使用 VMware vRealize Orchestrator 進行開發

vRealize Orchestrator 7.6

vmware®

您可以在 VMware 網站上找到最新的技術文件，網址如下：

https://docs.vmware.com/tw/

如果您對於本文件有任何意見，歡迎寄至：

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

目錄

使用 VMware vRealize Orchestrator 開發

使用 VMware vRealize Orchestrator 開發提供開發自訂 VMware® vRealize Orchestrator 工作流程及動作的資訊與說明。

另外，說明文件包含有關需執行指令碼 Orchestrator 元素的資訊及提供 JavaScript 範例。使用 VMware vRealize Orchestrator 開發也提供有關如何建立資源與套件的說明。

適合對象

此資訊適用於想要建立自訂 Orchestrator 工作流程和動作，以及自訂建置區塊的開發人員。

備註 本指南中所述的程序是基於 vRealize Orchestrator 舊版用戶端的使用者介面。

開發工作流程

1

您在 Orchestrator 用戶端介面中開發工作流程。工作流程的開發涉及使用工作流程編輯器、內建的 Mozilla Rhino JavaScript 指令碼引擎，以及 Orchestrator 和 vCenter Server API。

- 工作流程的重要概念

  工作流程由架構、屬性和參數組成。工作流程架構是工作流程的主要元件，因為它定義了所有工作流程元素以及這些元素之間的邏輯連線。工作流程屬性和參數是工作流程用於傳輸資料的變數。每當工作流程執行時，Orchestrator 都會儲存一個工作流程 Token，記錄該特定工作流程執行程序的詳細資料。

- 工作流程開發程序的階段

  開發工作流程的程序涉及一連串的階段。端視您開發的工作流程類型而定，您可以遵循不同序列的階段，也可以略過階段。例如，您不需要自訂指令碼即可建立工作流程。

- 開發工作流程的最佳作法

  VMware 對於由多位使用者以及在叢集化環境中開發 Orchestrator 工作流程的情況，提供多種最佳作法的建議。

- Orchestrator 用戶端的存取權限

  依預設，只有 Orchestrator 管理員 LDAP 群組的成員能存取 Orchestrator 用戶端。

- 在開發期間測試工作流程

  即使您尚未完成作流程或包含結束元素，您也可以在開發程序期間的任何時間點測試工作流程。

- 建立與編輯工作流程

  您會在 Orchestrator 用戶端中建立工作流程，並以工作流程編輯器編輯。工作流程編輯器是 Orchestrator 用戶端的 IDE，用於開發工作流程。

- 提供一般工作流程資訊

  在工作流程編輯器的一般索引標籤中，您可提供工作流程名稱和說明、定義工作流程行為的屬性和某些層面、設定版本號碼、檢查簽章，並設定使用者權限。

- 定義屬性與參數

  建立工作流程後，您必須定義工作流程的全域屬性、輸入參數和輸出參數。

- 工作流程架構

  工作流程架構是工作流程的圖形化表示，以互連工作流程元素的流程圖來呈現工作流程。工作流程架構會定義工作流程的邏輯流程。

- 開發外掛程式

  Orchestrator 允許透過開放式外掛程式架構整合管理及系統管理解決方案。您使用 Orchestrator 用戶端執行及建立外掛程式工作流程和存取外掛程式 API。

- 工作流程開始時取得使用者的輸入參數

  如果工作流程需要輸入參數，它將開啟對話方塊，以供使用者輸入執行時所需的輸入參數值。在使用者編輯器的**呈現**索引標籤中，整理此對話方塊的內容和配置或呈現方式。

- (選擇性) 工作流程執行時要求使用者互動

  工作流程有時會在執行時需要外部來源的其他輸入參數。這些輸入參數可來自於其他應用程式或工作流程，或者使用者可直接提供。

- 呼叫工作流程內的工作流程

  工作流程會在執行期間呼叫其他工作流程。工作流程會因為需要其他工作流程的結果作為自身執行的輸入參數，而啟動另一個工作流程，或是會啟動一個工作流程而繼續獨立運作。工作流程也會在未來的指定時間啟動工作流程，或同時啟動多個工作流程。

- 在物件選項上執行工作流程

  您可在物件選項上執行工作流程，自動化重複工作。例如：您可建立在虛擬機器資料夾中擷取所有虛擬機器快照的工作流程，或您可建立關閉指定主機上的所有虛擬機器電源的工作流程。

- 開發長時間執行的工作流程

  處於等待狀態的工作流程會耗用系統資源，因為它會不斷輪詢需要進行回應的物件。如果您知道工作流程可能要等待很長的時間才會收到所需的回應，則您可將長時間執行的工作流程元素新增至工作流程。

- 組態元素

  組態元素是一種屬性清單，包含可用來在整個 Orchestrator 伺服器部署間設定常數的屬性。

- 工作流程使用者權限

  Orchestrator 會定義您可套用至群組的權限層級，以便允許或拒絕群組存取工作流程。

- 驗證工作流程

  Orchestrator 提供工作流程驗證工具。驗證工作流程有助於識別工作流程中的錯誤，並檢查資料是否從一個元素正確傳送到另一個元素。

- 偵錯工作流程

  Orchestrator 提供工作流程偵錯工具。您可偵錯工作流程以在任何活動開始時，於工作流程執行期間的編輯模式檢查輸入與輸出參數和屬性、取代參數或屬性數值，以及從上次失敗的活動中恢復工作流程。

- 正在執行工作流程

  Orchestrator 工作流程會按照事件的邏輯流程執行。

- 恢復失敗工作流程執行

  若工作流程失敗，Orchestrator 提供一個從上次失敗活動中恢復工作流程執行的選項。

- 產生工作流程說明文件

  您可隨時以 PDF 格式匯出選擇的工作流程或工作流程資料夾說明文件。

- 使用工作流程版本紀錄

  您可使用版本紀錄還原工作流程為上一個版本。您可還原工作流程狀態為先前或之後的工作流程版本。您也可比較目前工作流程狀態與儲存工作流程版本間的差異。

- 還原刪除的工作流程

  您可還原已從工作流程程式庫中刪除的工作流程。

- 開發簡單範例工作流程

  開發簡單範例工作流程以示範工作流程開發程序中最常見的步驟。

- 開發複雜工作流程

  開發複雜範例工作流程以示範工作流程開發程序中最常見的步驟，以及更進階的案例，例如建立自訂決定與迴圈。

工作流程的重要概念

工作流程由架構、屬性和參數組成。工作流程架構是工作流程的主要元件，因為它定義了所有工作流程元素以及這些元素之間的邏輯連線。工作流程屬性和參數是工作流程用於傳輸資料的變數。每當工作流程執行時，Orchestrator 都會儲存一個工作流程 Token，記錄該特定工作流程執行程序的詳細資料。

工作流程參數

執行工作流程時，會收到輸入參數，並產生輸出參數。

輸入參數

大多數工作流程都需要執行一組特定的輸入參數。輸入參數是工作流程在啟動時處理的引數。使用者、應用程式、其他工作流程或某項動作會將輸入參數傳遞到某個工作流程，讓該工作流程在啟動時進行處理。

例如，如果某個工作流程重設虛擬機器，則該工作流程需要虛擬機器的名稱用做輸入參數。

輸出參數

工作流程的輸出參數代表工作流程執行的結果。輸出參數會隨工作流程或工作流程元素的執行而變更。執行工作流程時，可接收其他工作流程的輸出參數作為輸入參數。

例如，如果某個工作流程建立虛擬機器的快照，則該工作流程的輸出參數是產生的快照。

工作流程屬性

工作流程元素會處理收到的輸入參數資料，並且將產生的資料設定為工作流程屬性或輸出參數。

唯讀工作流程屬性是工作流程的全域常數。可寫入屬性是工作流程的全域變數。

您可以使用屬性在工作流程的元素之間傳輸資料。您可透過下列方式取得屬性：

- 在您建立工作流程時定義屬性
- 將工作流程元素的輸出參數設定為工作流程屬性

- 繼承組態元素的屬性

工作流程架構

工作流程架構是圖形化表示，以互連工作流程元素的流程圖來呈現工作流程。工作流程架構是最重要的工作流程元素，因為它會決定工作流程的邏輯。

工作流程簡報

使用者在執行工作流程時，會在工作流程簡報中提供工作流程的輸入參數數值。當您整理工作流程簡報後，需考慮工作流程輸入參數的型別及數量。

工作流程 Token

工作流程 Token 代表正在執行或已執行的工作流程。

工作流程是定義一般步驟順序及一般必要輸入參數集合的程序抽象描述。當您在執行含有一系列真實輸入參數的工作流程時，您會收到此抽象工作流程的執行個體，並依您提供的指定輸入參數表現行為。此指定的完整或執行中的工作流程執行個體稱之為工作流程 Token。

工作流程 Token 屬性

工作流程 Token 屬性皆屬於含工作流程 Token 執行的指定參數。工作流程 Token 屬性屬於含您執行工作流程 Token 的工作流程全域屬性及指定輸入與輸出參數彙總。

工作流程開發程序的階段

開發工作流程的程序涉及一連串的階段。端視您開發的工作流程類型而定，您可以遵循不同序列的階段，也可以略過階段。例如，您不需要自訂指令碼即可建立工作流程。

一般而言，您可透過下列階段開發工作流程。

1 建立新的工作流程，或透過標準程式庫重複建立既有的工作流程。

2 提供工作流程的一般資訊。

3 定義工作流程的輸入參數。

4 配置和連結工作流程架構來定義工作流程的邏輯流程。

5 將各個架構元素的輸入和輸出參數繫結至工作流程屬性。

6 撰寫可編寫指令碼工作元素或自訂自訂元素的所需指令碼。

7 建立工作流程呈現來定義使用者執行工作流程時看見的輸入參數對話方塊配置。

8 驗證工作流程。

開發工作流程的最佳作法

VMware 對於由多位使用者以及在叢集化環境中開發 Orchestrator 工作流程的情況，提供多種最佳作法的建議。

- 每個開發人員都有一個專用於建立和開發工作流程的測試獨立式 Orchestrator 執行個體。
- 工作流程會以 maven 專案形式儲存在共用原始碼控制系統。
- 若要確保 Orchestrator 生產部署達到最佳效能，建議最好在排程時段匯入工作流程。
- 將工作流程匯入 Orchestrator 叢集時，請使用其本機主機名稱或 IP 位址將 Orchestrator 用戶端連線至其中一個節點，而非使用負載平衡器虛擬伺服器的位址。

  備註 對工作流程所作的任何修改都會在下次執行時生效。

Orchestrator 用戶端的存取權限

依預設，只有 Orchestrator 管理員 LDAP 群組的成員能存取 Orchestrator 用戶端。

Orchestrator 管理員可設定至少 **檢視** 權限，授予 Orchestrator 用戶端存取權給其他使用者群組。

若要允許您存取 Orchestrator 用戶端，管理員必須將您新增至 Orchestrator 管理員 LDAP 群組，或設定 **檢視**、**檢查**、**編輯**、**執行** 或 **管理** 權限給您所屬的群組。

在開發期間測試工作流程

即使您尚未完成作流程或包含結束元素，您也可以在開發程序期間的任何時間點測試工作流程。

依預設，Orchestrator 會先檢查工作流程是否有效，之後您才能執行工作流程。您可以在工作流程開發期間停用自動驗證，以便執行部分工作流程來進行測試。

備註 您完成工作流程開發時，別忘了重新啟動自動驗證。

程序

1 在 Orchestrator 用戶端功能表中，按一下**工具 > 使用者喜好設定**。

2 按一下**工作流程**索引標籤。

3 取消選取**先驗證工作流程再執行**核取方塊。

此時您即已停用自動工作流程驗證。

建立與編輯工作流程

您會在 Orchestrator 用戶端中建立工作流程，並以工作流程編輯器編輯。工作流程編輯器是 Orchestrator 用戶端的 IDE，用於開發工作流程。

您可以透過編輯現有工作流程開啟工作流程編輯器。

- 建立工作流程

  您可以在 Orchestrator 用戶端的工作流程階層清單中建立工作流程。
- 編輯工作流程

  您可以編輯工作流程來變更現有的工作流程，或開發新的空白工作流程。
- 從標準程式庫中編輯工作流程

  Orchestrator 提供工作流程標準程式庫，讓您用於自動化虛擬基礎架構中的作業。標準程式庫中的工作流程皆鎖定在唯讀狀態。
- 工作流程編輯器索引標籤

  工作流程編輯器包含您可編輯之工作流程元件的索引標籤。

建立工作流程

您可以在 Orchestrator 用戶端的工作流程階層清單中建立工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 (選擇性) 在工作流程階層清單的根目錄或是清單中的資料夾上按一下滑鼠右鍵，然後選取**新增資料夾**以建立新的工作流程資料夾。

4 (選擇性) 輸入新資料夾的名稱。

5 在新資料夾或現有資料夾上按一下滑鼠右鍵，然後選取**新增工作流程**。

6 為新工作流程命名，然後按一下**確定**。

在您選擇的資料夾中隨即建立一個新的空白工作流程。

後續步驟

您可編輯該工作流程。

編輯工作流程

您可以編輯工作流程來變更現有的工作流程，或開發新的空白工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 展開工作流程階層清單，可瀏覽至您要編輯的工作流程。

4 若要開啟工作流程以進行編輯，請在工作流程上按一下滑鼠右鍵，然後選取**編輯**。

工作流程編輯器會開啟工作流程供您進行編輯。

從標準程式庫中編輯工作流程

Orchestrator 提供工作流程標準程式庫，讓您用於自動化虛擬基礎架構中的作業。標準程式庫中的工作流程皆鎖定在唯讀狀態。

若要從標準程式庫中編輯工作流程，您必須建立該工作流程的複本。您可編輯複製工作流程或自訂工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 (選擇性) 在工作流程資料夾的根階層按右鍵，然後選擇 **新資料夾** ，建立包含要編輯工作流程的資料夾。

