HP Operations Orchestration

适用于 Windows 和 Linux

软件版本： 10.10

动作开发人员指南

文档发布日期： 2014 年 5 月

软件发布日期： 2014 年 5 月

法律声明

担保

HP 产品和服务的唯一担保已在此类产品和服务随附的明示担保声明中提出。此处的任何内容均不构成额外担保。HP 不会为此处出现的技术或编辑错误或遗漏承担任何责任。

此处所含信息如有更改，恕不另行通知。

受限权利声明

机密计算机软件。必须拥有 HP 授予的有效许可证，方可拥有、使用或复制本软件。按照 FAR 12.211 和 12.212，并根据供应商的标准商业许可的规定，商业计算机软件、计算机软件文档与商品技术数据授权给美国政府使用。

版权声明

商标声明

Adobe™ 是 Adobe Systems Incorporated 的商标。

此产品包含“zlib”通用压缩库的接口，版权所有© 1995-2002 Jean-loup Gailly and Mark Adler。

AMD 及 AMD 箭头符号是 Advanced Micro Devices, Inc. 的商标

Google™ 和 Google Maps™ 是 Google Inc. 的商标

Intel®、Itanium®、Pentium® 和 Intel® Xeon® 是 Intel Corporation 在美国及其他国家/地区的商标。

Java 是 Oracle 和/或其附属公司的注册商标。

Microsoft®、Windows®、Windows NT®、Windows® XP 和 Windows Vista® 是 Microsoft Corporation 在美国的注册商标。

Oracle 是 Oracle Corporation 和/或其附属公司的注册商标。

UNIX® 是 The Open Group 的注册商标。

文档更新

此文档的标题页包含以下标识信息：

- 软件版本号，用于指示软件版本。
- 文档发布日期，该日期将在每次更新文档时更改。
- 软件发布日期，用于指示该版本软件的发布日期。

要检查是否有最新的更新，或者验证是否正在使用最新版本的文档，请访问：**http://h20230.www2.hp.com/selfsolve/manuals**

需要注册 HP Passport 才能登录此站点。要注册 HP Passport ID，请访问：**http://h20229.www2.hp.com/passport-registration.html**

或单击“HP Passport”登录页面上的“**New users - please register**”链接。

此外，如果订阅了相应的产品支持服务，则还会收到更新的版本或新版本。有关详细信息，请与您的 HP 销售代表联系。

支持

请访问 HP 软件联机支持网站：**http://www.hp.com/go/hpsoftwaresupport**

此网站提供了联系信息，以及有关 HP 软件提供的产品、服务和支持的详细信息。

HP 软件联机支持提供客户自助解决功能。通过该联机支持，可快速高效地访问用于管理业务的各种交互式技术支持工具。作为尊贵的支持客户，您可以通过该支持网站获得下列支持：

- 搜索感兴趣的知识文档
- 提交并跟踪支持案例和改进请求
- 下载软件修补程序
- 管理支持合同
- 查找 HP 支持联系人
- 查看有关可用服务的信息
- 参与其他软件客户的讨论
- 研究和注册软件培训

大多数提供支持的区域都要求您注册为 HP Passport 用户再登录，很多区域还要求用户提供支持合同。要注册 HP Passport ID，请访问：

http://h20229.www2.hp.com/passport-registration.html

要查找有关访问级别的详细信息，请访问：

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Now 可访问 HPSW 解决方案和集成门户网站。此网站将帮助您寻找可满足您业务需求的 HP 产品解决方案，包括 HP 产品之间的集成的完整列表以及 ITIL 流程的列表。此网站的 URL 为 **http://h20230.www2.hp.com/sc/solutions/index.jsp**

目录

目录 3
开发 HP OO 的扩展 4
创建 @Action 5
开发插件 5
准备使用 Maven 原型创建插件 5
使用 Maven 原型创建插件 6
开发 @Action 9
"Hello World!"示例 10
将参数传递给 @Action 10
返回值 11
添加 @Action 注释 11
注释 11
@Action 数据定义示例 14
测试扩展 14
将扩展作为项目构建的一部分进行测试 14
从命令行独立测试扩展 14
.NET 扩展 15
旧动作 18

