true
```

命令行表达式 SQL 语句规范化程序

如果当前没有使用正则表达式 SQL 规范化程序，并且您有 where 子句中使用双引号 (" ") 将值引起来的 SQL 语句，请使用以下命令行命令对 SQL 语句进行规范化：

```
-DSQLAgentNormalizeDoubleQuoteString=true
```

重要信息！ 您可以使用正则表达式 SQL 规范化程序（而不是此命令）对双引号中的 SQL 语句进行规范化。有关更多信息，请参阅正则表达式 SQL 语句规范化程序 (see page 174)。

关闭语句度量标准

有些应用程序会生成大量的唯一 SQL 语句，从而在 SQL 代理中造成度量标准爆发。您可以在 SQL 代理中关闭 SQL 语句度量标准。

注意：关闭语句度量标准后，不会丢失后端或顶级 JDBC 度量标准。

关闭语句度量标准

1. 打开 sqlagent.pbd 文件并找到 SQL 语句。例如：

   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements
   executeQuery(Ljava/lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends|{database}|SQL|{commandtype}|Query|{sql}"
   ```

2. 从希望关闭的跟踪指令中删除 *{sql}*。例如：

   ```
   TraceOneMethodWithParametersIfFlagged: SQLAgentStatements
   executeQuery(Ljava/ lang/String;)Ljava/sql/ResultSet; DbCommandTracer
   "Backends|{database}|SQL|{commandtype}|Query"
   ```

3. 保存 *sqlagent.pbd* 文件。

   将在 SQL 代理中关闭 SQL 语句度量标准。

SQL 度量标准

SQL 代理度量标准显示在 Introscope Workstation 调查器的“后端”节点下。可在“后端|<backendName>|SQL”节点下找到 SQL 语句度量标准。

注意： *“平均响应时间 (毫秒)”* 将仅显示返回数据阅读器的查询，即通过 *ExecuteReader()* 方法执行的查询。此度量标准表示在数据阅读器的 *Close()* 方法上花费的平均时间。

特定于 SQL 数据的度量标准类型包括：

- 连接计数－内存中实时连接对象的数目。

  当调用驱动程序的 *connect()* 方法时将打开连接，当通过 *close()* 方法关闭连接调用时将关闭连接。SQL 代理维护对连接集的弱引用。当连接对象是收集的垃圾时，计数将反映更改。

- 平均结果处理时间 (ms)－查询的平均处理时间。

  此度量标准表示处理结果集花费的平均时间，包括从 *executeQuery()* 调用结束到调用 ResultSet 的 *close()* 方法的时间。

注意：已检测的 XA 数据源可能不报告提交或回滚度量标准。其他已检测的数据源可能不报告提交或回滚度量标准，除非这些度量标准包含数据。

第 12 章： 启用 JMX 报告

本节介绍了有关启用 Java 代理以报告 JMX 数据的信息。

此部分包含以下主题：

Java 代理 JMX 支持

CA Introscope® 可以收集应用程序服务器或 Java 应用程序作为与 JMX 兼容的 MBean 公开的管理数据。CA Introscope® 会在“调查器”度量标准树中显示这些 JMX 数据。CA Introscope® 支持收集使用以下应用程序服务器的 JMX 信息：

- Glassfish－Glassfish 不需要其他配置即可报告 JMX 度量标准
- JBoss－将 JBoss 配置为报告 JMX 度量标准的说明位于将 JBoss 配置为使用 Java 代理 (see page 34)中
- Tomcat－将 Tomcat 配置为报告 JMX 度量标准的说明位于将 Apache Tomcat 配置为使用 Java 代理 (see page 33)中
- WebLogic－您可以将 WebSphere 和 WebLogic 配置为使用 JMX 报告 (see page 185)
- WebSphere－您可以将 WebSphere 和 WebLogic 配置为使用 JMX 报告 (see page 185)

注意：CA Introscope® 支持内置到 Sun JMX 规范中的任何 MBean。有关 Sun JMX 规范的详细信息，请参阅 Java 管理扩展。

CA Introscope® 会将 JMX 数据转换为 CA Introscope® 度量标准格式，并在调查器中的以下节点下进行显示：

<域>|*<主机>*|*<进程>*|*<代理>*|JMX|

Introscope 对 WebLogic JMX 度量标准的支持

WebLogic 提供以下 MBean 作为 JMX 度量标准的源：

- RuntimeServiceMBean：每服务器运行时度量标准，包括活动的有效配置
- DomainRuntimeServiceMBean：域范围内的运行时度量标准
- EditServiceMBean：允许用户编辑永久性配置

CA Introscope® 仅轮询 RuntimeServiceMBean，原因如下：

- RuntimeServiceMBean 支持本地访问（效率问题）。
- RuntimeServiceMBean 包含预期相关的大部分数据。

默认的 JMX 度量标准转换过程

默认情况下，CA Introscope® 会转换 JMX 度量标准以在调查器中显示。在以下情况下，CA Introscope® 会转换 MBean：

- 您使用 WebLogic。
- 您尚未配置主键转换来简化 JMX 度量标准 (see page 183)。

注意：如果您指定的主键没有与之匹配的 MBean，CA Introscope® 将使用默认的转换方法。

在默认转换方法中，CA Introscope® 同时显示属性的名称和值，并在度量标准树中按字母顺序列出名称/值对。

<域>|<主机>|<进程>|<代理>|JMX|*<域名>|<键 1>=<值 1>|<键 2>=<值 2>:<度量标准>*

例如，假定一个具有以下特征的 WebLogic MBean：

- 域名：WebLogic
- 键/值对：category=server、type=jdbc
- 度量标准名称：connections

如果在 IntroscopeAgent.profile 的属性 introscope.agent.jmx.name.primarykeys 中未指定任何主键，MBean 属性将转换为以下 CA Introscope® 度量标准：

<域>|*<主机>*|*<进程>*|*<代理>*|JMX|Weblogic|*类别=服务器*|*类型=jdbc:连接*

在 CA Introscope® 度量标准中，键/值对按字母顺序显示。

使用主键转换简化 JMX 度量标准

您可以随意配置度量标准在 JMX 节点下的显示顺序。可以在代理配置文件中定义主键。该主键可用于标识 MBean 的 ObjectName。

如果您未配置主键转换，CA Introscope® 将转换 JMX 数据 (see page 182)。使用默认转换，度量标准将按字母顺序在调查器中的 JMX 节点下列出。这种将 JMX 数据转换成度量标准的方法可简化度量标准名称。您可以控制键/值对信息在生成的度量标准中的顺序。

该行为是在代理配置文件中的 introscope.agent.jmx.name.primarykeys 属性中配置的。primarykeys 属性中的值指定 MBeans JMX ObjectName 中能够唯一标识 MBean 的部分。例如，WebLogic MBean 的 ObjectName 包含下列键：

- 指定 MBean 类型的类型键。
- 指定 MBean 所代表资源的名称的名称键。

ObjectName 中的键/值对可能会因 MBean 的类型而异。

CA Introscope® 可以根据以下规则转换并显示由 introscope.agent.jmx.name.primarykeys 属性值标识的 MBean：

- 仅显示键值信息，不显示键名称。
- 值按照 primarykeys 属性中定义的顺序排序。
- 这些值区分大小写。

例如，假定一个具有以下特征的 WebLogic MBean：

- 域名：WebLogic
- MBean ObjectName 键/值对：category=server、type=jdbc
- 度量标准名称：connections

如果配置：

introscope.agent.jmx.name.primarykeys=type,category，则 connections 属性将按以下结构显示在调查器树中：
<Introscope 域>|<主机>|<进程>|<代理>|JMX|Weblogic|jdbc|服务器:连接

注意：WebLogic 9.0 不具有通用主键。它使用在默认转换方法中找到的键/值对度量标准命名约定。这样，WebLogic 9.0 的 JMX 度量标准树将具有不同的结构。

详细信息：

默认的 JMX 度量标准转换过程 (p. 182)

使用 JMX 筛选管理度量标准量

定义 JMX 筛选可确定将在 CA Introscope® 中收集和显示哪些 JMX MBean 信息。如果未设置筛选，代理会将*所有* JMX MBean 信息报告给企业管理器，这样会增加系统开销。

筛选器是在代理配置文件 IntroscopeAgent.profile 的 introscope.agent.jmx.name.filter 属性中设置的。筛选器是关键字，以逗号分隔的字符串的形式输入属性中。CA Introscope® 支持包含星号 (*) 和问号 (?) 通配符的筛选字符串。

CA Introscope® 会将筛选字符串与 JMX 生成的度量标准进行匹配。如果找到匹配结果，匹配的度量标准将报告给 CA Introscope®。

要限制返回的度量标准的量，请在定义时尽可能缩小筛选字符串的范围。例如，定义匹配 MBean 属性并存在于多个 MBean 上的筛选字符串。报告来自*每个* MBean 的度量标准。如果您只对所选 MBean 的某个属性感兴趣，可以在筛选字符串中使用 MBean 名称来限定属性名称。

例如，假设您希望捕获 JMSDestinationRuntime MBean 的 MessagesCurrentCount 属性值。

如果 MessagesCurrentCount 的完全限定度量标准名称为：
SuperDomain|host-name|Process|Agent-name|JMX|comp-1|
JMSDestinationRuntime|comp-2:MessagesCurrentCount

在 IntroscopeAgent.profile 中将 introscope.agent.jmx.name.filter 定义为：
JMX|comp-1|JMSDestinationRuntime|comp-2:MessagesCurrentCount

用于 WebLogic 的 JMX 筛选器

在 WebLogic 的 *IntroscopeAgent.profile* 文件中，已定义以下关键字：

- ActiveConnectionsCurrentCount
- WaitingForConnectionCurrentCount
- PendingRequestCurrentCount
- ExecuteThreadCurrentIdleCount
- OpenSessionsCurrentCount

将 WebSphere 和 WebLogic 配置为使用 JMX 报告

如何配置 CA Introscope® 以支持 JMX 取决于您使用的应用程序服务器。此信息介绍了如何配置 CA Introscope® 以收集并显示来自 WebLogic Server 和 WebSphere 的 JMX 数据。

请执行以下步骤：

1. 关闭托管应用程序。
2. 仅对于 WebSphere 代理，在 IntroscopeAgent.profile 中将 introscope.agent.jmx.enable 设置为 true。

   注意：在 WebSphere 代理配置文件中，默认值为 false。
3. 在 IntroscopeAgent.profile 中，配置主键。
   - 对于 WebLogic 9.0，注释掉：
     `introscope.agent.jmx.name.primarykeys`
   - 对于其他 WebLogic 版本，取消注释：
     `introscope.agent.jmx.name.primarykeys`

   注意：如果修改属性的值，则值必须区分大小写，并且必须使用逗号分隔多个键。
4. 在 IntroscopeAgent.profile 中，确认已取消注释 introscope.agent.jmx.name.filter 属性。

   注意：筛选必须包括 MBean 属性，如版本号。例如，给定以下完整度量标准名称：

   ```
   *SuperDomain*|MyServer01|WebSphere|LODVMAPM032Node02Cell/server1|JMX|WebSphere|cell=LODVMAPM032Node02Cell|mbeanIdentifier=default_host/hello|name=hello-2_1_1_2_war#hello-2.1.1.2.war|node=LODVMAPM032Node02|platform=dynamicproxy|process=server1|spec=1.0|type=SessionManager|version=6.1.0.0|StatsImpl|LiveCount:HighWaterMark
   ```

包含版本号属性的筛选版本将是：

```
PlantsByWebSphere|name=PlantsByWebSphere#PlantsByWebSphere.war|node=LODVMAPM033Node01|platform=dynamicproxy|process=server1|spec=1.0|type=SessionManager|version=6.1.0.0
```

或者更简单点说：

```
Plant*re|*version*
```

5. 在属性中输入需要的字符串，并以逗号分隔。

   为了使 CA Introscope® 能够匹配筛选的字符串，字符串必须拼写准确并区分大小写。

6. 保存更改。
7. 启动托管应用程序。
8. 通过在 WebLogic Server 中配置一个 CA Introscope® 启动类，或在 WebSphere 中配置一个自定义服务，可以启用 JMX 数据。

详细信息：

为 WebLogic 创建启动类 (p. 38)

启用 JSR-77 数据并查看 WAS 上的 JMX 度量标准

您可以在 WebSphere 中配置 Introscope 以收集、保留和报告 JSR-77 JMX MBean 对象的度量标准。JSR-77（J2EE 管理规范）概括了 J2EE 体系结构的可管理部分，并定义了用于访问管理信息的接口。

注意：JSR-77 支持需要 JVM 1.4 或更高版本。如果 JVM 版本为 1.4，应用程序服务器还必须支持 JSR-77。

请执行以下步骤：

1. 关闭托管应用程序。
2. 在 WebSphere 中配置自定义服务 (see page 47)。
3. 在 IntroscopeAgent.profile 中，确认以下属性为 true：

   ```
   introscope.agent.jmx.enable=true
   ```

4. 在 IntroscopeAgent.profile 中，通过设置以下属性启用 JSR-77：

   ```
   introscope.agent.jmx.name.jsr77.disable=false
   ```

5. 在 IntroscopeAgent.profile 中取消注释以下属性，配置度量标准转换的主键方法：

   ```
   introscope.agent.jmx.name.primaryKeys=J2EEServer,Application,
   j2eeType,JDBCProvider,name,mbeanIdentifier
   ```

   注意：只有随针对 WebSphere 的 Introscope 提供的 IntroscopeAgent.profile 包含此属性定义。

6. 取消注释并设置以下属性，以标识希望 JSR-77 度量标准报告的度量标准：

   ```
   introscope.agent.jmx.name.filter
   ```

   尽管并不要求进行筛选，但强烈建议您这样做。

7. 取消注释以下属性并根据需要进行更新，以指定要在 JSR-77 度量标准中排除的 Mbean 属性：

   ```
   introscope.agent.jmx.ignore.attributes=server
   ```

注意：有关详细信息，请参阅 http://ca.com/support 上的知识库文章 TEC534087：《WebSphere Performance Viewer Data vs. Introscope PMI Data》（WebSphere 性能查看器数据与 Introscope PMI 数据）。

详细信息：

使用主键转换简化 JMX 度量标准 (p. 183)
使用 JMX 筛选管理度量标准量 (p. 184)

第 13 章： 配置平台监控

本节包含配置 Introscope 平台监视器的说明。

此部分包含以下主题：

平台监视器

利用平台监视器，Java 代理可以将系统度量标准（包括 CPU 统计信息）报告给企业管理器。平台监视器包含在 Introscope 代理的安装程序中。

考虑以下信息：

- 除 Windows Server 2003 和 AIX 之外的所有操作系统中的平台监视器在安装 Java 代理后都会自动启用。Windows Server 2003 和 AIX 平台监视器只需进行很少配置，即可投入使用。
- 平台监视器的二进制文件独立于应用程序服务器和操作系统位模式。此外，平台监视器的二进制文件完全依赖于 JVM 体系结构。
- 存在多个进程时，HP-UX 的 Java 代理平台监视器会报告 CPU 的使用率。例如，如果存在 4 个进程，则 CPU 的最大使用率为 400%（4 个进程使用 100% 的 CPU）。如果某个进程占用 110%，则它正在使用 1.1 个 CPU。

注意：有关系统要求，请参阅 *《兼容性指南》*。

Java 代理可生成以下平台度量标准：

- **进程 ID**
- **处理器计数**—表示 CPU 的数目
- **使用率百分比 (进程)**—对于 Java 代理进程，表示此进程使用的容量占所有处理器总容量的百分比。不论存在多少个处理器，此度量标准仅生成一个数字。
- **使用率百分比 (聚合)**—对于此处理器，表示系统中所有进程的总使用率（以百分比表示）。在调查器树中，每个处理器都显示为一个资源。

在 Windows Server 2003 上启用平台监视器

要在 Windows Server 2003 上运行平台监视器，必须具有 *admin* 权限。

请注意，要在 Windows 上运行平台监控，必须启用系统对象。

确定是否已启用系统对象

1. 转到“开始”>“运行”。
2. 键入 *perfmon*，然后单击“运行”。
3. 在对话框中，单击“添加”。
4. 在“添加”对话框中，如果“进程”和“处理器”性能对象位于下拉列表框中，则已启用系统对象。

启用系统对象

1. 转到“开始”>“附件”>右键单击命令提示符 >“运行身份”>“Administrator”。
2. 运行命令：*lodctr /r*

   将启用“进程”和“处理器”对象。

在 AIX 上启用平台监视器

您可以在 AIX 上启用平台监视器。

请执行以下步骤：

1. 安装 Java 代理之后，请验证以下文件是否安装在 wily/core/ext 目录中：
   - introscopeAIXPSeries32Stats.jar
   - introscopeAIXPSeries64Stats.jar
   - libIntroscopeAIXPSeries32Stats.so
   - libIntroscopeAIXPSeries64Stats.so
2. 安装 Perfstat 库。从 IBM FTP 站点安装以下包：
   - bos.perf.libperfstat
   - bos.perf.perfstat
3. 重新启动计算机。

   修补程序将生效，并且平台监视器将处于启用状态。

禁用平台监视器

可以通过将相应的 .jar 文件移至不同的目录来禁用平台监视器。

请执行以下步骤：

1. 转到 /wily/core/ext 目录。
2. 选择与您的平台对应的 introscope<*platform*>.jar 文件，例如 introscopeSolarisAmd32Stats.jar。
3. 将 .jar 文件从 /wily/core/ext 目录移至其他目录。

配置在 HP-UX 上访问平台监视器的权限

使用 .zip 或 .tar 安装程序在 HP-UX 上安装平台监控器需要管理员将下列文件的读写权限更改为 777：

```
wily/ext/introscopeHpuxItaniumStats.jar
wily/ext/introscopeHpuxItaniumStats.so
wily/ext/introscopeHpuxParisc32Stats.jar
wily/ext/introscopeHpuxParisc32Stats.so
```

示例（root 用户）：

```
chmod 777 /<Agent_Home>/wily/ext/introscopeHpuxItaniumStats.jar
```

请在代理安装之后、启动之前完成这些更改。

如果使用的是上面列出的前两个文件，您还必须在启动代理之前安装 gcc。您可以搜索 HP 支持网站以下载 C 和 C++ 的编译器 gcc。

平台监控故障排除

在大多数情况下，平台监视器可以检测操作系统，并在操作系统受支持的情况下运行。在少数情况下，不发生以上检测，此时可以在 Java 代理配置文件中显式指定操作系统以帮助确保平台监视器运行。

请执行以下步骤：

1. 打开 IntroscopeAgent.profile。
2. 在“Platform Monitor Configuration”标题下，找到操作系统的精确匹配值并将该属性取消注释。例如，对于安装有 Sun SPARC 硬件的 Sun Solaris 64 位操作系统，可以查找并取消注释以下值：

   ```
   #introscope.agent.platform.monitor.system=SolarisSparc64
   ```

3. 重新启动托管应用程序。

Windows 上的平台监控故障排除

在 Windows 平台上，Java 代理可能出现以下错误：

```
11/28/06 08:29:55 AM PST [ERROR] [IntroscopeAgent] An error occurred polling for platform data
```

如果该错误不常出现，则可能是由于来自 Windows 本身的暂时性错误导致的，没有危害性。在 Windows 以外的平台上，或始终发生该错误的情况下，此错误表示的状况更严重些，应报告给 Introscope 的 CA Support。

SAP Netweaver

由于缺少 SAP 用户帐户的权限，平台监视器可能无法在 SAP Netweaver 上正常工作。

默认情况下，SAP 用户帐户未在“性能监视器用户”下注册，可通过导航至“计算机管理”>“系统工具”>“本地用户和组”>“组”>“性能监视器用户”找到该选项。已在“性能监视器用户”下注册的用户有权访问 Perfmon 相关数据 (HKEY_PERFORMANCE_DATA)。

如果您的 SAP Netweaver 和 Introscope 性能监视器遇到问题，可通过将 SAP 用户帐户添加到“性能监视器用户”组然后重新启动 Windows 计算机解决此问题。