4 展開標準工作流程的**程式庫**階層式清單，瀏覽至要編輯的工作流程。

5 在工作流程上按一下滑鼠右鍵進行編輯。

編輯 選項停用。工作流程是唯讀的。

6 在工作流程上按一下滑鼠右鍵，然後選取**複製工作流程**。

7 為複製的工作流程提供名稱。

依預設，Orchestrator 會以 *workflow_name* 的複本為複製的工作流程命名。

8 按一下 **工作流程資料夾** 數值，搜尋儲存複製工作流程的資料夾。

選擇您在 步驟 3 中建立的資料夾。若您未建立資料夾，請選取不在標準工作流程程式庫中的資料夾。

9 按一下 **是** 或 **否** ，複製工作流程版本紀錄至複本。

選項	說明
Yes	原始工作流程的版本紀錄已在複本中複製。
No	複本的版本還原為 0.0.0。

10 按一下**複製**，複製工作流程。

11 在複製工作流程上按一下滑鼠右鍵，然後選取**編輯**。

工作流程編輯器開啟。您可編輯複製的工作流程。

您已從標準程式庫中複製工作流程。您可編輯複製的工作流程。

工作流程編輯器索引標籤

工作流程編輯器包含您可編輯之工作流程元件的索引標籤。

表 1-1. 工作流程編輯器索引標籤

索引標籤	說明
一般	編輯工作流程名稱、提供工作流程作業的說明、設定版本號碼、查看使用者權限、定義 Orchestrator 伺服器重新啟動時的工作流程行為，以及定義工作流程的全域屬性。
輸入	定義執行工作流程時需要的參數。這些輸入參數是工作流程處理的資料。工作流程的行為會根據這些參數而變更。
輸出	定義工作流程執行完成時產生的值。其他工作流程或動作可在執行時使用這些值。
架構	建置工作流程。從**架構**索引標籤左邊的工作流程選擇區拖曳工作流程架構元素，即可建置工作流程。在**架構**索引標籤的下半部，按一下架構圖中的元素可讓您定義和編輯元素的行為。
呈現	定義使用者執行工作流程時出現的使用者輸入對話方塊配置。您可以安排參數和屬性成為呈現步驟和群組，以便於識別輸入參數對話方塊中的參數。您可設定參數內容來定義限制，限制使用者可在呈現中提供的輸入參數。
參數參照	檢視哪些工作流程元素在工作流程的邏輯流程中耗用屬性和參數。對於您在**呈現**索引標籤上定義的這些參數和屬性，此索引標籤也會顯示限制。
工作流程 Token	檢視各個工作流程的執行詳細資料。這些資訊包含工作流程的狀態、執行工作流程的使用者、目前元素的業務狀態，以及工作流程開始和結束的時間與日期。
事件	檢視執行工作流程時各個個別事件的資訊。此資訊包括事件的說明、觸發器事件的使用者、事件的類型和起因，以及事件發生的時間和日期。
權限	設定與使用者或使用者群組的工作流程互動的權限。

提供一般工作流程資訊

在工作流程編輯器的**一般**索引標籤中，您可提供工作流程名稱和說明、定義工作流程行為的屬性和某些層面、設定版本號碼、檢查簽章，並設定使用者權限。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**一般**索引標籤。

2 按一下**版本**數字可設定工作流程的版本號碼。

版本註解對話方塊隨即開啟。

3 輸入此工作流程版本的註解，並按一下**確定**。

例如，如果您剛才建立工作流程，請輸入**初始建立**。

隨即建立新的工作流程版本。您稍後可以將工作流程的狀態還原為此版本。

4 設定**伺服器重新啟動行為**值，以定義 Orchestrator 伺服器重新啟動時工作流程的行為。

- 保留**恢復工作流程執行**的預設值，使工作流程在伺服器停止時中斷執行的時間點恢復。
- 按一下**恢復工作流程執行**，並選取**不恢復工作流程執行 (設定為 FAILED)**，避免 Orchestrator 伺服器重新啟動時工作流程重新啟動。

如果工作流程取決於執行的環境，請避免工作流程重新啟動。例如，如果工作流程需要特定 vCenter Server，而且您重新設定 Orchestrator 連接至不同的 vCenter Server，您重新啟動 Orchestrator 伺服器後重新啟動工作流程將造成工作流程失敗。

5 在**說明**文字方塊中，輸入工作流程的詳細說明。

6 按一下工作流程編輯器底端的**儲存**。

工作流程編輯器左下角的綠色訊息確認您已儲存變更。

您已設定工作流程行為的層面、設定版本，並定義使用者可對工作流程執行的作業。

後續步驟

您必須定義工作流程屬性和參數。

定義屬性與參數

建立工作流程後，您必須定義工作流程的全域屬性、輸入參數和輸出參數。

工作流程屬性會儲存工作流程內部處理的資料。工作流程輸入參數是由外部來源 (例如使用者或另一個工作流程) 所提供的資料。工作流程輸出參數是工作流程執行完成時所交付的資料。

- 定義工作流程參數

  您可以使用輸入與輸出參數，將資料傳入與傳出工作流程。
- 定義工作流程屬性

  工作流程屬性為工作流程處理的資料。
- 屬性及參數命名限制

  您可使用 OGNL 運算式，在工作流程執行時動態決定輸入參數。在 OGNL 處理期間，Orchestrator OGNL 剖析器使用您無法在工作流程屬性或參數名稱中使用的特定關鍵字。

定義工作流程參數

您可以使用輸入與輸出參數，將資料傳入與傳出工作流程。

您可以在工作流程編輯器中定義工作流程的參數。輸入參數是工作流程執行時所需的初始資料。使用者在執行工作流程時提供輸入參數的值。輸出參數是工作流程執行完成時所傳回的資料。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的適當索引標籤。

- 按一下**輸入**建立輸入參數。
- 按一下**輸出**建立輸出參數。

2 在參數索引標籤內按一下滑鼠右鍵，然後選取**新增參數**。

3 按一下參數名稱以變更。

輸入參數的預設名稱為 `arg_in_X`，輸出參數則為 `arg_out_X`，其中 *X* 是數字。

4 (選擇性) 若要變更參數類型的值，請按一下該值，然後從可用值清單中選取所需的值。

依預設，參數類型的值為「字串」。

5 在**說明**文字方塊中，新增參數的說明。

6 (選擇性) 如果您決定參數應為屬性，而非參數，請在該參數上按一下滑鼠右鍵，然後選取**改為屬性**，將參數變更為屬性。

您已定義工作流程的輸入或輸出參數。

後續步驟

定義工作流程的參數後，請建置工作流程架構。

定義工作流程屬性

工作流程屬性為工作流程處理的資料。

備註 當您建立工作流程架構時，您也可以定義工作流程架構元素中的工作流程屬性。通常當您建立處理屬性的工作流程架構元素時，能夠更容易地定義屬性。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**一般**索引標籤。

一般索引標籤的下半部會出現屬性窗格。

2 在屬性窗格中按一下滑鼠右鍵，並選取**新增屬性**。

新屬性隨即出現在屬性清單，其中的預設類型為「字串」。

3 按一下屬性名稱以變更。

預設名稱為 `attX`，其中 *X* 為數字。

備註 工作流程屬性的名稱不得與任何工作流程的參數名稱相同。

4 按一下屬性類型，從可能值清單中選取新的類型。

預設屬性類型為「字串」。

5 按一下屬性值，以根據屬性類型設定或選取值。

6 在**說明**文字方塊中，新增屬性的說明。

7 如果屬性是常數，而非變數，請按一下屬性名稱左側的核取方塊，讓其值為唯讀。

鎖頭圖示表示唯讀核取方塊的資料行。

8 (選擇性) 如果您決定屬性應為輸入或輸出參數，而非屬性，請在屬性上按一下滑鼠右鍵，然後選取**改為輸入/輸出參數**，將屬性變更為參數。

您已為工作流程定義屬性。

後續步驟

您可以定義工作流程的輸入與輸出參數。

屬性及參數命名限制

您可使用 OGNL 運算式，在工作流程執行時動態決定輸入參數。在 OGNL 處理期間，Orchestrator OGNL 剖析器使用您無法在工作流程屬性或參數名稱中使用的特定關鍵字。

使用保留的 OGNL 關鍵字作為屬性名稱的前置詞不會中斷 OGNL 處理。例如：您可命名參數 `trueParameter`。保留的關鍵字不區分大小寫。

您無法在工作流程屬性及參數名稱中使用下列關鍵字。

表 1-2. 屬性及參數名稱中禁用的關鍵字

禁用關鍵字	禁用關鍵字	禁用關鍵字
abstract	eof	_memberAccess
back_char_esc	esc	native
back_char_literal	exponent	package
boolean	export	private
byte	extends	public
char	false	root
char_literal	final	short
class	flt_literal	static
_classResolver	flt_suff	string_esc
const	ident	string_literal
context	implements	synchronized
debugger	import	this
dec_digits	in	_traceEvaluations
dec_flt	int	true
default	int_literal	_typeConverter
delete	interface	volatil
digit	_keepLastEvaluation	with
double	_lastEvaluation	WithinBackCharLiteral
dynamic_subscript	letter	WithinCharLiteral
enum	long	WithinStringLiteral

工作流程架構

工作流程架構是工作流程的圖形化表示，以互連工作流程元素的流程圖來呈現工作流程。工作流程架構會定義工作流程的邏輯流程。

- 檢視工作流程架構

  在 Orchestrator 用戶端的**架構**索引標籤中，您可檢視工作流程的架構。

- 在工作流程架構中建置工作流程

  工作流程架構包含一系列架構元素。工作流程架構元素是工作流程的建置區塊，可代表決定、指令碼式工作、動作、例外狀況處理常式，甚至是其他工作流程。

- 架構元素

  工作流程編輯器的**架構**索引標籤上的功能表列出工作流程架構元素。您可以使用**架構**索引標籤中的架構元素建置工作流程。

- 架構元素內容

  架構元素有您可以在工作流程選擇區的**架構**索引標籤中定義和編輯的內容。

- 連結和繫結

  元素之間的連結決定工作流程的邏輯流程。繫結會將輸入和輸出參數繫結到工作流程屬性，以便將其他元素的資料填入元素。

- 決定

  工作流程會實作決定功能，以根據布林值 `true` 或 `false` 陳述式來定義動作的不同進程。

- 例外狀況處理

  例外狀況處理能夠偵測到架構元素執行時發生的任何錯誤。例外狀況處理會定義架構元素在錯誤發生時的行為。

- 使用錯誤控點

  您可使用標準錯誤控點定義指定工作流程配置元素中發生錯誤時的行為。您可使用全域錯誤控點定義標準錯誤控點未捕捉到錯誤時的行為。

- Foreach 元素與複合型別

  您可在開發的工作流程中插入 Foreach 元素，執行逐一查看多數參數或屬性的子工作流程。若要改善工作流程的理解程度與可讀性，您可集中多組不同型別的工作流程參數，而這種在單一型別中採邏輯連接的項目稱之為複合型別。

- 新增切換活動至工作流程

  您可新增基本切換活動至工作流程架構，並根據工作流程屬性或參數定義切換情況。

檢視工作流程架構

在 Orchestrator 用戶端的**架構**索引標籤中，您可檢視工作流程的架構。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 瀏覽至工作流程階層清單中的工作流程。

3 按一下 [工作流程]。

  右窗格將出現該工作流程的資訊。

4 選取右窗格中的**架構**索引標籤。

您將看見工作流程的圖形化表示。

在工作流程架構中建置工作流程

工作流程架構包含一系列架構元素。工作流程架構元素是工作流程的建置區塊，可代表決定、指令碼式工作、動作、例外狀況處理常式，甚至是其他工作流程。

您可以在工作流程編輯器中建置工作流程，做法是將架構元素從工作流程編輯器左側的工作流程選擇區拖曳至工作流程架構圖上。

編輯工作流程架構

您可以建立架構元素的順序來定義工作流程的邏輯流程，藉此建置工作流程。

依預設，工作流程架構中的所有元素皆已連結。元素間的連結以箭頭表示。當您將新元素新增至工作流程架構時，您必須將它拖曳至箭頭上，或拖曳至未與下個元素連結的現有工作流程元素上。將工作流程元素新增至架構後，您可以刪除現有的連結，並建立新的連結以定義工作流程的邏輯流程。

您可從現有工作流程的架構中將一個元素或選取的元素複製到您正在編輯的工作流程架構。請參閱複製工作流程配置元素。

工作流程架構必須至少有一個**結束工作流程**元素，但其可以有多個。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1. 按一下工作流程編輯器中的**架構**索引標籤。
2. 將架構元素從左窗格的**一般**功能表拖曳至工作流程架構。
3. 按兩下您拖曳至工作流程架構的元素、輸入適當的名稱，然後按 Enter。

   您必須在工作流程內容中提供元素及其唯一名稱。

   您無法重新命名**等待計時器**、**等待事件**、**結束工作流程**或**擲回例外狀況**元素。
4. (選擇性) 在架構的元素上按一下滑鼠右鍵，然後選取**複製**。
5. (選擇性) 在架構的適當位置上按一下滑鼠右鍵，然後選取**貼上**。

   複製與貼上現有的架構元素，可讓您將類似的元素快速新增至架構。所複製元素的所有設定皆會顯示在貼上的元素中，唯業務狀態除外。相應地調整貼上的元素設定。
6. 將架構元素從**基本**、**記錄**或**網路**功能表拖曳至工作流程架構。

   您可以在**基本**、**記錄**或**網路**功能表中編輯元素的名稱。您無法編輯其指令碼。
7. 將**一般**功能表中的架構元素拖曳至工作流程架構。

   當您將動作或工作流程拖曳至工作流程架構時，會顯示一個對話方塊供您搜尋要插入的動作或工作流程。
8. 在**篩選器**文字方塊中，輸入要在工作流程中插入的工作流程或動作的名稱或部分名稱。

   在對話方塊中會顯示符合搜尋的工作流程或動作。
9. 按兩下工作流程或動作加以選取。

   您已將工作流程或動作插入工作流程架構中。
10. 重複此程序，直到您將所有需要的架構元素新增至工作流程架構中為止。

後續步驟

為您已新增至工作流程架構的元素定義內容，並將這些內容連結及繫結在一起。

複製工作流程配置元素

您可從現有工作流程的配置中複製元素或選擇的元素至編輯工作流程配置。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**配置**索引標籤。

2 在左窗格中，選擇您要複製配置元素的工作流程。

- 按一下 **全部工作流程** 並從工作流程階層式清單中選擇工作流程。
- 在搜尋文字方塊中鍵入工作流程名稱，然後按 Enter。

3 在選取工作流程上按一下滑鼠右鍵，然後選擇**開啟**。

出現顯示工作流程內容的視窗。

4 在工作流程視窗中，按一下 **配置** 索引標籤。

5 選擇一或多個工作流程配置元素，在選項上按一下滑鼠右鍵，然後選擇 **複製**。

6 在您編輯工作流程的 **配置** 索引標籤中，按一下滑鼠右鍵，然後選擇 **貼上**。

您已複製其中一個工作流程的工作流程配置元素。

後續步驟

您必須連結並繫結複製的配置元素至現有工作流程配置。

升階輸入與輸出參數

您可升階子系元素的輸入與輸出參數為父系工作流程。

您可升階已在工作流程編輯器 **一般** 索引標籤中定義的自訂屬性。您只能透過取代輸入參數為相符型別屬性，升階預先定義的屬性。

備註 若您升階預先定義的屬性並指派自訂數值，則會建立複製屬性以避免覆寫該原始屬性數值。複製屬性會保留原始屬性的名稱並在屬性名稱結尾遞增數值。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**配置**索引標籤。