开发 HP OO 的扩展

本文档为 Java 和 .NET 开发人员提供各种准则，以供开发用于扩展 HP Operations Orchestration 的动作。

备注：需要具备 Java 或 .NET 知识。

您可以采用编程方式扩展 HP Operations Orchestration。这意味着第三方可将功能添加到 HP Operations Orchestration，然后将其作为内容引入流执行引擎。

引入新内容需要构建扩展，并将其部署到 HP OO Central。您可以采用 Java 或 .NET 编写动作。

在 HP Operations Orchestration 10.x 中，扩展称为“插件”(在先前版本中，扩展称为 IAction)。插件是在运行引擎中运行的一段代码。这段代码可以为自己定义独立的类路径。独立的类路径可确保不同的插件能使用相冲突的依赖关系。例如，插件 A 可使用版本为 1.0 的 X 依赖关系，而插件 B 也可以使用相同的 X 依赖关系，不过版本为 2.0。现在，无论是否出现类路径相冲突的问题，您都仍能在相同的流中同时使用这两个插件。

插件包含一个或多个动作，并且可引用所有必需的依赖关系。

在 HP Operations Orchestration 10.x 中，有一个用于开发动作的新 @Action 接口。@Action 是指类中的方法。有关详细信息，请参阅开发 @Action (第 9 页)。

虽然所有插件均采用 Java 运行，但是 HP Operation Orchestration 也支持 .NET 动作。采用 .NET 编写的动作将由包含的 Java 插件引用。有关详细信息，请参阅.NET 扩展 (第 15 页)。

备注：现在，HP OO 9.x 中的 IAction 接口已弃用。用户编写新内容时，不应实施 IAction 接口，而应编写 @Action 的接口。

创建 @Action

构建 @Action 的建议方式是作为 Maven 插件构建。

您应使用 Apache Maven 3.0.3 - 3.0.5 构建插件。

开发插件

此部分描述如何开发插件。

使用 Maven 原型 **com.hp.oo.sdk:oo-plugin-archetype** 可以创建插件和 Studio 项目的结构。

准备使用 Maven 原型创建插件

安装 Maven

在计算机路径中具有 **bin** 目录的计算机上安装 Maven。这将支持您从文件系统的任何位置运行 **mvn**。

创建本地 Maven 存储库

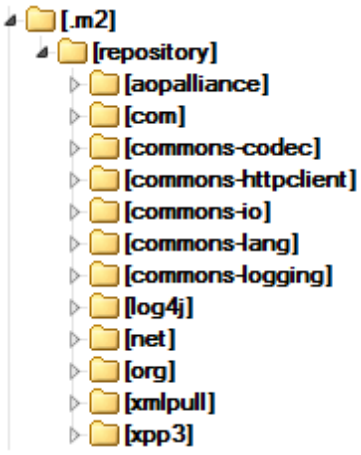
- 将 `sdk-dotnet-<版本>.zip` 和 sdk-java-<版本>.zip 展开至：

