保护应用程序

Astra Control Center

NetApp
November 21, 2023

目录

保护应用程序

保护概述

您可以使用 Astra 控制中心为应用程序创建备份，克隆，快照和保护策略。备份应用程序可帮助您的服务和关联数据尽可能地可用；在灾难情形下，从备份还原可以确保应用程序及其关联数据的完全恢复，而不会造成任何中断。备份，克隆和快照有助于防止常见威胁，例如勒索软件，意外数据丢失和环境灾难。 "了解 Astra 控制中心提供的数据保护类型以及何时使用"。

应用程序保护工作流

您可以使用以下示例工作流开始保护应用程序。

[一个] 备份所有应用程序

要确保您的应用程序立即受到保护， "为所有应用程序创建手动备份"。

[两个] 为每个应用程序配置一个保护策略

要自动执行未来备份和快照， "为每个应用程序配置一个保护策略"。例如，您可以从每周备份和每日快照开始，这两种备份均保留一个月。强烈建议使用保护策略自动执行备份和快照，而不是手动备份和快照。

[三个] 可选：调整保护策略

随着应用程序及其使用模式的变化，根据需要调整保护策略以提供最佳保护。

[四个] 发生灾难时，请还原您的应用程序

如果发生数据丢失，您可以通过进行恢复 "还原最新备份" 每个应用程序的第一个。然后，您可以还原最新的快照（如果可用）。

通过快照和备份保护应用程序

通过使用自动保护策略或临时创建快照和备份来保护应用程序。您可以使用 Astra UI 或 "Astra Control API" 保护应用程序。

> 如果您使用 Helm 部署应用程序，则 Astra 控制中心需要 Helm 版本 3 。完全支持管理和克隆使用 Helm 3 部署的应用程序（或从 Helm 2 升级到 Helm 3 ）。不支持使用 Helm 2 部署的应用程序。

> 在 OpenShift 集群上创建用于托管应用程序的项目时，系统会为该项目（或 Kubernetes 命名空间）分配一个 SecurityContext UID 。要使 Astra 控制中心能够保护您的应用程序并将应用程序移动到 OpenShift 中的其他集群或项目，您需要添加策略，使应用程序能够作为任何 UID 运行。例如，以下 OpenShift 命令行界面命令会为 WordPress 应用程序授予相应的策略。
>
> ```
> oc new-project WordPress oc adm policy add-SCS-to-group anyuid system : serviceaccounts : WordPress oc adm policy add-SCS-to-user privileged -z default -n WordPress
> ```