2 新增工作流程或動作元素至工作流程配置。

下列通知會出現在配置窗格的頂端。

您是否要新增活動的參數作為目前工作流程的輸入 / 輸出？

3 在通知上按一下 **設定**。

出現含有可用選項的快顯視窗。

4 為各輸入參數選擇對應型別。

選項	說明
Input	引數對應至輸入工作流程參數。
略過	引數對應至 NULL 數值。
值	引數對應至含有可從數值欄位中設定的含數值屬性。

5 為各輸出參數選擇對應型別。

選項	說明
Output	引數對應輸出工作流程參數。
略過	引數對應至 NULL 數值。
本機變數	引數對應至屬性。

6 按一下 **升階**。

您已將參數升階為父系工作流程。

修改搜尋結果

您使用**搜尋**文字方塊尋找元素，例如工作流程或動作。如果搜尋傳回部分結果，您能夠修改搜尋傳回的結果數。

您使用元素的搜尋時，綠色訊息方塊表示搜尋列出全部的結果。黃色訊息方塊表示搜尋僅列出部分的結果。

程序

1 (選擇性) 如果您在工作流程編輯器中編輯工作流程，按一下**儲存並關閉**即可結束編輯器。

2 從 Orchestrator 用戶端功能表中，選取**工具 > 使用者喜好設定**。

3 按一下**一般**索引標籤。

4 在 **Finder 最大大小**文字方塊中，輸入要搜尋傳回的結果數。

5 按一下 [使用者喜好設定] 對話方塊中的**儲存並關閉**。

您至此已修改搜尋傳回的結果數。

架構元素

工作流程編輯器的**架構**索引標籤上的功能表列出工作流程架構元素。您可以使用**架構**索引標籤中的架構元素建置工作流程。

表 1-3. 架構元素和圖示

架構元素名稱	說明	圖示	工作流程編輯器中的位置
啟動工作流程	工作流程的起點。所有工作流程均包含此元素。一個工作流程只能有一個啟動元素。啟動元素有一個輸出但沒有輸入，而且無法從工作流程架構中移除。		永遠顯示在**架構**索引標籤上
可編寫指令碼工作	自行定義的一般用途工作。使用者必須在此元素中撰寫 JavaScript 函數。		**一般**工作流程選擇區
決定	布林值函數。決定元素將使用一個輸入參數，並傳回 `true` 或 `false`。該元素所做的決定類型取決於輸入參數的類型。決定元素會依照決定元素收到的輸入參數來將工作流程分成不同的方向。若收到的輸入參數與預期的值對應，工作流程將沿著特定路由繼續進行。若輸入內容並非預期的值，工作流程將在替代路徑上繼續進行。		**一般**工作流程選擇區
自訂決定	布林值函數。自訂決定可以使用多個輸入參數，並根據自訂指令碼處理這些輸入參數。傳回 `true` 或 `false`。		**一般**工作流程選擇區
決定活動	布林值函數。決定活動會執行工作流程，並將其輸出參數繫結至 `true` 或 `false` 路徑。		**一般**工作流程選擇區
使用者互動	讓使用者將新的輸入參數傳遞至工作流程。您可以設計使用者互動元素向輸入參數提供要求的方式，並針對使用者可提供的參數設定限制條件。您可以設定權限來決定哪些使用者可提供輸入參數。執行中的工作流程到達使用者互動元素時，會進入被動狀態，並提示使用者進行輸入。您可以設定使用者必須提供輸入的逾時時間。工作流程會根據使用者傳遞過來的資料繼續執行，或在逾時時間結束時傳回例外狀況。工作流程等待使用者回應時，工作流程 Token 將處於 `waiting` 狀態。		**一般**工作流程選擇區
等待計時器	此為長時間執行之工作流程使用的架構元素名稱。執行中的工作流程到達等待計時器元素時，會進入被動狀態。您可以設定工作流程繼續執行的絕對日期。工作流程等待日期時，工作流程 Token 將處於 `waiting-signal` 狀態。		**一般**工作流程選擇區
等待事件	用於長時間執行的工作流程中。執行中的工作流程到達等待事件元素時，會進入被動狀態。您可以定義一個工作流程在繼續執行前，所等待的觸發器事件。工作流程等待該事件時，工作流程 Token 將處於 `waiting-signal` 狀態。		**一般**工作流程選擇區
結束工作流程	工作流程的終點。架構中可以有多個結束元素，來代表工作流程的各種可能結果。結束元素有一個輸入但沒有輸出。工作流程到達結束工作流程元素時，工作流程 Token 會進入 `completed` 狀態。		**一般**工作流程選擇區

表 1-3. 架構元素和圖示 (續)

架構元素名稱	說明	圖示	工作流程編輯器中的位置
擲回例外狀況	建立例外狀況並停止工作流程。此元素可以在工作流程架構中出現多次。例外狀況元素有一個輸入參數，該參數限定為字串類型，而且不能有輸出參數。工作流程到達例外狀況元素時，工作流程 Token 會進入 `failed` 狀態。		**一般**工作流程選擇區
工作流程附註	讓您為工作流程的區段加上註解。您可以延伸附註來說明工作流程的區段。以及變更附註的背景色彩來區分工作流程區域。工作流程附註僅提供視覺化資訊，協助您瞭解架構。		**一般**工作流程選擇區
動作元素	從 Orchestrator 程式庫呼叫動作。工作流程到達動作元素時，會呼叫並執行該動作。		**一般**工作流程選擇區
工作流程元素	同時啟動其他工作流程。工作流程到達其架構中的工作流程元素時，會在其程序進行時執行該工作流程。呼叫的工作流程執行完成後，原始工作流程才會繼續執行。		**一般**工作流程選擇區
Foreach 元素	對陣列中的每個元素執行工作流程。例如，您可以對資料夾中的所有虛擬機器，執行 [重新命名虛擬機器] 工作流程。		**一般**工作流程選擇區
非同步工作流程	以非同步方式啟動工作流程。工作流程到達非同步工作流程元素時，會啟動該工作流程並繼續執行。原始工作流程不會等被呼叫的工作流程完成後才執行。		**一般**工作流程選擇區
排程工作流程	建立一個在特定時間執行的工作後，該工作流程便會繼續執行。		**一般**工作流程選擇區
巢狀工作流程	同時啟動多個工作流程。您可以選擇將本機工作流程與不同 Orchestrator 伺服器中的遠端工作流程設為巢狀。您也可以透過不同的認證執行工作流程。工作流程會等待所有巢狀工作流程執行完成，才會繼續執行。		**一般**工作流程選擇區
控點錯誤	處理特定工作流程元素的錯誤。工作流程可以透過建立例外狀況、呼叫其他工作流程或執行自訂指令碼來處理錯誤。		**一般**工作流程選擇區
預設錯誤處理常式	處理標準錯誤處理常式未偵測到的工作流程錯誤。您可以使用任何架構元素來處理錯誤。		**一般**工作流程選擇區

表 1-3. 架構元素和圖示 (續)

架構元素名稱	說明	圖示	工作流程編輯器中的位置
切換	依照工作流程屬性或參數，切換為替代的工作流程路徑。		一般工作流程選擇區
預先定義的工作	不可編輯的指令碼式元素，可執行工作流程常用的標準工作。預先定義的工作如下： **基本** ■ 睡眠 ■ 變更認證 ■ 等待結束日期 ■ 等待自訂事件 ■ 傳送自訂事件 ■ 增加計數器 ■ 減少計數器 **記錄** ■ 系統記錄 ■ 系統警告 ■ 系統錯誤 ■ 伺服器記錄 ■ 伺服器警告 ■ 伺服器錯誤 ■ 系統+伺服器記錄 ■ 系統+伺服器警告 ■ 系統+伺服器錯誤 **網路** ■ HTTP post ■ HTTP get		基本、記錄和網路工作流程選擇區

架構元素內容

架構元素有您可以在工作流程選擇區的**架構**索引標籤中定義和編輯的內容。

編輯架構元素的全域內容

您在元素的 [資訊] 索引標籤中定義架構元素的全域內容。

必要條件

確認工作流程編輯器的 **配置** 標籤包含元素。

程序

1 按一下工作流程編輯器中的**架構**索引標籤。

2 按一下**編輯**圖示 (✏) 選取要編輯的元素。

隨即顯示一個對話方塊，其中列出元素的內容。

3 按一下**資訊**索引標籤。

4 在**名稱**文字方塊中提供此架構元素的名稱。

此名稱會顯示在工作流程架構圖的架構元素中。

5 從**互動**下拉式功能表中選取說明。

互動內容可讓您選取此元素如何與工作流程之外物件互動的標準說明。此內容僅供提供資訊。

6 (選擇性) 在**業務狀態**文字方塊中提供業務狀態說明。

業務狀態內容是此元素功能的簡短說明。當工作流程正在執行時，工作流程 Token 會在執行時顯示每個元素的「業務狀態」。此功能對於追蹤工作流程狀態很有幫助。

7 (選擇性) 在**說明**文字方塊中，輸入架構元素的說明。

架構元素內容索引標籤

您可以按一下已拖曳至工作流程架構的元素，以便存取架構元素的內容。元素的內容會出現在工作流程編輯器底端的索引標籤中。

不同的架構元素有不同的內容索引標籤。

表 1-4. 各個架構元素的內容索引標籤

架構元素內容索引標籤	說明	套用至架構元素類型
屬性	元素向外部來源 (例如使用者、事件或計時器) 要求的屬性。屬性可以是逾時限制、時間和日期、觸發器，或使用者認證。	■ 使用者互動 ■ 等待事件 ■ 等待計時器
決定	定義決定陳述式。決定元素收到符合或不符合決定陳述式的輸入參數，導致兩種可能的動作。	決定
結束工作流程	停止工作流程，因為工作流程成功完成，或因為遇到錯誤並傳回例外狀況。	■ 結束 ■ 例外狀況
例外狀況	此架構元素在發生例外狀況事件時的行為。	■ 動作 ■ 非同步工作流程 ■ 例外狀況 ■ 巢狀工作流程 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作 ■ 使用者互動 ■ 等待事件 ■ 等待計時器 ■ 工作流程
外部輸入	使用者必須在工作流程執行時的某個時間提供的輸入參數。	使用者互動

表 1-4. 各個架構元素的內容索引標籤 (續)

架構元素內容索引標籤	說明	套用至架構元素類型
輸入	此元素的輸入繫結。輸入繫結會定義架構元素如何從其工作流程中的前一個元素接收輸入。	■ 動作 ■ 非同步工作流程 ■ 自訂決定 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作 ■ 工作流程
資訊	架構元素的一般內容和說明。**資訊**索引標籤會按照架構元素的類型來顯示資訊。	■ 動作 ■ 非同步工作流程 ■ 自訂決定 ■ 決定 ■ 巢狀工作流程 ■ 備註 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作 ■ 使用者互動 ■ 等待事件 ■ 等待計時器 ■ 工作流程
輸出	此元素的輸出繫結。輸出繫結會定義架構元素如何將輸出參數繫結至工作流程屬性，或繫結至工作流程輸出參數。	■ 動作 ■ 非同步工作流程 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作 ■ 工作流程
呈現	定義在執行工作流程時需要使用者輸入的情況下，其所看見之輸入參數對話方塊的配置。	使用者互動
指令碼	顯示定義此架構元素行為的 JavaScript 函數。對於非同步工作流程、排程工作流程和動作元素，此指令碼為唯讀。對於可編寫指令碼工作和自訂決定元素，您可以在此索引標籤中編輯 JavaScript。	■ 動作 ■ 非同步工作流程 ■ 自訂決定 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作
視覺繫結	顯示此架構元素的參數和屬性如何繫結至工作流程之中前後元素的參數和屬性所用的圖形化表示。這是元素之輸入和輸出繫結的另一個表示。	■ 動作 ■ 非同步工作流程 ■ 預先定義的工作 ■ 排程工作流程 ■ 可編寫指令碼工作 ■ 工作流程
工作流程	選取要進行巢狀的工作流程。	巢狀工作流程

連結和繫結

元素之間的連結決定工作流程的邏輯流程。繫結會將輸入和輸出參數繫結到工作流程屬性，以便將其他元素的資料填入元素。

若要瞭解連結和繫結，您必須瞭解工作流程的邏輯流程和工作流程的資料流程兩者的差異。

工作流程的邏輯流程

工作流程的邏輯流程是工作流程執行時，從架構中的一個元素到下一個元素的工作流程進展。您可以連結架構中的元素來定義工作流程的邏輯流程。

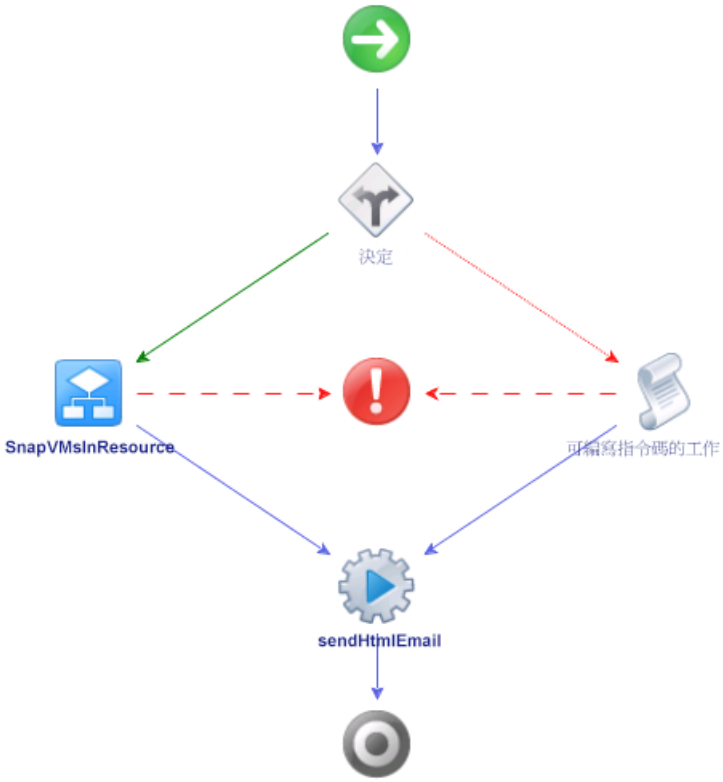
標準路徑是所有元素如預期執行的情況下，工作流程在邏輯流程中進行的路徑。例外狀況路徑是所有元素未如預期執行的情況下，工作流程在邏輯流程中進行的路徑。

工作流程架構中不同樣式的箭頭，表示工作流程能夠在其邏輯流程中進行的不同路徑。

- 藍色箭頭表示工作流程從一個元素進行到下一個元素的標準路徑。
- 綠色箭頭表示布林值決定元素傳回 `true` 的情況下，工作流程進行的路徑。
- 紅色點狀箭頭表示布林值決定元素傳回 `false` 的情況下，工作流程進行的路徑。
- 紅色虛線箭頭表示工作流程元素未正確執行的情況下，工作流程進行的例外狀況路徑。

下圖顯示範例工作流程架構，說明工作流程能夠進行的不同路徑。

圖 1-1. 工作流程的邏輯流程進行的不同工作流程路徑



此範例工作流程能夠在其邏輯流程中進行下列路徑。

- 標準路徑、`true` 決定結果、無例外狀況。
  a 決定元素傳回 `true`。
  b `SnapVMsInResourcePool` 工作流程成功執行。
  c `sendHtmlEmail` 動作成功執行。
  d 工作流程在 `completed` 狀態中成功結束。
- 標準路徑、`false` 決定結果、無例外狀況。
  a 決定元素傳回 `false`。
  b 可編寫指令碼工作元素定義的作業成功執行。
  c `sendHtmlEmail` 動作成功執行。
  d 工作流程在 `completed` 狀態中成功結束。
- `true` 決定結果、例外狀況。
  a 決定元素傳回 `true`。
  b `SnapVMsInResourcePool` 工作流程發生錯誤。
  c 工作流程傳回例外狀況，並在 `failed` 狀態中停止。
- `false` 決定結果、例外狀況。
  a 決定元素傳回 `false`。
  b 可編寫指令碼工作元素定義的作業發生錯誤。
  c 工作流程傳回例外狀況，並在 `failed` 狀態中停止。

