# 为您的应用程序开发一个插件

## SnapCenter Software 4.7

NetApp
January 18, 2024

# 目录

# 为您的应用程序开发一个插件

## 概述

### 通过 SnapCenter 服务器，您可以将应用程序作为 SnapCenter 的插件进行部署和管理。您选择的应用程序可以插入到 SnapCenter 服务器中，以实现数据保护和管理功能。

通过 SnapCenter ，您可以使用不同的编程语言开发自定义插件。您可以使用 Perl ， Java ，批处理或其他脚本语言开发自定义插件。

要在 SnapCenter 中使用自定义插件，必须执行以下任务：

- 按照本指南中的说明为您的应用程序创建一个插件
- 创建问题描述文件
- 导出自定义插件以将其安装在 SnapCenter 主机上
- 将此插件 zip 文件上传到 SnapCenter 服务器

### 所有 **API** 调用中的通用插件处理

### 对于每个 API 调用，请使用以下信息：

- 插件参数
- 退出代码
- 记录错误消息
- 数据一致性

使用插件参数

在每次进行 API 调用时，都会向插件传递一组参数。下表列出了参数的特定信息。

| 参数 | 目的 |
| --- | --- |
| Action | 确定工作流名称。例如， discover ， backup ， fileOrVolRestore 或 cloneVolAndLun |
| Resources | 列出要保护的资源。资源由 UID 和类型标识。此列表将按以下格式显示给此插件：<br><br>"<UID> ， <type> ； <UID> ， <type>" 。例如， " 实例 1 ，实例；实例 2\\DB1 ，数据库 " |
| 应用程序名称 | 确定正在使用的插件。例如 DB2 ， MySQL 。SnapCenter 服务器内置了对所列应用程序的支持。此参数区分大小写。 |

| 参数 | 目的 |
| --- | --- |
| app_ignore_error | （Y 或 N ）如果遇到应用程序错误，则会导致 SnapCenter 退出或不退出。如果您要备份多个数据库，而不希望单个故障停止备份操作，则此功能非常有用。 |
| <resource_name>_app_instance_username | 已为资源设置 SnapCenter 凭据。 |
| <resource_name>_app_instance_password | 已为资源设置 SnapCenter 凭据。 |
| <resource_name>_<custom_param> | 每个资源级别自定义密钥值均可供插件使用，并以 "<resource_name>_ " 开头。例如，如果名为 MySQLDB 的资源的自定义密钥为 "master_slave" ，则该密钥将作为 MySQLDB_master_slave 可用 |

使用退出代码

此插件通过退出代码将操作状态返回给主机。每个代码都有一个特定的含义，此插件使用正确的退出代码来指示相同的含义。

下表介绍了错误代码及其含义。

| 退出代码 | 目的 |
| --- | --- |
| 0 | 操作成功。 |
| 99 | 不支持或不实施请求的操作。 |
| 100 | 操作失败，跳过静默并退出。默认情况下，取消静默状态为。 |
| 101. | 操作失败，请继续执行备份操作。 |
| 其他 | 操作失败，运行 unquiesce 并退出。 |

记录错误消息

错误消息将从插件传递到 SnapCenter 服务器。此消息包括消息，日志级别和时间戳。

下表列出了级别及其用途。

| 参数 | 目的 |
| --- | --- |
| 信息 | 信息性消息 |
| 警告 | 警告消息 |

| 参数 | 目的 |
| --- | --- |
| error | 错误消息 |
| 调试 | 调试消息 |
| 跟踪 | 跟踪消息 |

保持数据一致性

自定义插件可在执行相同工作流的操作之间保留数据。例如，插件可以在暂停结束时存储数据，可在取消静默操作期间使用。

要保留的数据将通过插件作为结果对象的一部分进行设置。它采用特定格式，并在每种插件开发模式下进行详细介绍。

# 基于 **Perl** 的开发

## 在使用 Perl 开发插件时，必须遵循某些约定。

- 内容必须可读
- 必须实施强制操作 setenv ， quiesce 和 unquiesce
- 必须使用特定语法将结果传递回代理
- 这些内容应保存为 <plugin_name>.pm 文件