配置保护策略

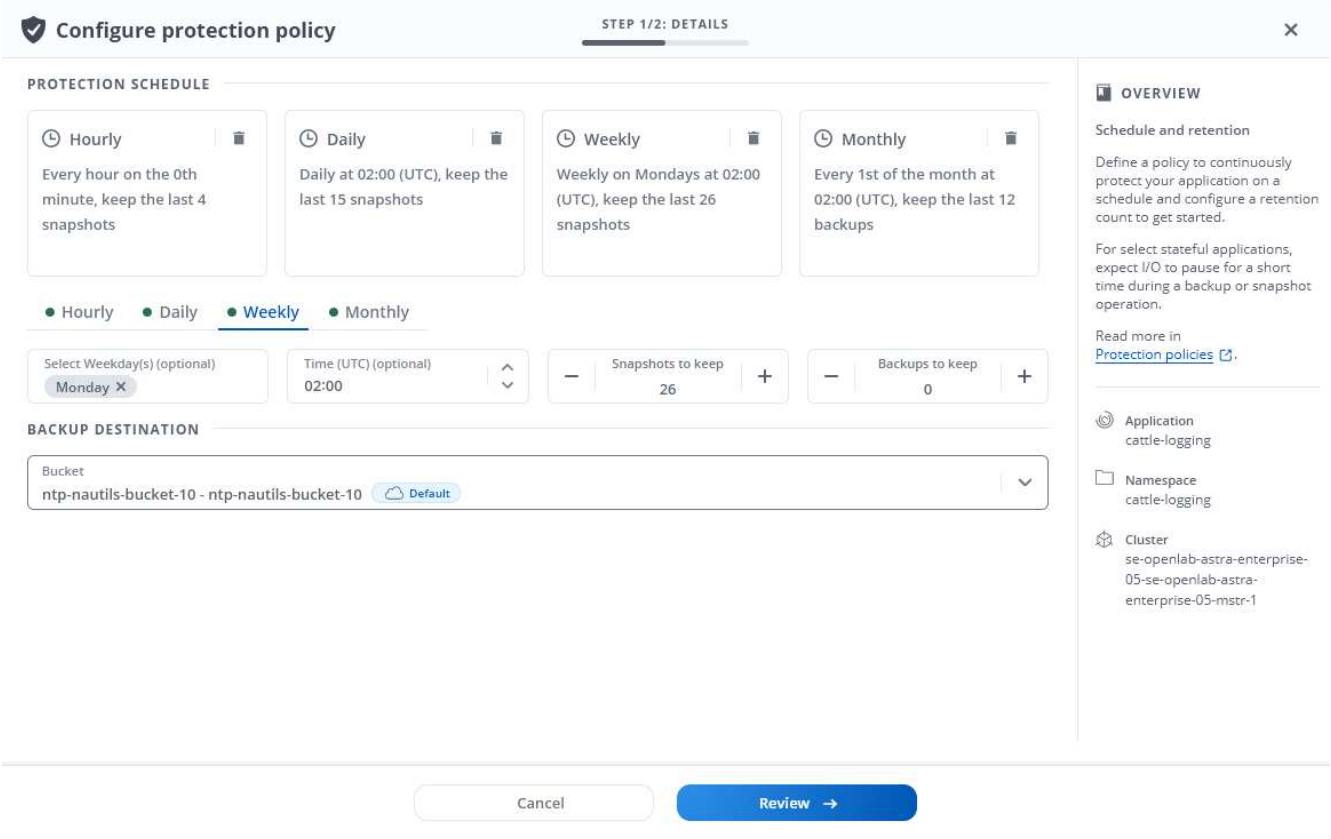
保护策略通过按定义的计划创建快照，备份或这两者来保护应用程序。您可以选择每小时，每天，每周和每月创建快照和备份，并且可以指定要保留的副本数。例如，保护策略可能会创建每周备份和每日快照，并将备份和快照保留一个月。创建快照和备份的频率以及保留时间取决于组织的需求。

步骤

1. 选择 * 应用程序 * ，然后选择应用程序的名称。
2. 选择 * 数据保护 * 。
3. 选择 * 配置保护策略 * 。
4. 通过选择每小时，每天，每周和每月保留的快照和备份数量来定义保护计划。

   您可以同时定义每小时，每天，每周和每月计划。在设置保留级别之前，计划不会变为活动状态。

   以下示例将为快照和备份设置四个保护计划：每小时，每天，每周和每月。



5. 选择 * 审阅 * 。
6. 选择 * 设置保护策略。 *

结果

Astra 控制中心通过使用您定义的计划和保留策略创建和保留快照和备份来实施数据保护策略。

创建快照

您可以随时创建按需快照。

步骤

1. 选择 * 应用程序 * 。
2. 从所需应用程序的 * 操作 * 列的选项菜单中，选择 * 快照 * 。
3. 自定义快照的名称，然后选择 * 审阅 * 。
4. 查看快照摘要并选择 * 快照 * 。

结果

快照过程开始。如果在 * 数据保护 * > * 快照 * 页面的 * 操作 * 列中的状态为 * 可用 * ，则快照将成功。

创建备份

您也可以随时备份应用程序。

> Astra 控制中心中的 S3 存储分段不会报告可用容量。在备份或克隆由 Astra 控制中心管理的应用程序之前，请检查 ONTAP 或 StorageGRID 管理系统中的存储分段信息。

步骤

1. 选择 * 应用程序 * 。
2. 从所需应用程序的 * 操作 * 列的选项菜单中，选择 * 备份 * 。
3. 自定义备份的名称。
4. 选择是否从现有快照备份应用程序。如果选择此选项，则可以从现有快照列表中进行选择。
5. 通过从存储分段列表中选择来选择备份的目标。
6. 选择 * 审阅 * 。
7. 查看备份摘要并选择 * 备份 * 。