元素連結

連結會連接各個架構元素，並定義工作流程從某個元素到下個元素的邏輯流程。

元素通常只會設定一個傳出連結來連結至工作流程中的另一個元素，及一個例外狀況連結來連結至定義其例外狀況行為的元素。傳出連結會定義工作流程的標準路徑。例外狀況連結會定義工作流程的例外狀況路徑。在大多數情況下，單一架構元素會收到來自多個元素的傳入標準路徑連結。

下列元素是前述陳述式的例外狀況。

- 啟動工作流程元素無法接收傳入連結，且沒有例外狀況連結。
- 例外狀況元素可接收多個傳入例外狀況連結，且沒有傳出或例外狀況連結。
- 決定元素有兩個傳出連結，可定義工作流程採取的路徑 (視決定的結果為 `true` 或 `false` 而定)。決定元素沒有例外狀況連結。
- 結束工作流程元素不能有傳出連結，或例外狀況連結。

建立標準路徑連結

標準路徑連結決定工作流程的正常執行。

當您將一個元素連結至另一個元素時，您一律以元素在工作流程中執行的順序連結元素。您一律從先執行的元素開始，以建立兩個元素之間的連結。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 確認工作流程編輯器的 **配置** 標籤包含元素。

程序

1. 將指標放置在您要連接至另一個元素的元素上。

   該元素右側會顯示藍色與紅色箭頭。

2. 將指標放置在藍色箭頭上。

   藍色箭頭隨即放大。

3. 在藍色箭頭上按一下滑鼠左鍵並按住不放，然後將指標移至目標元素。

   藍色箭頭會出現在兩個元素之間，在目標元素四周會出現綠色矩形。

4. 放開滑鼠左鍵。

   藍色箭頭會停留在兩個元素之間。

標準路徑現在已與元素連結。

後續步驟

元素已聯結，但您尚未定義資料流程。您必須定義輸入與輸出繫結，將傳入與傳出資料繫結至工作流程屬性。

工作流程的資料流程

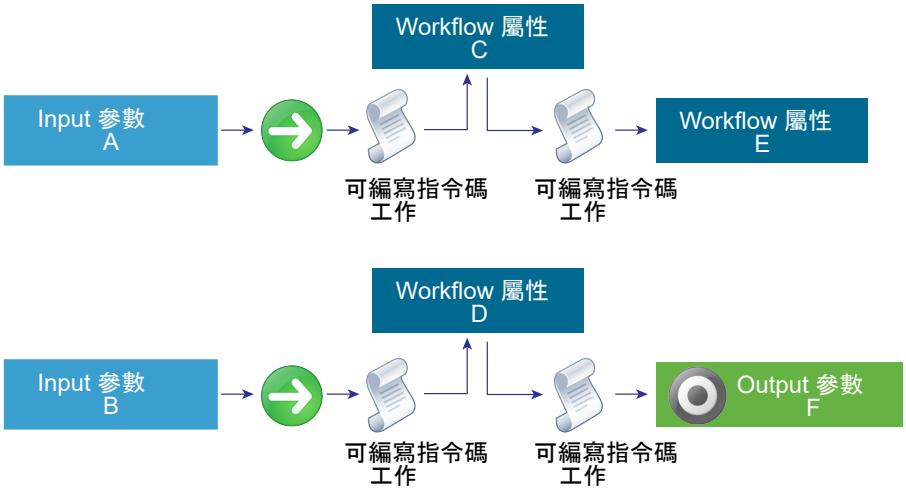
工作流程中的資料流程是當工作流程的每個元素執行時，工作流程元素輸入與輸出參數繫結至工作流程屬性的方式。您可以使用架構元素繫結定義工作流程的資料流程。

執行工作流程架構中的元素時，需要輸入參數形式的資料。該元素會取得其輸入參數的資料，方法是繫結至您在建立工作流程時設定的工作流程屬性，或繫結至工作流程中的前述元素在執行時設定的屬性。

元素會處理該資料，有可能予以轉換，並以輸出參數形式產生其執行的結果。元素會將其產生的輸出參數繫結至其所建立的新工作流程屬性。架構中的其他元素會繫結至這些新工作流程屬性，以作為其輸入參數。工作流程會在執行結束時產生屬性，作為其輸出參數。

下圖顯示非常簡單的工作流程。藍色箭頭代表工作流程的元素連結與邏輯流程。紅線顯示工作流程的資料流程。

圖 1-2. 工作流程資料流程範例



工作流程期間的資料流程如下所示。

1 工作流程從輸入參數 a 與 b 開始。

2 第一個元素會處理參數 a，並將處理結果繫結至工作流程屬性 c。

3 第一個元素會處理參數 b，並將處理結果繫結至工作流程屬性 d。

4 第二個元素會將工作流程屬性 c 作為輸入參數，加以處理，然後將產生的輸出參數繫結至工作流程屬性 e。

5 第二個元素會將工作流程屬性 d 作為輸入參數，加以處理，並產生輸出參數 f。

6 工作流程結束，並產生工作流程屬性 f 作為其輸出參數，亦即其執行的結果。

元素繫結

您必須將所有工作流程元素輸入與輸出參數，繫結至工作流程屬性。繫結會設定元素中的資料，並定義元素的輸出與例外狀況行為。連結會定義工作流程的邏輯流程，而繫結則會定義資料流程。

必須設定元素繫結，才能在元素中設定資料、在處理後從元素產生輸出參數，以及處理元素執行時可能發生的任何錯誤。

IN 繫結 設定架構元素的傳入資料。您需要將元素的本機輸入參數繫結至來源工作流程屬性。**輸入**索引標籤會在 [本機參數] 資料行中列出元素的輸入參數。**輸入**索引標籤會在 [來源參數] 資料行中列出本機參數繫結的工作流程屬性。此索引標籤還會顯示參數類型與參數說明。

OUT 繫結 變更工作流程屬性，並在元素執行完成時產生輸出參數。**輸出**索引標籤會在 [本機參數] 資料行中列出元素的輸出參數。**輸出**索引標籤會在 [來源參數] 資料行中列出本機參數繫結的工作流程屬性。此索引標籤還會顯示參數類型與參數說明。

例外狀況繫結 若元素執行時碰到例外狀況，會連結至例外狀況處理常式。

IN 繫結會從繫結的來源參數讀取值。OUT 繫結會將值寫入繫結的來源參數。

您必須使用 IN 繫結將您在架構元素中使用的每個屬性，或輸入參數繫結至工作流程屬性。若元素變更其執行時收到的輸入參數值，您必須使用 OUT 繫結將這些值繫結至工作流程屬性。將元素的輸出參數繫結至工作流程元素，可讓工作流程架構中，該元素之後的其他元素使用這些輸出參數做為其輸入參數。

建立工作流程時常會發生一種錯誤，就是未繫結輸出參數值，來反映該元素對工作流程屬性所做的變更。

重要 若新增的元素需要您已在工作流程中定義之類型的輸入與輸出參數時，Orchestrator 會將繫結設為連至這些參數。您必須確認 Orchestrator 繫結的參數是正確的，以防工作流程為該元素可繫結的同一類型定義不同的參數。

定義元素繫結

在連結元素來建立工作流程的邏輯流程後，即可定義元素繫結以定義每個元素如何處理所接收及產生的資料。

必要條件

確認在工作流程編輯器的**架構**索引標籤中有工作流程架構，且您已在元素之間建立連結。

程序

1 按一下要設定繫結之元素的**編輯**圖示 (✏)。

隨即顯示一個對話方塊，其中列出元素的內容。

2 按一下**輸入**索引標籤。

輸入索引標籤的內容取決於您所選元素的類型。

- 如果您選取預先定義的工作、工作流程或動作元素，則**輸入**索引標籤會列出該類型元素的可能本機輸入參數，但繫結沒有設定。
- 如果您選取另一種類型的元素，您可以在**輸入**索引標籤上按一下滑鼠右鍵，並選取**繫結至工作流程參數/屬性**，然後從您為工作流程定義的輸入參數與屬性清單中選取。
- 如果所需的屬性尚未存在，您可以在**輸入**索引標籤中按一下滑鼠右鍵，然後選取**繫結至工作流程參數/屬性 > 在工作流程中建立參數/屬性**。

3 如果有適當的參數存在，請選擇要繫結的輸入參數，然後按一下**來源參數**文字方塊中的**未設定**按鈕。

隨即顯示要繫結的可能來源參數和屬性清單。

4 選擇要從建議的清單繫結至本機輸入參數的來源參數。

5 (選擇性) 如果您尚未定義要繫結的來源參數，請按一下參數選取對話方塊中的**在工作流程中建立參數/屬性**連結來建立來源參數。

6 按一下**輸出**索引標籤。

輸出索引標籤的內容取決於您所選元素的類型。

- 如果您選取預先定義的工作、工作流程或動作元素，則**輸出**索引標籤會列出該類型元素的可能本機輸出參數，但繫結沒有設定。

- 如果您選取另一種類型的元素，您可以在**輸出**索引標籤上按一下滑鼠右鍵，並選取**繫結至工作流程參數/屬性**，然後從您為工作流程定義的輸出參數與屬性清單中選取。
- 如果所需的屬性不存在，您可以在**輸入**索引標籤中按一下滑鼠右鍵，然後選取**繫結至工作流程參數/屬性 > 在工作流程中建立參數/屬性**。

7 選擇要繫結的參數。

8 按一下**來源參數 > 未設定**按鈕。

9 選擇要繫結至輸入參數的來源參數。

10 (選擇性) 如果您未定義要繫結的參數，請按一下參數選取對話方塊中的**在工作流程中建立參數/屬性**按鈕來建立參數。

您已定義元素接收的輸入參數及其產生的輸出參數，並已將這些參數繫結至工作流程屬性與參數。

後續步驟

您可以透過定義決定，藉此在工作流程的路徑中建立分叉。

決定

工作流程會實作決定功能，以根據布林值 `true` 或 `false` 陳述式來定義動作的不同進程。

決定是工作流程中的分叉。系統會根據您、其他的工作流程、應用程式，或工作流程正在其中執行之環境的相關資訊，做出工作流程決定。決定元素收到的輸入參數值可決定工作流程會採用分叉的哪一個分支。例如，工作流程決定可能會收到指定虛擬機器的電源狀態作為其輸入值。如果虛擬機器的電源開啟，工作流程會在整個邏輯流程中採用特定的路徑。如果虛擬機器的電源關閉，工作流程會採用不同的路徑。

決定一律為布林值函數。每個決定的唯一可能結果是 `true` 或 `false`。

自訂決定

自訂決定不同於標準決定，因為您會在指令碼中定義決定陳述式。自訂決定會根據您定義的陳述式傳回 `true` 或 `false`，如下例所示。

```
if (decision_statement){
    return true;
}else{
    return false;
}
```

建立決定元素連結

決定元素與工作流程中的其他元素不同，只有 `true` 或 `false` 輸出參數。決定元素沒有例外狀況連結。

必要條件

確認工作流程編輯器的**架構**索引標籤包含元素，其中包含至少一個未連結至其他元素的決定元素。

程序

1 將滑鼠指標放置在決定元素上，將該元素連結至其他兩個定義工作流程中兩個可能分支的元素。

該元素右側會顯示藍色箭頭與紅色箭頭。

2 將指標放置在藍色箭頭上，按住滑鼠左鍵不放，將指標移至目標元素。

兩個元素之間出現綠色箭頭，目標元素變成綠色。綠色箭頭代表當決定元素收到的輸入參數或屬性符合決定陳述式時，工作流程所採用的 `true` 路徑。

3 放開滑鼠左鍵。

綠色箭頭會停留在兩個元素之間。您已定義當決定元素收到預期的值時工作流程採取的路徑。

4 將指標放置在決定元素上，按住滑鼠左鍵不放，將指標移至目標元素。

兩個元素之間出現點狀的紅色箭頭，目標元素變成綠色。紅色箭頭代表當決定元素收到的輸入參數或屬性不符合決定陳述式時，工作流程所採用的 `false` 路徑。

5 放開滑鼠左鍵。

點狀紅色箭頭會停留在兩個元素之間。您已定義當決定元素收到未預期的輸入時工作流程採取的路徑。

您已定義工作流程根據決定元素收到的輸入參數或屬性，所採用的可能 `true` 或 `false` 路徑。

後續步驟

定義決定陳述式。請參閱使用決定建立工作流程分支。

刪除連結的決定元素

當您刪除工作流程配置中的連結決定元素時，您必須指定要刪除的工作流程路徑。

必要條件

確認工作流程編輯器 **配置** 索引標籤是否包含元素，其中包括至少一個含 `true` 和 `false` 路徑的決定元素。

程序

1 選擇決定元素並按下刪除。

出現含可用選項的對話方塊。

2 選擇要刪除的決定分支。

選項	說明
成功分支	刪除工作流程配置中的決定元素及所有遵循 `true` 決定路徑的元素。
失敗分支	刪除工作流程配置中的決定元素及所有遵循 `false` 決定路徑的元素。
兩種分支	刪除工作流程配置中的決定元素及所有遵循兩種決定路徑的元素。
無	僅刪除工作流程配置中的決定元素與其連結。遵循決定路徑的所有元素保留在工作流程配置中。

3 按一下**確定**。

使用決定建立工作流程分支

決定元素是簡單布林值函數，可用來在工作流程中建立分支。決定元素可決定收到的輸入是否符合您設定的決定陳述式。正如此決定的功能，工作流程會以兩個可能路徑中的一個繼續其進程。

必要條件

先確定您有決定元素，且該元素與工作流程編輯器其架構中的另外兩個元素連結，再定義決定。

程序

1 按一下決定元素的**編輯**圖示 (✏)。

隨即顯示一個對話方塊，其中列出決定元素的內容。

2 按一下對話方塊中的**決定**索引標籤。

3 按一下**未設定 (空值)** 連結以選取此決定的來源輸入參數。

隨即顯示一個對話方塊，其中列出此工作流程中定義的所有屬性與輸入參數。

4 按兩下清單中的輸入參數以選取輸入參數。

5 如果您沒有定義要繫結的來源參數，請按一下參數選取對話方塊中的**在工作流程中建立屬性/參數**連結來建立來源參數。

6 從下拉式功能表中選取決定陳述式。

功能表建議的陳述式為關聯式，會視所選的輸入參數類型而不同。

7 新增您要決定陳述式比對的值。

視您選取的輸入類型與陳述式而定，您可能會在值文字方塊中看見**未設定 (空值)** 連結。按一下此連結會提供給您預先定義的值選項。若未提供，則以字串為例，會顯示文字方塊供您輸入值。