第 14 章： 将 CA APM 与 CA LISA 集成

此部分包含以下主题：

如何将 CA APM 与 CA LISA 集成

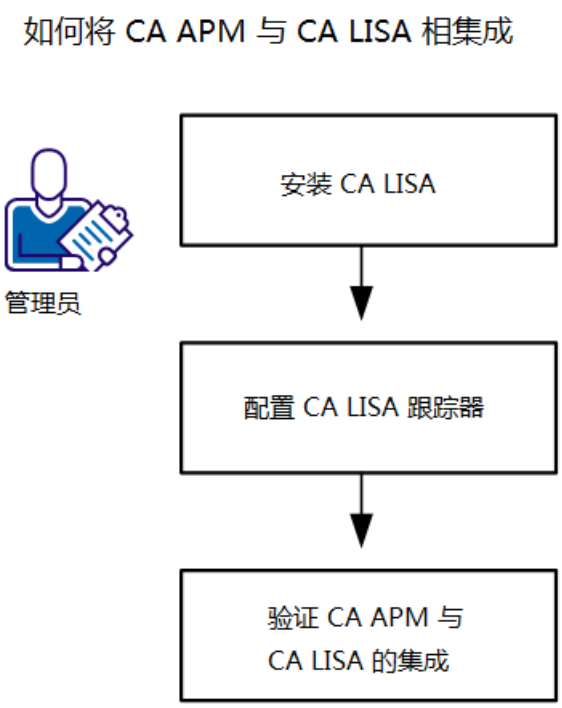
通过在预生产环境中从 CA LISA 执行负载测试和回归测试而生成综合事务，可以将 CA LISA 与 CA APM 集成以监控、检测、分类及诊断应用程序性能问题。

综合事务表示实际事务性能，可以用于：

- 在将应用程序发布到生产环境中之前，监控测试环境中的数据。监控综合数据可以确保应用程序在所定义的限制之内执行，并确认预期的服务级别协议、资源消耗和基准性能特征。
- 在软件开发生命周期的早期阶段检测并确定应用程序性能问题或瓶颈的根本原因，以便客户可以向生产环境提供更高质量的应用程序。
- 创建预生产环境中应用程序性能和资源使用级别的报告，作为生产容量规划的参考。

- 确保收集的度量标准可以呈现所监控的生产应用程序的重要方面。

下图说明了为将 CA LISA 与 CA APM 集成而以管理员身份执行的任务：



1. 安装 CA LISA (see page 197)
2. 配置 CA LISA 跟踪器 (see page 198)。
3. 验证 CA APM 和 CA LISA 集成 (see page 198)。

安装 CA LISA

您可以使用以下方法安装 CA LISA：

- 手工下载并解压缩 CA LISA 文件 (see page 195)
- 使用交互式安装程序安装 CA LISA (see page 197)

手工下载并解压缩 CA LISA 文件

下载并解压缩 CA LISA 文件，以便与 CA APM 进行集成。

重要信息！ 在安装 CA LISA 集成之前确认已安装 JRE。

请执行以下步骤：

1. 从 CA Support 上的 CA APM 软件下载区域下载 CALISAIntegrationNoInstaller<*版本号*>.windows.zip 或 CALISAIntegrationNoInstaller<*版本号*>.unix.tar。
2. 将此分发文件内容解压缩到不包含其他代理分发内容的任何目录。

   这些文件将被解压缩到指定的目录中。

检测 CA APM 以查看 CA LISA 中的测试事件度量标准和数据

针对以下 CA LISA 进程检测 CA APM 以查看 CA LISA 中的测试事件度量标准和数据：

- CA LISA Coordinator
- CA LISA Test Runner
- CA LISA Workstation

您可以检测其他 CA LISA 进程。但是，会尽可能少地报告与 CPU、内存使用情况以及代理和 JVM 身份相关的度量标准。

请执行以下步骤：

1. 使用与 Windows 或 Linux 进程可用的可执行文件或 shell 脚本文件相同的名称创建相应的 .vmoptions 文件，并将其保存到 *<LISA 主目录>*\bin 目录。例如，要检测 CA LISA Workstation，请在 *<LISA 主目录>*\bin 目录中创建 LISAWorkstation.vmoptions 文件。

   以下列表显示了可用的可执行文件或 shell 脚本文件名以及可以检测的进程：

   - Coordinator 进程－为 CoordinatorServer.exe 或 CoordinatorServer.sh 创建一个相应的扩展名为 .vmoptions 的文件。
   - Registry 进程－为 Registry.exe、Registry.sh、TestRegistry.exe 或 TestRegistry.sh 文件创建一个相应的扩展名为 .vmoptions 的文件。
   - Simulator 进程－为 Detached_Simulator.exe、Detached_Simulator.sh、Simulator.exe 或 Simulator.sh 文件创建一个相应的扩展名为 .vmoptions 的文件。
   - TestRunner 进程－为 TestRunner.exe 或 TestRunner.sh 文件创建一个相应的扩展名为 .vmoptions 的文件。

- Virtual Service Environment 进程－为 VirtualServiceEnvironmentService.exe 或 VirtualServiceEnvironmentService.sh 文件创建一个相应的扩展名为 .vmoptions 的文件。
- Workstation 进程－为 LISAWorkstation.exe 或 LISAWorkstation.sh 文件创建一个相应的扩展名为 .vmoptions 的文件。

2. （可选）使用与 Windows 服务可用的可执行文件或 shell 脚本文件相同的名称创建相应的 .vmoptions 文件，并将其保存到 *<LISA 主目录>*\bin 目录。

   下表显示了可执行文件名称以及可以检测的服务：

   - Coordinator 服务－为 CoordinatorService.exe 创建一个相应的扩展名为 .vmoptions 的文件。
   - Simulator 服务－为 SimulatorService.exe 文件创建一个相应的扩展名为 .vmoptions 的文件。
   - Registry 服务－为 TestRegistryService.exe 文件创建一个相应的扩展名为 .vmoptions 的文件。
   - Virtual Service Environment 服务－为 extensionVirtualServiceEnvironmentService.exe 文件创建一个相应的扩展名为 .vmoptions 的文件。

3. 通过添加一行以定义 Java com.wily.introscope.agent.agentName 系统属性来更改代理节点名称。例如，要为节点 CA LISA Workstation Agent 命名，请添加以下行：

   ```
   -Dcom.wily.introscope.agent.agentName=CA LISA Workstation Agent
   ```

4. 向 .vmoptions 文件中添加以下行：

   ```
   -javaagent:<代理主目录>/Agent.jar
   -Dcom.wily.introscope.agentProfile=<Agent_HOME>/core/config/IntroscopeAgent.profile
   ```

   其中，*<代理主目录>* 是 CA LISA 专用代理的安装路径。通常，*<代理主目录>* 路径应为绝对路径，但它可以是相对于运行 CA LISA 进程的当前目录的路径。

5. （可选）为要添加的每个 JVM 命令行选项或系统属性分别输入一行。
6. 如果正在检测的 CA LISA 进程在 Java 1.7 下运行，请添加 -XX:-UseSplitVerifier。
7. 重新启动应用程序服务器。

   此时将应用检测设置。

使用安装程序安装 CA LISA

可以使用安装程序将 CA LISA 与 CA APM 集成。

请执行以下步骤：

1. 选择适用于您操作系统的安装程序：
   - CALISAIntegrationInstaller<*版本号*>.unix.tar
   - CALISAIntegrationInstaller<*版本号*>.windows.zip
2. 在 CA LISA 服务器上执行选定安装程序。
   a. 在安装期间，指定 CA LISA 安装的主目录以及要用于安装的目录。
   b. 选择要检测的 CA LISA 进程。
   c. 指定 EM 和端口。

      如检测 CA APM 以查看 CA LISA 中的测试事件度量标准和数据 (see page 195)中所述，安装程序将在 *<LISA 主目录>*\bin 目录中创建所需的 vmoptions 文件。

      注意：如果正在检测的 CA LISA 应用程序在 Java 1.7 下运行，则必须在安装和添加 -XX:-UseSplitVerifier 之后手工编辑 vmoptions 文件。
3. 在 EM 安装期间，选择 CA APM Integration for CA LISA 作为监控选项。

   管理模块和类型视图将显示 CA LISA 数据。

   注意：如果在运行安装程序时没有选择 CA APM Integration for CA LISA 监控选项，可以通过从位于 EM 安装目录中的 <企业管理器主目录>\examples\CAAPMIntegrationForCALISA 文件夹复制文件来手工执行此步骤。

配置 CA LISA 跟踪器

配置 CA LISA 跟踪器以自定义要监控的 CA LISA 系统组件。通过 CA LISA 跟踪器，可以控制向以下节点报告度量标准的最低级别，从而限制向 CA APM 报告的度量标准的数目：

- 测试用例
- 模拟器
- 测试步骤

度量标准的最低报告级别位于“测试用例”节点上。

请执行以下步骤：

1. 转到 *<LISA_Home>*\core\config 并打开 lisa.pbd 配置文件，进行必要的配置以满足您的环境要求，然后保存该文件。
2. 转到 *<LISA_Home>*\core\config 并打开 IntroscopeAgent.profile 配置文件，进行必要的配置以满足您的环境要求，然后保存该文件。
3. 重新启动应用程序服务器。

   此时将应用配置设置。

验证 CA APM 和 CA LISA 集成

可以通过调用 CA LISA 进程并验证已检测的节点是否位于“调查器”树中来验证 CA APM 和 CA LISA 集成。

注意：只有当已从 CA LISA Coordinator、CA LISA Test Runner 或 CA LISA Workstation 运行 CA LISA 测试后，使用 CA LISA 测试调用的并显示在“调查器”树中“<CA LISA>｜测试用例”节点下的度量标准才会报告给企业管理器。

请执行以下步骤：

1. 使用 CA LISA Workstation 运行测试。有关使用 CA LISA 的更多信息，请参阅 CA LISA 文档集。
2. 启动企业管理器（如果它尚未运行）。
3. 启动 CA LISA 进程（如果它尚未运行）。

   重新启动 CA LISA 进程将启动代理。
4. 启动 Workstation 并打开调查器。

5. 找到位于以下节点的数据：

   超级域 | *<主机名>* | CA LISA | *<CA LISA Workstation>* | CA *LISA* | 测试用例 | *<测试用例名称>*

   - <Host_Name> 是承载安装 CA LISA 代理的计算机的节点。
   - <CA LISA Workstation> 是 LISAWorkstation.vmoptions 文件中指定的代理名称所在的节点。有关配置此节点的更多信息，请参阅检测 CA APM 以查看 CA LISA 中的测试事件度量标准和数据 (see page 195)。
   - <Test Case Name> 是在验证过程的第一步运行的测试用例的名称。

6. 展开文件夹。

   活动的度量标准将显示在已配置的节点下。

第 15 章： 将 CA APM Cloud Monitor 与 CA APM 相集成

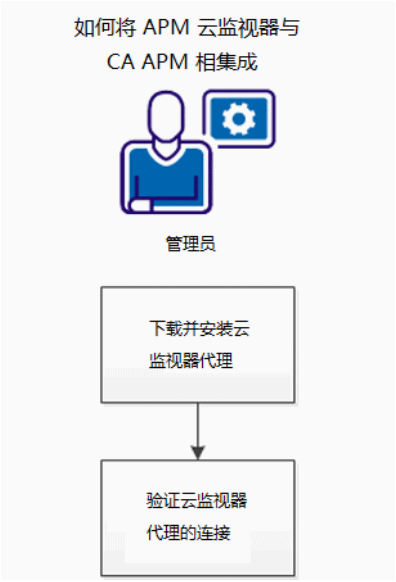
通过 CA APM Cloud Monitor，您可以执行以下任务：

- 了解来自 40 多个国家/地区的 60 多个监控站的完整用户体验。
- 监控实际浏览者以准确衡量用户体验。
- 监控由 SaaS 供应商和 MSP 提供的应用程序使其对 SLA 负责。
- 从防火墙外部测试应用程序响应时间（使用综合事务）来了解全球最终用户经验，并且甚至在没有实际用户流量的时候监控性能。
- 复制实际用户事务以监控整个应用程序基础架构的性能，从而快速地识别、诊断和解决问题。

如何将 CA APM Cloud Monitor 与您的 CA APM 部署相集成

作为 APM 管理员，您可以将 CA APM Cloud Monitor 集成到 CA APM 部署中，以便增加额外的监控和分类能力。

以下流程图显示了如何在 CA APM 内部部署中集成 CA APM Cloud Monitor。



1. 下载并安装 CA APM Cloud Monitor 代理。(see page 202)
2. 验证代理连接。(see page 202)

下载并安装 CA APM Cloud Monitor 代理

在此任务中，您要下载并安装集成 CA APM Cloud Monitor 和 CA APM 所需的 CA APM Cloud Monitor 代理。

注意：您可以在网络中的任何计算机上安装 CA APM Cloud Monitor 代理。其功能是通过 Internet 接收 CA APM Cloud Monitor 数据，并将数据转发给企业管理器，因此它可以是任何计算机（虽然您应选择具有服务器级别 CPU/RAM 的计算机）。

下载并安装 CA APM Cloud Monitor 代理：

1. 下载 CA APM Cloud Monitor 代理存档文件。将文件保存到您的计算机。

   Windows、Linux 和 Solaris 的安装程序位于 CA APM 软件下载网站上。

2. 安装代理。
   a. 启动 CA APM Cloud Monitor 安装程序文件。
   b. 接受许可条款。
   c. 指定 <*代理主目录*> 目录。
   d. 指定代理将连接到的企业管理器主机和端口。
   e. 在 Cloud Monitor 帐户设置屏幕中，输入您的 CA APM Cloud Monitor 用户 ID 和密码。这与用来登录 CA APM Cloud Monitor 网站的凭据相同。

CA APM Cloud Monitor 代理安装完成。

注意：有关启动代理且验证连接方面的说明，请参阅“Validate the CA APM Cloud Monitor Agent Connection (see page 202)”。

验证 CA APM Cloud Monitor 代理连接

要验证是否已正确安装 CA APM Cloud Monitor 代理，请启动代理，并使用 CA APM WebView 或 Workstation 检查数据。通过确认来自 CA APM Cloud Monitor 的传入数据是当前的来验证连接。

注意：要在 WebView 或 Workstation 中查看 CA APM Cloud Monitor 数据，必须在 CA APM Cloud Monitor 中设置文件夹和监视器。如果尚未执行此步骤，请参阅 *《CA APM Workstation 用户指南》* 中与使用 CA APM Cloud Monitor 相关的部分。

请执行以下步骤：

1. 导航到安装 CA APM Cloud Monitor 代理的目录，然后运行以下文件之一以启动代理。
   - 在 Windows 中，双击 APMCloudMonitor.bat。
   - 在 Linux 或 Solaris 上，运行 APMCloudMonitor.sh。

   代理在一个控制台窗口中启动。
2. 启动 CA APM WebView 或 Workstation。
3. 在 CA APM Cloud Monitor 中找到数据。
   a. 如果未打开“调查器”窗口，请选择“文件”>“新建调查器”。
   b. 浏览至“度量标准浏览器”选项卡。
   c. 展开以下节点：

      SuperDomain | *<Host_Name>* | APMCloudMonitor | APMCloudMonitorAgent | APM Cloud Monitor

      <Host_Name> 通常为安装 CA APM Cloud Monitor 代理的计算机，但在“度量标准浏览器”树中会显示 APMCloudMonitor.properties 文件中的属性 apmcm.agent.hostName 的值。
   d. 展开文件夹以查看各个度量标准，并确认这些度量标准是最新的。

故障排除

如果您无法在企业管理器安装过程期间选择“Cloud Monitor”作为监控选项，WebView/Workstation 不会允许全面显示 CA APM Cloud Monitor 数据。

改正此问题：

- 将 <*企业管理器主目录*>\examples\CAAPMIntegrationForCloudMonitor\ 目录的内容复制到 <企业管理器主目录>/config/modules/ 和 <企业管理器主目录>/ext/xmltv。

  这样，系统即可准备好管理模块和其他文件，以便正确地显示 Cloud Monitor 度量标准数据。

如何限制数据

为了改善性能，您可以限制 CA APM Cloud Monitor 代理向企业管理器发送的数据量。

通过配置 CA APM Cloud Monitor 属性来限制数据

您可以使用文件 APMCloudMonitor.properties 中的属性来筛选 CA APM Cloud Monitor 代理向企业管理器发送的数据。

要通过配置 CA APM Cloud Monitor 代理属性来筛选数据，请编辑 <*CloudMonitor 代理主目录*>/CloudMon/conf/APMCloudMonitor.properties 中的度量标准筛选部分。

有关这些设置的信息，请参阅“APMCloudMonitor.properties”的属性参考。

通过删除检查点来限制数据

CA APM Cloud Monitor 具有对五大洲内超过六十个检查点工作站的访问权限。它从这些工作站中进行随机选择，并检查工作站到您的网站或应用程序的可用性和性能。随着时间的推移，所有启用的工作站都将执行此检查，从而导致记录来自超过六十个站点的数据。

您可以删除一些可用的检查点工作站，以限制 CA APM Cloud Monitor 发送到 CA APM 的数据量。

请执行以下步骤：

1. 登录到 CA APM Cloud Monitor 网站：cloudmonitor.ca.com。
2. 选择“订阅”>“首选项”。

   默认情况下，会选择所有工作站。
3. 更改默认选择：
   - 清除各个工作站的复选框，*或执行以下操作：*
   - 单击“清除”全部清除，然后从列表顶部的组中进行选择。

     例如，要仅选择北美的工作站：

     a. 单击“清除”。

     b. 单击“北美”。
4. 在页面的底部，单击“更改”。

通过调整排定来限制数据

默认情况下，监视器定期（每天每小时）检查可用性以及性能。随着时间的推移，这可能会导致向 CA APM 返回的数据比您希望的要多。

请执行以下步骤：

1. 登录到 Cloud Monitor 网站：cloudmonitor.ca.com。
2. 选择“设置”>“监视器”。
3. 选择单个监视器，然后选择“更多选项”。
4. 重置以下设置：
   - 检查之间的延迟
   - 检查周期
   - 仅在这些日子检查
   - 维护排定

附录 A： Java 代理属性

配置 IntroscopeAgent.profile 位置

代理引用 IntroscopeAgent.profile 中针对其基本连接的属性和命名属性。安装代理时，代理配置文件安装在 *<Agent_Home>*/wily/core/config 目录中。

Introscope 按以下顺序在这些位置中查找代理配置文件：

- 在系统属性 com.wily.introscope.agentProfile 中定义的位置
- 在 com.wily.introscope.agentResource 中定义的位置
- *<working directory>*/wily/core/config 目录

注意：在 Windows 计算机上添加路径时，反斜杠 (\) 必须用另一个反斜杠转义，如下：C:\\Introscope\\lib\\Agent.jar。

更改 IntroscopeAgent.profile 的位置

1. 使用以下方法之一定义新位置：
   - 使用 **-D** 选项在 Java 命令行定义一个系统属性，以指定 IntroscopeAgent.profile 文件位置的完整路径：
   - -D com.wily.introscope.agentProfile。
   - 使 IntroscopeAgent.profile 在类路径上的资源中可用。设置 *com.wily.introscope.agentResource* 以指定包含代理配置文件的资源的路径。

     注意：如果更改 IntroscopeAgent.profile 的位置，也必须更改 AutoProbe 日志位置。有关详细信息，请参阅管理 ProbeBuilder 日志 (see page 135)。
2. 将 ProbeBuilder 指令（PBD 和 PBL 文件）移至与代理配置文件相同的位置－对这些指令位置的引用相对于配置文件位置。

如果使用 Sun ONE，则将代理配置文件的新位置添加到 Sun ONE *server.xml* 文件

更改 Sun ONE 的 IntroscopeAgent.profile 的位置

1. 要将 Introscope 信息添加到 Sun ONE 7.0 的启动脚本中，请以管理员或 Root 用户的身份登录。
2. 打开 *<SunOne_Home>*/domains/domain1/server1/config/ 中的 server.xml 文件
3. 向 server.xml 中的 jvm-options 节添加一行：

```
<jvm-options>
-Dcom.wily.introscope.agentProfile=SunOneHome/wily/core/config/IntroscopeAgent.profile
</jvm-options>
```

命令行属性覆盖

在 Introscope 中，可使用命令行覆盖企业管理器、代理、Workstation 和 WebView 的特定属性。对于 Java 代理，如果群集环境中具有共享代理的多个副本，并且您要定制被监控的每个应用程序的某些代理设置，则此方法非常有用。