可用操作包括

- 设置
- version
- 暂停
- 取消静默
- clone_pre ， clone_post
- restore_pree ， restore
- 清理

## 常规插件处理

### 使用 **Results** 对象

每个自定义插件操作都必须定义结果对象。此对象会将消息，退出代码， stdout 和 stderr 发送回主机代理。

Results 对象：

```
my $result = {
```

```
      exit_code => 0,
      stdout => "",
      stderr => "",
};
```

返回结果对象：

```
return $result;
```

保持数据一致性

在执行同一工作流期间，可以在操作之间保留数据（清理除外）。这可通过密钥值对来实现。关键值数据对作为结果对象的一部分进行设置，并在同一工作流的后续操作中保留和使用。

以下代码示例将设置要保留的数据：

```
my $result = {
  exit_code => 0,
  stdout => "",
  stderr => "",
};
  $result->{env}->{'key1'} = 'value1';
  $result->{env}->{'key2'} = 'value2';
  ….
  return $result
```

上述代码设置了两个键值对，这些键值对可在后续操作中用作输入。可以使用以下代码访问这两个键值对：

```
sub setENV {
    my ($self, $config) = @_;
    my $first_value = $config->{'key1'};
    my $second_value = $config->{'key2'};
    …
}
```

```
 === Logging error messages
```
每个操作都可以将消息发送回主机代理，主机代理将显示和存储内容。消息包含消息级别，时间戳和消息文本。支持多行消息。

```
Load the SnapCreator::Event Class:
my $msgObj = new SnapCreator::Event();
my @message_a = ();
```

使用 msgObj 通过使用收集方法捕获消息。

```
$msgObj->collect(\@message_a, INFO, "My INFO Message");
$msgObj->collect(\@message_a, WARN, "My WARN Message");
$msgObj->collect(\@message_a, ERROR, "My ERROR Message");
$msgObj->collect(\@message_a, DEBUG, "My DEBUG Message");
$msgObj->collect(\@message_a, TRACE, "My TRACE Message");
```

将消息应用于结果对象：

```
$result->{message} = \@message_a;
```

使用插件存根

自定义插件必须公开插件存根。这些方法是 SnapCenter 服务器根据工作流调用的。

| 插件存根 | 可选 / 必需 | 目的 |
| --- | --- | --- |
| 设置 | Required | 此存根用于设置环境和配置对象。<br><br>任何环境解析或处理都应在此处完成。每次调用存根时，都会在之前调用 setenv 存根。只有 Perl 模式插件才需要此功能。 |
| version | 可选 | 此存根用于获取应用程序版本。 |
| 发现 | 可选 | 此存根用于发现代理或主机上托管的实例或数据库等应用程序对象。<br><br>在响应过程中，此插件应以特定格式返回发现的应用程序对象。只有在应用程序与适用于 Unix 的 SnapDrive 集成时，才会使用此存根。<br><br>支持 Linux 文件系统（Linux Flavors）。不支持 AIX/Solaris （Unix 模式）。 |