  Windows：`%HOMEPATH%\.m2\repository`。

  Linux：`$HOME/.m2/repository`。

  > **备注：**
  >
  > 这些文件均位于“SDK 文件夹”下的 ISO 中。

以下是在正确提取文件的情况下的目录结构示例：



注册插件原型

- 打开命令提示符，并输入以下命令：

  ```
  mvn archetype:crawl
  ```

  这将更新 $HOME/.m2/repository 下的 Maven 原型目录。

使用 Maven 原型创建插件

创建示例项目

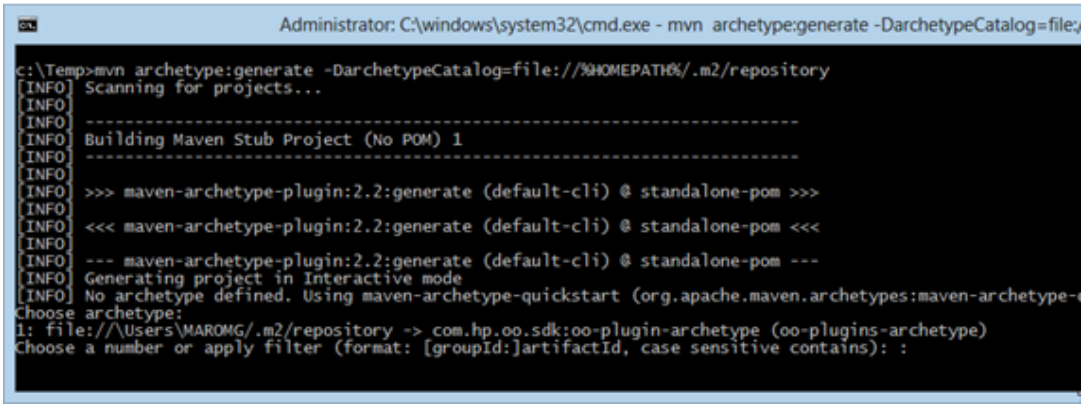
1. 转至要创建示例插件项目的路径，并在命令行中输入以下命令：

   ```
   mvn archetype:generate -DarchetypeCatalog=file://$HOME/.m2/repository
   ```

   > **备注：**对于 Windows，使用 %HOMEPATH%。

   这将启动项目创建。将显示在目录中找到的原型列表。按代表原型 **com.hp.oo.sdk:oo-plugin-archetype** 的数字，然后按 **Enter**。

   在下例中，您按 **1**。

2. 在原型创建期间，输入以下详细信息，输入一个按一次 **Enter**：
   - groupId：结果 Maven 项目的组 ID。在下例中使用的是 acmeGroup。
   - artifactId：结果 Maven 项目的项目 ID，在下例中使用的是 acmeArtifact。
   - package：项目中文件的包。此选项的默认值与 groupId 相同。

```
Define value for property 'groupId': : acmeGroup
Define value for property 'artifactId': : acmeArtifact
[INFO] Using property: version = 1.0.0
Define value for property 'package':  acmeGroup: :
Confirm properties configuration:
groupId: acmeGroup
artifactId: acmeArtifact
version: 1.0.0
package: acmeGroup
 Y: :
```

