Java CAPS 管理和监视 API

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

文件号码 820–5473
2008 年 6 月

081016@21288

目录

Java CAPS 管理和监视 API 5
Java CAPS 管理客户机 6
JavaDoc 8
目标 8
API 使用入门 9
▼ 首先使用 API 创建应用程序 9
通过 API 连接到服务器 10
连接类型定义 11
CAPSManagementClientFactory 定义 12
警报管理 API 13
数据库支持 14
AlertConfigurationService 14
AlertNotificationService 21
服务 — JavaCAPSManagementAPI 23
管理服务 23
运行时管理服务 24
配置服务 27
部署服务 27
安装服务 28
性能测量服务 29
警报管理服务 31
警报监控服务 31
警报通知服务 32
警报配置服务 32
JMS 管理服务 34
日志管理服务 36
BPEL 管理服务 36
主数据管理 (Master Data Management, MDM) 服务 38

Sun 适配器管理服务 38
管理客户机的目标选项行为 39
使用 Java 代码样例编写 Java 代码以访问 API 40
设置数据库 40
▼ 使用 Enterprise Manager 设置数据库 40
▼ 使用脚本实用程序设置数据库 41
将 Oracle 和其他数据库用于警报持久性 42
▼ 为警报持久性设置 Oracle 数据库 42
Oracle 脚本示例 45
设置脚本引擎 47
下载、安装和设置脚本环境 47
设置脚本环境以调用 Java CAPS 管理和监视 API 48
▼ 修改 env.bat 中的环境变量 48
使用脚本语言执行 Java CAPS 管理和监视 API 48
执行管理服务 49
执行配置服务 50
执行部署服务 50
执行安装服务 51
执行运行时管理服务 51
执行 JMS 管理服务 52
执行 BPEL 管理服务 53
执行 HTTP 管理服务 53
执行通知服务 54
集成到 NetBeans IDE 中的 JRuby 54

Java CAPS 管理和监视 API

本主题提供了有关常见管理和监视任务的信息。在 Java™ Composite Application Platform Suite (Java CAPS) 安装期间，不会安装管理和监视 API；它们将打包为 `EM_API_KIT.zip`。此 ZIP 文件位于 `ESB_API_Kit.zip` 中。您需要将 `EM_API_KIT.zip` 解压缩到 Java CAPS 6 安装的根目录中。如果将该文件解压缩到其他目录中，则需要在 `env.bat` 中设置 JAVA_HOME、JAVA_CAPS 和 ENGINE_HOME 环境变量。有关如何执行此操作的信息，请参见第 48 页中的 “修改 env.bat 中的环境变量”。ZIP 文件中还包含 JavaDoc、Java 样例以及说明如何使用 API 的 Groovy 脚本样例。

如果您有任何问题，请参见 `http://goldstar.stc.com/support` 中的 Java CAPS Support Web 站点。

提示 – 要访问 Sun Developer Network (SDN) Web 站点中 HTML 格式的所有 Java CAPS 文档，请转至 `http://developers.sun.com/docs/javacaps/`。

您还可以通过单击 Enterprise Manager 中“帮助”访问 SDN Web 站点上的 Java CAPS 文档。

通过 Java CAPS Uploader 中的“帮助”选项卡可打开“使用 Java CAPS Uploader 安装 Java CAPS 组件”主题。

您需要了解的信息

- 第 6 页中的 “Java CAPS 管理客户机”
- 第 9 页中的 “API 使用入门”
- 第 10 页中的 “通过 API 连接到服务器”
- 第 13 页中的 “警报管理 API”
- 第 23 页中的 “服务 — JavaCAPSManagementAPI”
- 第 40 页中的 “使用 Java 代码样例编写 Java 代码以访问 API”
- 第 40 页中的 “设置数据库”
- 第 42 页中的 “将 Oracle 和其他数据库用于警报持久性”
- 第 47 页中的 “设置脚本引擎”

可以使用以下项来公开管理和监视：

- 通用 API

  可以从 Java Composite Application Platform Suite (Java CAPS) 安装程序中安装客户机 API 工具包，该工具包与 Groovy 脚本引擎、Groovy 库和样例打包在一起。

- JSR 223 脚本

  可以从为 Java 平台提供 JSR 223 脚本支持的任何语言中执行所有管理功能。目前，它包含 25 种不同的脚本语言，其中包括 Groovy、JRuby、Jython、JACL 等。

- Sun Java System Application Server 管理控制台

  Sun Java System Application Server 管理控制台中内置了所有 JBI 管理用例。这样，管理员便可使用浏览器远程管理其域运行时环境和组件。

- 命令行界面

  通过使用随 Sun Java System Application Server 提供的 AS Admin 命令行界面，您可以管理应用服务器环境。所有 JBI 管理用例已集成到 AS Admin 命令行界面中。此外，还提供了用于管理非 JBI 组件的命令行界面供您使用。

- Ant

  通过使用 asant，您可以在 Sun Java System Application Server 上运行 ant 脚本。所有 JBI 管理用例已集成到 Sun Java System Application Server 的 asant 中，尽管您可以使用任何 ant 脚本，但也可以执行这些管理用例。

- Java CAPS Enterprise Manager

  可以使用 Enterprise Manager 远程管理非 JBI 组件。

- NetBeans 管理插件

  通过使用随 NetBeans IDE 提供的 JBI 管理器，开发者可以管理 JBI 运行时组件容器和复合应用程序。通过使用复合应用程序项目系统 (Composite Application Project System, CASA)，开发者可以在开发期间部署和管理生命周期操作。

Java CAPS 管理客户机

本主题介绍了 Java CAPS 管理客户机以及将 API 集与基于 JBI 的 Java CAPS 运行时环境和不基于 JBI 的 Java CAPS 运行时环境结合使用的客户机。

- 脚本客户机 — 使用任何 JSR 223 脚本客户机，如 Groovy、JRuby、Jython 或 JACL。有关其他信息，请参见第 47 页中的 “设置脚本引擎”。

  注 – Java CAPS 发行版 6 软件包附带提供了一些 Groovy 样例。

- CLI 客户机 — 通过您自己的命令行界面客户机来执行这些 API，这些客户机是使用预定义的功能代码（首选 Java 代码）构建的。

- 其他 Web/GUI 客户机 — 通过自定义 Web/GUI 客户机来执行这些 API。

典型客户机用法示例

```
try {
  // Get the Management Client
  ManagementClient client = ManagementClientFactory.getInstance
  ("localhost", 4848, "admin", "adminadmin");

  // Get the Administration Service
  AdministrationService administrationService =
  client.getService(AdministrationService.class);
  // ... Use the Administration Service ...

  // Get the Configuration Service
  ConfigurationService configurationService =
  client.getService(ConfigurationService.class);
  // ... Use the Configuration Service ...

  // Get the Deployment Service
  DeploymentService deploymentService =
  client.getService(DeploymentService.class);
  // ... Use the Deployment Service ...

  // Get the Installation Service
  InstallationService installationService =
  client.getService(InstallationService.class);
  // ... Use the Installation Service ...

  // Get the Runtime Management Service
  RuntimeManagementService runtimeManagementService =
  client.getService(RuntimeManagementService.class);
  // ... Use the Runtime Management Service ...

  // Get the JMS Management Service
  JmsManagementService jmsManagementService =
  client.getService(JmsManagementService.class);
  // ... Use the JMS Management Service ...

  // Get the Alert Management Service
  AlertManagementService alertManagementService =
  client.getService(AlertManagementService.class);
  // ... Use the Alert Management Service ...

  // Get the Log Management Service
  LogManagementService logManagementService =
  client.getService(LogManagementService.class);
  // ... Use the Log Management Service ...
```

```
} catch (ManagementRemoteException exception) {
  // Format the exception and print it.
  String formattedExceptionResult=
  ManagementRemoteExceptionProcessor.processTaskException(exception);
  System.out.println(formattedExceptionResult);
}
```

JavaDoc

JavaDoc 包含 API 中的所有类的完整列表。Java CAPS 软件包以压缩文件形式附带提供了 JavaDoc。

目标

目标提供了管理操作的范围。通过将命令定向到多个目标，可以有效地扩大该命令的范围。如果指定多个目标，将分别报告每个目标是否成功。也就是说，不会“累积”所有目标上的操作结果以报告摘要状态。下表描述了每种目标类型的范围。

注 – 其中的两个目标选项名称为常量："domain" 和 "server"。它们表示一个运算符实例，可以将其替换为特定于当前模板的名称。

表 1 目标操作

目标名称	范围
domain	针对域本身执行命令。对于 JBI 目的，这相当于“添加到系统信息库”活动。
server	针对嵌入的 DAS 服务器实例执行命令。
群集名称	针对群集中的所有实例执行命令。
实例名称	针对单个独立实例执行命令。
群集实例名称	针对群集中的特定实例执行命令。

注 – 此软件包以压缩文件形式附带提供了与该主题有关的样例，例如，`AdministrationServiceSample.groovy`。

API 使用入门

用户和开发者可以使用 Java Composite Application Platform Suite (Java CAPS) API 来创建应用程序和 Web 页。

▼ 首先使用 API 创建应用程序

以下任务包含使用 Java CAPS API 创建应用程序所需的所有内容。如果使用在 Java CAPS 安装期间安装的 Sun Java System Application Server (SJSAS) 进行连接，则不需要使用工作目录中的任何其他 JAR 文件。不过，如果要进行远程连接，则需要使用工作目录中的以下 JAR 文件：

- 如果仅通过 RMI 协议连接（使用 JSR-160 MX URL），则需要使用以下 JAR 文件：

```
%CAPS_MANAGEMENT_HOME%\api\caps.management.client.jar;
%SJSAS_HOME%\jbi\lib\jbi-admin-common.jar;
%SJSAS_HOME%\lib\javaee.jar;
```