| 插件存根 | 可选 / 必需 | 目的 |
| --- | --- | --- |
| discovery_complete | 可选 | 此存根用于发现代理或主机上托管的实例或数据库等应用程序对象。<br><br>在响应过程中，此插件应以特定格式返回发现的应用程序对象。只有在应用程序与适用于 Unix 的 SnapDrive 集成时，才会使用此存根。<br><br>支持 Linux 文件系统（Linux Flavors）。不支持 AIX 和 Solaris （Unix 模式）。 |
| 暂停 | Required | 此存根负责执行暂停，这意味着将应用程序置于可以创建 Snapshot 副本的状态。此操作在执行 Snapshot 副本操作之前调用。要保留的应用程序元数据应设置为响应的一部分，在对相应的存储 Snapshot 副本执行后续克隆或还原操作期间，应以配置参数的形式返回。 |
| 取消静默 | Required | 此存根负责执行静默，这意味着将应用程序置于正常状态。创建 Snapshot 副本后会调用此命令。 |
| clone_pre | 可选 | 此存根负责执行克隆前任务。此操作假定您使用的是内置的 SnapCenter 服务器克隆接口，并在执行克隆操作时触发。 |
| clone_post | 可选 | 此存根负责执行克隆后任务。这假定您使用的是内置的 SnapCenter 服务器克隆接口，并且只有在执行克隆操作时才会触发。 |
| restore_pre | 可选 | 此存根负责执行预存储任务。此操作假定您使用的是内置的 SnapCenter 服务器还原界面，并且是在执行还原操作时触发的。 |
| 还原 | 可选 | 此存根负责执行应用程序还原任务。这假定您使用的是内置的 SnapCenter 服务器还原界面，并且只有在执行还原操作时才会触发。 |

| 插件存根 | 可选 / 必需 | 目的 |
| --- | --- | --- |
| 清理 | 可选 | 此存根负责在执行备份，还原或克隆操作后执行清理。清理可以在正常工作流执行期间进行，也可以在工作流出现故障时进行。您可以通过引用配置参数操作来推断调用清理时使用的工作流名称，该操作可以是 backup ， cloneVolAndLun 或 fileOrVolRestore 。配置参数 error_message 用于指示执行工作流时是否存在任何错误。如果已定义 error_message ，而不是 NULL ，则在执行工作流失败期间会调用清理。 |
| APP_VERSION | 可选 | SnapCenter 使用此存根来获取此插件管理的应用程序版本详细信息。 |

插件软件包信息

每个插件都必须具有以下信息：

```
package MOCK;
our @ISA = qw(SnapCreator::Mod);
=head1 NAME
MOCK - class which represents a MOCK module.
=cut
=head1 DESCRIPTION
MOCK implements methods which only log requests.
=cut
use strict;
use warnings;
use diagnostics;
use SnapCreator::Util::Generic qw ( trim isEmpty );
use SnapCreator::Util::OS qw ( isWindows isUnix getUid
createTmpFile );
use SnapCreator::Event qw ( INFO ERROR WARN DEBUG COMMENT ASUP
CMD DUMP );
my $msgObj = new SnapCreator::Event();
my %config_h = ();
```

操作

您可以对自定义插件支持的各种操作进行编码，例如 setenv ， Version ， Quiesce 和 Unquiesce 。

#### setenv 操作

使用 Perl 创建的插件需要执行 setenv 操作。您可以设置 ENV 并轻松访问插件参数。

```
sub setENV {
    my ($self, $obj) = @_;
    %config_h = %{$obj};
    my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
    };
    return $result;
}
```

#### 版本操作

版本操作将返回应用程序版本信息。

```
sub version {
  my $version_result = {
    major => 1,
    minor => 2,
    patch => 1,
    build => 0
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
  $version_result->{message} = \@message_a;
  return $version_result;
}
```

#### 暂停操作

暂停操作会对 Resources 参数中列出的资源执行应用程序暂停操作。

```
sub quiesce {
  my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
  $result->{message} = \@message_a;
  return $result;
}
```

取消暂停操作

要取消应用程序静默，需要执行 " 取消暂停 " 操作。资源列表位于 Resources 参数中。

```
sub unquiesce {
  my $result = {
      exit_code => 0,
      stdout => "",
      stderr => "",
  };
  my @message_a = ();
  $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
  $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::unquiesce");
  $result->{message} = \@message_a;
  return $result;
}
```