这些步骤假设您已在要监控的应用程序服务器上安装并配置代理，且该代理成功连接到企业管理器。

使用命令行覆盖代理属性

1. 打开在其中修改了 Java 命令以启动代理的文件。

   此文件的位置因在环境中使用的应用程序服务器而异。
2. 添加 *-D* 命令以覆盖属性。例如，可添加以下命令以使代理也使用 *weblogic-full.pbl* 文件：

   ```
   -Dintroscope.autoprobe.directivesFile=weblogic-full.pbl
   ```

   将此命令置于打开的文件中其他 *-D* 命令的旁边。

   注意：当您使用此命令覆盖可进行热部署的属性时，将无法再对该属性进行热部署。此外，如果稍后在配置文件中修改属性，您将在 Workstation 中收到一条警告消息，指示您修改了覆盖属性，更改将无效。为避免此状况，请先删除覆盖命令，然后在配置文件中修改属性。

3. 保存该文件。
4. 重新启动代理。

   在上面使用的示例中，您将可以在 Workstation 的代理节点中看到其他 WebLogic 度量标准。

   重要信息！ 系统属性成为 Introscope 属性的属性空间的一部分，使 *java.io.tmpdir* 等对使用 IndexedProperties 的任何项目均可见。

代理故障转移

如果 Java 代理断开与其主企业管理器的连接，“代理故障转移”属性可指定代理将故障转移到哪个企业管理器，以及尝试重新连接到其主企业管理器的频率。

introscope.agent.enterprisemanager.connectionorder

指定代理断开与默认企业管理器的连接时使用的备份企业管理器连接顺序。

属性设置

代理可以连接到的其他企业管理器的名称。

默认

企业管理器由默认主机名、端口号和套接字工厂属性定义。

示例

```
introscope.agent.enterprisemanager.connectionorder=DEFAULT
```

注释

- 使用逗号分隔列表。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds

指定被拒绝的代理尝试重新连接到以下企业管理器的间隔秒数：

- 基于在代理配置文件 introscope.agent.enterprisemanager.connectionorder 属性值中配置的顺序的企业管理器。
- 根据 loadbalancing.xml 配置允许连接的任何企业管理器。

如果代理无法连接到企业管理器，可以按下列方式处理连接：

- 尝试连接到下一个允许的企业管理器。
- 不连接到任何不允许连接的企业管理器。

注意：有关配置 loadbalancing.xml 和代理与企业管理器的连接的信息，请参阅《*CA APM 配置和管理指南*》。

默认

默认时间间隔是 120 秒。

示例

```
introscope.agent.enterprisemanager.failbackRetryIntervalInSeconds=120
```

注释

必须重新启动托管应用程序，对此属性所做的更改才能生效。

默认情况下，此属性注释掉。

该属性在允许代理跨以下 CA APM 组件进行连接的环境中非常有用：

- 群集。
- 收集器和独立企业管理器。
- 群集、收集器和独立企业管理器的任意组合。

如果代理可以连接到不同群集中的企业管理器且未设置该属性，则以下示例显示了会发生的状况：

1. 连接到群集中企业管理器的代理断开连接。
2. 代理以不允许的模式连接到群集 2 中的企业管理器。
3. 代理不知道群集 1 中允许的企业管理器何时变得可用。

该属性强制代理持续尝试连接到允许的企业管理器，直到某个企业管理器可供连接。

代理 HTTP 隧道

可以将代理配置为使用隧道技术发送信息，从而使代理可远程连接到企业管理器。为此，必须将代理配置为连接到企业管理器的嵌入式 Web 服务器，该服务器上承载了 HTTP 隧道 Web 服务。

要在 *IntroscopeAgent.profile* 中将 HTTP 隧道通信配置为新代理连接，请指定：

- 运行企业管理器的计算机主机名。有关详细信息，请参阅 introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 213)。
- 与企业管理器 Web 服务器的连接端口。这是在代理将连接到的企业管理器的 *IntroscopeEnterpriseManager.properties* 中指定的 *introscope.enterprisemanager.webserver.port* 属性值。
- HTTP 隧道套接字工厂。指定以下客户端套接字工厂：
  `com.wily.isengard.postofficehub.link.net.HttpTunnelingSocketFactory`

代理服务器的代理 HTTP 隧道

以下属性仅适用于配置为通过 HTTP 进行隧道处理，且必须使用代理服务器连接到企业管理器的代理：

- introscope.agent.enterprisemanager.transport.http.proxy.host (see page 212)
- introscope.agent.enterprisemanager.transport.http.proxy.port (see page 212)
- introscope.agent.enterprisemanager.transport.http.proxy.username (see page 212)
- introscope.agent.enterprisemanager.transport.http.proxy.password (see page 213)

有关详细信息，请参阅为 HTTP 隧道配置代理服务器 (see page 57)。

introscope.agent.enterprisemanager.transport.http.proxy.host

指定代理服务器主机名。

默认

已注释掉；未指定。

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.http.proxy.port

指定代理服务器端口号。

默认

已注释掉；未指定。

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.http.proxy.username

如果代理服务器需要代理对其进行身份验证，请指定用于身份验证的用户名。

默认

已注释掉；未指定。

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.http.proxy.password

如果代理服务器需要代理对其进行身份验证，请指定用于身份验证的密码。

默认

已注释掉；未指定。

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

代理 HTTPS 隧道

可以将代理配置为使用 HTTPS 发送信息，从而使代理可远程连接到企业管理器：

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 213)
- introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT (see page 214)
- introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT (see page 214)

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

指定运行代理默认与之连接的企业管理器的计算机主机名。

默认

localhost

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

指定托管企业管理器的计算机上的端口号。如果使用 HTTPS 隧道，侦听来自代理的连接的默认端口为 8444。默认情况下，此属性注释掉。

默认

8444

示例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=8444
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

指定客户端套接字工厂，以便在使用 HTTPS 时用于从代理到企业管理器的连接。

默认

com.wily.isengard.postofficehub.link.net.HttpsTunnelingSocketFactory

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

代理内存开销

只有在某些极端情况下代理内存开销才会明显。降低内存消耗的典型代价是可能会延长响应时间。但是，每个应用程序都是唯一的，内存使用情况和响应时间之间的权衡可能会因应用程序本身而异。

introscope.agent.reduceAgentMemoryOverhead

此属性可指定要使用的代理配置。如果想要尝试减少代理内存开销，请取消注释该属性。默认情况下，此属性设置为 true，且已注释掉。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.reduceAgentMemoryOverhead=true
```

注释

必须重新启动托管应用程序，以便对此属性所做的更改生效。

代理度量标准老化

代理度量标准老化功能可定期从代理内存缓存中删除死度量标准。死度量标准是指在所配置的一段时间内没有报告新数据的度量标准。删除旧的度量标准有助于提高代理性能，避免潜在的度量标准爆发。

注意：如果无意中将代理设置为报告的度量标准多于系统可以处理的度量标准，将发生度量标准爆发。如果报告了太多度量标准，代理会影响应用程序服务器的性能，在极个别情况下，还会使服务器完全无法运行。

仅当组中的所有度量标准均被视为删除候选项时，才会删除组中的度量标准。目前，只有 *BlamePointTracer* 和 *MetricRecordingAdministrator* 度量标准可按组删除。其他度量标准均单独删除。

MetricRecordingAdministrator 具有以下用于创建、检索或删除度量标准组的接口：

- *getAgent().IAgent_getMetricRecordingAdministrator.addMetricGroup*

  字符串组件，收集度量标准。组件名称是度量标准组的度量标准资源名称。度量标准必须位于同一个度量标准节点下，才能称之为一个组。度量标准是 *com.wily.introscope.spec.metric.AgentMetric* 数据结构的集合。您只能将 *AgentMetric* 数据结构添加到此集合中。

- *getAgent().IAgent_getMetricRecordingAdministrator.getMetricGroup*

  字符串组件。根据组件名称（即度量标准资源名称），您可以获得度量标准集合。

- *getAgent().IAgent_getMetricRecordingAdministrator.removeMetricGroup*

  字符串组件。度量标准组基于组建名称（即度量标准资源名称）进行删除。

- *getAgent().IAgent_getDataAccumulatorFactory.isRemoved*

  检查度量标准是否已删除。如果在扩展中保留一个累积器实例，则使用此接口。如果累积器由于度量标准老化而被删除，可使用此接口防止保留死引用。

重要信息！如果要创建使用 *MetricRecordingAdministrator* 界面的扩展（例如，用于与其他 CA Technologies 产品配合使用），请确保删除自己的累积器实例。如果度量标准因未被调用而过期，并且稍后数据可用于该度量标准，则旧的累积器实例将不会创建新的度量标准数据点。要避免这种情况，请不要删除您自己的累积器实例，并改用 *getDataAccumulatorFactory* 接口。

配置代理度量标准老化

默认情况下，代理度量标准老化处于打开状态。您可以选择使用属性 introscope.agent.metricAging.turnOn (see page 217) 关闭此功能。如果从 *IntroscopeAgent.profile* 中删除此属性，默认情况下将关闭代理度量标准老化。

代理度量标准老化将针对代理中的心跳运行。心跳使用属性 introscope.agent.metricAging.heartbeatInterval (see page 217) 进行配置。请务必将心跳的频率维持在较低状态。较高的心跳频率将影响代理和 CA Introscope® 的性能。

在每个心跳期间，会检查一组特定的度量标准。这可通过属性 introscope.agent.metricAging.dataChunk (see page 218) 进行配置。将此值维持在较低状态也很重要，因为较高的值将影响性能。默认值是每个心跳检查 500 个度量标准。将逐个检查这 500 个度量标准以查看是否是删除候选项。例如，如果将此属性设置为每个心跳检查 500 个度量标准区块，并且代理内存中共有 10,000 个度量标准，则它将在对性能造成较小影响的前提下花费更长时间来检查所有 10,000 个度量标准。但是，如果将此属性设置为较高的值，您可以更快地检查这 10,000 个度量标准，但是开销可能会很大。

如果度量标准在特定时间段之后没有收到新数据，该度量标准将成为删除候选项。您可以使用属性 introscope.agent.metricAging.numberTimeslices (see page 218) 来配置此时间段。默认情况下，此属性设置为 3000。如果某个度量标准满足删除以上条件，将执行检查以查看其所在组中的所有度量标准是否是删除候选项。如果也满足此项要求，则删除该度量标准。

introscope.agent.metricAging.turnOn

打开或关闭代理度量标准老化。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.metricAging.turnOn=true
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.metricAging.heartbeatInterval

指定检查待删除度量标准的时间间隔（以秒为单位）。

默认

1800

示例

```
introscope.agent.metricAging.heartbeatInterval=1800
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.metricAging.dataChunk

指定在每个时间间隔内检查的度量标准数。

默认

500

示例

```
introscope.agent.metricAging.dataChunk=500
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.metricAging.numberTimeslices

在成为删除候选项之前，此属性会指定要检查的时间间隔数（没有任何新数据）。

默认

3000

示例

```
introscope.agent.metricAging.numberTimeslices=3000
```

注释

对此属性所做的更改会立即生效，不需要重新启动托管应用程序。

introscope.agent.metricAging.metricExclude.ignore.0

使指定的度量标准不被删除。要防止一个或多个度量标准老化，请将度量标准名称或度量标准筛选添加到列表中。

属性设置

以逗号分隔的度量标准列表。可以在度量标准名称中使用星号 (*) 作为通配符。

默认

默认情况下，度量标准名称以 Threads 开头 (*Threads**)。

示例

```
introscope.agent.metricAging.metricExclude.ignore.0=Threads*
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

代理度量标准限定

您可以配置代理以便大致限定发送给企业管理器的度量标准数目。如果生成的度量标准数目超过该属性值，则代理停止收集和发送新度量标准。

introscope.agent.metricClamp

将代理配置为大致限定发送给企业管理器的度量标准数目。

默认

5000

示例

```
introscope.agent.metricClamp=5000
```

注释

- 如果不设置该属性，则不会发生度量标准限定。旧的度量标准仍将报告值。
- 对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

- 该限定属性与位于 apm-events-thresholds-config.xml 文件中的 introscope.enterprisemanager.agent.metrics.limit 属性一起使用。

  注意：有关 introscope.enterprisemanager.agent.metrics.limit 属性的信息，请参阅《*CA APM 配置和管理指南*》。

  如果 introscope.enterprisemanager.agent.metrics.limit 限定值在 introscope.agent.metricClamp 值之前触发，企业管理器会读取代理度量标准，但是不会在调查器度量标准浏览器树中报告它们。

  如果 introscope.agent.metricClamp 限定值在 introscope.enterprisemanager.agent.metrics.limit 限定值之前触发，代理将停止向企业管理器发送度量标准。

代理命名

您可以配置属性以获取应用程序服务器的 Java 代理名称及更多信息。

详细信息：

了解 Java 代理名称 (p. 117)

introscope.agent.agentAutoNamingEnabled

指定是否使用代理自动命名获取支持的应用程序服务器的 Java 代理名称。

属性设置

True 或 False

默认

因应用程序服务器而异，请参阅下面的“注意”。

示例

```
introscope.agent.agentAutoNamingEnabled=false
```

注释

- 在以下应用程序服务器中**启用**代理自动命名：WebLogic、WebSphere 和 JBoss
- 此属性需要为 WebLogic 指定启动类，并为 WebSphere 指定自定义服务。
- 在以下应用程序服务器中**禁用**代理自动命名：Oracle 应用程序服务器、Interstage、Sun ONE 和 Tomcat。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

重要信息！对于 WebLogic、WebSphere 和 JBoss，默认情况下，属性 *introscope.agent.agentAutoNamingEnabled* 设置为 **TRUE**。

introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds

指定代理在连接到企业管理器之前等待命名信息的时间（秒）。

默认

120

示例

```
introscope.agent.agentAutoNamingMaximumConnectionDelayInSeconds=120
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.agentAutoRenamingIntervalInMinutes

指定时间间隔（分钟），在该时间段内，代理将检查以确认是否已重命名。

默认

10

示例

```
introscope.agent.agentAutoRenamingIntervalInMinutes=10
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.agentName

如果其他代理命名方法失败，请取消注释该属性以提供默认代理名称。

属性设置

对于任何安装，如果该属性的值无效，或者该属性已从配置文件中删除，则代理名称将为 *UnnamedAgent*。

示例

```
#introscope.agent.agentName=AgentName
```

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 在特定于应用程序服务器的代理安装程序所附带的代理配置文件中，默认值可反映应用程序服务器，例如 *WebLogic Agent*。
- 在默认代理安装程序附带的代理配置文件中，属性值为 *AgentName*，并且该行已注释掉。

introscope.agent.agentNameSystemPropertyKey

如果要使用 java 系统属性值指定代理名称，请使用此属性。

默认

未指定。

示例

```
introscope.agent.agentNameSystemPropertyKey
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.disableLogFileAutoNaming

指定使用自动命名选项时是否禁用代理日志文件的自动命名。

将此属性设置为 **true** 会禁用通过代理名称或时间戳对代理、AutoProbe 和 LeakHunter 的日志文件进行自动命名。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.disableLogFileAutoNaming=false
```

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 仅当可以使用 Java 系统属性或应用程序服务器自定义服务确定代理名称时，日志文件自动命名才会生效。

introscope.agent.clonedAgent

使您能够在同一台计算机上运行应用程序的相同副本。如果在同一台计算机上运行应用程序的相同副本，请将该属性设置为 true。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.clonedAgent=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.customProcessName

指定应显示在 Introscope 企业管理器和 Workstation 中的进程名称。

默认

因应用程序服务器而异。

示例

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

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 在特定于应用程序服务器的代理安装程序所附带的代理配置文件中，默认值可反映应用程序服务器，例如“WebLogic”。
- 在默认代理安装程序附带的代理配置文件中，该属性已注释掉。

introscope.agent.defaultProcessName

如果未提供自定义进程名称且代理无法确定主应用程序类的名称，则该值将用作进程名称。

默认

UnknownProcess

示例

```
introscope.agent.defaultProcessName=UnknownProcess
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.display.hostName.as.fqdn

该属性指定是否将代理名称显示为完全限定域名 (fqdn)。要启用完全限定域名，请将该属性值设置为 true。默认情况下，代理显示主机名。

注意：对于 Catalyst 集成，请将该属性设置为 true。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.display.hostName.as.fqdn=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

代理记录（业务记录）

您可以控制代理处理业务事务记录的方式。

注意：有关代理业务记录的详细信息，请参阅*《CA APM 事务定义指南》*。

introscope.agent.bizRecording.enabled

对代理启用或禁用业务事务记录。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.bizRecording.enabled=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

要进一步为代理配置业务事务记录，请参阅应用程序分类视图的其他属性。

详细信息：

应用程序分类视图 (p. 229)
应用程序分类图业务事务 POST 参数 (p. 234)
应用程序分类视图托管套接字配置 (p. 236)

代理线程优先级

以下属性控制代理线程的优先级：

- introscope.agent.thread.all.priority (see page 227)

introscope.agent.thread.all.priority

控制代理线程的优先级。

属性设置

可以按从 1（低）到 10（高）的范围对此进行设置。

默认

已注释掉；5。

示例

```
introscope.agent.thread.all.priority=5
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

代理到企业管理器的连接

您可以控制代理如何连接到企业管理器。

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

指定运行代理默认与之连接的企业管理器的计算机主机名。

默认

localhost

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

指定托管企业管理器的计算机上用于侦听来自代理的连接的端口号。

默认

默认端口取决于要配置的通信通道的类型。对于代理和企业管理器之间的直接通信，默认端口为 5001。

示例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5001
```

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 如果想要通过 HTTPS（基于 SSL 的 HTTP）连接到企业管理器，默认端口为 8444。如果想要通过 SSL 连接到企业管理器，默认端口为 5443。但是，默认情况下，这些默认设置已注释掉。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

指定默认客户端套接字工厂，以用于从代理到企业管理器的连接。

默认

默认客户端套接字工厂取决于要配置的通信通道的类型。对于代理与企业管理器之间的直接通信，默认的套接字工厂如下所示：

com.wily.isengard.postofficehub.link.net.DefaultSocketFactory

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT

指定运行代理默认与之连接的企业管理器的计算机的 IP 地址。

默认

默认情况下，未定义该属性。

示例

```
introscope.agent.enterprisemanager.transport.tcp.local.ipaddress.DEFAULT=<地址>
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT

指定运行代理默认与之连接的企业管理器的计算机本地端口。

默认

默认情况下，未定义该属性。

示例

```
introscope.agent.enterprisemanager.transport.tcp.local.port.DEFAULT=CA Portal
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

应用程序分类视图

您可以配置应用程序分类视图数据。

注意：有关如何使用应用程序分类视图的信息，请参阅 *《CA APM Workstation 用户指南》*。

introscope.agent.appmap.enabled

为应用程序分类视图启用或禁用对受监控代码的跟踪。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.appmap.enabled=true
```

注释

默认情况下处于启用状态。

introscope.agent.appmap.metrics.enabled

为应用程序分类视图节点启用或禁用度量标准跟踪。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.appmap.metrics.enabled=false
```

注释

默认情况下，此属性注释掉。

introscope.agent.appmap.queue.size

设置应用程序分类视图的缓冲区大小。

属性设置

正整数。

默认

1000

示例

```
introscope.agent.appmap.queue.size=1000
```

注释

- 该值必须是正整数。
- 如果该值设置为 0，则缓冲区没有边界。
- 默认情况下，此属性注释掉。

introscope.agent.appmap.queue.period

设置将应用程序分类视图数据发送到企业管理器的频率（以毫秒为单位）。

属性设置

正整数。

默认

1000

示例

```
introscope.agent.appmap.queue.period=1000
```

注释

- 必须是正整数。
- 如果值设置为 0，将使用默认值。
- 默认情况下，此属性注释掉。

introscope.agent.appmap.intermediateNodes.enabled

启用或禁用包括应用程序前端和后端节点之间的中间节点的功能。

属性设置

True 或 False

默认

False

示例