此时将完成构建，且项目也已创建。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 18.112s
[INFO] Finished at: Mon Jun 17 21:17:50 IDT 2013
[INFO] Final Memory: 13M/490M
[INFO] ------------------------------------------------------------------------
c:\Temp>
```

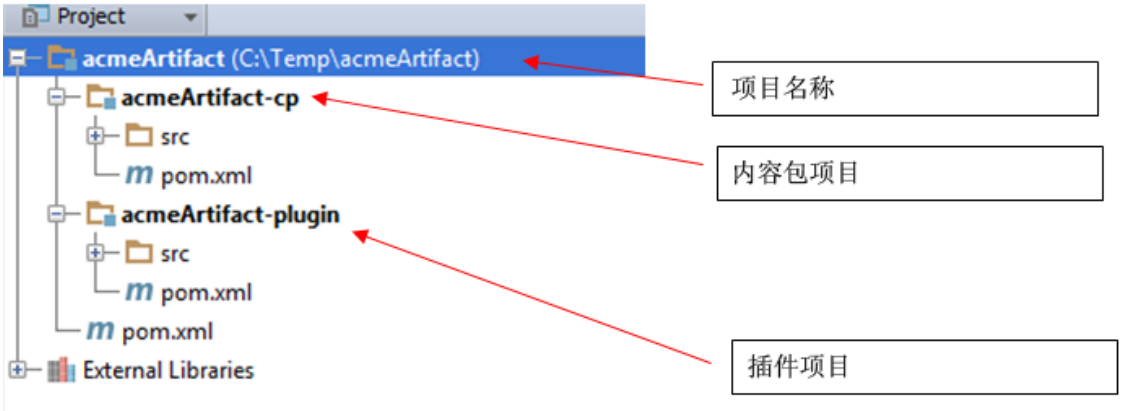
在 Java IDE 中打开项目

上一步创建了新的 Java 项目以及基于 Maven 的模型。

在 Java IDE 应用程序中打开此项目。

该项目包含两个模块，均以相同前缀作为提供的项目 ID。其中一个项目是内容包项目，另一个是插件项目。

例如：



父项目

在如图所示的示例中，父项目称为 **acmeArtifact**。

默认情况下它包含两个模块：一个是内容包，另一个是插件。此项目旨在将 @Action 及其相关操作和流分组到单个内容包中。

例如，如果在开发 Office 集成，则可以创建数个插件项目(每个 Office 版本一个项目)。但是，还会有一个包含操作和流的内容包项目。这就是建议的最佳实践。

插件项目

在示例中，插件项目称为 **acmeArtifact-plugin**。

此模块包含 @Action。当(使用 Maven)构建此项目时，将编译内部代码，并且可以在 Studio 中打开结果 JAR 文件，还可以从 @Action 内部创建操作。

在此模块中可以找到示例 @Action。可以删除它并编写自己的 @Action。

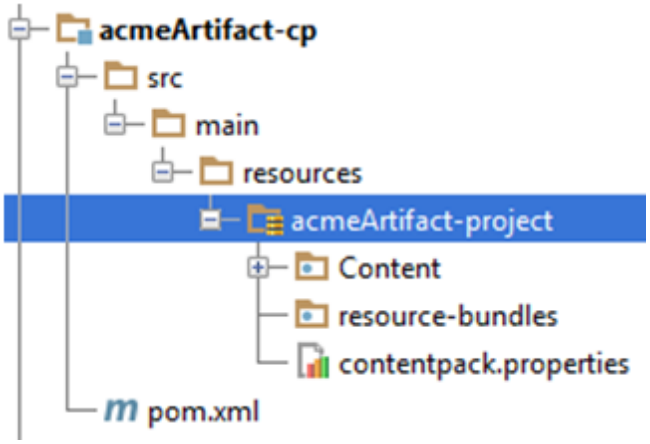
内容包项目

在示例中，内容包项目称为 **acmeArtifact-cp**。

此模块表示内容包。它包含任何所依赖的插件模块(例如 **acmeArtifact-plugin**)，以及在其中定义的任何流、操作和配置项。

使用 Studio 编辑内容包模块

内容包模块包含可在 Studio 项目中打开和编辑的 Studio 项目。您可以将项目文件夹(在此示例中即 **acmeArtifact-project** 文件夹)导入到 Studio 中，以便编辑项目。

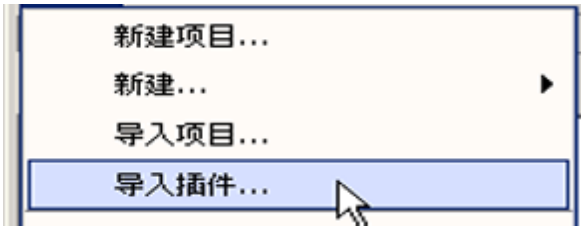


如果在项目内创建流、操作或配置项，则可以使用 Maven 构建此项目。生成的 JAR 文件将是一个可稍后部署到 Central 中或由其他用户在 Studio 重新打开的内容包。

使用插件模块在项目内部创建操作

如果要使用属于同一 **acmeArtifact** 项目的插件(例如 **acmeArtifact-plugin**)创建操作，请执行以下步骤：

1. 将 **acmeArtifact-project** 项目导入到 Studio 中。
2. 使用 Maven 构建 **acmeArtifact-plugin** 模块。
3. 将插件导入到 Studio。



插件的路径即是本地 Maven 存储库中 **acmeArtifact-plugin.jar** 文件的路径。

4. 在 Studio 中创建新的操作并在“创建操作”对话框中选择插件。

使用 Maven 创建内容包

如果要使用 Maven(而不是 Studio)创建内容包，可以在 **acmeArtifact-cp** 模块的 POM 文件中添加依赖关系：

```
<plugin>
    <groupId>${sdk.group}</groupId>
    <artifactId>oo-contentpack-maven-plugin</artifactId>
    <version>${sdk.version}</version>
    <dependencies>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>acmeArtifact-plugin</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <id>generate-contentpack-plugin</id>
            <phase>process-sources</phase>
            <goals>
                <goal>generate-contentpack</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <destinationFolder>${project.build.outputDirectory}</destinationFolder>
        <artifactItems>
            <artifactItem>
                <groupId>${project.groupId}</groupId>
                <artifactId>acmeArtifact-plugin</artifactId>
                <version>${project.version}</version>
            </artifactItem>
        </artifactItems>
    </configuration>
</plugin>
```

开发 @Action

@Action 是类中的方法，因此它可以是任何类中的任何方法。此方法也称为扩展。

使用 @Action 执行操作时，将在流执行期间调用此 @Action。

"Hello World!"示例

要将方法标记为 @Action，请使用 @com.hp.oo.sdk.content.annotations.Action 对其进行注释。以下内容是简单的“Hello World!”@Action 示例：