# 原生模式

## SnapCenter 支持使用非 Perl 编程或脚本语言来创建插件。这称为原生模式编程，可以是脚本或批处理文件。

本机模式插件必须遵循以下特定约定：

此插件必须是可执行的

- 对于 Unix 系统，运行代理的用户必须对此插件具有执行权限

- 对于 Windows 系统， PowerShell 插件的后缀必须为 .ps1 ，而其他 Windows 脚本的后缀必须为 .cmd 或 .bat ，并且必须可由用户执行
- 插件必须对命令行参数做出响应，例如 "-quiesce" ， "-unquiesce"
- 如果未实施某项操作或功能，插件必须返回退出代码 99
- 插件必须使用特定语法将结果传递回服务器

## 常规插件处理

### 记录错误消息

每个操作都可以将消息发送回服务器，服务器将显示和存储内容。消息包含消息级别，时间戳和消息文本。支持多行消息。

格式。

```
SC_MSG#<level>#<timestamp>#<message>
SC_MESSAGE#<level>#<timestamp>#<message>
```

### 使用插件存根

SnapCenter 插件必须实施插件存根。这些方法是 SnapCenter 服务器根据特定工作流调用的。

| 插件存根 | 可选 / 必需 | 目的 |
|---|---|---|
| 暂停 | Required | 此存根负责执行暂停。它会将应用程序置于一种可以创建 Snapshot 副本的状态。此操作在执行存储 Snapshot 副本操作之前调用。 |
| 取消静默 | Required | 此存根负责执行静默。它会将应用程序置于正常状态。此操作在执行存储 Snapshot 副本操作后调用。 |
| clone_pre | 可选 | 此存根负责执行克隆前任务。此操作假定您使用的是内置的 SnapCenter 克隆接口，并且只有在执行操作 "clone_vol 或 clone_LUN" 时才会触发此接口。 |
| clone_post | 可选 | 此存根负责执行克隆后任务。这假定您使用的是内置的 SnapCenter 克隆接口，并且只有在执行 "clone_vol 或 clone_lun" 操作时才会触发此接口。 |

| 插件存根 | 可选 / 必需 | 目的 |
| --- | --- | --- |
| restore_pre | 可选 | 此存根负责执行还原前任务。这假定您使用的是内置的 SnapCenter 还原界面，并且仅在执行还原操作时触发。 |
| 还原 | 可选 | 此存根负责执行所有还原操作。此操作假定您未使用内置还原界面。它会在执行还原操作时触发。 |

示例

**Windows PowerShell**

检查是否可以在您的系统上执行此脚本。如果无法执行此脚本，请为此脚本设置 Set-ExecutionPolicy bypass ，然后重试此操作。

```
if ($args.length -ne 1) {
  write-warning "You must specify a method";
  break;
}
function log ($level, $message) {
  $d = get-date
  echo "SC_MSG#$level#$d#$message"
}
function quiesce {
  $app_name = (get-item env:APP_NAME).value
  log "INFO" "Quiescing application using script $app_name";
  log "INFO" "Quiescing application finished successfully"
}
function unquiesce {
  $app_name = (get-item env:APP_NAME).value
  log "INFO" "Unquiescing application using script $app_name";
  log "INFO" "Unquiescing application finished successfully"
}
  switch ($args[0]) {
    "-quiesce" {
    quiesce;
}
"-unquiesce" {
    unquiesce;
}
default {
    write-error "Function $args[0] is not implemented";
    exit 99;
  }
}
exit 0;
```

# Java 模式

Java 自定义插件直接与数据库，实例等应用程序交互。

## 限制

在使用 Java 编程语言开发插件时，您应注意一些限制。

| 插件特征 | **Java** 插件 |
| --- | --- |
| 复杂性 | 从低到中 |

| 插件特征 | **Java** 插件 |
| --- | --- |
| 内存占用空间 | 最多 10-20 MB |
| 其他库的依赖关系 | 用于应用程序通信的库 |
| 线程数 | 1. |
| 线程运行时 | 不到一小时 |