结果

Astra 控制中心创建应用程序的备份。

> 如果网络发生中断或异常缓慢，备份操作可能会超时。这会导致备份失败。

> 无法停止正在运行的备份。如果需要删除备份，请等待备份完成，然后按照中的说明进行操作 [删除备份]。删除失败的备份， "使用 Astra Control API"。

> 在执行数据保护操作（克隆，备份，还原）并随后调整永久性卷大小后，在 UI 中显示新卷大小之前，最长会有 20 分钟的延迟。数据保护操作将在几分钟内成功完成，您可以使用存储后端的管理软件确认卷大小的更改。

查看快照和备份

您可以从数据保护选项卡查看应用程序的快照和备份。

步骤

1. 选择 * 应用程序 * ，然后选择应用程序的名称。

2. 选择 * 数据保护 * 。

   默认情况下会显示快照。

3. 选择 * 备份 * 可查看备份列表。

删除快照

删除不再需要的计划快照或按需快照。

步骤

1. 选择 * 应用程序 * ，然后选择应用程序的名称。
2. 选择 * 数据保护 * 。
3. 从选项菜单的 * 操作 * 列中为所需快照选择 * 删除快照 * 。
4. 键入单词 "delete" 确认删除，然后选择 * 是，删除 snapshot* 。

结果

Astra 控制中心会删除快照。

删除备份

删除不再需要的计划备份或按需备份。

> 无法停止正在运行的备份。如果需要删除备份，请等待备份完成，然后按照以下说明进行操作。删除失败的备份， "使用 Astra Control API"。

1. 选择 * 应用程序 * ，然后选择应用程序的名称。
2. 选择 * 数据保护 * 。
3. 选择 * 备份 * 。
4. 从选项菜单的 * 操作 * 列中为所需备份选择 * 删除备份 * 。
5. 键入单词 "delete" 确认删除，然后选择 * 是，删除备份 * 。

结果

Astra 控制中心删除备份。

还原应用程序

Astra Control 可以从快照或备份还原应用程序。将应用程序还原到同一集群时，从现有快照进行还原的速度会更快。您可以使用 Astra Control UI 或 "Astra Control API" 还原应用程序。

关于此任务

- 强烈建议在还原应用程序之前为其创建快照或备份。这样、您可以在还原失败时从快照或备份克隆。
- 如果您使用 Helm 部署应用程序，则 Astra 控制中心需要 Helm 版本 3 。完全支持管理和克隆使用 Helm 3

部署的应用程序（或从 Helm 2 升级到 Helm 3 ）。不支持使用 Helm 2 部署的应用程序。

- 如果要还原到其他集群，请确保此集群使用相同的永久性卷访问模式（例如 ReadWriteMany ）。如果目标永久性卷访问模式不同，还原操作将失败。
- 任何按命名空间名称 /ID 或命名空间标签限制命名空间的成员用户都可以将应用程序克隆或还原到同一集群上的新命名空间或其组织帐户中的任何其他集群。但是，同一用户无法访问新命名空间中的克隆或还原应用程序。通过克隆或还原操作创建新命名空间后，帐户管理员 / 所有者可以编辑成员用户帐户并更新受影响用户的角色约束，以授予对新命名空间的访问权限。
- 在 OpenShift 集群上创建用于托管应用程序的项目时，系统会为该项目（或 Kubernetes 命名空间）分配一个 SecurityContext UID 。要使 Astra 控制中心能够保护您的应用程序并将应用程序移动到 OpenShift 中的其他集群或项目，您需要添加策略，使应用程序能够作为任何 UID 运行。例如，以下 OpenShift 命令行界面命令会为 WordPress 应用程序授予相应的策略。

  ```
  oc new-project WordPress oc adm policy add-SCS-to-group anyuid system ：
  serviceaccounts ： WordPress oc adm policy add-SCS-to-user privileged -z
  default -n WordPress
  ```