```
public class MyActions {
    @Action
    public void sayHello() {
        System.out.println("Hello World!");
    }
}
```

默认情况下，创建的 @Action 将根据对其进行定义的方法进行命名。在“Hello World!”示例中，@Action 的名称为 sayHello。@Action 名称将在操作定义中使用。此操作是向 Studio 和流创建人公开 @Action 的一种方式。每项操作都指向特定的 groupId、artifactId、version 和 @Action 名称(GAV + @Action 名称)。

您可以自定义 @Action 名称，并提供一个与方法名称不同的名称。您可以使用 @Action 注释的值参数来完成此操作。以下代码可定义相同的“Hello World!”@Action，但是需将其命名为 my-hello-action：

```
public class MyActions {
    @Action(“my-hello-action”)
    public void sayHello() {
        System.out.println("Hello World!");
    }
}
```

将参数传递给 @Action

@Action 将向流上下文公开，并可从中请求参数。流上下文将保留流的状态。例如，假如以下 @Action 已添加两个数字，并将结果输出到控制台：

```
@Action
public void sum(int x, int y){
    System.out.println(x+y);
}
```

参数将按名称从上下文中获取。sum 方法从上下文请求两个整型参数 x 和 y。调用 @Action 时，HP Operations Orchestration 将上下文中的值 x 和 y 分配到具有相同名称的方法参数。

与 @Action 一样，可以自定义参数名称，并请求 HP Operations Orchestration 在使用自定义名称的同时解析此值。在以下示例中，sum 方法请求将上下文 op1 参数分配给 x 参数，并将 op2 分配给 y 参数：

```
@Action
public void sum(@Param("op1") int x, @Param("op2") int y){
    System.out.println(x+y);
}
```

com.hp.oo.sdk.content.constants 包下的类 **ResponseNames**、**ReturnCodes**、**InputNames** 和 **OutputNames** 包括可以在 @Action 中使用的常用常量。例如，输入名称(如 HOST、USERNAME、PASSWORD、PORT 等)或响应名称(如 SUCCESS、FAILURE、NO_MORE 等)。

返回值

@Action 与任何 Java 方法一样，也可以返回单个值。返回的值可视为 @Action 的返回结果，也可在操作中用作 return result。@Action 也可以将多个结果返回到操作中。这可以通过返回 Map<String, String> 来完成，其中，Map 键是结果的名称，关联的值是结果值。返回 Map<String, String> 是 @Action 在运行时将多个输出传递到操作的一种方式。

添加 @Action 注释

@Action 注释用于在 Studio 中生成新的操作。生成基于 @Action 的操作之后，全新操作的初始属性(描述、输入、输出、响应)将从 @Action 注释定义中获取。

在开发插件时，您必须为只返回一个值的动作正确添加注释。注释必须使用特殊名称 singleResultKey 声明输出。常量 ActionExecutionGoal.SINGLE_RESULT_KEY 十分有用，例如：

```
@Action(name = "modulo-ten",
    description = "returns the last digit",
    outputs = @Output(ActionExecutionGoal.SINGLE_RESULT_KEY),
    responses = @Response(text = ResponseNames.SUCCESS,
    field = OutputNames.RETURN_RESULT,
    value = "0", matchType = MatchType.ALWAYS_MATCH,
    responseType = ResponseType.RESOLVED)
  )
  public int moduloTen(@Param("number") int number) {
      return number % 10;
  }
```

备注: 使用 @Action 注释对于您而言十分重要，否则将更难使用从这些 @Action 创建的操作。

注释

添加元数据意味着添加或设置相关注释及其属性。下表描述了 @Action、@Output、@Response 和 @Param 的注释：

动作

属性：

- value(可选)：@Action 的名称
- description(可选)

- Output[](可选)：输出数组(如下所示)
- Response[](可选)：响应数组(如下所示)

注释：

有两个选项可设置 @Action 的名称：

1. value 属性：