- 如果通过 HTTP/HTTPS 协议进行连接，则需要使用以下 JAR 文件：

```
%CAPS_MANAGEMENT_HOME%\api\caps.management.client.jar;
%SJSAS_HOME%\jbi\lib\jbi-admin-common.jar;
%SJSAS_HOME%\lib\javaee.jar;
%SJSAS_HOME%\lib\appserv-deployment-client.jar;
%SJSAS_HOME%\lib\appserv-ext.jar;
%SJSAS_HOME%\lib\appserv-rt.jar;
%SJSAS_HOME%\lib\jmxremote_optional.jar
```

1 使用此示例连接到 JMX URL

```
String hostName = "localhost";
int jrmpPortNumber = 8686;
String userName = "admin", password = "adminadmin";
boolean isRemoteConnection = true;
String jrmpURLString = "service:jmx:rmi:///jndi/rmi://" + hostName
     + ":" + jrmpPortNumber + "/jmxrmi";
CAPSManagementClient managementClient =
     CAPSManagementClientFactory.getInstance(jrmpURLString,
     userName, password, isRemoteConnection);
```

2 获取编写应用程序所需的服务，例如，AdministrationService。

```
// get services
          CAPSAdministrationService administrationService =
          managementClient.getService(CAPSAdministrationService.class);
```

3 在创建应用程序（如 JBIRuntime）后，调用该应用程序。

```
// use the service,
          System.out.println("The JBI Runtime is
          "+(administrationService.isJBIRuntimeEnabled()?
          "Enabled." : "NOT Enabled."));
```

通过 API 连接到服务器

目前，Java CAPS 提供了 7 个便于您使用 API 连接到 Sun Java System Application Server 的选项。

CAPSManagementClientFactory 客户机用法

选项 1：host, port, userName, password

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                                  4848,
                                                                  "admin",
                                                                  "adminadmin");
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 2：host, port, userName, password, connectionType

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                                  4848,
                                                                  "admin",
                                                                  "adminadmin",
                                                                  ConnectionType.HTTP);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 3：url, userName, password, isRemoteConnection

```
/** Only relevant piece of code is shown */
ManagementClient client = CAPSManagementClientFactory.getInstance(
       "service:jmx:rmi:///jndi/rmi://localhost:22287/management/rmi-jmx-  connector",
       "admin", "adminadmin", false);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 4：MBeanServerConnection

```
/** Only relevant piece of code is shown */
MBeanServerConnection connection = ... // Get the MBeanServerConnection
ManagementClient client = CAPSManagementClientFactory.getInstance(connection);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 5：MBeanServerConnection, isRemoteConnection (true/false)

```
/** Only relevant piece of code is shown */
MBeanServerConnection connection = ... // Get the MBeanServerConnection
ManagementClient client = CAPSManagementClientFactory.getInstance(connection, true);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 6：host, port, userName, password, connectionType, promtUserForMasterPassword(true/false)

```
/** Only relevant piece of code is shown */
ManagementClient client =
CAPSManagementClientFactory.getInstance("127.0.0.1",
                                                    8686,
                                                    "admin",
                                                    "adminadmin",
                                                    ConnectionType.JRMP,



                                                false);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

选项 7：hostName, portNumber, userName, password, connectionType, keyStoreFileLocation, masterPassword, promptForMasterPassword (true/false)

```
/** Only relevant piece of code is shown */
ManagementClient client =
CAPSManagementClientFactory.getInstance
                                                    ("127.0.0.1",
                                                    8686,
                                                    "admin",
                                                    "adminadmin",
                                                    ConnectionType.JRMP,
                                                    "C:/CAPS6/Glassfish/
                                                    domains/domain1/
                                                    config/keystore.jks",
                                                    "changeit",
                                                    true);
// ... Invoke operations on the returned CAPSManagementClient object ...
```

连接类型定义

```
public enum ConnectionType {
    HTTP("s1ashttp"),
    HTTPS("s1ashttps"),
    JRMP("jmxrmi"),
```

```
    IIOP("iiop");

    // ... Implementation ...
    /** @return the protocol */
    public String getProtocol();
    /** @return the protocol description */
    public String getDescription();
}
```

CAPSManagementClientFactory 定义

```
/** Only relevant piece of code is shown */
public class CAPSManagementClientFactory {

    // Option 1 - host, port, userName, password
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
                String userName, String password) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 2 - host, port, userName, password, connectionType
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
                String userName, String password, ConnectionType connectionType)
                throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 3 - url, userName, password, isRemoteConnection
    public static CAPSManagementClient getInstance(String url, String userName,
                String password, boolean isRemoteConnection) throws
ManagementRemoteException {

        // ... Implementation ...
    }


    // Option 4 - MBeanServerConnection
    public static CAPSManagementClient getInstance(MBeanServerConnection connection)
            throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 5 - MBeanServerConnection, isRemoteConnection
    public static CAPSManagementClient getInstance(MBeanServerConnection connection,
```

```
                boolean isRemoteConnection) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 6 - host, port, userName, password, connectionType,
    promtUserForMasterPassword(true/false)
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
            String userName, String password, ConnectionType connectionType,
            boolean promptForPasswordFlag) throws ManagementRemoteException {

        // ... Implementation ...
    }

    // Option 7 - hostName, portNumber, userName, password, connectionType,
    keyStoreFileLocation,
    //            masterPassword, promptForMasterPassword (true/false)
    public static CAPSManagementClient getInstance(String hostName, int portNumber,
            String userName, String password, ConnectionType connectionType,
            String trustStoreFilePath, String trustStorePassword,
            boolean promptForPasswordFlag) throws ManagementRemoteException {

        // ... Implementation ...
    }
}
```

注 – 此软件包以压缩文件形式附带提供了与该主题有关的 Java CAPS 管理 API 样例，例如，`AdministrationServiceSample.groovy`。

警报管理 API

发行版 6 增加了警报管理服务功能，以便为用户提供更多的控制。现在，该服务拆分为三个服务：

- 警报监控服务，可用于在 Enterprise Manager 中监视和控制事件数据库中存储的警报
- Sun Java System Application Server (SJSAS) 上的警报配置服务，对事件管理组件中生成的警报的持久性提供更多控制。
- SJSAS 上的警报通知服务，对从事件管理组件传送到脚本客户机的警报类型提供更多控制

注意 – 启用持久性和禁用日志功能表示，在传送到所有当前客户机后，完成事件删除操作。

启用持久性和启用日志功能表示不删除事件。切记，如果启用了可删除策略，则会删除这些事件。

数据库支持

警报管理 API 支持 5 个数据库：

- Derby，随 Sun Java System Application Server 安装提供
- Oracle
- Sybase
- DB2
- PointBase

注 – 在使用 Java CAPS API 之前，您必须先启动数据库。

AlertConfigurationService

通过使用 AlertConfigurationService，Enterpriser Manager 和其他管理客户机可以管理和控制将警报从应用程序或 JBI 组件传送到其客户机的可靠性。Enterprise Manager 在启动后始终接收警报；而所有其他客户机需要进行注册才能接收警报。需要管理的配置包括：在数据库中启用和禁用警报持久性以及管理警报删除的策略。

注 – 对于早期版本的 Java CAPS 用户，他们必须启用持久性和日志功能才能获得以前使用的功能。

持久保存的警报删除策略

复合删除策略是通过控制以下三项内容来定义的：

- 警报计数

  可以持久保存的最大警报数。在执行该策略时，将删除最早生成的警报。
- 警报存留期

  可以持久保存警报的最长时间（即“存留期”）。
- 警报级别

  每个生成的警报都有一个关联的优先级，从低到高依次为：
  - INFO

- WARNING
- MINOR
- MAJOR
- CRITICAL
- FATAL

将删除优先级低于所定义级别的所有警报。

该实现基于“先到先得”策略，如警报配置服务 API 中所述。该策略是定义的所有策略项的组合(“与”条件)，即使执行的某个策略项的结果取消执行下一个策略项，也会应用该策略。

默认情况下，将持久性设置为“未启用”。如果将持久性默认设置更改为“已启用”，但未“启用”日志功能，则不需要设置该策略。不过，如果将日志功能设置为“启用”，则可以为警报计数、警报存留期和警报级别设置该策略。