步骤

1. 选择 * 应用程序 * ，然后选择应用程序的名称。
2. 选择 * 数据保护 * 。
3. 如果要从快照还原，请保持选中 * 快照 * 图标。否则，请选择 * 备份 * 图标以从备份中还原。
4. 从要还原的快照或备份的 * 操作 * 列的选项菜单中，选择 * 还原应用程序 * 。
5. * 还原详细信息 * ： 指定已还原应用程序的详细信息。默认情况下，将显示当前集群和命名空间。保留这些值不变，以便原位还原应用程序，从而将应用程序还原到其自身的早期版本。如果要还原到其他集群或命名空间，请更改这些值。
   - 输入应用程序的名称和命名空间。
   - 选择应用程序的目标集群。
   - 选择 * 审阅 * 。

   > 如果还原到先前已删除的命名空间、则在还原过程中会创建一个同名的新命名空间。任何有权管理先前删除的命名空间中的应用程序的用户都需要手动还原对新重新创建的命名空间的权限。

6. * 还原摘要 * ： 查看有关还原操作的详细信息，键入 "restore" ，然后选择 * 还原 * 。

结果

Astra 控制中心会根据您提供的信息还原应用程序。如果您已原位还原应用程序，则任何现有永久性卷的内容将替换为还原应用程序中的永久性卷的内容。

> 在执行数据保护操作(克隆、备份、还原)并随后调整永久性卷大小后、在Web UI中显示新卷大小之前、最多会有20分钟的延迟。数据保护操作将在几分钟内成功完成，您可以使用存储后端的管理软件确认卷大小的更改。

克隆和迁移应用程序

克隆现有应用程序以在同一个 Kubernetes 集群或另一个集群上创建重复的应用程序。当 Astra 控制中心克隆应用程序时，它会为您的应用程序配置和永久性存储创建一个克隆。

如果您需要将应用程序和存储从一个 Kubernetes 集群移动到另一个集群，则克隆可以助您一臂之力。例如，您可能希望通过 CI/CD 管道以及在 Kubernetes 命名空间之间移动工作负载。您可以使用 Astra UI 或 "Astra Control API" 克隆和迁移应用程序。

您需要的内容

要将应用程序克隆到其他集群，您需要一个默认存储分段。添加第一个存储分段时，它将成为默认存储分段。

关于此任务

- 如果您部署的应用程序明确设置了 StorageClass ，并且需要克隆该应用程序，则目标集群必须具有最初指定的 StorageClass 。将显式设置了 StorageClass 的应用程序克隆到不具有相同 StorageClass 的集群将失败。
- 如果克隆操作员部署的 Jenkins CI 实例，则需要手动还原永久性数据。这是应用程序部署模式的一个限制。
- Astra 控制中心中的 S3 存储分段不会报告可用容量。在备份或克隆由 Astra 控制中心管理的应用程序之前，请检查 ONTAP 或 StorageGRID 管理系统中的存储分段信息。
- 在应用程序备份或应用程序还原期间，您可以选择指定存储分段 ID 。但是，应用程序克隆操作始终使用已定义的默认分段。没有选项可用于更改克隆的分段。如果要控制使用哪个存储分段，您可以选择 "更改存储分段默认值" 或者执行 "backup" 后跟 A "还原" 请单独使用。
- 任何按命名空间名称 /ID 或命名空间标签限制命名空间的成员用户都可以将应用程序克隆或还原到同一集群上的新命名空间或其组织帐户中的任何其他集群。但是，同一用户无法访问新命名空间中的克隆或还原应用程序。通过克隆或还原操作创建新命名空间后，帐户管理员 / 所有者可以编辑成员用户帐户并更新受影响用户的角色约束，以授予对新命名空间的访问权限。