```
@Action("aflPing")
public void ping(…)
```

或者

```
@Action(value="aflPing")
public void ping(…)
```

2. 方法名称为：

```
@Action
public void ping(…)
```

名称将按上述顺序进行检查。第一个接受检查的是 value 属性。如果不存在，则将选择方法名称。

参数

属性：

- value：输入的名称
- required(可选)：默认情况下为 false
- encrypted(可选)：默认情况下为 false
- description(可选)

注释：

这不仅对于 @Action 数据而言十分重要，对于执行也十分重要。

输入将为操作或流提供执行所需的数据。每个输入均会映射到一个变量。您可以为流、操作或步骤创建输入。

在 Studio 中，输入可以：

- 设置为特定的值
- 从通过其他步骤收集的信息中获取
- 在流开始时，由运行流的人员输入。

有关详细信息，请参阅《HP OO 10 Studio 创建指南》；有关执行功能的详细信息，请参阅将参数传递给 @Action (第 10 页)。

输出

属性：

- value：输出的名称
- description(可选)

注释：

为了使 Studio 中的操作拥有多个输出，@Action 自身必须声明这些输出。通过创建返回值为 Map<String, String> 的 @Action，可将值分配给多个输出。

为了使 Studio 中的操作只有一个输出，@Action 自身必须在返回值中声明此输出，并使用 SINGLE_RESULT_KEY 进行绑定。

输出是操作或流生成的数据，例如成功代码、输出字符串、错误字符串或失败消息。

在 Studio 中，不同的操作输出种类包括：

- 原始结果：返回的完整数据(返回代码、数据输出和错误字符串)。
- 属于原始结果一部分的主输出和其他输出。

有关详细信息，请参阅《HP OO 10 Studio 创建指南》。

响应

属性：

- text：每次响应转换显示的文本
- field：要评估的字段
- value：field 中的预期值
- description：(可选)
- isDefault：表示这是否是默认响应。默认值为 false。@Action 中仅有一个响应可将此值设置为 true。
- mathType：要针对值激活的匹配器类型。例如，如果我们已定义(field = fieldName，value = 0，matchType = COMPARE_GREATER)，这就意味着如果字段 fieldName 拥有的值大于 0，则将选择此响应。
- responseType：响应的类型(Success、Failure、Diagnosed、No_Action 或 Resolve_By_Name)。
- isOnFail：表示这是否是“失败”响应。默认值为 false。@Action 中仅有一个响应可将此值设置为 true。
- ruleDefined：表示响应是否已定义规则。没有定义规则的响应可用作默认响应。一个 @Action 中仅能有一个未定义规则的响应。

注释：

响应有可能是操作或流的结果。响应包含一个规则：field 与 value 匹配。

有关详细信息，请参阅《HP OO 10 Studio 创建指南》。

@Action 数据定义示例

```
@Action(value = "aflPing",
        description = "perform a dummy ping",
        outputs = {@Output(value = RETURN_RESULT, description ="returnResult description"),
                   @Output(RETURN_CODE),
                   @Output("packetsSent"),
                   @Output("packetsReceived"),
                   @Output("percentagePacketsLost"),
                   @Output("transmissionTimeMin"),
                   @Output("transmissionTimeMax"),
                   @Output("transmissionTimeAvg")},
        responses = {@Response(text = "success", field = RETURN_CODE, value = PASSED),
                     @Response(text = "failure", field = RETURN_CODE, value = FAILED)})

public Map<String, String> doPing(
    @Param(value = "targetHost",
           required = true,
           encrypted = false,
           description = "the host to ping") String targetHost,
    @Param("packetCount") String packetCount,
    @Param("packetSize") String packetSize) {
    …
}
```

测试扩展

将扩展作为项目构建的一部分进行测试

由于 @Action 是一种简单的 Java 方法，因此可以使用标准的 Java 测试工具(如 JUnit)进行测试，并充分利用 Maven 项目的正常生命周期阶段。

由于 @Action 本身是一种常规方法，因此它不需要调用任何 HP Operations Orchestration 组件。在测试案例中，调用可以是直接的 Java 方法调用。

从命令行独立测试扩展

打包到插件之后，您可以从命令行调用扩展，以供测试。以下是 @Action 示例：