警报配置服务 API

```
* Enable alerts persistence in the alerts database.
     * enabling it allow for reliable alerts delivery in
     * case of delivery channel failure or Application
     * server restart.
     *
     * @param enableJournaling
     *                    true - will prevent the system from
     *                           removing alerts after they
     *                           are delivered. The alert stay
     *                           in the database until the
     *                           user removes them.
     *                    false - The system will remove the alert
     *                            upon acknowledgment from
     *                            the reliable client in case one was
     *                            setup or upon send the alert to
     *                            all the non reliable client/s.
     * @throws ManagementRemoteException if JMX related exception is
     *         thrown or the list of target name is null or empty.
     */
     public void enableAlertsPersistence(Boolean enableJournaling) throws
     *        ManagementRemoteException;

     /**
     * Disable alerts persistence in the alerts database.
     *
     * @throws ManagementRemoteException if JMX related exception is
     *         thrown or the list of target name is null or empty.
     */
     public void disableAlertsPersistence() throws ManagementRemoteException;
```

```
 /**
 * @return the last setting of alert persistence enabling operation.
 *         true if enable otherwise false.
 * @throws ManagementRemoteException if JMX related exception.
 */
 public  Boolean isAlertsPersistenceEnabled()throws
 * ManagementRemoteException;

/**
 * @return the last setting of alert journal enabling operation.
 *         true if enable otherwise false.
 * @throws ManagementRemoteException if JMX related exception.
 */
public Boolean isAlertsJournalEnabled()throws ManagementRemoteException;

/**
 * set the JNDI name of the data source database to be used
 * for persistence. if not provided at least once the persistence
 * will be disabled even if enableAlertsPersistence was set to true.
 *
 * @param jndiName - of the data source to be used in conjunction with
 *   enableAlertsPersistence set to true

 * @throws ManagementRemoteException if JMX related exception is
 *         thrown orjndiName parameter is null or empty.
 */
public void setPersistenceDataSourceJndiName(String jndiName) throws
* ManagementRemoteException;
/**

 * @return the last set JNDI name for the alert persistence data source.
 * @throws ManagementRemoteException
 */
public String  getPersistenceDataSourceJndiName() throws ManagementRemoteException;

/**
 * set the database type to be used for persistence.
 * Derby is the assumed default database. If different database is
 * used this method should be called prior to enabling the persistence.
 *

 * @param dbtype - one of the predefined typed defined in {@link
 *             com.sun.caps.management.api.alerts.AlertPersistenceDBType}
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistenceDataBaseType(AlertPersistenceDBType dbtype) throws
 *             ManagementRemoteException;
```

```
/**
 * @return The return value represent the last set DB type
 *             {@link com.sun.caps.management.api.alerts.AlertPersistenceDBType}
 *             for each.
 * @throws ManagementRemoteException if JMX related exception is thrown.     */
public AlertPersistenceDBType  getPersistenceDataBaseType() throws
ManagementRemoteException;

/**
 * Set the maximum time a persisted alert will be stored in the alert database
 * before it will be deleted as part of the removal policy
 * @param timeToLive - maximum time in millisecond.
 *
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsMaxAge(Long timeToLive) throws
ManagementRemoteException;

/**
 * return the last setting for the allowed persisted alert age.
 * A value of 0 current time which could cause all persisted alerts to be deleted.
 * A negative value this policy element is ignored.
 * @return  the returned value  representing as time in milliseconds set for each.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsMaxAge() throws ManagementRemoteException;

/**
 * set the maximum number of  alerts allowed to be persisted  before it will be
 * deleted as part of the removal policy
 * @param size - Maximum number of alerts.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsMaxCount(Long size) throws ManagementRemoteException;

/**
 * return the last setting for the  maximum of  alerts allowed to be persisted.
 * A value of 0 mean no alerts persisted. It behave as if the user
 *  set enableAlertsPersistence to false
 *
 * A negative value this policy element is ignored.
 * @return the returned value represent the maximum number of alerts allowed to be
 *         persisted on each target.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsMaxCount() throws ManagementRemoteException;
```

```
/**
 *  The priority based alert level that will be part of the removal policy.
 *  the priorities are as follows (from low to high):
 *  INFO,WARNING,MINOR,MAJOR,CRITICAL,FATAL.
 *  all alerts from the provided level and below will be candidates for removal.
 *
 * @param level - an AlertLevelType representing the level.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsLevel(AlertLevelType level) throws
ManagementRemoteException;
/**
 * @return  the returned value represent the last setting for the level of alerts
 *          that allowed to be removed from persistence for each target.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public AlertLevelType getPersistedAlertsLevel() throws ManagementRemoteException;


/**
 * set the effective policy for the removal of persisted alerts.
 * @param policyList - an array of AlertRemovalPolicyType where the priority
 *                     of the given policy is defined by its position in the list.
 *                     i.e the lower the index that policy will be applied first.
 *                     possible values are:
 *                     ALERTS_AGE,ALERTS_COUNT,ALERTS_LEVEL.
 *
 *                     null value or empty array indicate no policy will be
 *                     enforced.

 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsRemovelPolicy(AlertRemovalPolicyType[] policyList)
*          throws ManagementRemoteException;

/**
 * @return the return value representing an array the last setting the policy used
 *         when persisted alerts are to be removed. An empty array mean no policy
 *         is enforced.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public String[] getPersistedAlertsRemovalPolicy() throws ManagementRemoteException;


/**
 * enable or disable the ability to use removal policy.
 * @param enableExecuation -  true the current setting is enforced. False the
 *           current policy is ignored.
```

```
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void enablePersistedAlertsPolicyExecution(Boolean enableExecution) throws
*            ManagementRemoteException;

/**
 * @return the returned value represent the last setting that enable/disable the
 *         removal policy
 * @throws ManagementRemoteException
 */
public Boolean isPersistedAlertsPolicyExecutionEnabled()throws
*          ManagementRemoteException;

/**
 * set how often the persisted alerts removal policy will be executed.
 * @param excutionInterval - The interval is defined in milliseconds.

 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public void setPersistedAlertsRemovelPolicyExecInterval(Long excutionInterval)
*          throws ManagementRemoteException;

/**
 * @return the returned value representing The last interval setting of
 *         the persisted alerts removal policy will be executed.
 * @throws ManagementRemoteException if JMX related exception is thrown.
 */
public Long getPersistedAlertsRemovelPolicyExecInterval() throws
 *         ManagementRemoteException;

/**
 * Set the persisted alerts table name.
 * Note: if the same database is used across the whole enterprise. it
 *       must be unique for each domain used.
 * @param tableName - the table name to be used for the stored alerts
 * @throws ManagementRemoteException
 */
public void setAlertTableName(String tableName) throws ManagementRemoteException;


/**
 * @return The current assign persisted alerts table name.
 * @throws ManagementRemoteException
 */
public String getAlertTableName() throws ManagementRemoteException;
/**
 *
```

```
 * @return return the total number of alerts currently persisted. This value
 * is volatile and may V between two sequential calls to this method.
 * @throws ManagementRemoteException
 */
public  Integer getPersistedAlertsCount() throws ManagementRemoteException;

/**
 * the API allows the caller to set all the parameters defined in the other API in
 * this interface. All the setting are applied prior to enabling the persistence.
 * @param enableJournaling
 *                    true - will prevent the system from removing alerts after
 *                           they are delivered. The alert stay in the database
 *                           until the user remove them.
 *                    false - The system will remove the alert upon acknowledgment
 *                            from the reliable client in case one was setup or
 *                            upon send the alert to all the non reliable client/s.
 * @param jndiName - JNDI name of the data source database to be used for
 * persistence.
 * @param dbtype - one of the predefined typed defined in
 *                    {@link com.sun.caps.management.api.alerts.
 *                     AlertPersistenceDBType}
 * @param timeToLive -  maximum time in millisecond.
 * @param maxCount - Maximum number of alerts.
 * @param level -  an AlertLevelType representing the level.
 * @param policyList - an array of AlertRemovalPolicyType where the priority
 *                     of the given policy is defined by its position in the list.
 *                     i.e the lower the index that policy will be applied first.
 *                     possible values are:
 *                     ALERTS_AGE,ALERTS_COUNT,ALERTS_LEVEL.
 * @param enablePolicyExecution - true the current setting is enforced. False the
 *         current policy is ignored.
 * @param interval The interval the policy will be executed (is defined in
 *         milliseconds).
 * @param inMemoryCacheSize - The interval is defined in milliseconds.

 * @throws ManagementRemoteException if JMX related exception is thrown or
 *         the list of target names is null or empty.
 */
 public void enableAlertsPersistence(Boolean enableJournaling,String jndiName,
                                     AlertPersistenceDBType dbtype,Long timeToLive,
                                     Long maxCount,AlertLevelType level,
                                     AlertRemovalPolicyType[] policyList,
                                     Boolean enablePolicyExecution,Long interval,
                                     Integer inMemoryCacheSize) throws
 *                                   ManagementRemoteException;
```

AlertNotificationService

警报通知服务用于向客户机（如 Groovy）通知事件。不过，要使客户机能够接收通知，它必须使用警报通知服务 API 进行订阅。在可靠客户机和非可靠客户机方面，发行版 6 与以前的发行版之间存在明显的差异。对于发行版 5.1.x，客户机是可靠的，这表示在确认事件传送之前无法删除事件。对于发行版 6，客户机可能是不可靠的，这表示无法保证客户机能够接收到每个事件。在发行版 6 中，设置为“可靠”的最后一个客户机将其前面的所有客户机设置为“不可靠”客户机。

警报通知服务 API

```
/**
     * request to the event management system to get an events that satisfy the
     * filter provided. The method will validate the call-back object for the
     * call-back method name and parameter (see below for more information).
     * This method allow the caller to register multiple time with diffrent
     * filtering,target and call-back parameters.
     *
     * @param filter - the filter that will be applied to the events prior to
     *                 forwarding them to this client. the valid keys for the map
     *                 are defined in  {@link com.sun.caps.management.api.alerts.
     *                 AlertNotificationFilterElementType}.
     *                 For the ALERTSEVERITY type the valid value are define in
     *                 {@link com.sun.caps.management.api.alerts.AlertLevelType#}.
     *
     * @param targetNames - the server instances that is subscription will initially be
     *                      filtered on. if targetNames is/are defined it/they have
     *                      precedence over the servername element in the filter
     *                      mentioned above.
     *
     * @param CallbackObject an instance of the client object that contain the
     *                       call back method to be called when event received
     *                       from the server.
     * @param methodName the method name to be invoke when event received from the
     *                       server.
     *       IMPORTANT: THE METHOD MUST HAVE ONE PARAMETER OF TYPE
     *                      {@link com.sun.caps.management.api.alerts.Alert"} ALERT.
     *
     * @param requireReliableDelivery - true mean this client request that all events
     *                       should be delivered to him reliably otherwise the client
     *                       may miss event.
     *       IMPORTANT: THE SYSTEM ALLOW ONLY ONE RELIABLE CLIENT. THE  LAST CLIENT
     *                       TO SET IT TO TRUE TAKES OVER THE RELIABLE DELIVERY. IT
     *                       WILL AFFECT ALL THE SUBSCRIPTIONS DONE BY THE CLIENT IN
     *                       THE CURRENT APPLICATION SESSION.
     *
     * @param exceptionCallBack an instance of the client object that contain the call
     *                       back method to be called when an connectivity exception
```

```
     *                       is generated by this service.
     *
     * @param exceptionMethodName the method name to be invoke when an exception is
     *                       generated by this service.
     *       IMPORTANT: THE METHOD MUST HAVE ONE PARAMETER OF TYPE
     *                       {@link java.lang.Exception} EXCEPTION.
     *
     *       NOTE: the exception call be should be the same for all subscriptions
     *                       calls otherwise the last the exception call back
     *                       defined by the last subscription will be used.
     *
     * @return Unique identification string that need to be used in the un-subscribe
     *                       operation.
     *
     * @throws ManagementRemoteException
     *         1. if fail to communicate with the event management system
     *         2. unable to invoke call back method because of invalid parameter.
     */
    public String subscribe(Map filter,String[] targetName,Object CallbackObject,String
     *          methodName, Boolean requireReliableDelivery,Object exceptionCallBack,
                String exceptionMethodName) throws ManagementRemoteException;

    /**
     * request the event management system to stop forwarding events to this client
     * based on the subscription the caller made using the subscribe method.
     * once all the caller unsubscribe all the IDs any events that are waiting
     * to be delivered to this client will be discarded.
     *
     * @param - subscriptionIDs A list of IDs return by the subscribe call/s  that the
     *          caller wish to unsubscribe from.
     *
     * @throws ManagementRemoteException if fail to communicate with the Domain server.
     */
    public void unsubscribe(String[] subscriptionIDs) throws ManagementRemoteException;

    /**
     * utility method that returns the parameters the client used to subscribe for
     * alerts for the given subscription ID.
     *
     * @param - subscriptionIDs list return by the subscribe call/s.
     *
     * return a map keyed IDs on the provided as a parameter and values as
     *        SubcriptionInformationinstances.
     *        @see com.sun.caps.management.api.alerts.SubcriptionInformation
     * @throws ManagementRemoteException if fail to communicate with the Domain server.
     */
    public Map getSubscriptionInformation(String[] subscriptionIDs);
```