```
#introscope.agent.appmap.intermediateNodes.enabled=true
```

注释

- 对此属性所做的更改可立即生效，不需要重新启动托管应用程序。
- 如果将此属性设置为 true，代理性能将会降低。
- 默认情况下，此属性注释掉。

应用程序分类视图和 Catalyst 集成

您可以为 Catalyst 集成配置应用程序分类视图数据。

注意：有关如何使用应用程序分类视图的信息，请参阅 *《CA APM Workstation 用户指南》*。

配置发送信息的功能

通过此属性，可以启用或禁用代理发送与 Catalyst 集成所需的其他信息的功能。

请执行以下步骤：

1. 在文本编辑器中打开默认的 IntroscopeAgent.profile 文件。

找到
introscope.agent.appmap.catalystIntegration.enabled=<false|true> 行，并如下所示设置值：

true

启用代理发送与 Catalyst 集成所需的其他信息的功能。

false

禁用该配置。

以下示例显示了格式：

```
introscope.agent.appmap.catalystIntegration.enabled=false
```

注意：默认情况下，此属性会注释掉。

2. 保存并关闭文件。

代理将设置为使用该配置。

配置可用网络的列表

introscope.agent.primary.net.interface.name 属性指定代理针对 Catalyst 集成使用的主计算机的主网络接口名称。您可以更改该属性的配置，并且更改将自动应用。

注意：代理日志记录级别设置成调试时，可用于配置的网络接口名称相关信息将显示在日志文件中。另外，您还可以使用 Network Interface 实用程序来确定该属性的主网络接口名称。

请执行以下步骤：

1. 在文本编辑器中打开默认的 IntroscopeAgent.profile 文件。
2. 找到行：introscope.agent.primary.net.interface.name=<false|true>，并指定名称值。

   下面的示例显示了该名称格式：

   ```
   introscope.agent.primary.net.interface.name=eth4
   ```

   注意：未定义默认值。不设置该属性时，代理会将第一个可用网络接口指定为主接口。您可以使用 Network Interface 实用程序来确定用于该属性的名称值。
3. （可选）通过指定子接口编号（从 0 开始）允许多个网络地址。

   以下示例显示了子接口编号格式：

   ```
   introscope.agent.primary.net.interface.name=eth4.1
   ```
4. 保存并关闭文件。

   此时配置文件设置为使用该配置。

详细信息：

使用网络接口实用工具 (p. 339)

应用程序分类图业务事务 POST 参数

您可以通过匹配 POST 参数，配置 Local Product Shorts 以执行更复杂的监控。

introscope.agent.bizdef.matchPost

此属性确定何时匹配 POST 参数。

属性设置

此属性的有效设置包括 *never*、*before* 或 *after*。

- 将属性设置为 **never** 可获取完整的代理功能和更好的性能。此设置使得应用程序可以使用 URL、cookie 或标头参数来识别所有业务事务，但是无法匹配任何通过 POST 参数单独识别的业务事务。
- 将属性设置为 **before** 可获取完整的代理性能。此设置允许应用程序使用 POST 参数来识别部分或全部业务事务，但是从不直接访问 servlet 数据流来获取 HTTP 表单请求。该属性设置为 before 时，部署的新应用程序也必须符合标准 API。

  重要信息！将此属性设置为 *before* 时，可能会对应用程序造成具有潜在危险的影响。在实施之前，请与 CA Technologies 代表一起检查该属性设置。

- 将属性设置为 **after**，可使业务事务与 POST 参数安全匹配，但代理功能将受到限制。将此属性设置为 *after* 时，代理将无法在进程之间映射由 POST 参数识别的业务事务，或为这些业务事务生成全套度量标准。与其他选项相比，此设置还会消耗稍多的 CPU 时间，但是如果需要 POST 参数功能，则该设置被视为是最安全的。它允许应用程序使用 POST 参数来识别部分或全部业务事务，但不保证从不直接访问 servlet 数据流。

示例

```
introscope.agent.bizdef.matchPost=after
```

注释

- *never*—从不试图匹配 POST 参数。这是最快的选项，但可能导致不准确的业务事务组件匹配。
- *before*—在 servlet 执行之前匹配 POST 参数。
- *after*—在 servlet 执行之后匹配 POST 参数模式。跨进程映射和一些度量标准将不可用。这是该参数的默认设置。

已知限制

使用代理记录定义的度量标准显示在调查器的应用程序分类视图中。在配置代理记录时，使用正则表达式时会有一些已知的限制。大部分限制与 POST 参数有关。

已知的限制包括：

- POST 参数值不支持行终止符 (.)。
- 如果 POST 参数定义依赖于业务事务定义，则仅向业务事务组件提供三个度量标准。这些度量标准为：
  - 平均响应时间
  - 每个时间间隔的响应数
  - 每个时间间隔的错误
- 如果 POST 参数定义依赖于业务事务定义，则事务跟踪组件中的业务组件名称将有一个通用名称，而不是业务服务、业务事务和业务事务组件的特定名称。这也适用于业务事务定义，这些定义依赖于不匹配的 POST 参数定义。
- JBoss 和 Tomcat 的某些版本可能会将头键保存为小写值，从而导致 *caseSensitiveName* 属性无法正确处理 *HEADER_TYPE*。

注意：有关代理记录的详细信息，请参阅 *《CA APM 事务定义指南》*。

应用程序分类视图托管套接字配置

通过以下属性，您可以启用或禁用应用程序分类视图中套接字度量标准的外观：

- introscope.agent.sockets.managed.reportToAppmap (see page 236)
- introscope.agent.sockets.managed.reportClassAppEdge (see page 237)
- introscope.agent.sockets.managed.reportMethodAppEdge (see page 237)
- introscope.agent.sockets.managed.reportClassBTEdge (see page 238)
- introscope.agent.sockets.managed.reportMethodBTEdge (see page 238)

有关如何使用应用程序分类视图的详细信息，请参阅*《CA APM Workstation 用户指南》*。

introscope.agent.sockets.managed.reportToAppmap

使托管套接字可以显示在应用程序分类视图中。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.sockets.managed.reportToAppmap=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.sockets.managed.reportClassAppEdge

使托管套接字可以将类级别应用程序边缘报告给应用程序分类视图。

属性设置

True 或 False

默认

False

示例

introscope.agent.sockets.managed.reportClassAppEdge=false

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.sockets.managed.reportMethodAppEdge

使托管套接字可以将方法级别应用程序边缘报告给应用程序分类视图。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.sockets.managed.reportMethodAppEdge=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.sockets.managed.reportClassBTEdge

使托管套接字可以将类级别业务事务边缘报告给应用程序分类视图。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.sockets.managed.reportClassBTEdge=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.sockets.managed.reportMethodBTEdge

使托管套接字可以将方法级别业务事务边缘报告给应用程序分类视图。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.sockets.managed.reportMethodBTEdge=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

AutoProbe

以下属性用于配置 AutoProbe：

- introscope.autoprobe.directivesFile (see page 239)
- introscope.autoprobe.enable (see page 239)

introscope.autoprobe.directivesFile

指定 AutoProbe 的 ProbeBuilder 指令文件。

默认

因安装程序而异。

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 如果此属性包括一个或多个目录并且已启用动态检测，代理将从指定目录加载指令文件，而无需重新启动应用程序。

introscope.autoprobe.enable

此属性可启用或禁用自动将探测器插入字节码。

属性设置

true 或 false

默认

true

示例

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

注意：将此属性设置为 false 将会关闭自动将探测器插入字节码，但不会关闭代理或代理报告。JVM 的重新启动对于任何更改的生效是必需的。

introscope.autoprobe.logfile

Introscope AutoProbe 始终尝试记录其进行的更改。将此属性设置为将日志文件的位置移动到默认位置以外的位置。

属性设置

绝对文件路径，或非绝对路径。非绝对名称会相对于该属性文件的位置进行解析。

默认

../../logs/AutoProbe.log

示例

```
introscope.autoprobe.logfile=../../logs/AutoProbe.log
```

要禁用日志记录，请按以下方式注释掉日志文件：

```
introscope.autoprobe.logfile=logs/AutoProbe.log
```

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

启动类检测管理器

以下属性用于配置启动类检测管理器：

- introscope.bootstrapClassesManager.enabled (see page 240)
- introscope.bootstrapClassesManager.waitAtStartup (see page 241)

启动类检测管理器在代理启动后检测一组类，以便轻松实现 Java NIO 和安全套接字层 (SSL) 的跟踪器，并提高代理性能。可以通过在 *IntroscopeAgent.profile* 中注释掉此属性来禁用它。

introscope.bootstrapClassesManager.enabled

启用或禁用启动管理器。

属性设置

True 或 False

默认

True

示例

```
introscope.bootstrapClassesManager.enabled=true
```

注释

- 此属性仅在运行 Java 1.5 或更高版本的 JVM 上起作用。
- 如果设置为 false，则不检测任何系统类。
- 如果未设置此属性，则默认值为 false。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.bootstrapClassesManager.waitAtStartup

设置代理在启动后等待检测启动类的时间（秒）。

属性设置

时间（秒）

默认

- 与 HP-UX、Interstage、WebLogic 或 WebSphere Application Server 结合使用时为 240 秒。
- 与 JBoss、Oracle、Sun 或 Tomcat 结合使用时为 5 秒。

示例

```
introscope.bootstrapClassesManager.waitAtStartup=5
```

注释

- 此属性仅在运行 Java 1.5 或更高版本的 JVM 上起作用。
- 当此属性处于活动状态时，可以否决指定为已跳过的类。如果正在检测已跳过的类，请联系 CA Technologies 代表或 CA Support。

CA CEM 代理配置文件属性

您可以配置与 CA CEM 相关的 IntroscopeAgent.profile 属性。CA Introscope® 代理配置文件位于 <*代理主目录*>\wily 目录中。

为了使 CA CEM 与 CA Introscope® 能够正常集成，所有与 CA CEM 相关的属性均预配置为必需选项。

introscope.autoprobe.directivesFile

您需要通过 directivesFile 属性配置启用 ServletHeaderDecorator/HTTPHeaderDecorator 和 CEMTracer。

指令文件属性指定了应从何处查找 AutoProbe 的指令文件 (PBD) 或指令列表 (PBL)。

AutoProbe 使用指令来启用您的应用程序，并确定代理将哪些度量标准报告给企业管理器。

设置

根据所安装的代理应用程序服务器，格式为 *<appserver>*-full.pbl 或 *<appserver>*-typical.pbl。

默认

```
default-typical.pbl
```

示例

```
introscope.autoprobe.directivesFile=weblogic-typical.pbl
```

注释

虽然可以直接向此属性列表的末尾添加“ServletHeaderDecorator.pbd”或“httpheaderdecorator.pbd”，但最佳做法是：

1. 找到属性中指定的 PBL 文件（在上面的示例中为 weblogic-typical.pbl）。
2. 在文本编辑器中打开该 PBL 文件。
3. 对于 Java 代理，取消注释以启用 ServletHeaderDecorator.pbd 行。
4. 对于 .NET 代理，取消注释以启用 httpheaderdecorator.pbd 行。
5. 保存对 PBL.profile 文件所做的更改。

introscope.agent.remoteagentconfiguration.allowedFiles

此属性可确定允许从任何计算机远程复制到代理目录的文件。

企业管理器可使用此属性中的文件名来识别要发送给代理的有效 CA CEM 域配置文件。域配置文件包含 CA CEM 业务服务和事务定义。

设置

使用有效的文件名。

默认

domainconfig.xml

示例

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

注释

此属性也适用于 Introscope 命令行 Workstation (CLW)“发送配置文件”命令。

有关详细信息，请参阅使用命令行 Workstation。

此属性适用于 CA CEM 版本。

introscope.agent.remoteagentconfiguration.enabled

如果此布尔值设置为 true，则允许将远程文件从其他计算机复制到代理。

企业管理器要求将此属性设置为 true，以便将 CA CEM 域配置文件发送到代理。域配置文件包含 CA CEM 业务服务和事务定义。

设置

true 或 false

默认

- 对于 Java 代理，设置为 true
- 对于 .NET 代理，设置为 false

示例

```
introscope.agent.remoteagentconfiguration.enabled=true
```

注释

远程用户也可以使用 CA Introscope® 命令行 Workstation (CLW)“发送配置文件”命令将 introscope.agent.remoteagentconfiguration.allowedFiles 属性中指定的文件复制到代理目录中。

此属性适用于 CA CEM 4.0 和 4.1 版本。仅当您已在“Introscope 设置”页面上选择“CEMTracer 4.0/4.1 支持”选项时，此属性才适用于 CA CEM 4.2/4.5。

通过“CEMTracer 4.0/4.1 支持”选项，可以随时将代理从 4.0 或 4.1 交错迁移到 4.2/4.5。仅在需要时使用此选项。

对于不兼容的代理（即，不支持的 .NET 代理、EPA 代理或其他非 Java 代理），请将 introscope.agent.remoteagentconfiguration.enabled 属性设置为 false。

introscope.agent.decorator.enabled

如果此布尔值设置为 true，它会将代理配置为将其他性能监控信息添加到 HTTP 响应标头中。ServletHeaderDecorator/HTTPHeaderDecorator 可将 GUID 附加到每个事务，并将 GUID 插入 HTTP 标头 x-apm-info。

此属性可启用 CA CEM 与 CA Introscope® 之间的事务关联。

设置

true 或 false

默认

- 对于 Java 代理，设置为 false
- 对于 .NET 代理，设置为 true

示例

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

introscope.agent.decorator.security

此属性确定发送到 CA CEM 的已修饰 HTTP 响应标头的格式。

设置

- clear－明文编码
- encrypted－标头数据已加密

默认

clear

示例

```
introscope.agent.decorator.security=clear
```

注释

默认设置 clear 适用于初始测试。但是，此设置可能会泄露不希望防火墙之外的用户知道的事务标头中的信息。请将该属性设置为 encrypted，以实现更安全的生产环境。

要将此属性设置为 encrypted，请使用支持的 JVM。

注意：有关 JVM 支持信息，请参阅《*兼容性指南*》。

introscope.agent.cemtracer.domainconfigfile

用于指定 CA CEM 业务服务和事务层次结构的 CA CEM 域配置文件的名称。CEMTracer 会在其安装目录中查找具有该名称的文件。

每当 CA CEM 管理员单击 CA CEM 中的“同步所有监视器”时，都会将域配置文件推送到企业管理器，然后企业管理器再转而将文件推送到每个连接的代理。

有关特定于 CA CEM 版本的信息，请参阅**注释**（见下文）。

设置

可以是任何有效的文件名。

默认

```
domainconfig.xml
```

示例

```
introscope.agent.cemtracer.domainconfigfile=domainconfig.xml
```

注释

此属性适用于 CA CEM 4.0 和 4.1 版本。仅当您已在“Introscope 设置”页面上选择“CEMTracer 4.0/4.1 支持”选项时，此选项才适用于 CA CEM 4.2/4.5。

通过“CEMTracer 4.0/4.1 支持”选项，可以随时将您的代理从 4.0 或 4.1 交错迁移到 4.2/4.5；请仅在需要时使用。

- 如果代理未连接到企业管理器，则不会发送域配置文件。
- 如果代理目录为只读，则无法写入域配置文件。
- 如果未在代理上启用 CEMTracer 4.0/4.1，则发送域配置文件后不会对其执行任何操作。

introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes

代理重新加载域配置文件的频率（以分钟为单位）。（4.0/4.1 代理不会在每次企业管理器发送域配置文件后自动重新加载。如果域配置文件没有更改，则代理不会进行重新加载。）

有关特定于 CA CEM 版本的信息，请参阅**注释**（见下文）。

设置

数值

默认

1

示例

```
introscope.agent.cemtracer.domainconfigfile.reloadfrequencyinminutes=1
```

注释

此属性适用于 CA CEM 4.0 和 4.1 版本。仅当您已在“Introscope 设置”页面上选择“CEMTracer 4.0/4.1 支持”选项时，此选项才适用于 CA CEM 4.2/4.5。

通过“CEMTracer 4.0/4.1 支持”选项，可以随时将您的代理从 4.0 或 4.1 交错迁移到 4.2；请仅在需要时使用。

introscope.agent.distribution.statistics.components.pattern

要从 BlamePointTracers 收集响应时间分布信息，请取消注释并编辑此属性。您可以使用此响应时间信息来创建平均响应时间度量标准。

有关更多信息：

如何将代理配置为收集分布统计信息度量标准 (p. 68)

配置会话 ID 收集

introscope.agent.transactiontracer.parameter.capture.sessionid 属性可启用或禁用 TransactionTracer 数据中会话 ID 的收集。默认情况下，在 TransactionTracer 数据中启用并记录该属性。禁用该属性时，无法在筛选中使用数据。

请执行以下步骤：

1. 在文本编辑器中打开 IntroscopeAgent.profile 文件。

查找下列行：

```
# Uncomment the following property to disable sessionid capture in
TransactionTracer data.
# By default, it is enabled and recorded in the TT Data.

# introscope.agent.transactiontracer.parameter.capture.sessionid=true
```

2. 按照说明，通过注释或取消注释以下行来启用或禁用该属性：

```
# introscope.agent.transactiontracer.parameter.capture.sessionid=true
```

3. 保存并关闭文件，然后重新启动代理。

   代理配置设置为使用为收集会话 ID 而指定的值。

ChangeDetector 配置

您可以控制本地代理与 ChangeDetector 配合使用的方式。

注意：有关使用 ChangeDetector 的详细信息，请参阅*《CA APM ChangeDetector 用户指南》*。

introscope.changeDetector.enable

指定是启用还是禁用 ChangeDetector。将该属性设置为 true 可启用 ChangeDetector。默认情况下该属性会注释掉，并设置为 false。如果启用 ChangeDetector，还应设置与 ChangeDetector 相关的其他属性。

属性设置

True 或 False

默认

False

示例

```
introscope.changeDetector.enable=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.changeDetector.agentID

指定 ChangeDetector 用来识别本地代理的文本字符串。默认情况下，此属性注释掉。如果启用 ChangeDetector，应取消注释此属性并将其设置为相应的值。

默认

默认值为 *SampleApplicationName*。

示例

```
introscope.changeDetector.agentID=SampleApplicationName
```

introscope.changeDetector.rootDir

为 ChangeDetector 文件指定根目录。根目录是 ChangeDetector 在其中创建本地缓存文件的文件夹。

属性设置

文本字符串形式的 ChangeDetector 文件根目录完整路径。

默认

默认路径为 *c://sw//AppServer//wily//change_detector*。

示例

```
introscope.changeDetector.rootDir=c://sw//AppServer//wily//change_detector
```

注释

请使用反斜线转义反斜线字符，如示例中所示。

introscope.changeDetector.isengardStartupWaitTimeInSec

指定启动代理后等待 ChangeDetector 尝试连接到企业管理器的时间。默认情况下，此属性注释掉。

默认

默认值为 15 秒。

示例

```
introscope.changeDetector.isengardStartupWaitTimeInSec=15
```

introscope.changeDetector.waitTimeBetweenReconnectInSec

指定 ChangeDetector 在重新尝试连接到企业管理器之前等待的秒数。默认情况下，此属性注释掉。

默认

默认值为 10 秒。

示例

```
introscope.changeDetector.waitTimeBetweenReconnectInSec=10
```

introscope.changeDetector.profile

指定 ChangeDetector 数据源配置文件的绝对路径或相对路径。默认情况下，此属性注释掉。

默认

默认值为 ChangeDetector-config.xml。

示例

```
introscope.changeDetector.profile=CDConfig\\changedetector-config.xml
```

注释

请使用反斜线转义反斜线字符，如示例中所示。

introscope.changeDetector.profileDir

指定包含数据源配置文件的目录的绝对路径或相对路径。如果设置了此属性，除了使用 *introscope.changeDetector.profile* 属性指定的任何文件以外，还会使用此目录中的所有数据源配置文件。默认情况下，此属性注释掉。

默认

默认值为 changeDetector_profiles。

示例

```
introscope.changeDetector.profileDir=c:\\CDconfig\\changeDetector_profiles
```