您已為決定元素定義陳述式。當決定元素收到輸入參數時，會將輸入參數的值與陳述式的值進行比較，然後判斷陳述式為 true 或 false。

後續步驟

您必須設定工作流程處理例外狀況的方式。

例外狀況處理

例外狀況處理能夠偵測到架構元素執行時發生的任何錯誤。例外狀況處理會定義架構元素在錯誤發生時的行為。

工作流程中的所有元素 (決定、開始與結束元素除外) 皆包含特定的輸出參數類型，僅供處理例外狀況之用。若元素在執行期間發生錯誤，會將錯誤訊號傳送到例外狀況處理常式。例外狀況處理常式會偵測到錯誤，並根據所收到的錯誤做出反應。若您定義的例外狀況處理常式無法處理特定錯誤，可以將元素的例外狀況輸出參數繫結至例外狀況元素，這會讓狀態為 `failed` 的工作流程結束執行。

例外狀況會做為工作流程元素內的 `try` 與 `catch` 序列。若您不需要處理元素中的指定例外狀況，就不必繫結該元素的例外狀況輸出參數。

例外狀況的輸出參數類型一律為 `errorCode` 物件。

建立例外狀況繫結

元素可設定定義工作流程在該元素中遇到錯誤時之行為的繫結。

必要條件

確認工作流程編輯器的 **配置** 標籤包含元素。

程序

1 將指標放置在您要定義例外狀況繫結的元素上。

元素右側會出現紅色箭頭。

2 將指標放在紅色箭頭上，直到它放大為止，按住滑鼠左鍵不放，將紅色箭頭拖曳至目標元素。

紅色虛線箭頭會連結兩個元素。目標元素會定義如果連結至工作流程的元素碰到錯誤時，工作流程的行為。

3 按一下連結至例外狀況處理元素之元素的**編輯** 圖示 ( )。

4 按一下架構元素內容索引標籤中的**例外狀況**索引標籤。

5 若要設定**輸出例外狀況繫結**值，請按一下**未設定**。

- 選取要從例外狀況屬性繫結對話方塊繫結至例外狀況輸出參數的參數，然後按一下**選取**。
- 按一下**在工作流程中建立參數/屬性**以建立例外狀況輸出參數。

6 按一下定義例外狀況處理行為的目標元素。

7 按一下架構元素內容索引標籤中的**輸入**索引標籤。

8 按一下**繫結至工作流程參數/屬性**圖示 ( )。

用於選取輸入參數的對話方塊隨即顯示。

9 選取例外狀況輸出參數，然後按一下**選取**。

10 在架構元素內容索引標籤中，按一下例外狀況處理元素的**輸出**索引標籤。

11 定義例外狀況處理元素的行為。

- 按一下**繫結至工作流程參數/屬性**圖示 ( )，為要產生的例外狀況處理元素選取輸出參數。
- 按一下**指令碼**索引標籤，並使用 JavaScript 定義例外狀況處理元素的行為。

您已定義元素處理例外狀況的方式。

後續步驟

您必須定義當使用者執行工作流程時如何從使用者取得輸入參數。

使用錯誤控點

您可使用標準錯誤控點定義指定工作流程配置元素中發生錯誤時的行為。您可使用全域錯誤控點定義標準錯誤控點未捕捉到錯誤時的行為。

新增錯誤處理常式至工作流程

您可在工作流程執行期間新增錯誤處理常式至工作流程元素，以定義特定工作流程元素錯誤的處理方式。您只能新增錯誤控點至沒有指定錯誤路徑的工作流程元素。

重要　包含**控點錯誤**元素的工作流程不相容於 Orchestrator 5.5.x 或更舊版本。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 拖曳 **控點錯誤** 元素至工作流程配置中的適當元素。

此時將顯示對話方塊。

2 在對話方塊的下拉式功能表，選擇應處理錯誤的方式。

選項	說明
擲回例外狀況	發生錯誤時擲回例外狀況。您可修改例外狀況繫結。
呼叫工作流程	發生錯誤時，會執行選取的工作流程。
自訂指令碼	發生錯誤時，自訂的指令碼會執行。

3 按一下**選取**。

您已新增錯誤處理常式至工作流程。工作流程到達此元素時，會先執行選取的動作再結束執行。

新增全域錯誤控點至工作流程

您可定義在工作流程執行期間，透過新增全域錯誤控點至工作流程配置處理未由標準錯誤控點捕捉錯誤的方式。您可在工作流程配置中，新增一個全域錯誤控點。

重要　包含**預設錯誤控點**元素的工作流程不相容於 Orchestrator 5.5.x 或更舊版本。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 拖曳 **預設錯誤控點** 元素至工作流程配置。

2 (選擇性) 在 **預設錯誤控點** 元素及 **擲回例外狀況** 元素之間新增配置元素，以指定處理全域工作流程錯誤的方式。

您已新增全域錯誤控點至工作流程。當在工作流程中出現未由標準錯誤控點捕捉的錯誤時，全域錯誤控點會先執行指定動作，再結束工作流程執行。

Foreach 元素與複合型別

您可在開發的工作流程中插入 Foreach 元素，執行逐一查看多數參數或屬性的子工作流程。若要改善工作流程的理解程度與可讀性，您可集中多組不同型別的工作流程參數，而這種在單一型別中採邏輯連接的項目稱之為複合型別。

使用 Foreach 元素

Foreach 元素會在大量輸入參數或屬性上重複執行子工作流程。您可選擇執行子工作流程的陣列，並可在執行工作流程時傳遞此陣列元素的數值。執行與陣列中定義元素數量相同次數的子工作流程。

若您的組態元素含有大量屬性，可在 Foreach 元素中逐一執行這些屬性的工作流程。

例如：假設您要重新命名的資料夾中有 10 部虛擬機器。若要重新命名，您必須在工作流程中插入 Foreach 元素並定義重新命名虛擬機器工作流程為元素中的子工作流程。重新命名虛擬機器工作流程接受兩個輸入參數，分別是虛擬機器與其新名稱。您可升階這些參數作為目前工作流程的輸入，最後將成為重新命名虛擬機器工作流程將逐一執行的陣列。當您在執行工作流程時，即可在資料夾指定 10 部虛擬機器及其新名稱。每次工作流程執行時，都會從大量虛擬機器中收到元素，以及從虛擬機器的新名稱中收到元素。

使用複合型別

複合型別為一超過一組輸入參數或屬性以上的群組，可採用邏輯連接但分屬於不同型別。在 Foreach 元素中，您可繫結參數群組為複合數值。透過此方式，Foreach 元素可在每次工作流程執行後一次接受大量參數數值。

例如：假設您即將重新命名一部虛擬機器。您需要虛擬機器物件及其新名稱。若您必須重新命名多部虛擬機器，則您需要兩個陣列，一個用於虛擬機器，一個則用於其名稱。這兩個陣列皆未明確連接。複合型別可讓您擁有一個各元素皆含虛擬機器與其新名稱的陣列。透過此方式，在多組數值的情況下，將明確指定兩組參數之間的連線，且不由工作流程配置暗示。

備註　您無法從 vSphere Web Client 執行含有複合型別的工作流程。

定義 Foreach 元素

若您想要透過在各後續執行傳遞不同參數或屬性的數值執行多次子工作流程，您可在父系工作流程中插入 Foreach 元素。

當您插入 Foreach 元素時，您必須選擇至少一個 Foreach 元素逐一查看的陣列。陣列元素在各後續工作流程執行中可擁有不同的數值。

若子工作流程擁有輸出參數，您應選擇 Foreach 元素用於累計工作流程輸出的輸出參數，以便子工作流程也能逐一查看。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在工作流程編輯器中，選擇 **配置** 索引標籤。

2 在**一般**功能表中拖曳工作流程配置中的 Foreach 元素。

3 從選擇器對話方塊中，選擇工作流程。

下列通知會出現在配置窗格的頂端。

您是否要新增活動的參數作為目前工作流程的輸入 / 輸出？

4 在通知上按一下 **設定**。

出現含有可用選項的快顯視窗。

5 為各輸入參數選擇對應型別。

選項	說明
Input	引數對應至輸入工作流程參數。
略過	引數對應至 NULL 數值。
值	引數對應至含有可從數值欄位中設定的含數值屬性。

6 為各輸出參數選擇對應型別。

選項	說明
Output	引數對應輸出工作流程參數。
略過	引數對應至 NULL 數值。
本機變數	引數對應至屬性。

7 按一下 **升階**。

8 在 Foreach 元素上按一下滑鼠右鍵，然後選擇 **同步化 > 同步化簡報**。

此時將顯示確認對話方塊。

9 按一下 **確定** 傳播 Foreach 元素簡報至目前的工作流程。

對話方塊顯示有關作業結果的資訊。

10 在**輸入**索引標籤上，確認子工作流程的參數皆新增為類型陣列的元素。

11 在**輸出**索引標籤上，確認子工作流程的參數皆新增為類型陣列的元素。

您已在工作流程中定義 Foreach 元素。Foreach 元素執行的工作流程將您已定義的參數或屬性陣列中的各元素當成參數。

如為未定義為陣列的參數或屬性，工作流程會採用各後續執行中的相同數值。

範例： 使用 Foreach 元素重新命名虛擬機器

您可使用 Foreach 元素，一次重新命名多部虛擬機器。您必須在工作流程中插入 Foreach 元素並將 vm 和 newName 參數升階為目前工作流程的輸入。當您執行工作流程時，可透過此方式指定重新命名的虛擬機器，並為虛擬機器建立新名稱。虛擬機器當成您為 vm 參數建立的陣列元素加入。虛擬機器的新名稱加入至您為 newName 參數建立的陣列中。

在 Foreach 元素中定義複合型別

您可以將多個邏輯連線的工作流程參數以稱為複合類型的全新類型組成群組。您可使用 Foreach 元素繫結參數群組為複合數值，以便在單一陣列中連接大量參數。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 確認您的工作流程中有 Foreach 元素。

程序

1 選擇 Foreach 元素的 **IN** 或 **OUT** 索引標籤。

2 選擇您要與其他本機參數分成複合型別群組的本機參數。

3 按一下 **IN** 或 **OUT** 索引標籤上方的 **繫結參數群組為複合數值**。

4 在 [繫結] 窗格中，選擇您要分組為複合類型的參數。

5 選擇 **繫結為迭代器**。

您已設定 Foreach 元素以逐一查看複合型別陣列。

6 按一下 **接受**。

您已定義複合型別並確定工作流程將逐一查看此複合型別的陣列。分組為複合類型的參數名為 *composite_type_name.parameter_name*。例如：若您建立 snapshots 複合型別，分組在型別中的參數可以是 snapshots.vm[in-parameter] 或 snapshots.name[in-parameter]。來自複合型別陣列中的各元素包含您分組在複合型別中的各元素單一執行個體。

範例： 重新命名虛擬機器

假設您想要一次重新命名 10 部虛擬機器。若要重新命名，您需在工作流程中插入 Foreach 元素並選擇在元素中重新命名虛擬機器工作流程。您建立複合類型，以明確連接 vm 和 newName 參數。您將複合型別繫結為迭代器，因此建立包含 vm 與 newName 參數的單一陣列。

新增切換活動至工作流程

您可新增基本切換活動至工作流程架構，並根據工作流程屬性或參數定義切換情況。

每個切換活動都有多個切換情況。每個切換情況都由相關屬性或參數的條件定義。若履行條件，工作流程會切換為您定義的對應工作流程元素。若未履行任何指定條件，工作流程會切換為您定義的預設工作流程元素。

重要　包含**切換**元素的工作流程不相容於 Orchestrator 5.5.x 或更舊版本。

必要條件

確認工作流程編輯器的 **配置** 標籤包含元素。

程序

1 拖曳 **切換** 元素至工作流程配置中的適當元素。

2 按一下**切換**元素的**編輯**圖示 ( )。

3 在 **情況** 索引標籤中，新增或刪除切換情況。

您可變更切換情況的優先順序。

4 定義各切換情況的條件。

5 為各切換情況選取對應的工作流程元素。

6 選擇預設的工作流程元素進行切換。

7 按一下**關閉**。

8 按一下**儲存**。

您已定義切換情況條件與工作流程路徑。

開發外掛程式

Orchestrator 允許透過開放式外掛程式架構整合管理及系統管理解決方案。您使用 Orchestrator 用戶端執行及建立外掛程式工作流程和存取外掛程式 API。

外掛程式概觀

Orchestrator 外掛程式必須包含一組標準的元件，而且必須遵循標準架構。這些做法有助於您建立盡可能最多種外部技術都適用的外掛程式。

- Orchestrator 外掛程式的結構

  Orchestrator 外掛程式的一般結構由執行特定功能的各種分層類型組成。

- 公開外部 API 至 Orchestrator

  您透過建立 Orchestrator 外掛程式從外部產品公開 API 至 Orchestrator 平台。您可建立用於任何技術的外掛程式公開 API，並可對應至 Orchestrator 可使用的 JavaScript 物件。

- 外掛程式的元件

  外掛程式包含一組標準的元件，這些元件向 Orchestrator 平台公開外掛技術中的物件。

- vso.xml 檔案角色

  您使用 `vso.xml` 檔案對應外掛技術物件、級別、方法和屬性至 Orchestrator 詳細目錄物件、指令碼型別、指令碼級別、指令碼方法和屬性。`vso.xml` 檔案也負責定義外掛程式的組態與啟動行為。

- 外掛程式介面卡

  外掛程式介面卡是 Orchestrator 伺服器外掛程式進入點。外掛程式介面卡可作為 Orchestrator 伺服器中的外掛技術資料存放區、建立外掛程式處理站，並管理外掛技術中發生的事件。

- 外掛程式處理站的角色

  外掛程式處理站定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。

- Finder 物件角色

  Finder 物件會識別並找出外掛技術中受管理物件型別的指定執行個體。Orchestrator 可藉由執行 finder 物件上的工作流程，修改及與在外掛技術中找到的物件互動。

- 指令碼物件角色

  指令碼物件為外掛技術中物件的 JavaScript 表示法。外掛程式的指令碼物件出現在 Orchestrator Javascript API 中，您可在工作流程和動作中的執行指令元素上使用。

- 事件控點角色

  事件是指 Orchestrator 在外掛技術中發現的物件狀態變更或屬性變更。Orchestrator 會藉由執行事件控點監控事件。

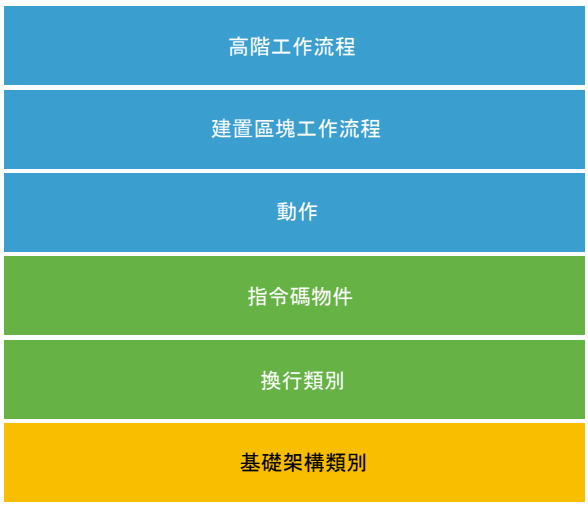
Orchestrator 外掛程式的結構

Orchestrator 外掛程式的一般結構由執行特定功能的各種分層類型組成。

Orchestrator 外掛程式的底部三圖層為基礎架構類別、換行類別及指令碼物件，皆可在外掛技術和 Orchestrator 間執行連線。

Orchestrator 外掛程式的使用者可視部分則為頂部三圖層，分別是動作、建置區塊及高階工作流程。

圖 1-3. Orchestrator 外掛程式的結構



基礎架構類別 在外掛技術與 Orchestrator 之間提供連線的類別組合。基礎架構類別包括依外掛程式定義執行的類別，如外掛工廠、外掛介面卡等。基礎架構類別也包括為一般工作和物件提供的功能類別，如協助程式、快取、詳細目錄等。