服务 — JavaCAPSManagementAPI

目前提供了 10 个 Java CAPS 管理客户机 API 服务：

- 管理
- 运行时管理
- 配置
- 部署
- 安装
- 性能测量
- 警报管理
- 警报监控
- 警报通知
- 警报配置
- JMS 管理
- 日志管理
- BPEL 管理
- 数据管理 (MDM)
- Sun 适配器管理
- 管理客户机的目标选项行为

此外，还提供了一个目标选项。其 Java CAPS 管理客户机行为也是服务的一部分。根据所指定的值，此选项将导致以不同方式执行安装和部署命令。有关详细信息，请参见第 39 页中的 “管理客户机的目标选项行为”。

注 – Java CAPS 6 软件包以压缩文件形式附带提供了服务样例文件（如 `AdministrationServiceSample.groovy`）和目标选项行为。

管理服务

管理服务支持

- 组件容器、共享库、服务组合件以及服务单元的描述符
- 使用和置备所有组件容器公开的端点
- 检索与每个端点关联的 WSDL 和 XSD 资源的操作

表 2 管理服务方法名称和描述

API 方法名称	描述
getComponentInstallationDescriptor	检索组件的 jbi.xml 部署描述符
getSharedLibraryInstallationDescriptor	检索共享库的 jbi.xml 部署描述符。

表 2 管理服务方法名称和描述 （续）

API 方法名称	描述
getServiceAssemblyDeploymentDescriptor	检索服务组合件的 jbi.xml 部署描述符。
getServiceUnitDeploymentDescriptor	检索服务单元的 jbi.xml 部署描述符。
isJBIRuntimeEnabled	检查是否启用了 JBI 运行时环境。
isServiceEngine	检查服务引擎。
isBindingComponent	检查绑定组件。
getConsumingEndpoints	检索组件的使用端点列表。
getProvisioningEndpoints	检索组件的置备端点列表。
getWSDLDefinition	检索与指定端点关联的主 WSDL。
getWSDLImportedResource	检索与指定端点和目标名称空间关联的 WSDL 或 XSD。
isClassicEnterpirseManagerUp	检查是否正在运行非 JBI 组件的 Enterprise Manager 服务器。

运行时管理服务

运行时管理服务：

- 支持生命周期操作，如启动、停止和关闭
- 支持 Java EE 应用程序的生命周期操作，如启用和禁用
- 提供为 JBI 和 Java EE 列出组件容器、共享库和应用程序的操作

表 3 运行时管理服务方法名称和描述

API 方法名称	描述
isTargetUp	检查目标（服务器、群集）是启动还是关闭。
listServiceEngines	列出服务引擎。
listBindingComponents	列出绑定组件。
listSharedLibraries	列出共享库。
listServiceAssemblies	列出服务组合件。
showServiceEngine	显示符合不同选项的服务引擎。
showBindingComponent	显示符合不同选项的绑定组件。
showSharedLibrary	显示符合不同选项的共享库。

表 3 运行时管理服务方法名称和描述 （续）

API 方法名称	描述
showServiceAssembly	显示符合不同选项的服务组合件。
listComponents	检索组件列表。
getState	检索运行时组件的状态（如 UP、DOWN、UNKNOWN 等）。
getProperties	检索组件的属性列表。
startComponent	启动组件。
stopComponent	停止组件。
restartComponent	重新启动组件。
shutdownComponent	关闭组件。
startServiceAssembly	启动服务组合件。
stopServiceAssembly	停止服务组合件。
shutdownServiceAssembly	关闭服务组合件。
listCompositeApplications	检索复合应用程序列表。
getRuntimeComponentProperties	获取运行时单元的属性列表。
getRuntimeComponentStatus	获取运行时组件的状态，即，UP、DOWN、UNKNOWN 等。
startRuntimeComponent	启动运行时组件。
restartRuntimeComponent	重新启动运行时组件。
stopRuntimecomponent	停止运行时组件。
listTargets	返回可以在该域中部署的所有目标。所有组和所有服务器（不属于任何组的服务器）。
listLifecycleModules	列出生命周期模块。
listExtensionModules	列出扩展模块。
listSystemConnectors	列出系统连接器模块。
listAppclientModules	返回在指定目标上部署的所有指定类型的模块。
listConnectorModules	返回部署的连接器数组。
listEjbModules	返回在指定目标上部署的所有指定类型的模块。
listWebModules	返回在指定目标上部署的所有指定类型的模块。

表 3 运行时管理服务方法名称和描述 （续）

API 方法名称	描述
listJavaEEApplications	返回部署的 JavaEE 应用程序列表。某些应用程序部署到域中并在其中进行注册。
enableJavaEEApplication	启用指定目标上的应用程序或模块。如果是群集，则还会启用该群集中的服务器实例的应用程序引用。
enableAppclientModule	启用指定目标上的应用程序或模块。如果是群集，则还会启用该群集中的服务器实例的应用程序引用。
enableConnectorModule	启用指定目标上的应用程序或模块。如果是群集，则还会启用该群集中的服务器实例的应用程序引用。
enableEjbModule	启用指定目标上的应用程序或模块。如果是群集，则还会启用该群集中的服务器实例的应用程序引用。
enableWebModule	启用指定目标上的应用程序或模块。如果是群集，则还会启用该群集中的服务器实例的应用程序引用。
disableJavaEEApplication	禁用指定目标上的应用程序或模块。如果是群集，则还会禁用该群集中的服务器实例的应用程序引用。
disableAppclientModule	禁用指定目标上的应用程序或模块。如果是群集，则还会禁用该群集中的服务器实例的应用程序引用。
disableConnectorModule	禁用指定目标上的应用程序或模块。如果是群集，则还会禁用该群集中的服务器实例的应用程序引用。
disableEjbModule	禁用指定目标上的应用程序或模块。如果是群集，则还会禁用该群集中的服务器实例的应用程序引用。
disableWebModule	禁用指定目标上的应用程序或模块。如果是群集，则还会禁用该群集中的服务器实例的应用程序引用。
isJavaEEComponentEnabled	返回配置中的应用程序状态。如果指定目标为 null/空/"domain"，则仅使用实际应用程序的“已启用”标志。否则，使用应用程序引用的“已启用”标志来确定启用状态。

配置服务

配置服务可用于：

- 配置组件容器和 JBI 运行时环境等。
- 为组件容器和 JBI 运行时环境配置日志记录
- 管理应用程序配置和应用程序变量
- 在部署前验证应用程序配置

表 4 配置服务方法名称和描述

API 方法名称	描述
getRuntimeLoggerLevels	检索所有运行时记录程序及其级别。
getRuntimeLoggerLevel	查找某个运行时记录程序的级别。
setRuntimeLoggerLevel	设置给定运行时记录程序的日志级别。
getComponentLoggerLevels	检索组件自定义记录程序及其级别。
setComponentLoggerLevel	设置给定自定义记录程序的组件日志级别
getComponentExtensionMBeanObjectNames	检索扩展 MBean 对象名称。
getComponentConfiguration	检索组件配置。
setComponentConfiguration	设置组件配置。
getRuntimeConfigurationMetaData	返回与指定属性关联的运行时配置元数据。元数据包含名称-值对，例如，default、descriptionID、descriptorType、displayName、displayNameId、isStatic、name、resourceBundleName、tooltip、tooltipId，等等。
setRuntimeConfiguration	设置运行时配置。
getRuntimeConfiguration	检索运行时配置。
getDefaultRuntimeConfiguration	检索默认运行时配置。
isServerRestartRequired	检查是否需要重新启动服务器以应用对某些配置参数所做的更改。

部署服务

通过使用部署服务，您可以部署和取消部署：

- JBI 服务组合件
- Java EE 工件，例如，企业归档文件（EAR 文件）、EJB 模块、Web 归档文件（WAR 文件）以及 Java EE 连接器、应用程序客户机等。

表 5 部署服务方法名称和描述

API 方法名称	描述
deployServiceAssembly	部署服务组合件。
deployServiceAssemblyFromDomain	从域目标中部署服务组合件。
undeployServiceAssembly	取消部署服务组合件。
deployJavaEEComponent	将组件部署到给定的一组目标中，这些目标可以是域、群集或单独实例。由于群集和单独实例在共享部署方面存在一些限制，因此，仅将组件数据部署到列表中的第一个目标，然后为指定目标组中的其余目标创建应用程序引用。
undeployJavaEEComponent	从给定的一组目标中取消部署组件，这些目标可以是域、群集或单独实例。由于群集和单独实例在共享部署方面存在一些限制，因此，除了目标组中的第一个目标以外，将删除所有其他目标的应用程序引用。然后，从目标组的第一个目标中取消部署组件数据。