注释

使用反斜杠来转义反斜杠字符。

introscope.changeDetector.compressEntries.enable

指定是否允许对 ChangeDetector 数据缓冲区进行压缩。如果在启动时占用内存，您可以将此属性设置为 true 以提高性能。

属性设置

True 或 False

默认

如果未在代理配置文件中设置此属性或已将其注释掉，则默认值为 false。

示例

```
introscope.changeDetector.compressEntries.enable=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.changeDetector.compressEntries.batchSize

此属性定义在以上 introscope.changeDetector.compressEntries.enable 属性中设置的压缩作业的批处理大小。

默认

1000

示例

```
introscope.changeDetector.compressEntries.batchSize=1000
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

WebLogic Server 中的跨进程跟踪

以下属性用于配置 WebLogic Server 中的跨进程跟踪：

- introscope.agent.weblogic.crossjvm (see page 251)

introscope.agent.weblogic.crossjvm

属性设置

True 或 False

默认

已注释掉；True

示例

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

跨进程事务跟踪

取消注释以下属性可启用对由于存在尾部筛选器而产生的下游跟踪的自动收集。

- introscope.agent.transactiontracer.tailfilterPropagate.enable (see page 251)

启用此属性并对尾部筛选器长时间运行事务跟踪会话，会导致将大量不需要的跟踪发送给企业管理器。

introscope.agent.transactiontracer.tailfilterPropagate.enable

控制尾部筛选的存在是否会触发从下游代理自动收集跟踪。此属性不影响对由于通过了头部筛选器而产生的自动下游跟踪的收集。

属性设置

True 或 False

默认

false；注释掉。

示例

```
introscope.agent.transactiontracer.tailfilterPropagate.enable=false
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

动态检测

您可以动态检测类和方法，而无需写入自定义 PBD、重新启动应用程序服务器或重新启动代理。

注意：有关从 Introscope Workstation 使用事务跟踪和动态检测的详细信息，请参阅《*CA APM Workstation 用户指南*》。

introscope.autoprobe.dynamicinstrument.enabled

适用于：在 JDK 1.5 下运行并使用 AutoProbe 的代理

该属性为代理启用动态 ProbeBuilding。

属性设置

True 或 False

默认

False

示例

```
introscope.autoprobe.dynamicinstrument.enabled=false
```

详细信息：

动态 ProbeBuilding (p. 73)

autoprobe.dynamicinstrument.pollIntervalMinutes

适用于：使用 AutoProbe 和动态 ProbeBuilding 在 JDK 1.5 下运行的代理

此属性可确定代理轮询新的和更改的 PBD 的频率。

默认

1

示例

```
autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs

根据观察，一些类加载器实现会返回较大的类文件。这是为了防止出现内存错误。

默认

1

示例

```
introscope.autoprobe.dynamicinstrument.classFileSizeLimitInMegs=1
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo

一次重新定义过多的类可能会占用大量 CPU。当 PBD 中的更改触发大量类的重新定义时，会以适当的速度批处理该过程。

默认

10

示例

```
introscope.autoprobe.dynamic.limitRedefinedClassesPerBatchTo=10
```

introscope.agent.remoteagentdynamicinstrumentation.enabled

启用或禁用动态检测的远程管理。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.remoteagentdynamicinstrumentation.enabled=true
```

注释

- 您必须重新启动受控应用程序，对此属性所做的更改才能生效。
- 动态检测是一个占用大量 CPU 的操作。请使用可最大程度减少正在检测的类的配置。

introscope.autoprobe.dynamicinstrument.pollIntervalMinutes

定义轮询 PBD 更改的轮询时间间隔（分钟）。

默认

1

示例

```
introscope.autoprobe.dynamicinstrument.pollIntervalMinutes=1
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

ErrorDetector

您可以控制代理与 ErrorDetector 的交互方式。

introscope.agent.errorsnapshots.enable

使代理能够捕获有关严重错误的事务详细信息。默认情况下，代理上安装有 Introscope ErrorDetector。该属性必须设置为 true，才能查看错误快照。

属性设置

True 或 False

默认

True

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.errorsnapshots.throttle

指定代理在 15 秒时间内可以发送的最大错误快照数目。

默认

10

示例

```
introscope.agent.errorsnapshots.throttle=10
```

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.errorsnapshots.ignore.<index>

指定一个或多个错误消息筛选。通过在属性名称后附加索引标识符（例如 .0、.1、.2...），可以根据需要指定任意数量的筛选。您可以使用通配符 (*)。将忽略满足您指定的条件的错误消息。对于符合您所定义的筛选的错误，不生成任何错误快照；也不会将有关这些错误的任何错误事件发送到企业管理器。

重要信息！此属性无法用于筛选 SOAP 错误消息。

默认

提供了示例定义，并注释掉，如下所示。

示例

```
introscope.agent.errorsnapshots.ignore.0=*com.company.HarmlessException*
introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code: 404*
```

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

扩展

您可以配置代理扩展的位置。

introscope.agent.extensions.directory

指定代理加载的所有扩展的位置。您可以指定该目录的绝对路径或相对路径。如果您不指定绝对路径，则您指定的值将相对于 *IntroscopeAgent.properties* 文件位置进行解析。

默认

默认位置为 *<Agent_Home>*/ext 目录中的 *ext* 目录。

示例

```
introscope.agent.extensions.directory=../ext
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.common.directory

配置代理扩展相关文件的位置。

默认

默认位置为 <Agent_Home>/common 目录中的 common 文件夹。

示例

```
introscope.agent.common.directory=../../common
```

GC 监视器

“GC 监视器”节点下的度量标准可报告有关垃圾回收器和内存池的信息。这些度量标准有助于检测到对性能产生负面影响的内存相关问题。您必须在代理配置文件中手工启用这些度量标准的收集。

introscope.agent.gcmonitor.enable

此属性启用或禁用垃圾回收器和内存池的度量标准。

属性设置

True 或 False

默认

默认值为 true。

示例

```
introscope.agent.gcmonitor.enable=true
```

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

您只能为监控 Sun 或 IBM JVM 的代理报告 GC 监视器度量标准。

Java NIO

Java 代理支持 Java New I/O（Java NIO 或 NIO）功能。Java NIO 是一个 API 集合，旨在提供对现代操作系统的低级别 I/O 操作的访问。Java NIO 度量标准捕获有关被检测应用程序如何使用 Java NIO 的信息。

注意： CA Introscope® Java NIO 度量标准和 Java NIO 信息的度量标准收集仅在 Java 1.5 JVM 或更高版本上可用。

CA Introscope® Java 代理收集 NIO 通道度量标准。

可以限制生成某些 NIO 度量标准。

默认情况下已启用 Java NIO 跟踪器组。可以关闭这些跟踪器组以进一步限制度量标准生成。

详细信息：

通道 (p. 258)
限制 Java NIO 度量标准 (p. 259)
默认跟踪器组和 Toggles 文件 (p. 86)

通道

通过 Java NIO 通道，可以将批量数据输入和输出 NIO 缓冲区以及外部系统。这是一个低级别数据传输机制，专门设计用于解决标准 Java I/O 内的性能和可伸缩性问题。

通道提供在缓冲区和外部系统之间移动字节的机制。Introscope 通道度量标准衡量经过通道的数据流速率。收集的 NIO 通道度量标准与当前使用标准 Java I/O 技术为文件和套接字 I/O 创建的度量标准相对应。单独收集以下通道类型的度量标准并将其显示在 Workstation 调查器中：

- 数据报通道
- 套接字通道

NIODatagramTracing 度量标准

尽管 UDP 是无连接协议，在 NIODatagramTracing 的说明中仍使用术语“连接”来描述 Java 代理如何收集数据报度量标准。Java 代理针对“发送到”或“接收自”的每个远程端点数据报单独收集度量标准。连接分为客户端或服务器，具体取决于从每个“发送到”或“接收自”端点观测到的第一个数据报的方向。

例如，如果第一个数据报“接收自”端点，则 Java 代理会将所有发送到或接收自该端点的其他数据报分类在 *NIO|通道|数据报|服务器|端口 {PORT}* 下，其中 *{PORT}* 是本地端口。

如果第一个数据报“发送到”端点，则会将所有发送到或接收自该端点的其他数据报分类在 *NIO|通道|数据报|客户端|{HOST}|端口 {PORT}* 下，其中 *{HOST}* 和 *{PORT}* 是远程端点。

Java 代理还生成客户端“连接”的后端度量标准，由“receive”方法读取数据报的情况除外。使用 DatagramChannel 连接方法创建的数据报通道被视为客户端连接，而不考虑观测到的第一个数据报的方向。

注意：使用 UDP 连接（带有连接方法）创建的数据报通道被视为客户端连接。

限制 Java NIO 度量标准

您可以配置属性来控制 Java NIO 检测的工作原理。还可以限制数据报和套接字度量标准的生成。这些属性仅影响 NIOSocketTracing 和 NIODatagramTracing 跟踪器组生成的详细信息度量标准。

注意：有关跟踪器组的详细信息，请参阅默认跟踪器组和 toggles 文件 (see page 86)。

introscope.agent.nio.datagram.client.hosts

将度量标准报告限制为指定主机的“客户端”UDP“连接”。

属性设置

以逗号分隔的主机列表。

默认

未定义（无值）

示例

```
introscope.agent.nio.datagram.client.hosts=hostA,hostB
```

注释

- 如果列表为空，将不应用任何主机限制。
- 可以通过名称或 IP 地址的文本表示形式（以 IPv4 或 IPv6 形式）来指定主机。
- 将在代理日志中报告无效的主机名并忽略这些名称。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。
- 已放弃重复主机名。如果多个主机名映射到一个单个 IP，则仅会保留其中一个名称。但是，该属性将匹配具有任何同义名称集的客户端连接。

introscope.agent.nio.datagram.client.ports

列出将报告 NIO 度量标准的端口。将只生成指定端口的“客户端”数据报度量标准。

属性设置

以逗号分隔的*端口*号列表。这里的*端口*指的是将数据报发送到或从中接收数据报的远程端口。

默认

未定义（无值）

示例

```
introscope.agent.nio.datagram.client.ports=123,456,789
```

注释

- 如果列表为空，将不应用任何端口限制。
- 将在代理日志中报告无效的端口号，并忽略这些端口号。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。
- 已放弃重复端口。在多个端口映射到单个 IP 的情况下，仅保留其中一个端口。

introscope.agent.nio.datagram.server.ports

列出将报告 NIO 度量标准的端口。将只生成指定端口的“服务器”数据报度量标准。

属性设置

以逗号分隔的*端口*号列表。这里的*端口*指的是通过其发送或接收数据报的本地端口。

默认

未定义（无值）

示例

```
introscope.agent.nio.datagram.server.ports=123,456,789
```

注释

- 如果列表为空，将不应用任何端口限制。
- 将在代理日志中报告无效的端口号，并忽略这些端口号。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.nio.socket.client.hosts

将度量标准报告限制为指定主机的“客户端”TCP“连接”。

属性设置

以逗号分隔的主机列表。

默认

未定义（无值）

示例

```
introscope.agent.nio.socket.client.hosts=hostA, hostB
```

注释

- 如果列表为空，将不应用任何主机限制。
- 可以通过名称或 IP 地址的文本表示形式（以 IPv4 或 IPv6 形式）来指定主机。
- 将在代理日志中报告无效的主机名并忽略这些名称。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.nio.socket.client.ports

列出将报告 NIO 度量标准的端口。将只生成指定端口的“客户端”套接字度量标准。

属性设置

以逗号分隔的*端口*号列表。这里的*端口*指的是将数据报发送到或从中接收数据报的远程端口。

默认

未定义（无值）

示例

```
introscope.agent.nio.socket.client.ports=123,456,789
```

注释

- 如果列表为空，将不应用任何端口限制。
- 将在代理日志中报告无效的端口号，并忽略这些端口号。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.nio.socket.server.ports

列出将报告 NIO 度量标准的端口。将只生成指定端口的“服务器”套接字度量标准。

属性设置

以逗号分隔的*端口*号列表。这里的*端口*指的是通过其发送或接收数据报的本地端口。

默认

未定义（无值）

示例

```
introscope.agent.nio.socket.client.ports=123,456,789
```

注释

- 如果列表为空，将不应用任何端口限制。
- 将在代理日志中报告无效的端口号，并忽略这些端口号。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

Java NIO 度量标准显示在 Workstation 调查器中代理的顶级节点下的 NIO 节点下面。任意“客户端”连接的其他 NIO 度量标准将显示在“后端”节点下。

通过注释掉要禁用的度量标准的 *TraceOneMethodIfFlagged* 或 *TraceOneMethodWithParametersIfFlagged* 指令，可以禁用单个 NIO 度量标准。但是，不应注释掉名称以 *BackendTracer* 或 *MappingTracer* 结尾的跟踪器。

例如，要禁用数据报的“并发读取线程”度量标准，请注释掉以下指令：

```
TraceOneMethodWithParametersIfFlagged: NIODatagramTracing read
NIODatagramConcurrentInvocationCounter "Concurrent Readers"
```

JMX

以下属性用于配置 JMX 度量标准：

- introscope.agent.jmx.enable (see page 263)
- introscope.agent.jmx.ignore.attributes (see page 264)
- introscope.agent.jmx.name.filter (see page 264)
- introscope.agent.jmx.name.jsr77.disable (see page 265)
- introscope.agent.jmx.name.primarykeys (see page 266)
- introscope.agent.jmx.excludeStringMetrics (see page 267)

introscope.agent.jmx.enable

启用 JMX 度量标准收集。

属性设置

True 或 False。

默认

因代理版本而异。

示例

```
introscope.agent.jmx.enable=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.jmx.ignore.attributes

控制要忽略哪些（如果有）JMX MBean 属性。

属性设置

以逗号分隔的关键字列表。

默认

已注释掉；*server*。

示例

```
introscope.agent.jmx.ignore.attributes=server
```

注释

- 如果 MBean 属性名称与列表中的一个名称匹配，将忽略该属性。
- 使列表保留为空将包含所有 MBean 属性。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.jmx.name.filter

指定以逗号分隔的筛选字符串列表，以确定 Introscope 收集并显示哪些 JMX 数据。

Introscope 报告 JMX 生成的与某个筛选字符串匹配的度量标准。筛选字符串可以包含星号 (*) 和问号 (?) 通配符：

- * 与零个或更多字符匹配
- ？匹配单个字符。

要匹配文本 * 或 ?，请用 \\ 对字符进行转义。

示例：

- *ab*c* 匹配包含 *ab*c* 的度量标准名称
- *ab*c* 匹配包含 *abc*、*abxc*、*abxxc* 等的度量标准名称
- *ab?c* 匹配包含 *abxc* 的度量标准名称
- *ab\\?c* 匹配包含 *ab?c* 的度量标准名称。

默认

注释掉。

对于 WebLogic：
`ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType`

示例

```
#introscope.agent.jmx.name.filter=ActiveConnectionsCurrentCount,WaitingForConnectionCurrentCount,PendingRequestCurrentCount,ExecuteThreadCurrentIdleCount,OpenSessionsCurrentCount,j2eeType
```

注释

- 保留为空将包含系统中可用的所有 MBean 数据。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.jmx.name.jsr77.disable

此属性控制 Introscope 是否收集并报告完整的 JSR77 数据，包括复杂的 JMX 数据。

此属性只能在 WebLogic 和 WebSphere 的 *IntroscopeAgent.profile* 文件中使用。

属性设置

True 或 False

默认

True

注释

- 应用程序服务器必须提供 JSR-77 管理支持，此属性才会起作用。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.jmx.name.primarykeys

用户定义的 MBean 信息的顺序，可简化名称转换。

属性设置

以逗号分隔的、应唯一标识特定 MBean 的键的有序列表。

默认

在默认的 *IntroscopeAgent.profile* 文件中已注释掉。

示例

```
introscope.agent.jmx.name.primarykeys=J2EEServer
```

注释

- WebLogic 的属性设置：
  - Type
  - Name
- 如果使用 WebLogic Server 9.0，请注释掉此属性。
- WebSphere 的属性设置：
  - J2EEServer
  - Application
  - j2eeType
  - JDBCProvider
  - name
  - mbeanIdentifier
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.jmx.excludeStringMetrics

控制是否包含串值度量标准。要启用串值度量标准，请将此属性值设置为 false。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.jmx.excludeStringMetrics=true
```

注释

- 排除串值度量标准可减少整体度量标准计数，从而提高代理和 EM 性能。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

LeakHunter

以下属性用于配置代理与 LeakHunter 的交互：

- introscope.agent.leakhunter.collectAllocationStackTraces (see page 268)
- introscope.agent.leakhunter.enable (see page 268)
- introscope.agent.leakhunter.leakSensitivity (see page 269)
- introscope.agent.leakhunter.logfile.append (see page 269)
- introscope.agent.leakhunter.logfile.location (see page 270)
- introscope.agent.leakhunter.timeoutInMinutes (see page 270)

introscope.agent.leakhunter.collectAllocationStackTraces

控制 LeakHunter 是否为潜在泄漏生成分配堆栈跟踪。将该属性设置为 *true* 将获得更精确的有关潜在泄漏分配的数据，但是需要额外的内存和 CPU 开销。因此，默认设置为 *false*。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.leakhunter.collectAllocationStackTraces=
false
```

注释

- 将该属性设置为 *true* 可能会在 CPU 使用和内存方面造成较高的系统开销。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.leakhunter.enable

控制是否启用 LeakHunter 功能。将值设置为 true 可启用 LeakHunter。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.leakhunter.enable=false
```

注释

- 启用该选项可能会导致 CPU 和内存使用率较高。仅当其他度量标准表明存在内存泄漏时才应启用该功能。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.leakhunter.leakSensitivity

控制 LeakHunter 泄漏检测算法的敏感度级别。较高的敏感度设置将导致报告较多的潜在泄漏，较低的敏感度设置则导致报告较少的潜在泄漏。

属性设置

泄漏敏感度范围必须是 1（低）到 10（高）之间的正整数值。

默认

5

示例

```
introscope.agent.leakhunter.leakSensitivity=5
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.leakhunter.logfile.append

指定在应用程序重新启动时替换日志文件还是向现有日志文件中添加信息。

属性设置

True 或 False

- 设置为 False 将替换日志文件。
- 设置为 True 将向现有日志文件中添加信息。

默认

False

示例

```
introscope.agent.leakhunter.logfile.append=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.leakhunter.logfile.location

控制 *LeakHunter.log* 文件的位置。可以指定文件名的绝对路径或相对路径。相对路径相对于 *<Agent_Home>* 目录进行解析。如果不希望 LeakHunter 将数据记录到日志文件中，请将值保留为空白或注释掉。

默认

默认路径为 *../../logs/LeakHunter.log*。将在 *<Agent_Home>logs* 目录中找到日志文件。

示例

```
introscope.agent.leakhunter.logfile.location=../../logs/LeakHunter.log
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.leakhunter.timeoutInMinutes

控制 LeakHunter 寻找新的潜在泄漏所用的时间长度（以分钟为单位）。在指定的时间过后，LeakHunter 将停止寻找新的潜在泄漏。它将继续跟踪以前识别的潜在泄漏。

属性设置

必须是正整数（不能是负数）。

默认

默认值为 120 分钟。

示例

```
introscope.agent.leakhunter.timeoutInMinutes=120
```

注释

- 如果希望 LeakHunter 始终寻找新的潜在泄漏，请将值设置为零。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.leakhunter.ignore.<number>

使用此属性可忽略匹配提供的任意模式的任意类。默认情况下，已提供十个类－注释掉要使用的任意类。

属性设置

以逗号分隔的类匹配模式列表。

默认

无

示例

```
introscope.agent.leakhunter.ignore.4=java.util.SubList
introscope.agent.leakhunter.ignore.5=com.sun.faces.context.BaseContextMap$EntrySet
introscope.agent.leakhunter.ignore.6=com.sun.faces.context.BaseContextMap$Key
```

注释

- 某些收集无法与 LeakHunter 结合使用。收集必须能够随时安全地从任何线程调用 *size()*，才能与 LeakHunter 结合使用。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。
- 可使用“*”通配符。

日志记录

以下属性用于配置代理日志记录选项：

- log4j.logger.IntroscopeAgent (see page 272)
- log4j.appender.logfile.File (see page 273)
- log4j.logger.IntroscopeAgent.inheritance (see page 273)
- log4j.appender.pbdlog.File (see page 274)
- log4j.appender.pbdlog (see page 274)
- log4j.appender.pbdlog.layout (see page 275)
- log4j.appender.pbdlog.layout.ConversionPattern (see page 275)
- log4j.additivity.IntroscopeAgent.inheritance (see page 276)

log4j.logger.IntroscopeAgent

此属性控制日志信息的日志记录级别和输出位置。

属性设置

详细信息级别值可以是：

- *INFO*
- *VERBOSE#com.wily.util.feedback.Log4JSeverityLevel*

目标值可以为：

- *console*
- *logfile*
- *console* 和 *logfile*

默认

INFO, console, logfile

示例