OpenShift 注意事项

- 如果您在集群之间克隆应用程序，则源集群和目标集群必须是 OpenShift 的同一分发版。例如，如果从 OpenShift 4.7 集群克隆应用程序，请使用同时也是 OpenShift 4.7 的目标集群。
- 在 OpenShift 集群上创建用于托管应用程序的项目时，系统会为该项目（或 Kubernetes 命名空间）分配一个 SecurityContext UID 。要使 Astra 控制中心能够保护您的应用程序并将应用程序移动到 OpenShift 中的其他集群或项目，您需要添加策略，使应用程序能够作为任何 UID 运行。例如，以下 OpenShift 命令行界面命令会为 WordPress 应用程序授予相应的策略。

```
oc new-project WordPress oc adm policy add-SCS-to-group anyuid system ：
serviceaccounts ： WordPress oc adm policy add-SCS-to-user privileged -z
default -n WordPress
```

步骤

1. 选择 * 应用程序 * 。
2. 执行以下操作之一：
   - 在 * 操作 * 列中选择所需应用程序的选项菜单。
   - 选择所需应用程序的名称，然后选择页面右上角的状态下拉列表。
3. 选择 * 克隆 * 。

4. * 克隆详细信息 * ：指定克隆的详细信息：
    - 输入名称。
    - 输入克隆的命名空间。
    - 选择克隆的目标集群。
    - 选择是要从现有快照还是备份创建克隆。如果不选择此选项，则 Astra 控制中心将根据应用程序的当前状态创建克隆。
5. * 源 * ：如果选择从现有快照或备份克隆，请选择要使用的快照或备份。
6. 选择 * 审阅 * 。
7. * 克隆摘要 * ：查看有关克隆的详细信息并选择 * 克隆 * 。

结果

Astra 控制中心会根据您提供的信息克隆该应用程序。如果新应用程序克隆在 * 应用程序 * 页面上处于 可用 状态，则克隆操作将成功。

> 在执行数据保护操作（克隆，备份，还原）并随后调整永久性卷大小后，在 UI 中显示新卷大小之前，最长会有 20 分钟的延迟。数据保护操作将在几分钟内成功完成，您可以使用存储后端的管理软件确认卷大小的更改。

管理应用程序执行挂钩

执行钩是一种自定义脚本，您可以在托管应用程序快照之前或之后运行该脚本。例如，如果您有一个数据库应用程序，则可以使用执行挂钩在快照之前暂停所有数据库事务，并在快照完成后恢复事务。这样可以确保应用程序一致的快照。

默认执行挂钩和正则表达式

对于某些应用程序， Astra Control 附带了 NetApp 提供的默认执行挂钩，用于处理快照前后的冻结和解冻操作。Astra Control 使用正则表达式将应用程序的容器映像与以下应用程序匹配：

- MariaDB
    - 匹配正则表达式： \bmariadb\b
- MySQL
    - 匹配正则表达式： \bmysql\b
- PostgreSQL
    - 匹配正则表达式： \bpostgresql\b

如果存在匹配项，则 NetApp 为该应用程序提供的默认执行挂钩将显示在该应用程序的活动执行挂钩列表中，这些挂钩将在该应用程序创建快照时自动运行。如果某个自定义应用程序的映像名称类似，恰好与其中一个正则表达式匹配（并且您不想使用默认执行挂钩），则可以更改映像名称， 或者禁用该应用程序的默认执行连接，而改用自定义连接。

您不能删除或修改默认执行挂钩。

有关自定义执行挂钩的重要注意事项

在为应用程序规划执行挂钩时，请考虑以下几点。

- Astra Control 要求以可执行 Shell 脚本的格式编写执行挂钩。
- 脚本大小限制为 128 KB 。
- Astra Control 使用执行挂钩设置和任何匹配条件来确定哪些挂钩适用于快照。
- 所有执行挂机故障均为软故障；即使某个挂机发生故障，仍会尝试使用其他挂机和快照。但是，如果挂机发生故障，则会在 * 活动 * 页面事件日志中记录一个警告事件。
- 要创建，编辑或删除执行挂钩，您必须是具有所有者，管理员或成员权限的用户。
- 如果执行挂机运行时间超过 25 分钟，则此挂机将失败，从而创建返回代码为不适用的事件日志条目。任何受影响的快照都将超时并标记为失败，并会生成一个事件日志条目，用于记录超时情况。