安装服务

安装服务可用于：

- 安装和卸载 JBI 组件容器和共享库
- 升级组件容器，而无需取消部署现有工件

表 6 安装服务方法名称和描述

API 方法名称	描述
installComponent	安装组件（服务引擎、绑定组件）。
uninstallComponent	卸载组件（服务引擎、绑定组件）。
installSharedLibrary	安装共享库。
uninstallSharedLibrary	卸载共享库。
installComponentFromDomain	从域目标中安装组件（服务引擎、绑定组件）。
installSharedLibraryFromDomain	从域目标中安装共享库。

表 6 安装服务方法名称和描述 （续）

API 方法名称	描述
upgradeComponent	升级组件（服务引擎、绑定组件）。按实际使用组件的方式升级组件。在升级处理期间，将调用组件的新升级 SPI 实现，以使组件能够执行完成升级所需的任何特殊处理。如果组件未提供升级 SPI 实现，仍可使用 updateComponent API 对组件进行更新。另外，在升级实现中允许更改组件的安装描述符，但组件名称除外（原因是显而易见的）。这样，便可建立新的共享库依赖关系、更改组件 SPI 实现的类名以及更改组件的类加载首选项（类路径和类加载顺序）。无论组件是否提供了新升级 SPI 实现，都允许进行这些更改。

性能测量服务

性能测量服务尝试为开发者和管理员描述性能特性。

- JBI 框架
  - 开始时间
  - 启动时间
  - 组件计数
  - 端点计数
  - 服务组合件计数
- 标准化消息路由器 (Normalized Message Router, NMR)
  - 每个组件花费的时间
  - 传送通道花费的时间
  - NMR 花费的时间
  - 每个组件/端点
  - 按组件/端点查询
- 组件端点
  - 运行时间
  - 发送/接收的请求总数，如“传入”消息
  - 发送/接收的回复总数，如“传出”消息
  - 发送/接收的错误总数
  - 完成的交换总数，如已使用/提供
  - 活动交换总数，如已使用/提供
  - 错误交换总数，如已使用/提供
  - 响应时间
  - 状态时间
  - 活动消息 ID
  - 等待消息 ID

表 7 性能测量服务方法名称和描述

API 方法名称	描述
clearPeformaceInstrumentationMeasurement	重置端点上的性能测量。
getPerformanceInstrumentationEnabled	检索性能测量启用标志。
getPerformanceInstrumentationMeasurement	检索指定端点的性能测量数据。
getPerformanceMeasurementCategories	检索性能统计信息类别。数组中的每个项是复合性能数据的关键所在，它还指明了测量类型（如标准化）。
setPerformanceInstrumentationEnabled	设置性能测量启用标志。
getFrameworkStatisticsAsTabularData	此方法用于提供给定目标中的 JBI 框架统计信息。这些数据是以表格形式显示的。
getFrameworkStatistics	此方法用于提供给定目标中的 JBI 框架统计信息。
getComponentStatistics	此方法用于提供给定目标中的给定组件的统计信息。
getComponentStatisticsAsTabularData	此方法用于提供给定目标中的给定组件的统计信息。这些数据是以表格形式显示的。
getEndpointStatisticsAsTabularData	此方法用于提供有关给定目标中的给定端点的统计信息。这些数据是以表格形式显示的。
getEndpointStatistics	此方法用于提供有关给定目标中的给定端点的统计信息。
getNMRStatisticsAsTabularData	此方法用于提供有关给定目标中的消息服务的统计信息。这些数据是以表格形式显示的。
getNMRStatistics	此方法用于提供有关给定目标中的消息服务的统计信息。
getServiceAssemblyStatisticsAsTabularData	此方法用于提供有关给定目标中的服务组合件的统计信息。这些数据是以表格形式显示的。
getServiceAssemblyStatistics	此方法用于提供有关给定目标中的服务组合件的统计信息。
enableMessageExchangeMonitoring	此方法用于启用有关消息交换的计时信息监视。
disableMessageExchangeMonitoring	此方法用于禁用有关消息交换的计时信息监视。

警报管理服务

表 8 警报管理服务方法名称和描述

API 方法名称	描述
getAlertAdministrationService	创建并返回警报监控服务。
getAlertConfigurationService	创建并返回警报配置服务。
getAlertNotificationService	创建并返回警报通知服务。

警报监控服务

表 9 警报监控服务方法名称和描述

API 方法名称	描述
getAlerts	检索所有警报。
getAlertFieldNames	检索警报实例数据库字段名称列表。
update	根据条件更新字段值。
delete	从持久性存储中删除警报对象。
observe	将警报实例的 observationState 列表设置为已观察。
resolved	根据条件将警报实例的 observationState 设置为已解析。
reset	将警报实例的 observationState 列表设置为其初始状态（未观察）。
UpdateComment	更新警报的注释字段。
resetAll	将所有表条目的 observationState 设置为其初始状态。
resolveAll	将所有表条目的警报 observationState 设置为已解析状态。
observeAll	将所有表条目的警报 observationState 设置为已观察状态。

警报通知服务

表 10 警报通知服务方法名称和描述

API 方法名称	描述
subscribe	请求事件管理系统获取符合提供的过滤器的事件。该方法将验证回调方法名称和参数的回调对象（有关详细信息，请参见下文）。通过使用该方法，调用方可使用不同的过滤器、目标和回调参数多次进行注册。
unsubscribe	请求事件管理系统停止将事件转发到此客户机（根据调用方使用 subscribe 方法进行的订阅）。在所有调用方取消订阅所有 ID 后，将放弃等待传送到此客户机的任何事件。
getSubscriptionInformation	这是一种实用程序方法，它返回客户机用于为给定订阅 ID 订阅警报的参数。

警报配置服务

表 11 警报配置服务方法名称和描述

API 方法名称	描述
enableAlertsPersistence	在警报数据库中启用警报持久性。如果启用，在传送通道失败或重新启动应用服务器时，可提供可靠的警报传送。
disableAlertsPersistence	在警报数据库中禁用警报持久性。
isAlertsPersistenceEnabled	返回警报持久性启用操作的最终设置。如果启用，则为 true，否则为 false。
isAlertsJournalEnabled	返回警报日志功能启用操作的最终设置。如果启用，则为 true，否则为 false。
setPersistenceDataSourceJndiName	设置用于持久性的数据源数据库的 JNDI 名称。如果未至少提供一次，即使将 enableAlertsPersistence 设置为 true，也会禁用持久性。
getPersistenceDataSourceJndiName	返回最终设置的警报持久性数据源 JNDI 名称。
setPersistenceDataBaseType	设置用于持久性的数据库类型。Derby 是假定的默认数据库。如果使用其他数据库，应在启用持久性之前调用该方法。

表 11 警报配置服务方法名称和描述 （续）

API 方法名称	描述
getPersistenceDataBaseType	返回值表示最终设置的数据库类型。
setPersistedAlertsMaxAge	设置在删除持久保存的警报（作为删除策略的一部分）之前在警报数据库中存储警报的最长时间。
getPersistedAlertsMaxAge	返回允许持久保存的警报存留期的最终设置。如果当前时间的值为 0，则可能会导致删除所有持久保存的警报。如果使用负值，则会忽略该策略元素。
setPersistedAlertsMaxCount	设置在删除某个警报（作为有效删除策略的一部分）前允许持久保存的最大警报数。如果启用了持久性并将计数设置为 0，则会关闭所打开的日志功能。
getPersistedAlertsMaxCount	返回允许持久保存的最大警报数的最终设置。如果值为 0，则表示不持久保存任何警报。
setPersistedAlertsLevel	基于优先级的警报级别（作为删除策略的一部分）。优先级从低到高依次为：INFO、WARNING、MINOR、MAJOR、CRITICAL 和 FATAL。您可以选择删除提供的级别以及以下级别的所有警报。
getPersistedAlertsLevel	返回的值表示警报级别的最终设置，将允许从每个目标中删除持久保存的这些警报。
setPersistedAlertsRemovelPolicy	为删除持久保存的警报设置有效的策略。
getPersistedAlertsRemovalPolicy	返回的值表示在删除持久保存的警报时策略将使用的最终设置数组。如果数组为空，则表示未执行任何策略。
enablePersistedAlertsPolicyExecution	允许或禁止使用删除策略的功能。
isPersistedAlertsPolicyExecutionEnabled	返回的值表示，启用/禁用持久保存的警报删除策略的最终设置。
setPersistedAlertsRemovelPolicyExecInterval	设置持久保存的警报删除策略的执行间隔。
getPersistedAlertsRemovelPolicyExecInterval	返回的值表示，持久保存的警报删除策略的执行间隔的最终设置。
setInMemoryAlertsCacheMaxSize	设置在传送到注册的侦听器之前可在内存中缓存的最大警报数。
getInMemoryAlertsCacheMaxSize	返回的值表示，警报高速缓存中的最大内存大小的最终设置。
setAlertTableName	设置持久保存的警报表名。

表 11 警报配置服务方法名称和描述 （续）

API 方法名称	描述
getPersistedAlertsCount	返回当前持久保存的警报总数。该值是可变的，可能会发生更改。
enableAlertsPersistence	通过使用该 API，调用方可以在此接口中设置其他 API 中定义的所有参数。将在启用持久性之前应用所有这些设置。