```
log4j.logger.IntroscopeAgent=INFO,console,logfile
```

要禁用代理日志记录，请按如下所示删除此属性中的选项：

删除前：

```
log4j.logger.IntroscopeAgent=INFO, console, logfile
```

删除后：

```
log4j.logger.IntroscopeAgent=
```

注释

- 对此属性所做的更改会立即生效，不需要重新启动托管应用程序。

log4j.appender.logfile.File

如果在 log4j.logger.IntroscopeAgent 中指定了 logfile，则指定 IntroscopeAgent.log 文件的名称和位置。文件名相对于包含代理配置文件的目录。

默认

IntroscopeAgent.log

示例

```
log4j.appender.logfile.File=../../logs/IntroscopeAgent.log
```

注释

系统属性（Java 命令行 -D 选项）作为文件名的一部分进行扩展。例如，如果 Java 以 *-Dmy.property=Server1* 开头，则 *log4j.appender.logfile.File=../../logs/Introscope-${my.property}.log* 扩展为 *log4j.appender.logfile.File=../../logs/Introscope-Server1.log*。

log4j.logger.IntroscopeAgent.inheritance

控制有关需要检测的类的日志消息的日志级别和目标。

属性设置

要配置尚未检测的类（由于其扩展父类型或接口）的日志记录，请将此属性设置为 *INFO, pbdlog*

有关继承类日志记录的信息，请参阅控制指令日志记录 (see page 78)。

默认

无

示例

```
log4j.logger.IntroscopeAgent.inheritance=INFO,pbdlog
```

log4j.appender.pbdlog.File

确定有关需要检测的类的消息的日志文件。

属性设置

要配置尚未检测的类（由于其扩展父类型或接口）的日志记录，请设置为 *pbdupdate.log*

默认

无

示例

```
log4j.appender.pbdlog.File=../../pbdupdate.log
```

log4j.appender.pbdlog

为有关需要检测的类的日志记录消息指定包。

属性设置

要配置尚未检测的类（由于其扩展父类型或接口）的日志记录，请将此属性设置为 *com.wily.introscope.agent.AutoNamingRollingFileAppender*

默认

无

示例

```
log4j.appender.pbdlog=com.wily.introscope.agent.AutoNamingRollingFileAppender
```

log4j.appender.pbdlog.layout

为记录有关需要检测的类的消息指定规则。

属性设置

要配置尚未检测的类（由于其扩展父类型或接口）的日志记录，请将此属性设置为 *com.wily.org.apache.log4j.PatternLayout*

默认

无

示例

```
log4j.appender.pbdlog.layout=com.wily.org.apache.log4j.PatternLayout
```

log4j.appender.pbdlog.layout.ConversionPattern

为记录有关需要检测的类的消息指定规则。

属性设置

要配置尚未检测的类（由于它们扩展超类型或接口）的记录，请将该属性设置为：

```
%d{M/dd/yy hh:mm:ss a z} [%-3p] [%c] %m%n
```

默认

无

示例

```
log4j.appender.pbdlog.layout.ConversionPattern=%d{M/dd/yy hh:mm:ss a z} [%-3p]
[%c] %m%n
```

log4j.additivity.IntroscopeAgent.inheritance

导致仅在 *pbdupdate.log* 文件中记录多级别继承的指令。

属性设置

True 或 False

默认

True

示例

```
log4j.additivity.IntroscopeAgent.inheritance=true
```

注释

要仅在 *pbdupdate.log* 中配置多级别继承指令的日志记录，请将此属性添加到代理配置文件并设置为 *false*。

度量标准计数

以下属性影响您将在调查器中查看 *"度量标准计数"* 度量标准的位置：

- introscope.ext.agent.metric.count (see page 277)

introscope.ext.agent.metric.count

控制您将在调查器中查看*“度量标准计数”*度量标准的位置。默认情况下，*“度量标准计数”*显示在*“自定义度量标准代理”*节点下。如果要在*“代理统计信息”*节点下查看*“度量标准计数”*度量标准，请将此属性添加到 *IntroscopeAgent.profile*。

属性设置

True 或 False

默认

在 *IntroscopeAgent.profile* 中不存在；False

示例

```
introscope.ext.agent.metric.count=true
```

注释

将此属性添加到 *IntroscopeAgent.profile* 并设置为 true，可在“代理统计信息”节点下查看“度量标准计数”度量标准。

多重继承

对于基于接口或超类的指令，代理无法检测多重继承，因此不检测这些类。启用以下属性可确定应用程序服务器或代理进程启动后的多种情况。这些属性记录需要检测但尚未进行检测的类，依赖于动态检测来影响更改。

- introscope.autoprobe.hierarchysupport.enabled (see page 278)
- introscope.autoprobe.hierarchysupport.runOnceOnly (see page 278)
- introscope.autoprobe.hierarchysupport.pollIntervalMinutes (see page 279)
- introscope.autoprobe.hierarchysupport.executionCount (see page 279)
- introscope.autoprobe.hierarchysupport.disableLogging (see page 280)
- introscope.autoprobe.hierarchysupport.disableDirectivesChange (see page 280)

introscope.autoprobe.hierarchysupport.enabled

对于使用 AutoProbe 和动态检测在 JDK 1.5 下运行的代理，可使用此属性来启用扩展父类型或接口的类的检测。

属性设置

True 或 False

默认

True

示例

```
introscope.autoprobe.hierarchysupport.enabled=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.hierarchysupport.runOnceOnly

如果已启用扩展父类型或接口的类的检测，可使用此属性来控制启用功能的实用工具是仅运行一次还是按照指定的时间间隔运行。

仅当需要定期进行检测时，才将此属性更改为 **true**。

属性设置

True 或 False

默认

False

示例

```
introscope.autoprobe.hierarchysupport.enabled=false
```

注释

- 在“日志记录”中定义与动态检测相关的日志记录属性。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.hierarchysupport.pollIntervalMinutes

检查由于多重继承无法进行检测的类的轮询时间间隔。在大多数情况下，这种情况只发生一次；但是，建议使用保守值进行应用程序服务器初始化。

默认

5

示例

```
introscope.autoprobe.hierarchysupport.pollIntervalMinutes=5
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.hierarchysupport.executionCount

如果需要轮询时间间隔运行有限次数，而不是仅运行一次或定期运行，请使用此属性指定运行轮询时间间隔的精确次数。始终使用此属性指定应运行的精确次数。

使用此属性覆盖仅运行一次设置。

属性设置

一个正整数。

默认

3

示例

```
introscope.autoprobe.hierarchysupport.executionCount=3
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.hierarchysupport.disableLogging

如果不需要记录被检测的类，请取消注释此属性。仅当启用动态检测时，才取消注释此属性。

属性设置

True 或 False

默认

True

示例

```
#introscope.autoprobe.hierarchysupport.disableLogging=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.autoprobe.hierarchysupport.disableDirectivesChange

取消注释此属性可仅记录更改并禁止触发动态检测。

属性设置

True 或 False

默认

True

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

平台监控

以下属性用于配置平台监控度量标准：

- introscope.agent.platform.monitor.system (see page 281)

introscope.agent.platform.monitor.system

要为其加载平台监视器的操作系统的名称。

属性设置

有关此属性的选项的详细信息，请参阅平台监控故障排除 (see page 191)。

默认

已注释掉；因平台而异。

示例

```
introscope.agent.platform.monitor.system=Solaris
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

远程配置

以下属性允许对 Java 代理进行远程配置：

- introscope.agent.remoteagentconfiguration.enabled (see page 281)
- introscope.agent.remoteagentconfiguration.allowedFiles (see page 282)

introscope.agent.remoteagentconfiguration.enabled

该属性用于启用或禁用代理的远程配置。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.remoteagentconfiguration.enabled=true
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.remoteagentconfiguration.allowedFiles

此属性列出允许远程传输到此代理的文件的精确列表。

属性设置

domainconfig.xml

默认

domainconfig.xml

示例

```
introscope.agent.remoteagentconfiguration.allowedFiles=domainconfig.xml
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

安全性

以下属性用于配置发送到 CA CEM 的 HTTP 标头的安全性：

- introscope.agent.decorator.security (see page 282)

introscope.agent.decorator.security

此属性确定发送到 CA CEM 的已修饰 HTTP 响应标头的格式。

属性设置

Clear：明文编码

Encrypted：标头数据已加密

默认

Clear

示例

```
introscope.agent.decorator.security=clear
```

Servlet 标头装饰器

以下属性启用 CA CEM 与 Introscope 间事务的关联：

- introscope.agent.decorator.enabled (see page 283)

introscope.agent.decorator.enabled

如果此布尔值设置为 true，它会将代理配置为将其他性能监控信息添加到 HTTP 响应标头中。ServletHeaderDecorator 将 GUID 附加到每个事务并将其插入 HTTP 标头，例如：x-apm-info

属性设置

True 或 False

默认

False

示例

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

套接字度量标准

I/O 套接字度量标准的生成可能会受以下参数的限制：

- introscope.agent.sockets.reportRateMetrics (see page 284)
- introscope.agent.io.socket.client.hosts (see page 284)
- introscope.agent.io.socket.client.ports (see page 285)
- introscope.agent.io.socket.server.ports (see page 285)

introscope.agent.sockets.reportRateMetrics

启用各个套接字输入/输出 (I/O) 带宽速率度量标准的报告。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.sockets.reportRateMetrics=false
```

注释

- 仅当启用 ManagedSocketTracing 并禁用 SocketTracing 时才起作用。有关详细信息，请参阅向后兼容性 (see page 131)。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.io.socket.client.hosts

将已检测的套接字客户端连接限制为具有指定远程主机的连接。

属性设置

以逗号分隔的值列表。

示例

```
introscope.agent.io.socket.client.hosts=
```

注释

- 如果任意单个值无效，将忽略该值。
- 如果任意参数未定义，或排除所有无效值后列表为空，将不对该参数应用任何限制。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.io.socket.client.ports

将已检测的套接字客户端连接限制为具有指定远程端口的连接。

属性设置

以逗号分隔的值列表。

示例

```
introscope.agent.io.socket.client.ports=
```

注释

- 如果任意单个值无效，将忽略该值。
- 如果任意参数未定义，或排除所有无效值后列表为空，将不对该参数应用任何限制。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.io.socket.server.ports

将已检测的套接字客户端连接限制为使用指定本地端口的连接。

属性设置

以逗号分隔的值列表。

示例

```
introscope.agent.io.socket.server.ports=
```

注释

- 如果任意单个值无效，将忽略该值。
- 如果任意参数未定义，或排除所有无效值后列表为空，将不对该参数应用任何限制。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

SQL 代理

您可以配置 SQL 代理的某些方面。

详细信息：

introscope.agent.sqlagent.normalizer.extension

此属性可指定用于覆盖预配置规范化方案的 SQL 规范化程序扩展的名称。

要使自定义规范化扩展生效，其清单属性 *com-wily-Extension-Plugin-{pluginName}-Name* 的值必须与此属性中给定的值匹配。

如果您指定逗号分隔的名称列表，代理将使用默认规范化程序扩展。

例如，根据以下内容设置，RegexSqlNormalizer 可用于规范化：

```
introscope.agent.sqlagent.normalizer.extension=ext1, ext2
```

此属性可限制将 SQL 语句中的多少内容（字节）显示在 SQL 代理度量标准的调查器树中。

属性设置

用于覆盖预配置规范化方案的 SQL 规范化程序扩展的名称。

默认

RegexSqlNormalizer

示例

```
introscope.agent.sqlagent.normalizer.extension=RegexSqlNormalizer
```

注释

如果使用默认设置，您还必须配置正则表达式 SQL 语句规范化程序属性：

- introscope.agent.sqlagent.normalizer.regex.matchFallThrough (see page 287)
- introscope.agent.sqlagent.normalizer.regex.keys (see page 288)
- introscope.agent.sqlagent.normalizer.regex.key1.pattern (see page 288)
- introscope.agent.sqlagent.normalizer.regex.key1.replaceAll (see page 289)
- introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat (see page 289)
- introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive (see page 290)

对此属性所做的更改将即时生效，无需重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.matchFallThrough

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。此属性设置为 true 时，它将根据所有正则表达式键组评估 SQL 字符串。

实施是连锁的。例如，如果 SQL 匹配多个键组，则 group1 的规范化 SQL 输出将作为 group2 的输入，依此类推。

如果该属性设置为 false，只要键组匹配，就会返回来自该组的规范化 SQL 输出。

属性设置

True 或 False

默认

false

示例

```
introscope.agent.sqlagent.normalizer.regex.matchFallThrough=false
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.keys

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。此属性用于指定正则表达式组键。按顺序对其进行评估。

默认

key1

示例

```
introscope.agent.sqlagent.normalizer.regex.keys=key1
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.key1.pattern

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。此属性用于指定将用于与 SQL 匹配的正则表达式模式。

属性设置

此处可使用 *java.util.Regex* 包允许的所有有效正则表达式条目。

默认

.*call(.*\)\.FOO(.*\)

示例

```
introscope.agent.sqlagent.normalizer.regex.key1.pattern=.*call(.*\)\.FOO(.*\)
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.key1.replaceAll

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。如果该属性设置为 false，它会将 SQL 查询中首次出现的匹配模式替换为替换字符串。如果设置为 true，它将使用替换字符串替换 SQL 查询中匹配模式的所有实例。

属性设置

True 或 False

默认

false

示例

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceAll=false
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。此属性指定替换字符串的格式。

属性设置

java.util.Regex 包和 *java.util.regex.Matcher* 类允许的所有有效正则表达式条目均可以在此处使用。

默认

$1

示例

```
introscope.agent.sqlagent.normalizer.regex.key1.replaceFormat=$1
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive

将此属性与 introscope.agent.sqlagent.normalizer.extension (see page 286) 结合使用，可设置正则表达式 SQL 语句规范化程序。此属性指定模式匹配是否区分大小写。

属性设置

true 或 false

默认

false

示例

```
introscope.agent.sqlagent.normalizer.regex.key1.caseSensitive=false
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.sqlagent.sql.artonly

introscope.agent.sqlagent.sql.artonly 属性可用于将代理配置为只创建和发送平均响应时间度量标准。该属性可影响后端下的所有 SQL 代理度量标准。该属性的值为 true 时，可以改善代理在 SQL 度量标准和事务跟踪方面的性能。

注意： 设置 introscope.agent.sqlagent.sql.turnoffmetrics (see page 291)=true 将覆盖该属性。

重要信息！ 必须设置以下跟踪器参数，此属性设置才能正常工作：

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

默认情况下，该属性处于关闭状态：

```
introscope.agent.sqlagent.sql.artonly=false
```

对该属性所做的更改将立即生效，可以使用“管理”界面进行更改。

注意： 该属性无法控制自定义度量标准，如连接计数。

introscope.agent.sqlagent.sql.rawsql

introscope.agent.sqlagent.sql.rawsql 属性可将代理配置为将*未规范化的* SQL 作为 SQL 组件的参数添加到事务跟踪。该属性的值为 true 时，可以改善代理在 SQL 度量标准和事务跟踪方面的性能。

默认情况下，该属性处于关闭状态：

introscope.agent.sqlagent.sql.rawsql=false

对该属性所做的更改将在重新启动托管应用程序之后生效。

重要信息！启用该属性会导致事务跟踪中显示密码和敏感信息。

introscope.agent.sqlagent.sql.turnoffmetrics

可以使用 introscope.agent.sqlagent.sql.turnoffmetrics 属性关闭 SQL 语句度量标准，从而将较少的度量标准从代理发送到企业管理器。该属性的值为 true 时，可以改善代理在 SQL 度量标准和事务跟踪方面的性能。

重要信息！必须设置以下跟踪器参数，此属性设置才能正常工作：

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

默认情况下，该属性处于关闭状态：

```
introscope.agent.sqlagent.sql.turnoffmetrics=false
```

此属性将覆盖 introscope.agent.sqlagent.sql.artonly 属性。

对该属性所做的更改将立即生效，可以使用“管理”用户界面进行更改。

introscope.agent.sqlagent.sql.turnofftrace

introscope.agent.sqlagent.sql.turnofftrace 属性可控制代理是否针对后端下的 SQL 语句创建事务跟踪组件，并将其发送给企业管理器。该属性的值为 true 时，可以改善代理在 SQL 度量标准和事务跟踪方面的性能。

重要信息！必须设置以下跟踪器参数，此属性设置才能正常工作：

```
SetTracerParameter: StatementToConnectionMappingTracer agentcomponent "SQL Agent"
```

默认情况下，该属性处于关闭状态：

```
introscope.agent.sqlagent.sql.turnofftrace=false
```

对该属性所做的更改将立即生效，可以使用“管理”用户界面进行更改。

SSL 通信

代理可通过 SSL 连接到企业管理器。使用以下属性可配置该通信：

- introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT (see page 213)
- introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT
- introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT

introscope.agent.enterprisemanager.transport.tcp.host.DEFAULT

指定运行代理默认与之连接的企业管理器的计算机主机名。

默认

localhost

示例

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

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT

指定托管企业管理器的计算机上用于侦听来自代理的连接的端口号。如果使用的是安全套接字层 (SSL) 协议，侦听来自代理的连接的默认端口则为 5443。

默认

5443

示例

```
introscope.agent.enterprisemanager.transport.tcp.port.DEFAULT=5443
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT

指定客户端套接字工厂，以便在使用 SSL 时用于从代理到企业管理器的连接。

默认

com.wily.isengard.postofficehub.link.net.SSLSocketFactory

示例

```
ntroscope.agent.enterprisemanager.transport.tcp.socketfactory.DEFAULT=com.wil
y.isengard.postofficehub.link.net.SSLSocketFactory
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT

包含可信任企业管理器证书的 truststore 的位置。如果未指定 truststore，则代理信任所有证书。

属性设置

代理工作目录的绝对路径或相对路径。

示例

```
introscope.agent.enterprisemanager.transport.tcp.truststore.DEFAULT=/var/trustedcerts
```

注释

在 Windows 上，反斜杠必须进行转义。示例：C:\\keystore

introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT

truststore 的密码。

示例

```
introscope.agent.enterprisemanager.transport.tcp.trustpassword.DEFAULT=
```

introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT

包含代理证书的 keystore 的位置。如果企业管理器要求客户端身份验证，则需要 keystore。

属性设置

代理工作目录的绝对路径或相对路径。

示例

```
introscope.agent.enterprisemanager.transport.tcp.keystore.DEFAULT=c:\\keystore
```

注释

在 Windows 上，反斜杠必须进行转义。示例：C:\\keystore

introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT

keystore 的密码。

示例

```
introscope.agent.enterprisemanager.transport.tcp.keypassword.DEFAULT=MyPasswo
rd768
```

introscope.agent.enterprisemanager.transport.tcp.ciphersuites.DEFAULT

设置已启用密码套件。

属性设置

密码套件的逗号分隔列表。

示例

```
introscope.agent.enterprisemanager.transport.tcp.
ciphersuites.DEFAULT=SSL_DH_anon_WITH_RC4_128_MD5
```

注释

如果未指定，则使用默认的已启用密码套件。

停顿度量标准

以下属性用于停顿度量标准：

- introscope.agent.stalls.thresholdseconds (see page 295)
- introscope.agent.stalls.resolutionseconds (see page 296)

有关停顿度量标准属性的详细信息，请参阅禁止将停顿作为事件进行捕获 (see page 167)。

introscope.agent.stalls.thresholdseconds

该属性指定执行过程被视为停顿过程之前的秒数。要确保准确的“停顿计数”度量标准，请将停顿阈值设置为 15 秒或更多。此设置允许企业管理器完成其搜集周期。

默认

默认值为 30 秒。

示例