**Java** 限制的原因

SnapCenter 代理的目标是确保持续，安全且稳定可靠的应用程序集成。通过支持 Java 插件，插件可能会导致内存泄漏和其他不需要的问题。这些问题很难解决，尤其是在目标是让事情简单易用的情况下。如果插件的复杂性不是太复杂，开发人员可能会出现错误的可能性就会小得多。Java 插件的风险在于，它们与 SnapCenter 代理本身在同一 JVM 中运行。当插件崩溃或内存泄漏时，它也可能对代理产生负面影响。

## 支持的方法

| 方法 | **Required** | **Description** | 何时被调用？由谁调用？ |
| --- | --- | --- | --- |
| version | 是的。 | 需要返回插件的版本。 | 由 SnapCenter 服务器或代理请求插件的版本。 |
| 暂停 | 是的。 | 需要对应用程序执行暂停。在大多数情况下，这意味着将应用程序置于 SnapCenter 服务器可以创建备份的状态（例如 Snapshot 副本）。 | 在 SnapCenter 服务器创建 Snapshot 副本或执行常规备份之前。 |
| 取消静默 | 是的。 | 需要对应用程序执行静默操作。在大多数情况下，这意味着将应用程序重新置于正常运行状态。 | 在 SnapCenter 服务器创建 Snapshot 副本或执行常规备份之后。 |
| 清理 | 否 | 负责清理插件需要清理的任何内容。 | SnapCenter 服务器上的工作流完成后（成功或出现故障）。 |
| clonePre | 否 | 应在执行克隆操作之前执行需要执行的操作。 | 当用户触发 "cloneVol" 或 "cloneLun" 操作并使用内置克隆向导（ GUI/CLI ）时。 |

| 方法 | Required | Description | 何时被调用？由谁调用？ |
| --- | --- | --- | --- |
| clonePost | 否 | 应在执行克隆操作后执行需要执行的操作。 | 当用户触发 "cloneVol" 或 "cloneLun" 操作并使用内置克隆向导（ GUI/CLI ）时。 |
| 还原前 | 否 | 应在调用还原操作之前执行需要执行的操作。 | 用户触发还原操作时。 |
| 还原 | 否 | 负责执行应用程序的还原 / 恢复。 | 用户触发还原操作时。 |
| 应用程序版本 | 否 | 以检索插件管理的应用程序版本。 | 作为备份 / 还原 / 克隆等每个工作流中 ASUP 数据收集的一部分。 |

## 教程

本节介绍如何使用 Java 编程语言创建自定义插件。

### 设置 **Eclipse**

1. 在 Eclipse 中创建一个新的 Java 项目 "TuttorialPlug"
2. 单击 * 完成 * 。
3. 右键单击 * 新项目 * → * 属性 * → * Java 构建路径 * → * 库 * → * 添加外部 JAR *
4. 导航到主机代理的 ./lib/ 文件夹，然后选择 JAR scAgent-5.0-core.jar 和 common-5.0.jar
5. 选择项目并右键单击 * 源文件夹 * → * 新增 * → * 软件包 * ，然后创建一个名为 com.netapp.snapcreator.agent.plugin.TutorialPlugin 的新软件包
6. 右键单击新软件包并选择新建 → Java 类。
   a. 输入名称作为 TuttorialPlugin 。
   b. 单击超类浏览按钮并搜索 "* 抽象插件 " 。只应显示一个结果：

```
 "AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".
.. 单击 * 完成 * 。
.. Java 类：
```

```
package com.netapp.snapcreator.agent.plugin.TutorialPlugin;
import
com.netapp.snapcreator.agent.nextgen.common.result.Describe
Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.VersionR
esult;
import
com.netapp.snapcreator.agent.nextgen.context.Context;
import
com.netapp.snapcreator.agent.nextgen.plugin.AbstractPlugin;
public class TutorialPlugin extends AbstractPlugin {
  @Override
  public DescribeResult describe(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public Result quiesce(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public Result unquiesce(Context context) {
    // TODO Auto-generated method stub
    return null;
  }
  @Override
  public VersionResult version() {
    // TODO Auto-generated method stub
    return null;
  }
}
```