JMS 管理服务

表 12 JMS 管理服务方法名称和描述

API 方法名称	描述
getServerProperties	返回服务器属性。
isServerReady	检查消息服务器是否准备就绪。
getXids	返回消息服务器上的事务列表。
rollbackXid	回滚消息服务器上的指定事务。
getTopicProperties	检索指定的主题属性。
getTopics	检索消息服务器上的主题列表。
getTopicsWithHeaders	返回消息服务器上包含标头属性的主题列表。
getTemporaryTopics	检索临时主题列表。
createTopic	在消息服务器上新建具有指定名称的指定主题。
deleteTopic	删除消息服务器上的指定主题。
getTopicMessage	返回指定主题的指定消息的消息体。
getTopicMsgProperties	返回指定主题的消息属性。
getTopicMsgPropertiesList	返回给定起始消息索引的消息及其属性列表。
getSubscribers	返回指定主题的订阅者列表。
changeTopicTextMessage	更改指定主题的指定文本消息内容。
changeTopicBytesMessage	更改指定主题的指定字节消息内容。
deleteTopicMessage	从指定主题中删除指定消息。
getTopicMessageType	返回指定主题中的指定消息的消息类型。
suspendTopic	暂停指定主题。

表 12 JMS 管理服务方法名称和描述 （续）

API 方法名称	描述
resumeTopic	恢复暂停的主题。
submitNewMessage	将新消息提交到消息服务器上的指定主题或队列。
createTopicDurableSubscriber	为消息服务器上的指定主题新建指定主题持久订阅者。
unsubscribeDurableSubscriber	取消订阅消息服务器上的指定持久订阅。
republishTopicMessage	将已记入日志的指定消息重新发送到消息服务器上的指定队列。
getTopicStatistics	返回消息服务器上的指定主题的统计信息。
getQeueueProperties	检索队列属性。
getQueues	检索消息服务器上的队列列表。
getQueuesWithHeaders	返回消息服务器上包含标头属性的队列列表。
getTemporaryQueues	检索临时队列列表。
createQueue	在消息服务器上新建具有指定名称的队列。
deleteQueue	删除消息服务器上的指定队列。
getQueueMsgProperties	返回指定主题的消息属性。
getQueueMsgPropertiesList	返回给定起始消息索引的消息及其属性列表。
getQueueMessage	返回队列的指定消息的消息体。
getReceivers	返回指定队列的接收者列表。
changeQueueTextMessage	更改队列的指定文本消息内容。
changeQueueBytesMessage	更改指定队列的指定字节消息内容。
deleteQueueMessage	从指定队列中删除指定消息。
getQueueMessageType	返回队列的指定消息的消息类型。
suspendQueue	暂停指定队列。
resumeQueue	恢复暂停的队列。
resendQueueMessage	将已记入日志的指定消息重新发布到消息服务器上的指定主题。
getQueueStatistics	返回消息服务器上的指定队列的统计信息。

日志管理服务

表 13 日志管理服务方法名称和描述

API 方法名称	描述
getLogString	读取页面、过滤器和搜索日志。仅以单个字符串的形式返回相应的行，而不返回任何其他元数据。
getLog	从日志中读取、过滤和搜索页面中的行；这些行将以图的形式返回。
getLogAsString	从日志中记录、过滤和搜索页面中的行；这些行将作为一个字符串返回。
listLoggerNames	列出在目标中注册的记录程序的名称。
listLoggerObjectNames	列出在目标中注册的记录程序的名称。
isLoggerRegistered	询问是否在目标服务器实例中注册记录程序实例。
registerLogger	在目标服务器实例中注册记录程序实例。
unregisterLogger	在目标服务器实例中取消注册记录程序实例。
setLogFile	设置记录程序的文件名。
getLogFile	获取记录程序的文件名。

BPEL 管理服务

BPEL 管理服务支持两种类型的 API 方法名称：

- Java CAPS（非 JBI 组件）
- Java CAPS（JBI 组件）

注 – 这些 API 对配置服务中为 BPEL 公开的 API 提供补充。

表 14 BPEL 管理服务 API 方法名称和描述（非 JBI 组件）

API 方法名称	描述
getBPELInstances	获取 BPEL 实例（如果给定可选 BPEL 流程 QName 和/或可选 BPStatus 或可选实例 ID）。如果实例 ID 存在，则忽略 BPEL 流程 QName 和 BPStatus。返回的最大实例数为 1000，用户可以为 maxRecords 指定较小的数值以限制返回的实例数。如果 BPInstanceQueryResult.overflow 为 true，则表明符合条件的实例数大于 1000，不会在结果列表中返回任何实例，用户应指定 maxRecords (<= 1000)、sortColumn 和顺序。
getBPELInstanceActivityStatus	获取 BPEL 实例的活动状态列表。
getBPELProcessIds	获取 BPEL 流程 QName（作为服务单元中的字符串）列表。
getBPELInstanceFault	获取错误 BPEL 实例的错误详细信息。
getInvokerInstance	获取调用特定 BPEL 实例的调用方（父）BPEL 实例的列表。
getInvokeeInstance	获取特定 BPEL 实例调用的被调用方（子）BPEL 实例的列表。
isMonitoringEnabled	询问是否启用监视。
isPersistenceEnabled	询问是否启用持久性。
resumeInstance	恢复给定业务流程实例的某个业务流程实例。
terminateInstance	终止给定业务流程实例的某个业务流程实例。
suspendInstance	暂停给定业务流程实例的某个业务流程实例。
getBPELInstanceFault	获取错误 BPEL 实例的错误详细信息。
suspendAllInstance	暂停 BPEL 流程的所有实例。
resumeAllInstance	恢复 BPEL 流程的所有实例。
terminateAllInstance	终止 BPEL 流程的所有实例。
changeVariableValue	更改 BPEL 变量值。请注意，只能更改叶节点。
getVariableValue	获取 BPEL 变量值。将返回 BPEL 变量的内容。
listBPELVaraibles	获取 BPEL 实例的 BPEL 变量信息。

表 15 BPEL 管理服务 API 方法名称和描述（JBI 组件）

API 方法名称	描述
setBusinessProcessInstanceVariableValue	在某个部件上使用 XPath 以设置属性值（如果给定业务流程实例、容器、部件、XPath 表达式和值）。

主数据管理 (Master Data Management, MDM) 服务

该 API 服务提供了在复杂的分布式企业业务环境中集成和管理数据和应用程序的功能，其中包括以下数据管理产品：

- Master Index Studio（以前称为 eView Studio 和 Single Patient View）
- Data Integrator（以前称为 eTL Integrator）
- Data Quality
- Data Services
- Data Migrator

表 16 主数据管理 (Master Data Management, MDM) 服务 API 方法名称和描述

API 方法名称	描述
listApplicationNames	返回当前部署的 MDM 应用程序列表。
getApplicationStatus	返回 MDM 应用程序状态。
getDatabaseStatus	返回数据库连接状态。
getWebModuleStatus	返回 MDM Web 应用程序模块状态。

Sun 适配器管理服务

表 17 Sun 适配器管理服务 API 方法名称和描述

API 方法名称	描述
start	启动组件；此操作的语义由实现进行定义。
restart	重新启动组件；此操作的语义由实现进行定义。
stop	停止组件；此操作的语义由实现进行定义。
getStatus	返回状态。
getProperties	返回属性。
isStartable	确定是否向用户显示“启动”按钮。

表 17 Sun 适配器管理服务 API 方法名称和描述 （续）

API 方法名称	描述
isRestartable	确定是否向用户显示“重新启动”按钮。
isStoppable	确定是否向用户显示“停止”按钮。
getType	返回适配器所监视的组件的类型。
getConfiguration	返回适配器所监视的组件的配置。
getState	返回适配器所监视的组件的状态。
getTargetState	返回适配器所监视的组件的目标状态。这是适配器组件的当前状态或将要变为的状态。此状态可能不同于 getState() 返回的状态。
getRAConfiguration	返回适配器的元数据详细信息。

管理客户机的目标选项行为

--target 选项导致以不同方式执行安装和部署命令。根据指定的选项值，这些差异可能是非常大的。

注 – 其中的两个目标选项名称为常量：“server” 和 “domain”。它们表示一个运算符实例，可以将其替换为特定于当前模板的名称。

表 18 目标选项值和行为

选项值	行为
server	针对嵌入的域管理服务器 (Domain Administration Server, DAS) 实例执行命令。
domain	如果目标选项为文字字符串 *domain*，则针对该域本身执行组件，而不是针对该域上运行的任何实例或群集。此选项值仅适用于安装或部署命令。
群集名称	如果指定了群集名称，则针对指定群集中的所有实例执行命令。
实例名称	如果指定了实例名称，则针对指定的特定实例执行命令。

使用 Java 代码样例编写 Java 代码以访问 API

可以使用 Java 代码样例，编写您自己的 Java 代码以访问 API。此软件包以压缩文件形式附带提供了这些样例。

当警报管理子系统收到警报时，它将向订阅者发送这些警报，如 Enterprise Manager 或 Groovy。该流程具有两种工作方式。

- 如果未启用持久性，则将警报直接发送到所有使用客户机 API 的订阅者。如果没有订阅者，则会删除警报。
- 如果启用持久性，则会将警报发送到数据库。如果没有订阅者，则会在数据库中持久保存警报，直至订阅者准备开始使用它们。您可以选择启用日志功能，这可确保永远不会删除警报。

设置数据库

Derby 是随 Java CAPS 提供的数据库。不过，您可以设置并使用其他数据库。但要记住，您仅限于使用 Java CAPS 支持的数据库。

- Derby
- Oracle
- Sybase
- DB2
- PointBase

作为以下过程的一个选项，您可以使用警报配置管理 API 编写 Groovy 脚本或小型 Java 实用程序以替换步骤 4-6。有关此选项的示例，请参见 *JavaCAPS6*`/ESB_API_KIT/samples` 目录（其中，*JavaCAPS6* 是将 `EM_API_KIT.zip` 解压缩到的目录）。

注意 – 请记住，以脚本或编程方式编写并执行的最后一条命令应该是，在指定数据库中启用持久性。以下是警报配置服务 API 中的方法示例：setPersistenceDataSourceJndiName、getPersistenceDataSourceJndiName、setPersistenceDataBaseType、getPersistenceDataBaseType 和 setAlertTableName（可选）。