```
public class TestActions {
    @Action
    public int sum(@Param("op1") int x, @Param("op2") int y){
        return x+y;
    }
}
```

假设 TestActions 类位于带有以下 groupId、artifactId 和 version (GAV) 的插件中：**com.mycompany:my-actions:1.0**

您可以按如下所示从命令行调用 sum @Action：

```
mvn com.mycompany:my-actions:1.0:execute -Daction=sum -Dop1=1 -Dop2=3 -X
```

此命令的结果是较长的跟踪。需要使用 -X 选项查看日志消息。在跟踪的末尾处，您将看到：

```
[DEBUG] Configuring mojo 'com.mycompany:my-actions:1.0::execute' with basic configurator -->
[DEBUG]   (f) actionName = sum
[DEBUG]   (f) session = org.apache.maven.execution.MavenSession@21cfa61c
[DEBUG] -- end configuration --
[DEBUG] Action result:action result = 4
```

.NET 扩展

要使用 .NET 动作创建内容，您需要执行以下操作：

1. 创建包含所需 @Action 实施的 DLL 文件，就像在版本 9.x 中一样。@Action 类应实施 IAction 接口。
2. 使用 `mvn install:install-file` 将创建的 DLL(包括引用的库)部署到本地 Maven 存储库。有关安装不是由 Maven 构建的项目的详细信息，请参阅 http://maven.apache.org/plugins/maven-install-plugin/usage.html
3. 生成包含 .NET 动作的 HP OO Maven 插件。要执行此操作，您需要：
   a. 创建 **pom.xml** 文件。有关 POM 参考，请参阅 http://maven.apache.org/pom.html。
   b. 在 <dependencies> 下，添加包含所有所需 DLL 的列表。使用 <type>dll</type> 定义所有 DLL 项目。
   c. 从包含 **pom.xml** 文件的文件夹运行 `mvn install` 命令。这是考虑到 Maven **bin** 文件夹包含在系统路径中。

结果是位于目标文件夹且安装到本地 Maven 存储库的 Maven 插件。目标文件夹位置与当前文件夹相对。

pom.xml 的内容为：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/mav
en-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>[my plugin groupId]</groupId>
    <artifactId>[my plugin artifactId]</artifactId>
    <version>[my plugin version]</version>

    <packaging>maven-plugin</packaging>

    <properties>
        <oo-sdk.version>[THE LATEST HP OO_SDK VERSION]</oo-sdk.version>
        <oo-dotnet.version>[THE LATEST HP OO_DOTNET VERSION]</oo-dotnet.version>
    </properties>

    <dependencies>
```

```
        <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-dotnet-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>

        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-dotnet-legacy-plugin</artifactId>
            <version>${oo-dotnet.version}</version>
            <type>dll</type>
        </dependency>

        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>IAction</artifactId>
            <version>9.0</version>
            <type>dll</type>
        </dependency>
        <!-- end of required dependencies -->

        <dependency>
            <groupId>[groupId-1]</groupId>
            <artifactId>[artifactId-1]</artifactId>
            <version>[version-1]</version>
            <type>dll</type>
        </dependency>

        <dependency>
            <groupId>[groupId-2]</groupId>
            <artifactId>[artifactId-2]</artifactId>
            <version>[version-2]</version>
            <type>dll</type>
        </dependency>

...

        <dependency>
            <groupId>[groupId-n]</groupId>
            <artifactId>[artifactId-n]</artifactId>
            <version>[version-n]</version>
            <type>dll</type>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.hp.oo</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-dotnet-plugin</goal>
                        </goals>
                    </execution>
```

```
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

在以下示例中：

- POM 文件的名称为 **example.pom.xml**。
- **my-dotnet-actions.dll** 包含所需的 @Action。
- 生成的 Maven 插件为 **com.example:my-dotnet-plugin:1.0**。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.
0.0.xsd">


    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>my-dotnet-plugin</artifactId>
    <version>1.0</version>
    <packaging>maven-plugin</packaging>

    <properties>
        <oo-sdk.version>2.190</oo-sdk.version>
        <oo-dotnet.version>1.30</oo-dotnet.version>
    </properties>

    <dependencies>
      <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-dotnet-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-dotnet-legacy-plugin</artifactId>
            <version>${oo-dotnet.version}</version>
            <type>dll</type>