实施所需的方法

暂停，取消静默和版本是每个自定义 Java 插件必须实施的强制方法。

以下是返回插件版本的版本方法。

```
@Override
public VersionResult version() {
    VersionResult versionResult = VersionResult.builder()
                                            .withMajor(1)
                                            .withMinor(0)
                                            .withPatch(0)
                                            .withBuild(0)
                                            .build();
    return versionResult;
}
```

```
Below is the implementation of quiesce and unquiesce method. These will be
interacting with   the application, which is being protected by SnapCenter
Server. As this is just a tutorial, the
application part is not explained, and the focus is more on the
functionality that SnapCenter   Agent provides the following to the plug-
in developers:
```

```
@Override
  public Result quiesce(Context context) {
    final Logger logger = context.getLogger();
    /*
      * TODO: Add application interaction here
    */
```

```
logger.error("Something bad happened.");
logger.info("Successfully handled application");
```

```
    Result result = Result.builder()
                    .withExitCode(0)
                    .withMessages(logger.getMessages())
                    .build();
    return result;
}
```

方法在上下文对象中传递。其中包含多个帮助程序，例如 Logger 和上下文存储，以及有关当前操作的信息（工作流 ID ，作业 ID ）。我们可以通过调用 final Logger logger = context.getLogger （）来获取此日志程序。logger 对象提供了其他日志记录框架中已知的类似方法，例如，登录回。在 result 对象中，您还可以指定退出代码。在此示例中，返回零，因为没有问题描述。其他退出代码可以映射到不同的故障情形。

正在使用结果对象

result 对象包含以下参数：

| 参数 | Default | Description |
| --- | --- | --- |
| 配置 | 空配置 | 此参数可用于将配置参数发送回服务器。它可以是插件要更新的参数。此更改是否实际反映在 SnapCenter 服务器的配置中取决于配置中的 APP_CONF_persistency=Y 或 N 参数。 |
| ExitCode | 0 | 指示操作的状态。"0" 表示操作已成功执行。其他值表示错误或警告。 |
| 标准输出 | 空列表 | 这可用于将 stdout 消息传输回 SnapCenter 服务器。 |
| 标准 | 空列表 | 这可用于将 stderr 消息传输回 SnapCenter 服务器。 |
| 消息 | 空列表 | 此列表包含插件要返回到服务器的所有消息。SnapCenter 服务器会在命令行界面或图形用户界面中显示这些消息。 |

SnapCenter 代理可提供构建程序 ("构建程序模式"）。这使得使用它们变得非常简单：

```
Result result = Result.builder()
                     .withExitCode(0)
                     .withStdout(stdout)
                     .withStderr(stderr)
                     .withConfig(config)
                     .withMessages(logger.getMessages())
                     .build()
```

例如，将退出代码设置为 0 ，为 stdout 和 stderr 设置列表，设置配置参数，并附加要发送回服务器的日志消息。如果您不需要所有参数，请仅发送所需的参数。由于每个参数都有一个默认值，因此，如果从以下代码中删除 .withExitCode （ 0 ），则结果不会受到影响：

```
Result result = Result.builder()
                       .withExitCode(0)
                       .withMessages(logger.getMessages())
                       .build();
```

版本结果

VersionResult 会向 SnapCenter 服务器通知插件版本。由于它还会从结果继承，因此它包含 config ， exitCode ， stdout ， stderr 和 messages 参数。

| 参数 | **Default** | **Description** |
| --- | --- | --- |
| major | 0 | 插件的主要版本字段。 |
| 次要 | 0 | 插件的次要版本字段。 |
| patch | 0 | 插件的修补程序版本字段。 |
| build | 0 | 此插件的 Build version 字段。 |