> 由于执行挂钩通常会减少或完全禁用其所运行的应用程序的功能，因此您应始终尽量缩短自定义执行挂钩运行所需的时间。

运行快照时，执行钩事件按以下顺序发生：

1. NetApp 提供的任何适用的默认快照前执行挂钩都会在相应的容器上运行。
2. 任何适用的自定义快照前执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义预快照挂钩，但在创建快照之前执行这些挂钩的顺序既不能保证也不可配置。
3. 执行快照。
4. 任何适用的自定义快照后执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义快照后挂钩，但这些挂钩在快照后的执行顺序既不能保证也不可配置。
5. NetApp 提供的任何适用的默认快照后执行挂钩都会在相应的容器上运行。

> 在生产环境中启用执行钩脚本之前，应始终对其进行测试。您可以使用 "kubectl exec" 命令方便地测试脚本。在生产环境中启用执行挂钩后，测试生成的快照以确保其一致。为此，您可以将应用程序克隆到临时命名空间，还原快照，然后测试应用程序。

查看现有执行挂钩

您可以查看应用程序的现有自定义或 NetApp 提供的默认执行挂钩。

步骤

1. 转到 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。

   您可以在显示的列表中查看所有已启用或已禁用的执行挂钩。您可以查看挂机的状态，源以及运行时间（快照前或快照后）。要查看与执行挂钩相关的事件日志，请转到左侧导航区域中的 * 活动 * 页面。

创建自定义执行挂钩

您可以为应用程序创建自定义执行挂钩。请参见 "执行钩示例" 有关挂机示例。要创建执行挂钩，您需要拥有所有者，管理员或成员权限。

创建用作执行挂钩的自定义 Shell 脚本时，请务必在文件开头指定适当的 shell ，除非您正在运行 Linux 命令或提供可执行文件的完整路径。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 选择 * 添加新挂钩 * 。
4. 在 * 挂机详细信息 * 区域中，根据挂机应运行的时间，选择 * 预 Snapshot * 或 * 后 Snapshot * 。
5. 输入此挂钩的唯一名称。
6. （可选）输入执行期间传递到挂机的任何参数，在输入的每个参数之后按 Enter 键以记录每个参数。
7. 在 * 容器映像 * 区域中，如果此挂钩应针对应用程序中包含的所有容器映像运行，请启用 * 应用于所有容器映像 * 复选框。如果该挂钩只能作用于一个或多个指定的容器映像，请在 * 要匹配的容器映像名称 * 字段中输入容器映像名称。
8. 在 * 脚本 * 区域中，执行以下操作之一：
   - 上传自定义脚本。
     i. 选择 * 上传文件 * 选项。
     ii. 浏览到文件并上传。
     iii. 为脚本指定一个唯一名称。
     iv. （可选）输入其他管理员应了解的有关该脚本的任何注释。
   - 从剪贴板粘贴到自定义脚本中。
     i. 选择 * 从剪贴板粘贴 * 选项。
     ii. 选择文本字段并将脚本文本粘贴到字段中。
     iii. 为脚本指定一个唯一名称。
     iv. （可选）输入其他管理员应了解的有关该脚本的任何注释。
9. 选择 * 添加挂钩 * 。

禁用执行挂钩

如果要暂时阻止执行挂钩在应用程序快照之前或之后运行，可以禁用执行挂钩。要禁用执行挂钩，您需要拥有所有者，管理员或成员权限。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 在 * 操作 * 列中选择要禁用的挂机的选项菜单。
4. 选择 * 禁用 * 。

删除执行挂钩

如果您不再需要执行挂钩，则可以将其完全移除。要删除执行挂钩，您需要拥有所有者，管理员或成员权限。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 在 * 操作 * 列中选择要删除的挂机的选项菜单。
4. 选择 * 删除 * 。

执行钩示例

使用以下示例了解如何构建执行挂钩。您可以将这些挂钩用作模板或测试脚本。

简单的成功示例

这是一个简单的钩子示例，它成功地将消息写入标准输出和标准错误。

```
#!/bin/sh

# success_sample.sh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
```

```
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.sh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

简单成功示例（ **bash** 版本）

这是一个简单的钩子示例，该钩子成功地将消息写入标准输出和标准错误，并写入 bash 。