換行類別 讓外掛技術物件模型配合您要在 Orchestrator 內部公開物件模型的類別組合。

指令碼物件 在外掛技術中提供換行類別存取權限、方法及屬性的 JavaScript 物件型別。在 `vso.xml` 檔案中，您定義的外掛技術換行類別、屬性及方法，將公開至 Orchestrator。

動作 您可在工作流程及指令碼工作中，直接使用的 JavaScript 函式組合。動作可取得多個輸入參數並擁有單一傳回數值。

建置區塊工作流程 工作流程組合，包括您要搭配外掛程式提供的所有一般功能。一般而言，建置區塊工作流程代表協調技術中的使用者介面作業。建置區塊工作流程可直接使用，或可加入高階工作流程內。

高階工作流程 包括外掛程式指定功能的工作流程組合。您可提供高階工作流程以符合具體需求或顯示外掛程式用法的複雜範例。

公開外部 API 至 Orchestrator

您透過建立 Orchestrator 外掛程式從外部產品公開 API 至 Orchestrator 平台。您可建立用於任何技術的外掛程式公開 API，並可對應至 Orchestrator 可使用的 JavaScript 物件。

外掛程式對應 Java 物件及方法至新增至 Orchestrator 指令碼 API 的 JavaScript 物件。若外部技術公開 Java API，您可直接對應 API 至 Orchestrator 的 JavaScript，以便在工作流程及動作中使用。

您可透過使用 WSDL（網路服務定義語言）、REST（具象狀態傳輸）或傳訊服務，建立適用以非 Java 語言公開 API 的應用程式外掛程式，以整合公開 API 至 Java 物件。接著您可對應整合的 Java 物件至 Orchestrator 的 JavaScript 進行使用。

外掛技術獨立於 Orchestrator 之外。即使您只有二進位代碼的存取權限，您仍可建立用於外部產品的 Orchestrator 外掛程式，例如：在 Java 封存（JAR 檔案），而非原始程式碼中。

外掛程式的元件

外掛程式包含一組標準的元件，這些元件向 Orchestrator 平台公開外掛技術中的物件。

外掛程式的主要元件有外掛程式介面卡、處理站和事件實作。您可以將介面卡、處理站和事件實作中定義的物件和作業，對應至名稱為 `vso.xml` 的 XML 定義檔中出現的 Orchestrator 物件。`Vso.xml` 檔會將外掛技術中的物件和函數，對應至 Orchestrator JavaScript API 中出現的 JavaScript 指令碼物件。`vso.xml` 檔也會將外掛技術的物件類型，對應至 Orchestrator **詳細目錄**索引標籤中出現的 Finder。

外掛程式包含下列元件。

外掛程式模組 外掛程式本身 (由 Java 類別集合所定義)、`vso.xml` 檔，以及與您透過外掛程式存取之物件互動的工作流程和動作套件。外掛程式模組是必要的。

外掛程式介面卡 定義外掛技術和 Orchestrator 伺服器之間的介面。介面卡是 Orchestrator 平台外掛程式的進入點。介面卡可建立外掛程式處理站、管理外掛程式的載入和卸載，並管理針對外掛技術中物件發生的事件。外掛程式介面卡是必要的。

外掛程式處理站 定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。對於 Orchestrator 與外掛技術之間開啟的用戶端工作階段，介面卡會建立處理

站。處理站可讓您在所有用戶端連線之間共用工作階段，或對於個別用戶端連線開啟一個工作階段。外掛程式處理站是必要的。

組態 Orchestrator 不會定義外掛程式儲存其組態的標準方式。您可以透過使用 Windows 登錄、靜態組態檔、將資訊儲存於資料庫，或儲存於 XML 檔的方式儲存組態資訊。在 Orchestrator 用戶端中執行組態工作流程，即可設定 Orchestrator 外掛程式。

Finder 對於 Orchestrator 如何尋找和代表外掛技術中的物件予以定義的互動規則。Finder 會從外掛技術向 Orchestrator 公開的一組物件擷取物件。您可在 `vso.xml` 檔定義物件之間的關係，以便瀏覽物件的網路。Orchestrator 代表**詳細目錄**索引標籤中的外掛技術物件模型。如果您要向 Orchestrator 公開外掛技術中的物件，Finder 是必要的。

指令碼物件 對於外掛技術中的物件、作業及屬性提供存取權限的 JavaScript 物件類型。指令碼物件定義 Orchestrator 如何透過 JavaScript 存取外掛技術的物件模型。您可以將外掛技術的類別和方法對應至 `vso.xml` 檔中的 JavaScript 物件。您可以存取 Orchestrator 指令碼 API 中的 JavaScript 物件，並將這些物件整合於 Orchestrator 指令碼式工作、動作和工作流程。如果您要將指令碼類型、類別和方法新增至 Orchestrator JavaScript API，指令碼物件是必要的。

詳細目錄 Orchestrator 使用 Finder 找到的外掛技術物件執行個體，會出現在 Orchestrator 用戶端的**詳細目錄**視圖中。您可以對詳細目錄中的物件執行工作流程，以便對這些物件執行作業。詳細目錄是選用的。您可以建立外掛程式僅將指令碼類型和類別新增至 Orchestrator JavaScript API，而不公開詳細目錄之中物件的任何執行個體。

事件 外掛技術中的物件發生的狀態變更。Orchestrator 可被動接聽外掛技術中發生的事件。Orchestrator 也可主動觸發外掛技術中的事件。事件是選用的。

vso.xml 檔案角色

您使用 `vso.xml` 檔案對應外掛技術物件、級別、方法和屬性至 Orchestrator 詳細目錄物件、指令碼型別、指令碼級別、指令碼方法和屬性。`vso.xml` 檔案也負責定義外掛程式的組態與啟動行為。

`vso.xml` 檔案執行下列主體角色。

啟動及組態行為 定義外掛程式啟動的方式，並找出外掛程式定義的所有執行組態。載入外掛程式介面卡。

詳細目錄物件 定義外掛技術中，外掛程式存取的物件型別。外掛程式原廠執行的 Finder 方法可找出這些物件的執行個體，並顯示在 Orchestrator 詳細目錄中。

指令碼型別 新增指令碼型別至 Orchestrator JavaScript API，表示詳細目錄中的不同物件型別。您可利用這些指令碼型別作為工作流程中的輸入參數。

指令碼級別 新增級別至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。

指令碼方法	新增方法至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。
指令碼屬性	新增外掛技術中的物件屬性至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。

外掛程式介面卡

外掛程式介面卡是 Orchestrator 伺服器外掛程式進入點。外掛程式介面卡可作為 Orchestrator 伺服器中的外掛技術資料存放區、建立外掛程式處理站，並管理外掛技術中發生的事件。

若要建立外掛程式介面卡，您可以建立實作 `IPluginAdaptor` 介面的 Java 類別。

您建立的外掛程式介面卡類別可管理外掛技術中的外掛程式處理站、事件和觸發器。`IPluginAdaptor` 介面提供您可用來執行這些工作的方法。

外掛程式介面卡執行下列主體角色。

建立處理站	對於從 Orchestrator 到外掛技術的每個連線而言，外掛程式介面卡的最重要角色是載入和卸載一個外掛程式處理站執行個體。外掛程式介面卡類別會呼叫 `IPluginAdaptor.createPluginFactory()` 方法，建立實作 `IPluginFactory` 介面的類別執行個體。
管理事件	外掛程式介面卡是 Orchestrator 伺服器與外掛技術之間的介面。外掛程式介面卡可管理 Orchestrator 對外掛技術中的物件執行和監看的事件。介面卡可透過事件發佈者管理事件。事件發佈者為由呼叫 `IPluginAdaptor.registerEventPublisher()` 方法所建立之介面卡的 `IPluginEventPublisher` 介面執行個體。事件發佈者會對外掛技術中的物件設定觸發器和量表，以便在物件發生特定事件或物件的值通過特定臨界值的情況下，允許 Orchestrator 啟動定義的動作。同樣地，您可以定義 `PluginTrigger` 和 `PluginWatcher` 執行個體來定義長時間執行之工作流程中的等待事件元素等待的事件。
設定外掛程式名稱	您可在 `vso.xml` 檔中提供外掛程式的名稱。外掛程式介面卡可從 `vso.xml` 檔取得此名稱，並在 Orchestrator 用戶端**詳細目錄**視圖中發佈該名稱。
安裝授權	您可以呼叫方法來安裝外掛技術在介面卡實作中需要的任何授權檔案。

如需 `IPluginAdaptor` 介面、其所有方法和外掛程式 API 其他所有類別的完整詳細資料，請參閱 Orchestrator 外掛程式 API 參考。

外掛程式處理站的角色

外掛程式處理站定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。

若要建立外掛程式處理站，您必須從 Orchestrator 外掛程式 API 實作和擴充 `IPluginFactory` 介面。您建立的外掛程式處理站類別會定義 Orchestrator 存取外掛技術中之物件所用的 Finder 函數。處理站允許 Orchestrator 伺服器按照物件的識別碼、物件與其他物件之間的關係或搜尋查詢字串來尋找物件。

外掛程式處理站執行下列主體工作。

尋找物件	您可以建立函數，按照物件名稱和類型來尋找物件。您可以使用 `IPluginFactory.find()` 方法，按照名稱和類型來尋找物件。
尋找與其他物件相關聯的物件	您可以建立函數，按照特定關係類型來尋找與特定物件相關聯的物件。您可以在 `vso.xml` 檔中定義關係。您也可以建立 Finder，按照特定關係類型尋找與所有父系相關聯的相依子物件。您可以實作 `IPluginFactory.findRelation()` 方法，按照特定關係類型尋找與特定父系物件相關聯的任何物件。您可以實作 `IPluginFactory.hasChildrenInRelation()` 方法，探索對於父系執行個體是否至少存在一個子物件。
定義查詢來按照您自己的準則尋找物件	您可以建立物件 Finder 實作您定義的查詢規則。您可以實作 `IPluginFactory.findAll()` 方法，尋找處理站呼叫此方法時與您定義的查詢規則相符合的所有物件。您將取得 `QueryResult` 物件的 `findAll()` 方法結果，其中包含與您定義的查詢規則相符合的所有物件清單。

如需 `IPluginFactory` 介面、其所有方法和外掛程式 API 其他所有類別的詳細資訊，請參閱 Orchestrator 外掛程式 API 參考。

Finder 物件角色

Finder 物件會識別並找出外掛技術中受管理物件型別的指定執行個體。Orchestrator 可藉由執行 finder 物件上的工作流程，修改及與在外掛技術中找到的物件互動。

在外掛技術中指定受管理物件型別的各執行個體必須擁有唯一識別碼，如此 Orchestrator finder 物件才能找到。外掛技術提供用於物件執行個體的唯一識別碼作為字串。工作流程執行時，Orchestrator 會設定物件的唯一識別碼並當成工作流程屬性數值尋找。需要指定類型物件作為輸入參數的工作流程會在該類型物件的特定執行個體上執行。

外掛程式新增至 Orchestrator JavaScript API 的 Finder 物件含有作為前置詞的外掛程式名稱。例如：vCenter Server API 中的 `VirtualMachine` 受管理物件型別會作為 `VC:VirtualMachine` JavaScript 型別出現在 Orchestrator 中。

例如：Orchestrator 藉由執行 finder 物件並使用虛擬機器 `id` 屬性作為其唯一識別碼，以透過 vCenter Server 外掛程式存取指定 `VC:VirtualMachine` 執行個體。您可將此物件執行個體當成屬性數值傳遞至工作流程元素。

Orchestrator 外掛程式對應外掛技術中的物件至 `vso.xml` 檔案中 `<finder>` 元素內的對等 Orchestrator finder 物件。`<finder>`元素可識別外掛技術中的方法或功能，以取得指定物件執行個體的唯一識別碼。`<finder>`元素也會定義物件間的關係，以透過和其他物件的關聯方式尋找物件。

Finder 物件出現在外掛程式下的 Orchestrator **詳細目錄** 索引標籤中。

指令碼物件角色

指令碼物件為外掛技術中物件的 JavaScript 表示法。外掛程式的指令碼物件出現在 Orchestrator Javascript API 中，您可在工作流程和動作中的執行指令元素上使用。

外掛程式中的指令碼物件作為 JavaScript 模組、類型和類別出現在 Orchestrator JavaScript API 中。大多數的 Finder 物件都有指令碼物件表示法。JavaScript 類別可新增方法和屬性至 Orchestrator JavaScript API，以代表外掛技術 API 的物件方法與屬性。外掛技術會提供 Orchestrator 獨立執行的物件、型別、級別、屬性和方法。例如：vCenter Server 外掛程式代表所有來自 vCenter Server API 中作為 Orchestrator JavaScript API 的 JavaScript 物件，並包含所有 vCenter Server API 定義的級別、方法和屬性的 JavaScript 表示法。您可使用在 Orchestrator 執行指令函式中定義的 vCenter Server 指令碼類別與方法和屬性。

例如：vCenter Server API 的 `VirtualMachine` 受管理物件型別可透過 `VC:VirtualMachine` finder 找到並作為 `VcVirtualMachine` JavaScript 級別出現在 Orchestrator JavaScript API 中。Orchestrator JavaScript API 中的 `VcVirtualMachine` JavaScript 級別負責定義所有與 vCenter Server API `VirtualMachine` 受管理物件相同的方法與屬性。

Orchestrator 外掛程式對應外掛技術中的物件、型別、級別、屬性和方法至 `vso.xml` 檔案中 `<scripting-objects>` 元素的對等 Orchestrator JavaScript 物件、型別、級別、屬性與方法。

事件控點角色

事件是指 Orchestrator 在外掛技術中發現的物件狀態變更或屬性變更。Orchestrator 會藉由執行事件控點監控事件。

Orchestrator 外掛程式允許您使用不同方式監控外掛技術中的事件。Orchestrator 外掛程式 API 允許您建立下列型別的事件控點，監控外掛技術中的事件。

接聽程式
被動監控外掛技術中的物件是否變更狀態。外掛技術或外掛程式執行可定義接聽程式監控的事件。接聽程式不會啟動事件，但會在發生事件時通知 Orchestrator。接聽程式會透過輪詢外掛技術或接收外掛技術的通知來偵測事件。發生事件時，正在等待事件的 Orchestrator 原則或工作流程會透過開始 Orchestrator 伺服器中的作業作為反應。接聽程式元件為選用。