例如：

```
VersionResult result = VersionResult.builder()
                                  .withMajor(1)
                                  .withMinor(0)
                                  .withPatch(0)
                                  .withBuild(0)
                                  .build();
```

使用上下文对象

上下文对象提供了以下方法：

| 上下文方法 | 目的 |
| --- | --- |
| 字符串 getWorkflowId（）； | 返回 SnapCenter 服务器在当前工作流中使用的工作流 ID 。 |
| config getconfig（）； | 返回正在从 SnapCenter 服务器发送到代理的配置。 |

工作流 **ID**

工作流 ID 是 SnapCenter 服务器用于引用特定正在运行的工作流的 ID 。

配置

此对象包含（大多数）用户可在 SnapCenter 服务器的配置中设置的参数。但是，由于安全原因，其中某些参数可能会在服务器端进行筛选。以下是有关如何访问 Config 并检索参数的示例：

```
final Config config = context.getConfig();
String myParameter =
config.getParameter("PLUGIN_MANDATORY_PARAMETER");
```

现在， "/" myParameter " 包含从 SnapCenter 服务器上的配置中读取的参数如果配置参数密钥不存在，则它将返回空字符串（ "" ）。

导出插件

要在 SnapCenter 主机上安装此插件，必须导出此插件。

在 Eclipse 中，执行以下任务：

1. 右键单击插件的基础软件包（在我们的示例 com.netapp.snapcreator.agent.plugin.TutorialPlugin 中）。
2. 选择 * 导出 * → * Java * → * JAR 文件 *
3. 单击 * 下一步 * 。
4. 在以下窗口中，指定目标 JAR 文件路径： tutorial_plugin.jar 插件的基础类名为 TuttorialPlugin 。 class ，必须将此插件添加到同名文件夹中。

如果插件依赖于其他库，则可以创建以下文件夹： lib/

您可以添加与插件相关的 JAR 文件（例如数据库驱动程序）。当 SnapCenter 加载此插件时，它会自动将此文件夹中的所有 JAR 文件与其关联，并将其添加到类路径中。

# SnapCenter 中的自定义插件

## SnapCenter 中的自定义插件

可以使用 SnapCenter 服务器将使用 Java ， Perl 或原生模式创建的自定义插件安装在主机上，以便为应用程序启用数据保护。您必须已导出此插件，才能使用本教程中提供的操作步骤将其安装到 SnapCenter 主机上。

创建插件问题描述文件

对于创建的每个插件，您都必须具有一个问题描述文件。问题描述文件介绍了此插件的详细信息。文件名必须为 Plugin_Descriptioner.xml 。

使用插件描述符文件属性及其重要性

| 属性 | Description |
| --- | --- |
| Name | 插件的名称。允许使用字母数字字符。例如， DB2 ， MySQL ， MongoDB<br><br>对于以原生模式创建的插件，请确保不提供文件扩展名。例如，如果此插件的名称是 MongoDB.sh ，请将此名称指定为 MongoDB 。 |