```
#!/bin/bash

# success_sample.bash
#
# A simple noop success hook script for testing purposes.
#
# args: None


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}
```

```
#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.bash"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

简单成功示例（**zsh** 版本）

这是一个简单的钩子示例，该钩子成功地将消息写入标准输出和标准错误，并写入 Z shell 。

```
#!/bin/zsh

# success_sample.zsh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#


#
# Writes the given message to standard output
```

```
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.zsh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

成功使用参数示例

以下示例演示了如何在挂机中使用 args 。

```
#!/bin/sh

# success_sample_args.sh
#
# A simple success hook script with args for testing purposes.
```

```
#
# args: Up to two optional args that are echoed to stdout

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample_args.sh"


# collect args
arg1=$1
arg2=$2

# output args and arg count to stdout
info "number of args: $#"
info "arg1 ${arg1}"
```

```
info "arg2 ${arg2}"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

快照前 / 快照后挂钩示例

以下示例演示了如何对快照前和快照后挂钩使用同一脚本。

```
#!/bin/sh

# success_sample_pre_post.sh
#
# A simple success hook script example with an arg for testing purposes
# to demonstrate how the same script can be used for both a prehook and
posthook
#
# args: [pre|post]

# unique error codes for every error case
ebase=100
eusage=$((ebase+1))
ebadstage=$((ebase+2))
epre=$((ebase+3))
epost=$((ebase+4))


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}
```

```
#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# Would run prehook steps here
#
prehook() {
    info "Running noop prehook"
    return 0
}

#
# Would run posthook steps here
#
posthook() {
    info "Running noop posthook"
    return 0
}


#
# main
#

# check arg
stage=$1
if [ -z "${stage}" ]; then
    echo "Usage: $0 <pre|post>"
    exit ${eusage}
fi

if [ "${stage}" != "pre" ] && [ "${stage}" != "post" ]; then
    echo "Invalid arg: ${stage}"
    exit ${ebadstage}
fi

# log something to stdout
info "running success_sample_pre_post.sh"
```

```
if [ "${stage}" = "pre" ]; then
    prehook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during prehook"
    fi
fi

if [ "${stage}" = "post" ]; then
    posthook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during posthook"
    fi
fi

exit ${rc}
```

故障示例

以下示例演示了如何处理挂机故障。

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}



#
# Writes the given information message to standard output
#
# $* - The message to write
```

```
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

详细故障示例

以下示例演示了如何处理挂机故障，并提供更详细的日志记录。

```
#!/bin/sh

# failure_sample_verbose.sh
#
# A simple failure hook script with args for testing purposes.
#
# args: [The number of lines to output to stdout]


#
# Writes the given message to standard output
#
```

```
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_verbose.sh"


# output arg value to stdout
linecount=$1
info "line count ${linecount}"

# write out a line to stdout based on line count arg
i=1
while [ "$i" -le ${linecount} ]; do
    info "This is line ${i} from failure_sample_verbose.sh"
    i=$(( i + 1 ))
done

error "exiting with error code 8"
exit 8
```

退出代码示例失败

以下示例显示了一个连接失败并显示退出代码。

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#
```

```
# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

失败后成功示例

以下示例显示了首次运行时发生故障的挂钩，但在第二次运行后仍会成功。

```
#!/bin/sh

# failure_then_success_sample.sh
#
# A hook script that fails on initial run but succeeds on second run for
testing purposes.
#
# Helpful for testing retry logic for post hooks.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}
```

```
#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_success sample.sh"


if [ -e /tmp/hook-test.junk ] ; then
    info "File does exist.  Removing /tmp/hook-test.junk"
    rm /tmp/hook-test.junk
    info "Second run so returning exit code 0"
    exit 0
else
    info "File does not exist.  Creating /tmp/hook-test.junk"
    echo "test" > /tmp/hook-test.junk
    error "Failed first run, returning exit code 5"
    exit 5
fi
```