原則
監控外掛技術中的特定事件，若發生事件則開始 Orchestrator 伺服器中的作業。原則可監控原則觸發器與原則量表。原則觸發器負責定義外掛技術中發生的事件，會造成執行中的原則開始 Orchestrator 伺服器中的作業，例如工作流程。原則量表負責定義外掛技術中物件屬性的數值範圍，如超過則會造成 Orchestrator 開始作業。原則為選用。

工作流程觸發器
若執行中的工作流程包含等待事件元素，則在到達該元素後將暫停執行並等待外掛技術中的事件發生。工作流程觸發器負責定義外掛技術中，工作流程中等待事件元素等候的事件。您可透過監看程式登錄工作流程觸發器。工作流程觸發器為選用。

監看程式
代表工作流程中等待事件元素監看外掛技術中的工作流程觸發器是否發生特定事件。發生事件時，監看程式會通知正在等待該事件的所有工作流程。監看程式為選用。

外掛程式的內容及結構

Orchestrator 外掛程式必須包含標準元件組合並符合標準檔案結構。如為符合標準檔案結構的外掛程式，必須包括指定資料夾及檔案。

若要建立 Orchestrator 外掛程式，您需定義 Orchestrator 存取及與外掛技術物件互動的方式。此外，您需對應所有物件與外掛技術的功能，以對應 `vso.xml` 檔案中的 Orchestrator 物件和功能。

`vso.xml` 檔案必須包括所有物件型別或 Orchestrator 公開作業的參考。外掛程式在外掛技術中找到的各物件必須擁有您提供的唯一識別碼。您定義在 `vso.xml` 檔案中的 `finder` 元素和物件元素的物件名稱。

外掛程式可當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但在任何情況下，檔案必須以 `.dar` 副檔名重新命名。

備註 您可使用 Orchestrator Control Center 匯入 DAR 檔案至 Orchestrator 伺服器。

- 在 vso.xml 檔案中定義應用程式對應

  您在 `vso.xml` 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。
- vso.xml 外掛程式定義檔案格式

  `vso.xml` 檔案會定義 Orchestrator 伺服器與外掛技術互動的方式。您必須在 `vso.xml` 檔案中包含要對 Orchestrator 公開的每一類型物件或作業的參照。
- 命名外掛程式物件

  對於外掛程式在外掛技術中找到的每個物件，您必須提供唯一識別碼。您定義在 `vso.xml` 檔案中的 `<finder>` 元素和 `<object>` 元素的物件名稱。
- 外掛程式物件命名慣例

  您必須在命名所有外掛程式中的物件時遵守 Java 類別命名慣例。
- 外掛程式的檔案結構

  外掛程式必須符合標準檔案結構，而且必須包括某些特定資料夾及檔案。您可以將外掛程式當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但是檔案必須以 `.dar` 副檔名重新命名。

在 vso.xml 檔案中定義應用程式對應

您在 `vso.xml` 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。

`vso.xml` 檔案會將下列資訊提供給 Orchestrator 伺服器：

- 外掛程式的版本、名稱與說明
- 外掛技術類別的參照及相關外掛程式介面卡的參照
- 當 Orchestrator 伺服器啟動時初始化外掛程式
- 代表外掛技術中物件類型的指令碼類型
- 物件類型之間定義物件在 Orchestrator 詳細目錄中如何顯示的關係
- 將外掛技術中的物件與作業對應至 Orchestrator JavaScript API 中函數與物件類型的指令碼類別
- 用於定義套用至某特定類型的所有物件之常數值清單的列舉
- Orchestrator 在外掛技術中監視的事件

vso.xml 檔案必須符合 Orchestrator 外掛程式的 XML 架構定義。您可以存取 VMware 支援站台的架構定義。

```
http://www.vmware.com/support/orchestrator/plugin-4-1.xsd
```

如需 vso.xml 檔案所有元素的說明，請參閱 vso.xml 外掛程式定義檔案元素。

vso.xml 外掛程式定義檔案格式

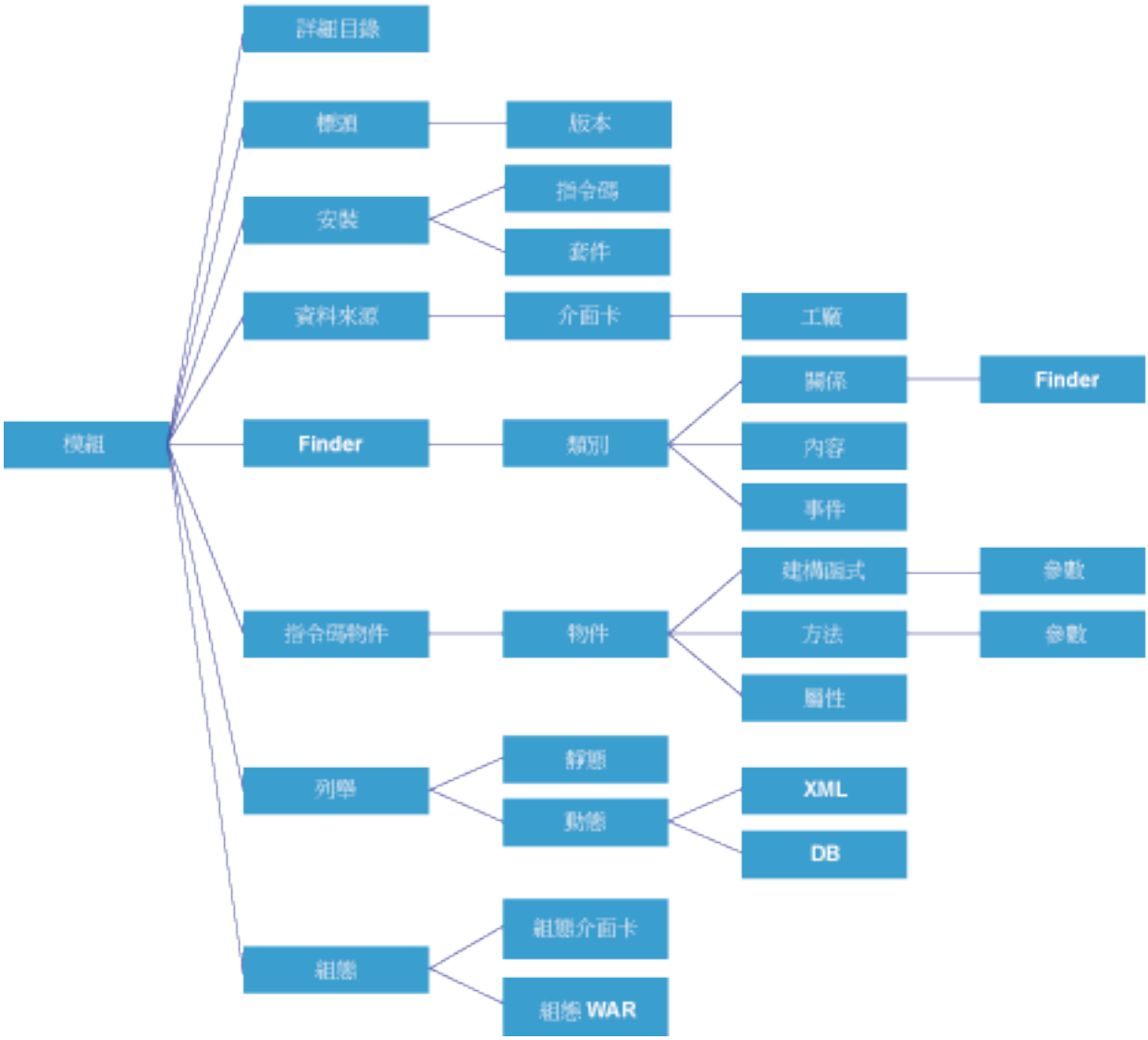
vso.xml 檔案會定義 Orchestrator 伺服器與外掛技術互動的方式。您必須在 vso.xml 檔案中包含要對 Orchestrator 公開的每一類型物件或作業的參照。

您在 vso.xml 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。

vso.xml 檔案為外掛程式的開放式架構與標準化實作的一部分，因此必須採用標準格式。

下圖顯示 vso.xml 外掛程式定義檔案的格式，以及元素如何互相嵌套。

圖 1-4. vso.xml 外掛程式定義檔案格式



命名外掛程式物件

對於外掛程式在外掛技術中找到的每個物件，您必須提供唯一識別碼。您定義在 `vso.xml` 檔案中的 `<finder>` 元素和 `<object>` 元素的物件名稱。

您在外掛技術的處理站執行尋找物件中定義的 Finder 作業。外掛程式尋找物件時，您可以在 Orchestrator 工作流程中使用這些物件，並且在工作流程元素之間傳送這些物件。您對於物件提供的唯一識別碼，能夠讓這些物件在工作流程中的元素之間傳遞。

Orchestrator 伺服器僅儲存所處理的每個物件本身的類型和識別碼，不會儲存 Orchestrator 如何取得物件的相關資訊。您必須在外掛程式實作中以一致的方式命名物件，以便您追蹤從外掛程式取得的物件。

工作流程進行時，如果 Orchestrator 伺服器停止，則您重新啟動伺服器時，工作流程會從伺服器停止時執行的工作流程元素恢復進行。工作流程使用識別碼擷取伺服器停止時元素正在處理的物件。

外掛程式物件命名慣例

您必須在命名所有外掛程式中的物件時遵守 Java 類別命名慣例。

重要 由於工作流程引擎執行資料序列化的方式，請勿在物件名稱中使用下列字串順序。在物件識別碼中使用這些字元順序會造成工作流程引擎間接剖析工作流程，並會在您執行工作流程時導致非預期行為。

- `#;#`
- `#,#`
- `#=#`

在您命名外掛程式中的物件時使用這些方針。

- 在名稱中的各文字使用首字母大寫。
- 請勿使用空格分隔文字。
- 在字母中，請使用標準字元 `A` 至 `Z` 和 `a` 至 `z`。
- 請勿使用特殊字元，如重音符號。
- 請勿使用數字作為名稱的第一個字元。
- 如有可能，請使用少於 10 個字元。

表 1-5. 外掛程式物件命名規定 顯示套用至個別物件型別的規定。

表 1-5. 外掛程式物件命名規定

物件類型	命名規定
外掛程式	■ 在 `vso.xml` 檔案中的 `<module>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 必須唯一。您無法在 Orchestrator 伺服器中執行兩個相同名稱的外掛程式。
Finder 物件	■ 在 `vso.xml` 檔案中的 `<finder>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 在外掛程式中必須為唯一。 Orchestrator 新增外掛程式名稱及冒號至 Orchestrator 指令碼 API finder 物件類型中的 finder 物件名稱。例如：vCenter Server 外掛程式中的 `VirtualMachine` 物件型別以 `VC:VirtualMachine` 出現在 Orchestrator 指令碼 API 中。
指令碼物件	■ 在 `vso.xml` 檔案中的 `<scripting-object>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 在 Orchestrator 伺服器中必須為唯一。 ■ 若要避免混淆指令碼物件與相同名稱的 finder 物件，或與其他外掛程式的指令碼物件混淆，請務必以外掛程式的名稱作為指令碼物件的前置詞，但不要新增分號。例如：vCenter Server 外掛程式中的 `VirtualMachine` 類別以 `VcVirtualMachine` 出現在 Orchestrator 指令碼 API 中。

外掛程式的檔案結構

外掛程式必須符合標準檔案結構，而且必須包括某些特定資料夾及檔案。您可以將外掛程式當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但是檔案必須以 `.dar` 副檔名重新命名。

DAR 封存檔案的內容必須使用下列資料夾結構和命名慣例。

表 1-6. DAR 封存檔案的結構

資料夾	說明
plug-in_name\VSO-INF\	包含 vso.xml 檔，其定義外掛技術中的物件與 Orchestrator 物件之間的對應。 VSO-INF 資料夾和 vso.xml 檔是必要的。
plug-in_name\lib\	包含外掛技術二進位檔的 JAR 檔案。也包含 JAR 檔案，其包括介面卡、處理站、通知處理常式和外掛程式其他介面的實作。 lib 資料夾和 JAR 檔案是必要的。
plug-in_name\resources\	包含外掛程式需要的資源檔案。resources 資料夾可包含下列類型的元素： ■ 映像檔，代表 Orchestrator **詳細目錄**索引標籤中的外掛程式物件。 ■ 指令碼，定義外掛程式啟動時的初始化行為。 ■ Orchestrator 套件，其可包含自訂工作流程、動作，以及與您使用外掛程式存取的物件進行互動的其他資源。 您可以將資源整理到子資料夾中。例如，resources\images\、resources\scripts\ 或 resources\packages\。 resources 資料夾是選用的。

您可使用 Orchestrator Control Center 將 DAR 檔案匯入至 Orchestrator 伺服器。

Orchestrator 外掛程式 API 參考

您開發 IPluginAdaptor 和 IPluginFactory 實作來建立外掛程式時，Orchestrator 外掛程式 API 定義 Java 介面和類別進行實作和擴充。

除非另有說明，否則所有類別均包含在 ch.dunes.vso.sdk.api 套件中。

IAop 介面

IAop 介面提供使用外掛技術取得與設定物件內容的方法。

```
public interface IAop
```

IAop 介面定義下列方法：

方法	傳回	說明
get(java.lang.String propertyName, java.lang.Object object, java.lang.Object sdkObject)	java.lang.Object	從外掛程式指定的物件中取得內容。
set(java.lang.String propertyName, java.lang.String propertyValue, java.lang.Object object)	作廢	設定外掛程式指定的物件之內容。

IDynamicFinder 介面

IDynamicFinder 介面會以程式設計方式傳回 Finder 的識別碼與內容，而不會透過在 vso.xml 檔案中定義識別碼與內容的方式傳回。

IDynamicFinder 介面定義下列方法。

方法	傳回	說明
getIdAccessor(java.lang.String type)	java.lang.String	提供 OGNL 運算式，以透過程式設計方式取得物件識別碼。
getProperties(java.lang.String type)	java.util.List<SDKFinderProperty>	以程式設計方式提供物件內容清單。

IPluginAdaptor 介面

您可以實作 IPluginAdaptor 介面來管理外掛程式處理站、事件和監看程式。IPluginAdaptor 介面會定義外掛程式與 Orchestrator 伺服器之間的介面卡。