| 属性 | **Description** |
| --- | --- |
| version | 插件版本。可以包括主要版本和次要版本。例如， 1.0 ， 1.1 ， 2.0 ， 2.1 |
| displayName | 要在 SnapCenter 服务器中显示的插件名称。如果写入了同一插件的多个版本，请确保所有版本的显示名称相同。 |
| 插件类型 | 用于创建插件的语言。支持的值包括 Perl ， Java 和原生。原生插件类型包括 Unix/Linux Shell 脚本，Windows 脚本， Python 或任何其他脚本语言。 |
| OSNAME | 安装此插件的主机操作系统名称。有效值为 Windows 和 Linux 。可以在多种操作系统类型上部署一个插件，例如 Perl 类型的插件。 |
| 操作系统配置 | 安装了插件的主机操作系统版本。 |
| 资源名称 | 插件可以支持的资源类型的名称。例如，数据库，实例，集合。 |
| 父级 | 在这种情况下， ResourceName 在层次结构上依赖于另一种资源类型，然后 Parent 确定父 ResourceType 。<br><br>例如，在 DB2 插件中， ResourceName " 数据库 " 具有父 " 实例 " 。 |
| RequireFileSystemPlugin | 是或否确定恢复选项卡是否显示在还原向导中。 |
| ResourceRequireAuthentication | 是或否确定自动发现或尚未自动发现的资源在发现存储后是否需要凭据来执行数据保护操作。 |
| RequireFileSystemClone | 是或否确定此插件是否需要为克隆工作流集成文件系统插件。 |

以下是自定义插件 DB2 的 Plugin_descriptor.xml 文件示例：

```
<Plugin>
<SMSServer></SMSServer>
<Name>DB2</Name>
<Version>1.0</Version>
<PluginType>Perl</PluginType>
<DisplayName>Custom DB2 Plugin</DisplayName>
<SupportedOS>
<OS>
<OSName>windows</OSName>
<OSVersion>2012</OSVersion>
</OS>
<OS>
<OSName>Linux</OSName>
<OSVersion>7</OSVersion>
</OS>
</SupportedOS>
<ResourceTypes>
<ResourceType>
<ResourceName>Database</ResourceName>
<Parent>Instance</Parent>
</ResourceType>
<ResourceType>
<ResourceName>Instance</ResourceName>
</ResourceType>
</ResourceTypes>
<RequireFileSystemPlugin>no</RequireFileSystemPlugin>
<ResourceRequiresAuthentication>yes</ResourceRequiresAuthentication>
<SupportsApplicationRecovery>yes</SupportsApplicationRecovery>
</Plugin>
```

创建 **ZIP** 文件

开发插件并创建描述符文件后，必须将插件文件和 Plugin_descriptor.xml 文件添加到文件夹中并将其压缩。

在创建 ZIP 文件之前，您必须考虑以下事项：

- 脚本名称必须与插件名称相同。
- 对于 Perl 插件， ZIP 文件夹必须包含一个包含脚本文件的文件夹，并且描述符文件必须位于此文件夹之外。文件夹名称必须与插件名称相同。
- 对于 Perl 插件以外的插件， ZIP 文件夹必须包含描述符和脚本文件。
- 操作系统版本必须为数字。

示例

- DB2 插件：将 db2.pm 和 Plugin_descriptor.xml 文件添加到 "db2.zip " 中。

- 使用 Java 开发的插件：将 JAR 文件，相关的 JAR 文件和 Plugin_descriptor.xml 文件添加到文件夹中并将其压缩。

上传插件 **ZIP** 文件

您必须将此插件 ZIP 文件上传到 SnapCenter 服务器，才能在所需主机上部署此插件。

您可以使用 UI 或 cmdlet 上传此插件。

- 用户界面： *
- 在 * 添加 * 或 * 修改主机 * 工作流向导中上传插件 ZIP 文件
- 单击 * " 选择以上传自定义插件 "*
- PowerShell： *
- upload-SmPluginPackage cmdlet

  例如， PS> Upload — SmPluginPackage — AbsolutePath c： \DB2_1.zip

  有关 PowerShell cmdlet 的详细信息，请使用 SnapCenter cmdlet 帮助或参阅 cmdlet 参考信息。

"《 SnapCenter 软件 cmdlet 参考指南》"。

部署自定义插件

现在，在 * 添加 * 和 * 修改主机 * 工作流中，可以在所需主机上部署上传的自定义插件。您可以将多个版本的插件上传到 SnapCenter 服务器，并且可以选择要在特定主机上部署的所需版本。

有关如何上传此插件的详细信息，请参见： "添加主机并在远程主机上安装插件软件包"