如果数据库对表名有限制，例如，Oracle 限制为 30 个字符，自动生成的表可能会超过该限制。请使用警报配置 API (SetAlertTableName) 来设置表名。切记，每个域必须具有唯一的表名，以防止某个域中的事件显示在其他域中。

▼ 使用 Enterprise Manager 设置数据库

1 启动要使用的域。

2. **使用 Sun Java System Application Server 管理控制台或 AS Admin 命令行实用程序设置连接池和资源。**

   注 – 记下为资源分配的名称。

   有关如何执行此任务的详细说明，请参见管理控制台联机帮助。

3. **启动选定的数据库。**

4. **启动 Enterprise Manager，然后在步骤 1 中添加所启动的域。**

5. **启动警报配置屏幕（有关信息，请参见《监视 Java EE 组件》）。**

   a. **选择与选定数据库匹配的数据库类型。**

   b. **输入 JNDI 名称。**

      注 – 这是在步骤 2 中创建的资源名称。

6. **启用持久性和日志功能，然后单击"保存"以提交更改。**

   注 – 在启用持久性时，您不需要启用日志功能；即，在启用持久性时，日志功能是可选的。

▼ 使用脚本实用程序设置数据库

1. **启动要使用的域。**

2. **使用 Sun Java System Application Server 管理控制台或 AS Admin 命令行实用程序设置连接池和资源。**

   注 – 记下为资源分配的名称。

   有关如何执行此任务的详细说明，请参见管理控制台联机帮助。

3. **启动选定的数据库。**

4. **编写脚本实用程序以调用相应的 API。**

```
setPersistenceDataSourceJndiName

setPersistenceDataBaseType(AlertPersistenceDBType dbtype)

(Optional) setAlertTableName(String tableName)

enableAlertsPersistence(Boolean enableJournaling)
```

或者，要执行上面的所有 API（可选），请使用：

```
enableAlertsPersistence(Boolean enableJournaling,String jndiName,
                                    AlertPersistenceDBType dbtype,Long timeToLive,
                                    Long maxCount,AlertLevelType level,
                                    AlertRemovalPolicyType[] policyList,
                                    Boolean enablePolicyExecution,Long interval,
                                    Integer inMemoryCacheSize) throws
*                                   ManagementRemoteException;
```

注 – 未设置此处的顺序，您可以对其进行更改，除非启用了持久性，此时它必须是最终的。

将 Oracle 和其他数据库用于警报持久性

除 Derby 以外，Oracle 和其他支持的数据库也可以执行警报持久性。不过，您必须进行一些更改才能使其正常工作。

注 – 如果打算使用 Oracle 持久保存警报，而不是使用 Derby，请确保阅读此主题。

▼ 为警报持久性设置 Oracle 数据库

1. **修改 `appserver/domains/domain1/jbi/config` 下面的 `eventmanagement.properties` 文件。**

   a. **将 DatabaseType 更改为 `ORACLE`。**

   b. **将 AlertTablename 更改为 `EMHOSTNAMEVistastccom8080`。**

注 – 必须执行此步骤，因为默认名称 EVENTMANAGEMENTHOSTNAMEistastccom8080 超过了 Oracle 对表名的限制（30 个字符）。

---

c. **将 DBJndiName 更改为 `OracleXPDB`。**

注 – 可以在管理控制台中创建该名称；步骤 2 中说明了这一点。

---

d. **将 PersistenceEnbled 更改为 `true`。**

e. **通过使用 Enterprise Manager，在普通警报管理配置代理中设置 DatabaseType、DBJndiName 和 PersistenceEnabled 的值。**

注 – 在启用持久性之前，可能已运行了数据库脚本。必须在文件中手动更改表名。

---

2 **在 Sun Java System Application Server 管理控制台中创建 JDBC 连接池和资源。**

a. **将 `classes12.zip` 的位置添加到类路径中，选择“JVM 设置”→“路径设置”→“类路径后缀”，然后重新启动域。**

注 – 要获取 Oracle 数据源，必须执行此操作。

---

b. **创建 Oracle 连接池。**

i. **输入一个名称（如 OracleXPPool），您可以选择所用的名称。**

ii. **选择 "`javax.sql.DataSource`" 作为资源类型。**

iii. **选择 "Oracle" 作为数据库供应商。**

iv. **设置相应的属性：**

用户：`eventdb_user`

数据库名称：`orcl`

注 – 根据您配置 Oracle 数据库方式的不同，此配置以及其他特定于数据库的配置可能会有所变化。

密码：`eventdb_user`

服务器名称：`hostname`

注 – 这是运行数据库的服务器。

端口号：1521

URL：jdbc:oracle:thin:@hostname:1521:orcl

注意 – 此 URL 可能会实际覆盖其他设置；它应该与其他设置相匹配。

v. **创建 JDBC 资源。**

输入 JNDI 名称，例如 OracleXPDB。

注 – 该名称应该与上述 `Alert Management Config Agent/eventmanagement.properties` 文件中设置的名称相匹配。

选择相应的池名称；在本示例中，我们使用 OracleXPPool。

3 **手动创建警报持久性和日志功能所需的用户和表等。**

注 – 对于 Derby，此操作是自动完成的。

`appserver/jbi/lib` 下面的 `jbi_rt.jar` 文件中打包了一些数据库脚本。但存在一些错误，因此，您需要手动更正这些脚本。

- 已修改了这些示例脚本以便用于 Oracle 10 GB 数据库；请参见第 45 页中的 “Oracle 脚本示例”。
- 以系统（管理员）用户身份依次运行 `create_event_store_user.sql` 和 `create_event_store_schema.sql` 脚本。
- 在此示例中进行的更正：
  - 设置表空间数据文件的绝对路径（与数据库安装有关）
  - 修改命令，以便与业务流程持久性相匹配
  - 注释掉第二个数据文件
  - 移动注释
  - 修复表名引用，以便与上面在 `eventmanagement.properties` 中设置的用户模式和表名相匹配
  - 修复列引用；即，将第二列的名称由 event_timestamp 更改为 timestamp
  - 将数据类型由时间戳更改为十进制
  - 修复序列引用

Oracle 脚本示例

truncate_event_store_schema.sql

```
TRUNCATE TABLE eventdb_user.EMHostNameVistastccom8080;
```

create_event_store_schema.sql

```
create table eventdb_user.EMHostNameVistastccom8080(
      id NUMBER CONSTRAINT ID_PK PRIMARY KEY,
            timeStamp decimal,
         physicalHostName varchar(256),
            environmentName varchar(256),
            logicalHostName varchar(256),
            serverType varchar(256),
            serverName varchar(256),
            componentType varchar(256),
            componentProjectPathName varchar(1024),
            componentName varchar(256),
            eventType varchar(256),
            severity integer,
            operationalState int,
            messageCode varchar(256),
            messageDetail varchar(4000),
            observationalState int,
            deploymentName varchar(256));
);
-- INSERT statement need to use it to insure autoincrement functionality
CREATE SEQUENCE eventdb_user.autoincrement_id;
create index  eventTime on eventdb_user.EMHostNameVistastccom8080(timeStamp);
```

create_event_store_user.sql

```
--Create a tablespace named EVENTDB_USER_DB. Change this value if a different
name is desired.
--Specify the name and the location of the file where the data related to
the tablespace
--created above will be stored. The location is by default the location determined by
--the database server/instance on which this script is run
--For example, for Windows c:\MyDatafiles\EVENTDB_USER_DB.dat, for Unix
/dev/home1/EVENTDB_USER_DB.dat
--Note that the name of the actual file need not be EVENTDB_USER_DB.dat
--Specify the size constraints

    -- Window and Oracle 9i there is a limitation on file size, it is 2 GB.
This by default creats 4GB, add more files if you need more than 4 GB.
    --- provide abosolute path if you preference is not default location
'C:\OracleDirectory\EVENTDB_USER_DB.dat' SIZE 2000M,
```

```
CREATE TABLESPACE EM_EVENTSTORE_DB
 DATAFILE
    'C:\oracle\product\10.2.0\oradata\orcl\EVENTDB_USER_DB.dat' SIZE 512M REUSE
AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED;

    -- 'C:\oracle\product\10.2.0\oradata\orcl\EVENTDB_USER_DB1.dat' SIZE 512M
REUSE AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED --- provide abosolute path
if you preference is not defaultlocation 'C:\OracleDirectory\EVENTDB_USER_DB1.dat'
SIZE 2000M
        -- when TABLESPACE is created with these options performance is degrading
gradually as more and more records added to schema  EXTENT MANAGEMENT LOCAL SEGMENT
SPACE MANAGEMENT AUTO


--Create a new user EVENTDB_USER. Change the name if so desired. Password will
be same as
--the user name by default. This username and password will be used to create the
--connection pool on the application server. Also specify the tablespace
and the quota on
--the tablespace the user has. Note that if you used a different tablespace
name above,
--you will have to specify that tablespace name here.

CREATE USER EVENTDB_USER IDENTIFIED BY EVENTDB_USER
DEFAULT TABLESPACE EM_EVENTSTORE_DB
QUOTA UNLIMITED ON EM_EVENTSTORE_DB
TEMPORARY TABLESPACE temp
QUOTA 0M ON system;

--Modify the user name if the default user name was changed

GRANT CREATE session to EVENTDB_USER;
GRANT CREATE table to EVENTDB_USER;
GRANT CREATE procedure to EVENTDB_USER;
```

drop_event_store_schema.sql

```
DROP TABLE eventdb_user.EMHostNameVistastccom8080;
drop sequence eventdb_user.autoincrement_id;
```

drop_event_store_user.sql

```
--Drop the user that was created earlier. Note that if you chose a
different name for the
--user while creating the user, you will have to specify that name here.
DROP USER EVENTDB_USER CASCADE;
```

```
--Drop the tablespace that was created earlier. Note that if you chose a
different name for
--the tablespace while creating the user, you will have to specify that name here.
DROP TABLESPACE EM_EVENTSTORE_DB INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;

--Manually delete the datafiles that were created. If you used the defaults
while creating
--the datafiles, the names would be EVENTDB_USER_DB1.dat'and 'EVENTDB_USER_DB2.dat'
```