```
introscope.agent.stalls.thresholdseconds=30
```

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.stalls.resolutionseconds

该属性指定代理检查停顿的频率。要确保准确的“停顿计数”度量标准，请勿将停顿解析度设置为少于 10 秒。此设置允许企业管理器完成其搜集周期。

默认

默认为每 10 秒。

示例

```
introscope.agent.stalls.resolutionseconds=10
```

注释

此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

线程转储

这些属性启用和配置 CA Introscope® 线程转储功能的代理方面：

- introscope.agent.threaddump.enable (see page 297)
- introscope.agent.threaddump.deadlockpoller.enable (see page 297)
- introscope.agent.threaddump.deadlockpollerinterval (see page 298)
- introscope.agent.threaddump.MaxStackElements (see page 298)

注意：有关配置线程转储的详细信息，请参阅如何启用和配置线程转储 (see page 65)。

introscope.agent.threaddump.enable

启用要在代理 JVM 上收集的线程转储，并允许用户查看“线程转储”选项卡。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.threaddump.enable=true
```

注释

- 对该属性所做的更改将即时生效，无需重新启动托管应用程序。
- 该属性与 *IntroscopeEnterpriseManager.properties* 文件 introscope.enterprisemanager.threaddump.enable 属性共同起作用，如果将后者设置为 true，可启用企业管理器线程转储功能。

introscope.agent.threaddump.deadlockpoller.enable

在度量标准浏览器树中启用“死锁计数”度量标准，以显示代理 JVM 中的当前死锁数。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.threaddump.deadlockpoller.enable=true
```

注释

- 对该属性所做的更改将即时生效，无需重新启动托管应用程序。

introscope.agent.threaddump.deadlockpollerinterval

CA Introscope® 对代理 JVM 轮询死锁线程的频率（毫秒）

属性设置

大于 0 的整数

默认

15000（毫秒）

示例

```
introscope.agent.threaddump.deadlockpollerinterval=15000
```

注释

- 重新启动托管应用程序，以便对此属性所做的更改生效。

introscope.agent.threaddump.MaxStackElements

线程堆栈跟踪中的总行数决定 CA Introscope® 线程转储的大小。此属性设置线程堆栈中允许的行数。

属性设置

大于 0 且小于等于 25,000 的整数

默认

12,000

示例

```
introscope.agent.threaddump.MaxStackElements=12000
```

注释

必须重新启动托管应用程序，对此属性所做的更改才能生效。

事务跟踪

以下属性用于事务跟踪和采样：

- introscope.agent.bizdef.turnOff.nonIdentifying.txn (see page 299)
- introscope.agent.transactiontracer.parameter.httprequest.headers (see page 300)
- introscope.agent.transactiontracer.parameter.httprequest.parameters (see page 300)
- introscope.agent.transactiontracer.parameter.httpsession.attributes (see page 301)
- introscope.agent.transactiontracer.userid.key (see page 301)
- introscope.agent.transactiontracer.userid.method (see page 302)
- introscope.agent.transactiontrace.componentCountClamp (see page 303)
- introscope.agent.crossprocess.compression (see page 304)
- introscope.agent.crossprocess.compression.minlimit (see page 305)
- introscope.agent.crossprocess.correlationid.maxlimit (see page 306)
- introscope.agent.transactiontracer.sampling.enabled (see page 306)
- introscope.agent.transactiontracer.sampling.perinterval.count (see page 307)
- introscope.agent.transactiontracer.sampling.interval.seconds (see page 307)
- introscope.agent.transactiontrace.headFilterClamp (see page 307)

有关详细信息，请参阅配置事务跟踪选项 (see page 159)。

introscope.agent.bizdef.turnOff.nonIdentifying.txn

启用或禁用对非标识事务的跟踪。

当此属性在 *introscopeAgent.profile* 中设置为 FALSE 时，将生成对非标识事务的跟踪。

默认情况下不会生成对非标识事务的跟踪，即使在 CEM UI 中启用了此功能。

属性设置

TRUE 或 FALSE

默认

TRUE

示例

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

introscope.agent.transactiontracer.parameter.httprequest.headers

指定要捕获的 HTTP 请求标头数据（在一个逗号分隔列表中）。使用逗号分隔列表。

默认

注释掉；*User-Agent*

示例

```
introscope.agent.transactiontracer.parameter.httprequest.headers=User-Agent
```

注释

IntroscopeAgent.profile 包含将此属性的值设为空值的已注释掉的语句。用户可以选择性地取消注释该语句并提供所需的标头名称。

introscope.agent.transactiontracer.parameter.httprequest.parameters

指定要捕获的 HTTP 请求参数数据（在一个逗号分隔列表中）。

默认

注释掉；常规参数。

示例

```
introscope.agent.transactiontracer.parameter.httprequest.parameters=parameter
1,parameter2
```

注释

IntroscopeAgent.profile 包含将此属性的值设为空值的已注释掉的语句。用户可以选择性地取消注释该语句并提供所需的参数名称。

introscope.agent.transactiontracer.parameter.httpsession.attributes

指定要捕获的 HTTP 会话属性数据（在一个逗号分隔列表中）。

默认

注释掉；常规参数。

示例

```
introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

注释

IntroscopeAgent.profile 包含将此属性的值设为空值的已注释掉的语句。用户可以选择性地取消注释该语句并提供所需的参数名称。

introscope.agent.transactiontracer.userid.key

用户定义的键字符串。

默认

注释掉；常规参数。

示例

```
#introscope.agent.transactiontracer.parameter.httpsession.attributes=attribute1,attribute2
```

注释

IntroscopeAgent.profile 包含将此属性的值设为空值的已注释掉的语句。如果在您的环境中使用 *HttpServletRequest.getHeader* 或 *HttpServletRequest.getValue* 访问用户 ID，则用户可以选择取消注释该语句并提供正确的值。

更多信息请参阅 introscope.agent.transactiontracer.userid.method (see page 302)。

introscope.agent.transactiontracer.userid.method

指定返回用户 ID 的方法。代理配置文件针对三个允许值中的每一个都包含一个已注释掉的属性定义。

基于 getRemoteUser、getHeader 或 getValue 是否访问了用户 ID 来取消注释相应语句。

属性设置

可用的值有：

- *HttpServletRequest.getRemoteUser*
- *HttpServletRequest.getHeader*
- *HttpServletRequest.getValue*

默认

注释掉；请参见上面的选项。

示例

IntroscopeAgent.profile 包括三个允许值中每一个的已注释掉的属性定义。可以取消注释要使用的属性。

```
introscope.agent.transactiontracer.userid.method=HttpServletRequest.getRemoteUser
#introscope.agent.transactiontracer.userid.method=HttpServletRequest.getHeader
#introscope.agent.transactiontracer.userid.method=HttpSession.getValue
```

introscope.agent.transactiontrace.componentCountClamp

限制事务跟踪中允许的组件数量。

默认

5000

重要信息！如果限定大小增加，则需要更多内存。在极个别情况下，可能需要调整 JVM 的最大堆大小，否则托管应用程序可能会耗尽内存。

示例

```
introscope.agent.transactiontrace.componentCountClamp=5000
```

注释

- 将在代理中丢弃任何超出限定值的事务跟踪，并在代理日志文件中记录一条警告消息。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。
- 到达设置的限制时，日志中将显示警告，跟踪将停止。
- 零**不是**有效值。请**不要**设置 introscope.agent.transactiontrace.componentCountClamp=0。

introscope.agent.crossprocess.compression

使用此属性可减小跨进程事务跟踪数据的大小。

属性设置

lzma、gzip、none

默认

lzma

示例

```
introscope.agent.crossprocess.compression=lzma
```

注释

- 此选项将增加代理 CPU 开销，但会减小跨进程标头的大小。
- *lzma* 压缩比 *gzip* 压缩更加高效，但可能会占用更多的 CPU。
- .NET 代理不支持 *gzip* 选项，因此，如需具备互操作性，请不要使用 *gzip*。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.crossprocess.compression.minlimit

使用此属性可设置要应用压缩的跨进程参数数据的最小长度。

属性设置

可以设置的值范围是 0 到 introscope.agent.crossprocess.correlationid.maxlimit (see page 306) 中设置的最大总限制的两倍。

如果设置为小于默认值 1500，则压缩将更加频繁地运行并消耗更多 CPU。在正常情况下，默认设置 1500 通常不会对 CPU 开销造成影响。

默认

1500

示例

```
introscope.agent.crossprocess.compression.minlimit=1500
```

注释

- 与上面的 introscope.agent.crossprocess.compression 属性一起使用。
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.crossprocess.correlationid.maxlimit

允许的跨进程参数数据的最大大小。

如果跨进程参数数据的总大小超过了此限制，即使应用了压缩，一些数据仍将被丢弃，并且某些跨进程关联功能将无法正常工作。

但是，此设置可以避免用户事务由于标头太大而导致网络传输失败。

默认

4096

示例

```
introscope.agent.crossprocess.correlationid.maxlimit=4096
```

注释

- 与上述 *introscope.agent.crossprocess.compression* 和 *introscope.agent.crossprocess.compression.minlimit* 属性一起使用
- 此属性是动态属性。您可以在运行时更改此属性的配置，更改会自动生效。

introscope.agent.transactiontracer.sampling.enabled

取消注释以下属性可禁用事务跟踪器采样。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.transactiontracer.sampling.enabled=false
```

注释

对此属性所做的更改可立即生效，不需要重新启动托管应用程序。

introscope.agent.transactiontracer.sampling.perinterval.count

此属性通常在企业管理器中进行配置。在代理中配置此属性将禁用在企业管理器中的配置。请参阅*《CA APM 配置和管理指南》*以获取更多信息。

默认

1

示例

```
introscope.agent.transactiontracer.sampling.perinterval.count=1
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.transactiontracer.sampling.interval.seconds

此属性通常在企业管理器中进行配置。在代理中配置此属性将禁用在企业管理器中的配置。

注意：有关详细信息，请参阅*《CA APM 配置和管理指南》*。

默认

120

示例

```
introscope.agent.transactiontracer.sampling.interval.seconds=120
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.transactiontrace.headFilterClamp

指定标头筛选中允许的最大组件深度。标头筛选会检查事务的开头，以便潜在地收集整个事务。标头筛选将检查每个组件，直到第一个出现问题的组件退出。如果不进行限定，在具有非常深的调用堆栈的事务中会出现问题。通过强制代理仅查询到某个固定深度，限定值可以限制该行为对内存和 CPU 使用率的影响。

默认

30

警告！ 如果限定大小增加，则需要更多内存。将影响垃圾回收行为，从而在应用程序范围内对性能造成影响。

示例

```
introscope.agent.transactiontrace.headFilterClamp=30
```

注释

- 对此属性所做的更改可立即生效，不需要重新启动托管应用程序。
- 将不再检查深度超过限定的事务跟踪是否存在收集；*除非*可使用一些其他机制（如采样或用户启动事务跟踪）选择要收集的事务。

introscope.agent.ttClamp

此属性限制代理在每个报告周期内报告的事务数量。

属性设置

整数。

默认

50

示例

```
introscope.agent.ttClamp=50
```

注释

- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。
- 如果未设置此属性（为空），则值默认设置为 200。

URL 分组

以下属性用于为前端度量标准配置 URL 组：

- introscope.agent.urlgroup.keys (see page 309)
- introscope.agent.urlgroup.group.default.pathprefix (see page 309)
- introscope.agent.urlgroup.group.default.format (see page 309)

有关详细信息，请参阅使用 URL 组 (see page 153)。

introscope.agent.urlgroup.keys

前端命名的配置设置。

默认

default

示例

```
introscope.agent.urlgroup.keys=default
```

注释

如果某个 URL 地址属于两个 URL 组，则 URL 组的键在此属性中的排列顺序很重要。狭义模式所定义的 URL 组应先于广义模式所指定的 URL 组。

例如，如果具有 alpha 键的 URL 组包含单个地址，而具有 beta 键的 URL 组包含网段上的所有地址（该网段包含第一个 URL 组中的地址），则 alpha 在 keys 参数中应先于 beta。

introscope.agent.urlgroup.group.default.pathprefix

前端命名的配置设置。

默认

*

示例

```
introscope.agent.urlgroup.group.default.pathprefix=*
```

introscope.agent.urlgroup.group.default.format

前端命名的配置设置。

默认

default

示例

```
introscope.agent.urlgroup.group.default.format=default
```

WebSphere PMI

以下属性用于配置 WebSphere PMI 度量标准：

- introscope.agent.pmi.enable (see page 311)
- introscope.agent.pmi.enable.alarmManagerModule (see page 311)
- introscope.agent.pmi.enable.beanModule (see page 312)
- introscope.agent.pmi.enable.cacheModule (see page 312)
- introscope.agent.pmi.enable.connectionPoolModule (see page 313)
- introscope.agent.pmi.enable.hamanagerModule (see page 313)
- introscope.agent.pmi.enable.j2cModule (see page 314)
- introscope.agent.pmi.enable.jvmpiModule (see page 314)
- introscope.agent.pmi.enable.jvmRuntimeModule (see page 315)
- introscope.agent.pmi.enable.objectPoolModule (see page 315)
- introscope.agent.pmi.enable.orbPerfModule (see page 316)
- introscope.agent.pmi.enable.schedulerModule (see page 316)
- introscope.agent.pmi.enable.servletSessionsModule (see page 317)
- introscope.agent.pmi.enable.systemModule (see page 317)
- introscope.agent.pmi.enable.threadPoolModule (see page 318)
- introscope.agent.pmi.enable.transactionModule (see page 318)
- introscope.agent.pmi.enable.webAppModule (see page 319)
- introscope.agent.pmi.enable.webServicesModule (see page 319)
- introscope.agent.pmi.enable.wlmModule (see page 320)
- introscope.agent.pmi.enable.wsgwModule (see page 320)
- introscope.agent.pmi.filter.objrefModule (see page 321)

对于 WebSphere 安装，这些属性位于 *IntroscopeAgent.websphere.profile* 文件，或默认的代理配置文件中。

introscope.agent.pmi.enable

启用从 WebSphere PMI 收集数据。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.pmi.enable=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.alarmManagerModule

设置为 true 时，启用 PMI 报警管理器数据收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.alarmManagerModule=false
```

注释

- 必须在 WebSphere 中打开该报警管理器数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.beanModule

启用 PMI Bean 数据收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.beanModule=false
```

introscope.agent.pmi.enable.cacheModule

设置为 true 时，启用 PMI 缓存数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.cacheModule=false
```

注释

- 必须在 WebSphere 中打开缓存数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.connectionPoolModule

启用 PMI connectionPool 数据的收集。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.pmi.enable.connectionPoolModule=true
```

introscope.agent.pmi.enable.hamanagerModule

设置为 true 时，启用 PMI 管理器数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.hamanagerModule=false
```

注释

- 必须在 WebSphere 中打开管理器数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.j2cModule

设置为 true 时，启用 PMI J2C 数据的收集。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.pmi.enable.j2cModule=true
```

注释

- 必须在 WebSphere 中打开 J2C 数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.jvmpiModule

启用 PMI JVM PI 数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.jvmpiModule=false
```

注释

要向此模块提供数据，必须在 WebSphere 中启用 JVMPI。

introscope.agent.pmi.enable.jvmRuntimeModule

启用 PMI JVM 运行时数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.jvmRuntimeModule=false
```

注释

- 要向此模块提供数据，必须在 WebSphere 中启用 JVMPI。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.objectPoolModule

设置为 true 时，启用 PMI 对象池数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.objectPoolModule=false
```

注释

- 必须在 WebSphere 中打开对象池数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.orbPerfModule

设置为 true 时，启用 PMI orbPerf 数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.orbPerfModule=false
```

注释

- 必须在 WebSphere 中打开 orbPerf 数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.schedulerModule

设置为 true 时，启用 PMI 排定程序数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.schedulerModule=false
```

注释

- 必须在 WebSphere 中打开排定程序数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.servletSessionsModule

启用 PMI servletSessions 数据的收集。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.pmi.enable.servletSessionsModule=true
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.systemModule

设置为 true 时，启用 PMI 系统数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.systemModule=false
```

注释

- 必须在 WebSphere 中打开系统数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.threadPoolModule

设置为 true 时，启用 PMI 线程池数据的收集。

属性设置

True 或 False

默认

True

示例

```
introscope.agent.pmi.enable.threadPoolModule=true
```

注释

- 必须在 WebSphere 中打开线程池数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.transactionModule

启用 PMI 事务数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.transactionModule=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.webAppModule

启用 PMI webApp 数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.webAppModule=false
```

注释

您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.webServicesModule

设置为 true 时，启用 PMI Web 服务数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.webServicesModule=false
```

注释

- 必须在 WebSphere 中打开 Web 服务数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.wlmModule

设置为 true 时，启用 PMI WLM 数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.wlmModule=false
```

注释

- 必须在 WebSphere 中打开 WLM 数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.enable.wsgwModule

设置为 true 时，启用 PMI WSGW 数据的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.enable.wsgwModule=false
```

注释

- 必须在 WebSphere 中打开 WSGW 数据类别，这些数据才能像 Introscope 数据一样可见。
- 您必须重新启动托管应用程序，对此属性所做的更改才能生效。

introscope.agent.pmi.filter.objrefModule

控制硬编码筛选器。

objref 筛选器筛选出以“@xxxxx”结尾的名称，其中“xxxxx”是数字字符串。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.pmi.filter.objrefModule=false
```

注释

您必须重新启动托管应用程序，对该属性所做的更改才能生效。

WLDF 度量标准

以下属性用于配置 WLDF 度量标准：

- introscope.agent.wldf.enable (see page 321)

introscope.agent.wldf.enable

该属性可启用对 WLDF 度量标准的收集。

属性设置

True 或 False

默认

False

示例

```
introscope.agent.wldf.enable=false
```

附录 B： 检测的备选方式

本节介绍了无法使用 JVM AutoProbe 时检测应用程序的备选方式。CA Technologies 建议在任何情况下优先使用 JVM AutoProbe，而不是本节中所述的备选方式。但是，如果无法对特定应用程序服务器使用 JVM AutoProbe，您可以使用本节中的说明检测应用程序。

此部分包含以下主题：

其他应用程序服务器上的 Java 代理部署

JVM AutoProbe 是检测应用程序的最常用方法。CA Technologies 强烈建议使用 JVM AutoProbe 检测您的应用程序。

但是，如果是在以下应用程序服务器上运行 JVM 1.4 或更低版本，您可以使用应用程序服务器 AutoProbe：

- Sun ONE 7.0

  应用程序服务器 AutoProbe 仅在 Sun ONE 版本 7 应用程序服务器上受支持。

- Oracle 10g 10.0.3

  应用程序服务器 AutoProbe 仅在 Oracle 10g 10.0.3 版本的应用程序服务器上受支持。

- WebSphere 或 WebLogic。

重要信息！ 以下平台不支持应用程序服务器 AutoProbe：

- 超过 1.5 版本的任何 JVM
- OS/400

重要信息！仅使用一种方法来检测应用程序。如果已使用 JVM AutoProbe，请不要再使用应用程序服务器 AutoProbe。

在启动应用程序服务器时，请勿使用连字符 (-) 作为类名的标识符。CA Introscope® 无法解析该字符，使用它可能会导致代理日志中出现类加载错误。

详细信息：

将 Sun ONE 配置为使用 AutoProbe (p. 324)
将 Oracle 配置为使用 AutoProbe (p. 325)
将 IBM WebSphere 配置为使用 Java 代理 (p. 42)
将 Oracle WebLogic 配置为使用 Java 代理 (p. 37)

将 Sun ONE 配置为使用 AutoProbe

适用于：Sun ONE 7.0

您可以将 Sun ONE 安装配置为使用 AutoProbe 检测应用程序。

请执行以下步骤：

注意：以下 .xml 示例中使用的“...”表示信息在 .xml 代码中未显示出来。此信息与示例无关。

1. 以管理员或 Root 用户的身份登录。
2. 导航到以下位置并打开 server.xml 文件：
   <Sun ONE 安装目录>/domains/*domain1/server1/config/*

   注意：项分隔符是冒号 (:)。
3. 将 wily/Agent.jar 的完整路径添加到 server.xml 文件中 java-config 元素的“server-classpath”属性中。例如：
   ```
   <java-config ...
   server-classpath="/sw/sun/sunone7/wily/Agent.jar:..." ...>
   ```
4. 导航到 java-config 元素：
   - 添加 bytecode-preprocessors 属性并将其值设置为 com.wily.introscope.api.sun.appserver.SunONEAutoProbe。