        </dependency>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>IAction</artifactId>
            <version>9.0</version>
            <type>dll</type>
        </dependency>
      <!-- end of required dependencies -->
        <dependency>
            <groupId>com.example</groupId>
            <artifactId>my-dotnet-actions</artifactId>
            <version>1.0</version>
            <type>dll</type>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>com.hp.oo</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-dotnet-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

旧动作

要使用旧动作创建内容，您需要执行以下操作：

1. 验证 JAR 是否包含所需动作的实施，就像在版本 9.x 中一样。动作类应实施 IAction 接口。

2. 使用 `mvn install:install-file` 将 JAR(包括引用的库)部署到本地 Maven 存储库。有关安装不是由 Maven 构建的项目的详细信息，请参阅 http://maven.apache.org/plugins/maven-install-plugin/usage.html

3. 生成包含旧动作库的 HP OO Maven 插件。要执行此操作，您需要：
   a. 创建 **pom.xml** 文件。有关 POM 参考，请参阅 http://maven.apache.org/pom.html。

   b. 在 `<dependencies>` 下，添加包含所有所需 JAR 的列表。

   c. 从包含 **pom.xml** 文件的文件夹运行 `mvn install` 命令。这是考虑到 Maven **bin** 文件夹包含在系统路径中。

结果是位于目标文件夹且安装到本地 Maven 存储库的 Maven 插件。目标文件夹位置与当前文件夹相对。

pom.xml 的内容为：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/mave
n-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>[my plugin groupId]</groupId>
    <artifactId>[my plugin artifactId]</artifactId>
    <version>[my plugin version]</version>
```

```
    <packaging>maven-plugin</packaging>

    <properties>
        <oo-sdk.version>[THE LATEST HP OO_SDK VERSION]</oo-sdk.version>
        <oo-dotnet.version>[THE LATEST HP OO_DOTNET VERSION]</oo-dotnet.version>
    </properties>


    <dependencies>
        <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-legacy-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>
        <!-- end of required dependencies -->

        <dependency>
            <groupId>[groupId-1]</groupId>
            <artifactId>[artifactId-1]</artifactId>
            <version>[version-1]</version>
        </dependency>

        <dependency>
            <groupId>[groupId-2]</groupId>
            <artifactId>[artifactId-2]</artifactId>
            <version>[version-2]</version>
        </dependency>

...

        <dependency>
            <groupId>[groupId-n]</groupId>
            <artifactId>[artifactId-n]</artifactId>
            <version>[version-n]</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.hp.oo</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-legacy-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

在以下示例中：

- POM 文件的名称为 **example.pom.xml**。
- **my-legacy-actions.jar** 包含所需动作。
- 生成的 Maven 插件为 **com.example:my-legacy-actions:1.0**。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/mav
en-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-legacy-actions-plugin</artifactId>
    <version>1.0</version>

    <packaging>maven-plugin</packaging>

    <properties>
        <oo-sdk.version>2.190</oo-sdk.version>
        <oo-dotnet.version>1.30</oo-dotnet.version>
    </properties>

    <dependencies>
        <!-- required dependencies -->
        <dependency>
            <groupId>com.hp.oo</groupId>
            <artifactId>oo-legacy-action-plugin</artifactId>
            <version>${oo-sdk.version}</version>
        </dependency>
        <!-- end of required dependencies -->

        <dependency>
            <groupId>com.example</groupId>
            <artifactId>my-legacy-actions</artifactId>
            <version>1.0</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>com.hp.oo</groupId>
                <artifactId>oo-action-plugin-maven-plugin</artifactId>
                <version>${oo-sdk.version}</version>
                <executions>
                    <execution>
                        <id>generate plugin</id>
                        <phase>process-sources</phase>
                        <goals>
                            <goal>generate-legacy-plugin</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
```

```
    </build>
</project>
```


Java™
COMPATIBLE