设置脚本引擎

可以从任何 JSR-223 脚本环境中调用 Java CAPS 管理和监视 API。本节介绍了如何设置这些脚本引擎（如 Groovy、JRuby、Jython (Java Python)、JACL (Java TCL)）或能够调用当前可用的 Java CAPS 管理和监视 API 的 25 个其他 JSR-223 脚本引擎之一。可以使用这些说明设置您自己的环境，以便从所选的 JSR-223 脚本环境中调用这些 API。

下载、安装和设置脚本环境

就本主题而言，我们重点介绍了以下 4 个脚本引擎：

- Groovy：Java CAPS 管理和监视 API 经测试可用于 Groovy 1.0、Groovy 1.1 beta 1 和最新的 Groovy 1.1 beta 2。请使用提供的安装说明安装该引擎，并确保其正常运行。从 `http://groovy.codehaus.org/` 中下载 Groovy。
- JRuby：Java CAPS 管理和监视 API 经测试可用于 JRuby 1.0 和最新的 JRuby 1.0.1。请使用提供的安装说明安装该引擎，并确保其正常运行。从 `http://jruby.codehaus.org/` 中下载 JRuby。
- Jython (Java Python)：Java CAPS 管理和监视 API 经测试可用于 Jython 2.2 RC1 和最新的 Jython 2.2。请使用提供的安装说明安装该引擎，并确保其正常运行。从 `http://www.jython.org/Project/index.html` 中下载 Jython。
- JACL (Java TCL)：Java CAPS 管理和监视 API 经测试可用于最新的 JACL 1.4.0。请使用提供的安装说明安装该引擎，并确保其正常运行。您可能需要按照下载中包含的 `readme.txt` 文件所述，修改 `bin` 文件夹中的 `jaclsh.bat`。修改后的 `bin/jaclsh.bat` 即如上所述。请从 `http://tcljava.sourceforge.net/docs/website/index.html` 中下载 JACL。

注 – Java CAPS 6 软件包以压缩文件形式附带提供了服务样例文件（如 `AdministrationServiceSample.groovy`）和目标选项行为。

设置脚本环境以调用 Java CAPS 管理和监视 API

在设置脚本环境以调用 Java CAPS 管理和监视 API 之前，需要使用以下两个文件：

- `env.bat`
- `caps.management.client.jar`

注 – Java CAPS 软件包附带提供了这些文件。

在下载 `env.bat` 和 `caps.management.client.jar` 后，请修改 `env.bat` 中的环境变量。

要修改环境变量，请单击第 48 页中的 “修改 env.bat 中的环境变量”。

▼ 修改 env.bat 中的环境变量

1. **将 JAVA_HOME 变量设置为安装 JDK 或 JRE 的 JDK/JRE 主文件夹。**
   例如：`set JAVA_HOME=C:\java\jdk1.6.0`。

2. **将 SJSAS_HOME 变量设置为安装 Sun Java System Application Server 的 SJSAS 主文件夹。**
   例如：`set SJSAS_HOME=C:\CAPS6\SJSAS`。

3. **将 CAPS_MANAGEMENT_HOME 变量设置为本地保存 Java CAPS 管理和监视 API Java 归档文件的文件夹。**
   例如：`set CAPS_MANAGEMENT_HOME=C:\scripting\engines\common`。

4. **将 ENGINE_HOME 变量设置为安装脚本引擎的脚本引擎主文件夹。**
   例如：`set ENGINE_HOME=C:\scripting\engines\groovy\groovy-1.1-beta-2`。

注 – 现在，您就已做好了运行脚本样例准备。

使用脚本语言执行 Java CAPS 管理和监视 API

在运行脚本样例之前，假定您安装了以下脚本引擎之一：

- Groovy（位于 `C:\scripting\engines\groovy\groovy-1.1-beta-2` 中）
- JRuby（位于 `C:\scripting\engines\jruby\jruby-1.0.1` 中）
- Jython（位于 `C:\scripting\engines\jython\jython2.2` 中）
- JACL（位于 `C:\scripting\engines\jacl\jacl140` 中）

它还假定正在使用以下设置运行 Sun Java System Application Server：

主机	HTTP 管理端口	管理员用户名	管理员密码
localhost	4848	admin	adminadmin

提示 – 如果您的环境与之不同，请根据您的环境或安装更改以下步骤或脚本。

目前，Sun 支持以下这些服务：

- 管理服务
- 配置服务
- 部署服务
- 安装服务
- 运行时管理服务

以后还会支持其他服务。

执行管理服务

在执行管理服务之前，必须在环境中安装和/或部署以下项：

- 名为 sun-http-binding 的绑定组件
- 名为 sun-wsdl-library 的共享库
- 名为 SynchronousSampleApplication 的服务组合件，它包含一个名为 SynchronousSampleApplication-SynchronousSample 的服务单元

提示 – 在运行管理服务脚本之前，必须从 NetBeans 中创建 SynchronousSampleApplication，然后部署并启动它。

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行管理服务

`AdministrationServiceTest.groovy` 中附加了用于执行管理服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\AdministrationServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行配置服务

在执行配置服务之前，您必须在环境中安装以下项：

- 名为 sun-http-binding 的绑定组件
- 名为 sun-bpel-engine 的服务引擎

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行配置服务

`AdministrationServiceTest.groovy` 中附加了用于执行配置服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\ConfigurationServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行部署服务

部署服务尝试部署一个名为 SynchronousSampleApplication 的服务组合件。在运行部署服务脚本之前，必须先从 NetBeans 中创建 SynchronousSampleApplication，然后部署并启动它。请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行部署服务

`DeploymentServiceTest.groovy` 中附加了用于执行部署服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\DeploymentServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行安装服务

在执行安装服务之前，必须在环境中安装 `aspectserviceengine.jar`。

注意 – 请仔细阅读以下内容。

1. 名为 sun-aspect-engine 的服务引擎尝试停止、关闭并卸载其本身。
2. 然后，该服务引擎尝试安装附加的 `aspectserviceengine.jar` 中的 sun-aspect-engine。
3. 在运行安装服务样例脚本之前，请在环境中安装并启动 `aspectserviceengine.jar`。
4. 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行安装服务

`InstallationServiceTest.groovy` 中附加了用于执行安装服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\InstallationServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行运行时管理服务

在执行运行时管理服务之前，必须在环境中安装并部署以下项：

- 名为 sun-bpel-engine 的服务引擎
- 名为 SynchronousSampleApplication 的服务组合件（从 NetBeans 中创建）。

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行运行时管理服务

RuntimeManagmentServiceTest.groovy 中附加了用于执行运行时管理服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\RuntimeManagementServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行 JMS 管理服务

在执行 JMS 管理服务之前，必须先从 Java CAPS 6 安装程序中安装 Sun JMS IQ Manager，然后将其部署到环境中：

注 – 默认情况下，可以在安装过程中使用 Java CAPS 安装程序安装 Sun JMS IQ Manager。不过，您也可以在安装 Java CAPS 6 后随时安装 Sun JMS IQ Manager。

在 Groovy 中执行 JMS 管理服务

JMSManagmentServiceTest.groovy 中附加了用于执行 JMS 管理服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\JMSManagementServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行 BPEL 管理服务

在执行 BPEL 管理服务之前，您必须确保：

- 为 BPEL 服务引擎启用了监视
- 在环境中安装并部署了名为 sun-bpel-engine 的服务引擎

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行 BPEL 管理服务

`BPELManagmentServiceTest.groovy` 中附加了用于执行 BPEL 管理服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\BPELManagementServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行 HTTP 管理服务

在执行 HTTP 管理服务之前，必须在环境中安装并部署以下项：

- 名为 sun-http-binding 的绑定组件
- 包含某些 HTTP 使用端点的服务组合件

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行 HTTP 管理服务

`HTTPAdiministrationServiceTest.groovy` 中附加了用于执行 HTTP 管理服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\HTTPAdministrationServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

执行通知服务

对于通知服务测试，除了已启动并运行 AppServer 以外，没有任何其他前提条件。当 JBI 运行时环境中的任何生命周期发生变化时，通知服务会将这些通知发送到所有订阅的客户机。

注 – 请对该脚本进行更改，以使其适合在您的环境中运行。

在 Groovy 中执行通知服务

`NotificationServiceTest.groovy` 中附加了用于执行通知服务的 Groovy 脚本。请先根据需要修改该脚本，然后再在您的环境/安装中执行它。

```
cd C:\JavaCAPS6\managementapi
env.bat
cd scripting\groovy\scripts\
groovy ServiceTest\NotificationServiceTest.groovy
```

如果您习惯使用基于 Swing 的 groovyConsole，请使用它加载并执行脚本文件。

```
cd C:\scripting\groovy\scripts\
env.bat
groovyConsole
```

集成到 NetBeans IDE 中的 JRuby

如果具有最新版本的 NetBeans IDE 6.0，您可以创建自己的 JRuby 项目，并从 NetBeans IDE 中执行 JRuby 样例（它们具有 "rb" 扩展名）。

注 – NetBeans IDE 6.1 ML 附带提供了 Java CAPS 6。

▼ 创建 JRuby 项目

1. 创建新的 JRuby 项目。
2. 将 JRuby 脚本文件复制到该项目中。
3. 单击鼠标右键并打开“项目属性”选项卡，然后提供项目使用的相应 jar 文件。
4. 启动 Sun Java System Application Server。
5. 在 NetBeans 中打开一个脚本文件，右键单击源代码，然后单击“运行文件”以运行脚本。