     例如：
     ```
     <java-config ...
     bytecode-preprocessors="com.wily.introscope.api.sun.appserver.SunONEA
     utoProbe">
     ```

- 添加 jvm-options 元素并定义代理配置文件的位置。定义 com.wily.introscope.agentProfile 或 com.wily.introscope.agentResource。

  com.wily.introscope.agentProfile 的示例如下：

  ```
  <java-config ...>
  ...
  <jvm-options>-Dcom.wily.introscope.agentProfile=/sw/sun/sunone7/wily/
  core/config/IntroscopeAgent.profile </jvm-options>
  </java-config>
  ```

  com.wily.introscope.agentResource 的示例如下：

  ```
  <java-config ...>
  ...
  <jvm-options>-Dcom.wily.introscope.agentResource=<virtual path
  to>/IntroscopeAgent.profile</jvm-options>
  </java-config>
  ```

- （可选）如果已配置 com.wily.introscope.agentResource，请将资源文件添加到服务器类路径。

5. 通过配置跟踪器组来收集 servlet 数据。

将 Oracle 配置为使用 AutoProbe

适用于：Oracle 10g 10.0.3

您可以将 Oracle 安装配置为使用 AutoProbe 检测应用程序。

请执行以下步骤：

1. 将 Agent.jar 添加到应用程序服务器类路径。
2. 将系统属性 oracle.classpreprocessor.classes 的值设置为 com.wily.introscope.api.oracle.OracleAutoProbe。
3. 将系统属性 oracle.j2ee.class.preprocessing 的值设置为 true。
4. 在命令行上运行以下命令：

   -Dcom.wily.introscope.probebuilder.oracle.enable=true

5. 使用以下命令重新启动 Oracle Application Server 10g：

```
java
-Doracle.classpreprocessor.classes=com.wily.introscope.api.oracle.OracleA
utoProbe -Doracle.j2ee.class.preprocessing=true
-Dcom.wily.introscope.probebuilder.oracle.enable=true -classpath
oc4j.jar:<wily 安装目录的路径>/wily/Agent.jar com.evermind.server.OC4JServer
-config <oracle 安装目录的路径>/config/server.xml
```

重要信息！ Windows 主机上使用 Sun JDK 1.42 运行 Oracle 10g Release 2 的用户必须使用 ^（插入符号）字符来转义正斜杠。例如：

```
-Xbootclasspath^/p:<IntroscopeAgent.jar 路径>
```

6. 通过配置跟踪器组来收集 servlet 数据。

详细信息：

配置 HTTP servlet 跟踪 (p. 327)

配置 WebLogic Server

您可以配置 WebLogic Server 以使用 AutoProbe 检测应用程序。

请执行以下步骤：

1. 编辑应用程序启动脚本（如 *startMedRecServer.cmd*）中的类路径以包含 wily/Agent.jar 文件。
2. 使用 -D 选项在 Java 命令行上设置应用程序启动脚本中的以下属性，以激活 Introscope AutoProbe：

```
-Dweblogic.classloader.preprocessor=
com.wily.introscope.api.weblogic.PreProcessor
```

3. 通过配置 HTTP servlet 跟踪 (see page 327)来配置跟踪器组以收集 servlet 数据。

配置 HTTP servlet 跟踪

在将 AutoProbe 与应用程序服务器结合使用以检测应用程序之前，必须在 *toggles-full.pbd* 和 *toggles-typical.pbd* 文件中配置跟踪器组。这将允许收集 Servlet 数据。

您将关闭一个跟踪器组，而打开另一个跟踪器组。

配置 HTTP servlet 跟踪

1. 导航到 *<your-application-server-home>/wily/core/config/toggles-full.pbd* 文件并将其打开。
2. 转至 PBD 的“*HTTP Servlets Configuration*”部分。
3. 通过在行首放置井号**关闭** *HTTPServletTracing* 跟踪器组。例如：
   `#TurnOn: HTTPServletTracing`
4. 通过删除行首的井号**打开** *HTTPAppServerAutoProbeServletTracing* 跟踪器组。例如：
   `TurnOn: HTTPAppServerAutoProbeServletTracing`
5. 对 *<your-app-server-home>/wily/core/config/toggles-typical.pbd* 文件重复步骤 2-4。

创建 AutoProbe 连接器文件

降级的 JVM AutoProbe 方法需要具有一个连接器 .jar 文件才能正常运行。要创建 AutoProbe 连接器，请按以下过程操作。如果您的 JVM 版本是 1.5，请按照 JVM AutoProbe 中的说明进行操作。

请执行以下步骤：

1. 将工作目录更改为安装目录下的 wily/connectors。
2. 使用以下命令之一运行“创建 AutoProbe 连接器”工具：
   - 使用正在运行该工具的 JVM 指定 JVM：

     ```
     java -jar CreateAutoProbeConnector.jar -current
     ```
   - 通过在命令行中传递 JVM 目录来指定 JVM：

     ```
     java -jar CreateAutoProbeConnector.jar -jvm <directory>
     ```

   输出是具有以下形式的文件：wily/connectors/AutoProbeConnector.jar

3. （可选）重命名创建的 .jar 文件，使其更易于管理并且获得普遍接受。例如：
   - wily/connectors/AutoProbeConnector131_02_Sun.jar
   - wily/connectors/AutoProbeConnector130_IBM.jar

详细信息：

配置 HTTP servlet 跟踪 (p. 327)

为 JVM 运行 AutoProbe 连接器

在为 Sun 或 IBM JVM 创建 AutoProbe 连接器之后，可运行创建的文件来检测应用程序。连接器的运行方式取决于您使用的应用程序服务器。有关详细信息，请参阅适用于您所用应用程序服务器的部分。

为 SAP J2EE 6.20 运行 AutoProbe 连接器

1. 打开文件：
   <drive>:\usr\sap\<J2EE_ENGINE_ID>\j2ee\j2ee_<INSTANCE>\cluster\server\cmdline.properties
2. 将以下命令添加到 **JavaParameters** 部分：
   ```
   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>
   -Dcom.wily.introscope.agent.agentName=<yourAgentName>
   ```
   例如：
   ```
   Xbootclasspath/p:C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/connectors/AutoProbeConnector.jar;C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/Agent.jar
   -Dcom.wily.introscope.agentProfile=C:/usr/sap/P602/j2ee/j2ee_00/ccms/wily/core/config/IntroscopeAgent.profile
   ```
3. 重新启动 SAP 服务器。

为 NetWeaver 04/SAP J2EE 6.40 运行 AutoProbe 连接器

1. 运行 **SAP J2EE Configtool**。
2. 选择要修改的服务器。
3. 在“**Java 参数**”字段中添加以下新 java 参数：
   -Xbootclasspath/p:PathToAutoProbeConnectorJar;PathToAgentJar
   -Dcom.wily.introscope.agentProfile=<path-to-IntroscopeAgent.profile>

例如：

```
Xbootclasspath/p:D:/usr/sap/ccms/wily/connectors/AutoProbeConnector.jar;D
:/usr/sap/ccms/wily/Agent.jar
-Dcom.wily.introscope.agentProfile=D:/usr/sap/ccms/wily/core/config/Intro
scopeAgent.profile
```

注意：对于 Windows 中的 NetWeaver 6.40，这些 java 参数中的斜杠必须是正斜杠。

4. 单击**“磁盘”**以进行保存。
5. 对每台服务器重复步骤 2—4。
6. 重新启动 SAP 服务器。
7. 要验证是否已进行 Configtool 更改，请打开以下文件：
   <drive>:\usr\sap\ccms\P66\JC00\j2ee\cluster\instance.properties
8. 查找以 *ID<server_id>.JavaParameters* 开头的行，并确认其中包含您输入的行。

为 Sun ONE 运行 AutoProbe 连接器

1. 以管理员或 Root 用户的身份登录。

   您必须以管理员或 Root 用户的权限登录，才能向 Sun ONE 7.0 的启动脚本中添加 Introscope 信息。

2. 打开位于以下位置的 server.xml 文件：
   <SunONE install dir>/domains/domain1/server1/config/
3. 将以下行添加到 *server.xml* 文件中：

   ```
   <jvm-options>
   -Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar
   </jvm-options>
   ```

   项分隔符为冒号 (:)。例如：

   ```
   <jvm-options>
   -Xbootclasspath/p:/sw/sun/sunone7/wily/connectors/AutoProbeConnector.jar:
   /sw/sun/sunone7/wily/Agent.jar
   </jvm-options>
   ```

为 Oracle 10g 运行 AutoProbe 连接器

- 要运行 AutoProbe 连接器，请修改启动类路径：

  ```
  -Xbootclasspath/p:wily/connectors/AutoProbeConnectorJar:PathToAgentJar
  ```

Windows 主机上使用 Sun JDK 1.42 运行 Oracle 10g Release 2 的用户必须使用 ^（插入符号）字符来转义正斜杠。例如：

```
-Xbootclasspath^/p:<IntroscopeAgent.jar path>
```

不同版本的 WebLogic 使用不同版本的 Java 来运行。如果您使用的是 Java 1.4 或更低版本，请使用以下步骤来运行 AutoProbe 连接器。如果您使用的是 Java 1.5 或更高版本，请参阅 JVM AutoProbe 了解详细信息。

为 WebLogic 运行 AutoProbe 连接器

1. 在应用程序启动脚本中编辑启动类路径，以包括您使用以下命令创建的 *AutoProbeConnector.jar*（如 *startMedRecServer.cmd*）：

   -Xbootclasspath/p:PathToAutoProbeConnectorJar:PathToAgentJar

   在脚本结尾的 *JAVA_VM* 和 *JAVA_OPTIONS* 之后向最终的启动命令中添加 -X 开关。下面的摘录内容显示插入开关的正确位置：

   ```
   "$JAVA_HOME/bin/java" ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}
   -Xbootclasspath/p:${WL_HOME}/wily/connectors/AutoProbeConnector.jar:${WL_
   HOME}/wily/Agent.jar
   -Dweblogic.Name=${SERVER_NAME}
   -Dweblogic.management.username=${WLS_USER}
   -Dweblogic.management.password=${WLS_PW}
   -Dweblogic.ProductionModeEnabled=${PRODUCTION_MODE}
   -Djava.security.policy="${WL_HOME}/server/lib/weblogic.policy"
   weblogic.Server
   ```

2. 如果您使用的不是默认启动类路径，请在自定义启动类路径开头添加 *Agent.jar* 和 *AutoProbeConnector.jar* 文件。

为具有 JRockit JVM 的 WebLogic 运行 AutoProbe 连接器

- 启动 JVM 时添加以下命令行选项：

  ```
  -Xbootclasspath/a:<PathToAgentJar>
  -Xmanagement:class=com.wily.introscope.api.jrockit.AutoProbeLoader
  ```

为其他应用程序服务器运行 AutoProbe 连接器

- 要运行 AutoProbe 连接器，请使用以下命令将 *Agent.jar* 和 AutoProbe 连接器添加到应用程序服务器启动类路径中：

  ```
  -Xbootclasspath/p:wily/connectors/AutoProbeConnector.jar:PathToAgentJar
  ```

示例：使用 Xbootclasspath 检测 WAS

此示例演示了如何使用 -Xbootclasspath 选项检测 WebSphere 应用程序服务器。使用此选项，您可以覆盖启动时由 JVM 默认加载的实体（类、jar、目录、zip）。为要检测的 JVM 创建 AutoprobeConnector.jar 文件，因为您不能直接对 agent.jar 文件使用 Xbootclasspath。

请执行以下步骤：

1. 找到 WebSphere 应用程序服务器使用的 Java 可执行文件，例如，在 AppServer/java/jre/bin 目录下。
2. 打开命令提示符并输入以下命令：

   ```
   cd <agent_install_dir>/wily/connectors
   <path_to_WAS>/AppServer/java/jre/bin/java -jar
   CreateAutoProbeConnector.jar -current
   ```

   此时将创建 AutoprobeConnector.jar。

3. 输入以下命令：

```
-Xbootclasspath/p:<创建的 Autoprobe jar 文件的路径>/AutoprobeConnector.jar:<
代理路径>/Agent[NoRedef].jar
-Dcom.wily.introscope.agentProfile=<代理路
径>/IntroscopeAgent[NoRedef].profile
```

-Xbootclasspath/p:

指定要附加在默认启动类路径之前的目录、JAR 存档和 ZIP 存档的路径，并以冒号分隔。覆盖启动时由 JVM 默认加载的实体。

注意：在 UNIX 系统上，请在 Xbootclasspath 中使用冒号 (:)，而在 Windows 系统上，请使用分号 (;)。

关于手工运行 ProbeBuilder

手工运行 ProbeBuilder 是一种非动态的应用程序检测方法。手工运行 ProbeBuilder 时，它会在应用程序服务器运行之前检测磁盘上的类。当环境不支持 AutoProbe 或您不愿意使用 AutoProbe 时，可以使用手工 ProbeBuilding。

手工 ProbeBuilding 不应与其他检测方法结合使用，并且*应作为最后的手段*。

此处的手工 ProbeBuilding 说明假设您已经执行了以下安装和配置任务：

1. 已安装 Java 代理。有关详细信息，请参阅安装 Java 代理。
2. 已配置 Java 代理连接属性。有关详细信息，请参阅连接到企业管理器 (see page 55)。
3. 已配置 Java 代理名称。有关详细信息，请参阅 Java 代理命名 (see page 117)。
4. 已配置 ProbeBuilder 的选项。有关详细信息，请参阅 AutoProbe 和 ProbeBuilding 选项 (see page 71)。

为 z/OS 上的 WebSphere 配置 AutoProbe

适用于：z/OS 上的 WebSphere 6.1 和 7.0

您可以将 z/OS 安装上的 WebSphere 配置为使用 AutoProbe 检测应用程序。有关 AutoProbe 的详细信息，请参阅 AutoProbe 和 ProbeBuilding 选项 (see page 71)。

注意：使用以下过程检测 z/OS 上的 WebSphere 7.0 不会提供与使用用于 z/OS 上的 WAS 7 的 JVM 1.5 AutoProbe 方法时一样详细的度量标准。例如，不检测线程度量标准级别。

重要信息！如果您正在使用 Java Agent 9.0 或更高版本监控 z/OS 上的 WebSphere 7.0，则应用程序服务器进程会反复重新启动。要避免此问题，请升级到 WAS 7.0 内部版本级别 7.0.0.8 或更高版本。

请执行以下步骤：

1. 在 WebSphere 中，启动管理员控制台。
2. 选择“应用程序服务器”><*您的服务器*>>“进程定义”。

   此时会列出“控项”和“从项”。
3. 单击“从项”，然后单击 JavaVirtualMachine。
4. 设置“常规 JVM 参数”字段，以指定类加载器插件以及 IntroscopeAgent.profile 文件的位置。设置以下条目之一：

   `com.wily.introscope.agentProfile`

   或

   `com.wily.introscope.agentResource`

   该参数具有以下值（在一个参数中设置若干个属性）：

   ```
   -Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
   -Dcom.wily.introscope.agentProfile=<IntroscopeAgent.profile 的路径>
   ```

   或

   ```
   -Dcom.ibm.websphere.classloader.plugin=com.wily.introscope.api.websphere.WASAutoProbe
   -Dcom.wily.introscope.agentResource=<包含 IntroscopeAgent.profile 的资源的路径>
   ```
5. 将 Agent.jar 文件放在 <*WebSphere 实例目录*>/lib/ext 目录中。

   注意：不要将 Agent.jar 文件放在 WebSphere 安装目录中。

   下面显示了错误目录和正确目录的示例：

   ```
   错误：/usr/lpp/zWebSphere/V5R0M0/lib/ext
   正确：/WebSphere/V5R0M0/AppServer/lib/ext
   ```
6. 确认在 ./wily 目录中新建的所有 CA Introscope® 文件和目录都可供 WebSphere 进程进行读取访问。
7. 确认所有 *.log 文件都对 WebSphere 进程拥有写入访问权限。Java 代理和 ProbeBuilder 将这些文件写入 ./wily 文件夹。这些文件包括：
   - 所有 CA Introscope® 文件和目录
   - <*WAS 实例目录*>/lib/ext 中的 CA Introscope® 文件

8. 重新启动 WebSphere 应用程序服务器。
9. 当 WebSphere 显示“open for e-business”时，请打开管理员控制台。

   度量标准将开始报告。
10. 为使 AutoProbe 在已启用 Java2 安全性的 WebSphere 环境中正确运行，请向 Java2 安全策略中添加权限 (see page 47)。
11. 通过配置 HTTP servlet 跟踪 (see page 327)来收集 servlet 数据。

详细信息：

AutoProbe 和 ProbeBuilding 选项 (p. 71)

附录 C： 使用 PBD Generator

您可以使用 PBD Generator 工具检测自定义 Java 类文件以供代理使用。

此部分包含以下主题：

关于 CA PBD Generator

PBD Generator 实用工具可通过注释 Java 代码所使用的 Javadoc 标记来创建 PBD 文件，以便于检测供 Java 代理使用的自定义 Java 类文件。

Generator 会检查一组 Java 源文件，并检测包含 Javadoc 标记 *@instrument* 的类中的方法。

使用 PBD Generator 工具，可执行以下操作：

- 自动构建 PBD 文件，避免出现手工创建 PBD 文件可能引入的潜在错误。
- 将 PBD 的生成集成到生成系统中，从而自动创建并更新 PBD 文件，并将所有更改合并到 Java 源中。

您需要使用 *PBDGenerator.jar* 文件将 PBD Generator 集成到 Apache Ant 目标，然后将其作为 Ant Javadoc 任务运行，来配置 PBD Generator。

配置 PBD Generator

此工具应合并到基于 Ant 的生成系统中，作为 Ant 目标中的一项 Javadoc 任务。

以下 Javadoc 任务示例说明了此工具在 Ant 中的使用：

```
<javadoc sourcepath="/src/engineering/products/introscope/source"
    destdir="/src/engineering/products/introscope/source/generatedpbd"
    maxmemory="512m"
    packagenames="com.wily.introscope.console.thornhill.ui.util"
    verbose="false"
    private="true">
  <doclet name="com.wily.util.build.javadoc.PBDInstrumentDoclet"
    path="/Wily/tools/WilyPBDGenerator.jar">
    <param name="-d"
    value="/src/engineering/products/introscope/source/generatedpbd"/>
  </doclet>
</javadoc>
```

必需的 PBD Generator 参数

以下关键 PBD Generator 参数是必需的：

sourcepath

Java 源树的根目录

destdir

将从工具中输出的 PBD 文件的目录路径

packagenames

为执行检测而要检查的 Java 软件包的逗号分隔列表

doclet path

包含此工具的 PBD Generator jar 文件的路径

param name="-d"

此参数的值必须与 *destdir* 的值相同

使用 PBD Generator

您必须先将特殊的 Javadoc 标记插入待检测的 Java 源文件中，然后才可以使用 PBD Generator。

JavaDoc 标记的语法如下：

```
@instrument <valid metric prefix> <optional tracer name>
```

其中：

<valid metric prefix> 是任意有效的 Introscope 度量标准前缀－不含冒号字符 (:) 的字符串。可接受竖线字符 (|)。

<optional tracer name> 可以是 BlamePointTracer、FrontendMarker 或 BackendMarker。如果缺少跟踪器名称，则默认值为 BlamePointTracer。

附录 D： 使用网络接口实用工具

可以使用网络接口实用工具来确定代理用于 Catalyst 集成的主计算机的网络接口名称值。

此部分包含以下主题：

确定网络接口名称 (p. 339)

确定网络接口名称

网络接口实用工具为 introscope.agent.primary.net.interface.name 属性提供名称和子接口值。在代理使用的同一 JVM 和应用程序服务器上运行该实用工具。

请执行以下步骤：

1. 从命令行导航至以下目录：

   <Agent_Home>/wily/tools

2. 运行以下命令来调用实用工具：

   ```
   java -jar NetInterface.jar
   ```

   浏览器会在“网络接口”选项卡上显示 Java 支持的网络接口名称列表。

详细信息：

配置可用网络的列表 (p. 233)