IPluginAdaptor 執行個體負責工作階段管理。IPluginAdaptor 介面定義下列方法。

方法	傳回	說明
addWatcher(PluginWatcher watcher)	作廢	新增監視特定事件的監看程式
createPluginFactory(java.lang.String sessionID, java.lang.String username, java.lang.String password, IPluginNotificationHandler notificationHandler)	IPluginFactory	建立 IPluginFactory 執行個體。Orchestrator 伺服器使用處理站，按照物件的識別碼、與其他物件之間的關係等等，從外掛技術取得物件。 工作階段識別碼 可讓您識別執行中的工作階段。例如，使用者可登入兩個不同的 Orchestrator 用戶端，並同時執行兩個工作階段。 同樣地，開始工作流程將建立與開始工作階段的用戶端無關聯的工作流程。即使您關閉 Orchestrator 用戶端，工作流程仍將持續執行。
installLicenses(PluginLicense[] licenses)	作廢	安裝 VMware 提供的標準外掛程式有關的授權資訊。
registerEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)	作廢	在詳細目錄的元素上設定觸發器和量表
removeWatcher(java.lang.String watcherId)	作廢	移除監看程式
setPluginName(java.lang.String pluginName)	作廢	從 vso.xml 檔案取得外掛程式名稱
setPluginPublisher(IPluginPublisher pluginPublisher)	作廢	設定外掛程式的發佈者
uninstallPluginFactory(IPluginFactory plugin)	作廢	解除安裝外掛程式處理站。
unregisterEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)	作廢	從詳細目錄的元素中移除觸發器和量表

IPluginEventPublisher 介面

IPluginEventPublisher 介面會在事件通知匯流排上發佈量表和觸發器，以便 Orchestrator 原則進行監視。

您可以直接在外掛程式介面卡實作中建立 IPluginEventPublisher 執行個體，也可以在個別事件產生器類別中建立這些執行個體。

您可執行 IPluginEventPublisher 介面，將外掛技術事件發佈至 Orchestrator 原則引擎。您可以在外掛技術中建立方法來設定物件的原則觸發器和量表，以及事件接聽程式來接聽這些物件上的事件。

原則可執行量表或觸發器，監控外掛技術中的物件。原則量表負責監控物件屬性，若物件數值超過特定限制，則在 Orchestrator 伺服器中推播事件。若物件上發生定義的事件，原則觸發器會監控物件並在 Orchestrator 伺服器中推播事件。您向 IPluginEventPublisher 執行個體登錄原則量表和觸發器，以便 Orchestrator 原則可以監控它們。

IPluginEventPublisher 介面定義下列方法。

類型	傳回	說明
pushGauge(java.lang.String type, java.lang.String id, java.lang.String gaugeName, java.lang.String deviceName, java.lang.Double gaugeValue)	作廢	發佈量表，以便原則進行監視。使用下列參數： ■ type：要監視的物件所屬的類型。 ■ id：要監視的物件所用的識別碼。 ■ gaugeName：此量表的名稱。 ■ deviceName：該量表監視的屬性所用的類型名稱。 ■ gaugeValue：量表監視物件的值。
pushTrigger(java.lang.String type, java.lang.String id, java.lang.String triggerName, java.util.Properties additionalProperties)	作廢	發佈觸發器，以便原則進行監視。使用下列參數： ■ type：要監視的物件所屬的類型。 ■ id：要監視的物件所用的識別碼。 ■ triggerName：此觸發器的名稱。 ■ additionalProperties：觸發器監視的其他任何內容。

IPluginFactory 介面

IPluginAdaptor 傳回 IPluginFactory 執行個體。IPluginFactory 執行個體在外掛程式應用程式中執行命令，並尋找執行 Orchestrator 作業的物件。

IPluginFactory 介面定義下列欄位：

static final java.lang.String RELATION_CHILDREN

IPluginFactory 介面定義下列方法。

方法	傳回	說明
executePluginCommand(java.lang.String cmd)	作廢	使用外掛程式執行命令。VMware 建議不要使用此方法。
find(java.lang.String type, java.lang.String id)	java.lang.Object	使用外掛程式尋找物件。按識別碼及類型識別物件。
findAll(java.lang.String type, java.lang.String query)	QueryResult	使用外掛程式尋找屬於特定類型且符合查詢字串的物件。您可以在外掛程式的 IPluginFactory 實作中定義查詢的語法。如果不定義查詢語法，findAll() 將傳回指定類型的所有物件。
findRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)	java.util.List	決定物件是否有子系。
hasChildrenInRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)	HasChildrenResult	按特定關係尋找與指定父系相關的所有子系。
invalidate(java.lang.String type, java.lang.String id)	作廢	按類型及識別碼使物件失效。
void invalidateAll()	作廢	使快取中的所有物件失效。

IPluginNotificationHandler 介面

IPluginNotificationHandler 定義方法，通知 Orchestrator 有不同類型的事件在 Orchestrator 透過外掛程式存取的物件上發生。

IPluginNotificationHandler 介面定義下列方法。

方法	傳回	說明
getSessionID()	java.lang.String	傳回目前工作階段識別碼。
notifyElementDeleted(java.lang.String type, java.lang.String id)	作廢	通知系統已刪除指定類型和識別碼的物件。
notifyElementInvalidate(java.lang.String type, java.lang.String id)	作廢	通知系統物件的關係已變更。您可以使用 notifyElementInvalidate() 方法，通知 Orchestrator 物件之間的所有關係變更，而不僅止於讓物件失效的關係變更。例如，將子物件新增至父系代表兩個物件之間的關係變更。
notifyElementUpdated(java.lang.String type, java.lang.String id)	作廢	通知系統已修改物件的屬性。
notifyMessage(ch.dunes.vso.sdk.api.ErrorLevel severity, java.lang.String type, java.lang.String id, java.lang.String message)	作廢	發佈與目前模組相關的錯誤訊息

IPluginPublisher 介面

IPluginPublisher 介面在事件通知匯流排上發佈監看程式事件，以便長時間執行的工作流程等待事件元素進行監視。

工作流程觸發器開始外掛技術事件時，監看觸發器的外掛監看程式及透過 IPluginPublisher 執行個體登錄的監看程式會告知任何等待中的工作流程已發生事件。

IPluginPublisher 介面定義下列方法。

類型	值	說明
pushWatcherEvent(java.lang.String id, java.util.Properties properties)	作廢	在事件通知匯流排上發佈監看程式事件

WebConfigurationAdaptor 介面

WebConfigurationAdaptor 介面增量 IConfigurationAdaptor 與定義方法可在組態索引標籤中找出及安裝用於外掛程式的網路應用程式。

備註　自 Orchestrator 4.1 開始，WebConfigurationAdaptor 介面已過時。若要新增網路應用程式至組態，請執行 IConfigurationAdaptor 並使用 vso.xml 檔案中的 configuration-war 屬性來識別網路應用程式。

WebConfigurationAdaptor 介面定義下列方法。

方法	傳回	說明
getWebAppContext()	字串	找出網路應用程式的 WAR 檔案，取得組態索引標籤。從 DAR 檔案的 /webapps 目錄中提供名稱與路徑至 WAR 檔案作為字串。
setWebConfiguration(boolean webConfiguration)	布林值	判斷組態索引標籤的內容是否由網路應用程式定義。

PluginTrigger 類別

PluginTrigger 類別會建立觸發器模組，取得代表工作流程中的等待事件元素在外掛技術中監視的物件和事件相關的資訊。

PluginTrigger 類別定義方法以取得或設定監控物件型別與名稱、事件本質和逾時時間。

您建立專供工作流程中等待事件元素使用的 PluginTrigger 執行類別。您在定義事件及執行 IPluginEventPublisher.pushTrigger() 方法類別中定義用於 Orchestrator 原則的原則觸發器。

```
public class PluginTrigger
extends java.lang.Object
implements java.io.Serializable
```

PluginTrigger 類別定義下列方法：

方法	傳回	說明
getModuleName()	java.lang.String	取得觸發器模組的名稱。
getProperties()	java.util.Properties	取得觸發器內容的清單。

方法	傳回	說明
getSdkId()	java.lang.String	取得在外掛技術中監視的物件識別碼。
getSdkType()	java.lang.String	取得在外掛技術中監視的物件類型。
getTimeout()	完整	取得觸發器逾時時間。
setModuleName(java.lang.String moduleName)	作廢	設定觸發器模組的名稱。
setProperties(java.util.Properties properties)	作廢	設定觸發器內容的清單。
setSdkId(java.lang.String sdkId)	作廢	設定在外掛技術中監視的物件識別碼。
setSdkType(java.lang.String sdkType)	作廢	設定在外掛技術中監視的物件類型。
setTimeout(long timeout)	作廢	設定逾時時間 (以秒為單位)。負值將停用逾時。

建構函數

- PluginTrigger()
- PluginTrigger(java.lang.String moduleName, long timeout, java.lang.String sdkType, java.lang.String sdkId)

PluginWatcher 類別

PluginWatcher 類別會代表長時間執行的工作流程等待事件元素，監看外掛技術中已定義事件的觸發器模組。

PluginWatcher 類別定義建構函式，讓您可用來建立外掛監看程式執行個體。PluginWatcher 類別定義方法以取得或設定監看工作流程觸發器名稱及逾時時間。

```
public class PluginWatcher
extends java.lang.Object
implements java.io.Serializable
```

PluginWatcher 類別定義下列方法：

方法	傳回	說明
getId()	java.lang.String	取得觸發器的識別碼
getModuleName()	java.lang.String	取得觸發器模組名稱
getTimeoutDate()	完整	取得觸發器逾時日期
getTrigger()	作廢	取得觸發器
setId(java.lang.String id)	作廢	設定觸發器的識別碼
setTimeoutDate()	作廢	設定觸發器逾時日期

建構函數

PluginWatcher(PluginTrigger trigger)

QueryResult 類別

QueryResult 類別包含對於 Orchestrator 透過外掛程式存取之物件取得的 find 查詢結果。

```
public class QueryResult
extends java.lang.Object
implements java.io.Serializable
```

如果找到的結果總數大於查詢傳回的結果數，則 totalCount 值會大於 QueryResult 傳回的元素數。查詢傳回的結果數是在 vso.xml 檔的查詢語法中定義的。

QueryResult 類別定義下列方法：

方法	傳回	說明
addElement(java.lang.Object element)	作廢	將元素新增至 QueryResult
addElements(java.util.List elements)	作廢	將元素的清單新增至 QueryResult
getElements()	java.util.List	取得外掛程式應用程式中的元素
getTotalCount()	完整	取得外掛技術中出現的所有元素計數。
isPartialResult()	布林值	決定取得的結果是否完整
removeElement(java.lang.Object element)	作廢	移除外掛技術中的元素
setElements(java.util.List elements)	作廢	在外掛技術中設定元素
setTotalCount(long totalCount)	作廢	設定外掛技術中出現的元素總數

建構函數

- QueryResult()
- QueryResult(java.util.List ret)
- QueryResult(java.util.List elements, long totalCount)

SDKFinderProperty 類別

SDKFinderProperty 類別可定義方法來取得和設定 Orchestrator Finder 物件在外掛技術中找到的物件內容。IDynanmicFinder.getProperties 方法會傳回 SDKFinderProperty 物件。

```
public class SDKFinderProperty
extends java.lang.Object
```

SDKFinderProperty 類別定義下列方法：

方法	傳回	說明
getAttributeName()	java.lang.String	取得物件屬性名稱
getBeanProperty()	java.lang.String	取得 Java bean 的內容
getDescription()	java.lang.String	取得物件說明
getDisplayName()	java.lang.String	取得物件顯示名稱
getPossibleResultType()	java.lang.String	取得 Finder 傳回的可能結果類型

方法	傳回	說明
getPropertyAccessor()	java.lang.String	取得物件內容存取子
getPropertyAccessorTree()	java.lang.Object	取得物件內容存取子樹狀結構
isHidden()	布林值	顯示或隱藏物件
isShowInColumn()	布林值	顯示或隱藏資料庫資料行中的物件
isShowInDescription()	布林值	顯示或隱藏物件說明
setAttributeName(java.lang.String attributeName)	作廢	設定物件屬性名稱
setBeanProperty(java.lang.String beanProperty)	作廢	設定 Java bean 中的內容
setDescription(java.lang.String description)	作廢	設定物件說明
setDisplayName(java.lang.String displayName)	作廢	設定物件顯示名稱
setHidden(boolean hidden)	作廢	顯示或隱藏物件
setPossibleResultType(java.lang.String possibleResultType)	作廢	設定 Finder 傳回的可能結果類型
setPropertyAccessor(java.lang.String propertyAccessor)	作廢	設定物件內容存取子
setPropertyAccessorTree(java.lang.Object propertyAccessorTree)	作廢	設定物件內容存取子樹狀結構
setShowInColumn(boolean showInTable)	作廢	顯示或隱藏資料庫資料行中的物件
setShowInDescription(boolean showInDescription)	作廢	顯示或隱藏物件說明

建構函數

SDKFinderProperty(java.lang.String attributeName, java.lang.String displayName, java.lang.String beanProperty, java.lang.String propertyAccessor)

PluginExecutionException 類別

如果外掛程式執行作業時發生例外狀況，PluginExecutionException 類別將傳回錯誤訊息。

```
public class PluginExecutionException
extends java.lang.Exception
implements java.io.Serializable
```

PluginExecutionException 類別繼承 class java.lang.Throwable 的下列方法：

fillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace、printStackTrace、printStackTrace、setStackTrace、toStringfillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace

建構函數

PluginExecutionException(java.lang.String message)

PluginOperationException 類別

PluginOperationException 類別會處理外掛程式作業期間發生的錯誤。

```
public class PluginOperationException
extends java.lang.RuntimeException
implements java.io.Serializable
```

PluginOperationException 類別繼承 class java.lang.Throwable 的下列方法：

fillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace、printStackTrace、printStackTrace、setStackTrace、toString

建構函數

PluginOperationException(java.lang.String message)

HasChildrenResult 列舉

HasChildrenResult 列舉宣告指定的父系是否有子系。IPluginFactory.hasChildrenInRelation 方法會傳回 HasChildrenResult 物件。

```
public enum HasChildrenResult
extends java.lang.Enum<HasChildrenResult>
implements java.io.Serializable
```

HasChildrenResult 列舉定義下列常數：

- public static final HasChildrenResult Yes
- public static final HasChildrenResult No
- public static final HasChildrenResult Unknown

HasChildrenResult 列舉定義下列方法：

方法	傳回	說明
`getValue()`	int	傳回下列其中一個值。 **1** 父系有子系 **-1** 父系沒有子系 **0** 參數未知或無效
`valueOf(java.lang.String name)`	`static HasChildrenResult`	傳回此類型的列舉常數及指定的名稱。字串必須完全符合用於宣告此類型列舉常數的識別碼。請勿在列舉名稱中使用空白字元。
`values()`	`static HasChildrenResult[]`	依照常數被宣告的順序，傳回包含此列舉類型之常數的陣列。此方法會逐一查看常數，方法如下： `for (HasChildrenResult c : HasChildrenResult.values()) System.out.println(c);`

`HasChildrenResult` 列舉會從 `class java.lang.Enum` 繼承下列方法：

`clone`、`compareTo`、`equals`、`finalize`、`getDeclaringClass`、`hashCode`、`name`、`ordinal`、`toString`、`valueOf`

ScriptingAttribute 註解類型

`ScriptingAttribute` 註解類型會在外掛技術中加上物件中屬性的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value={METHOD,FIELD})
public @interface ScriptingAttribute
```

`ScriptingAttribute` 註解類型有下列值：

```
public abstract java.lang.String value
```

ScriptingFunction 註解類型

`ScriptingFunction` 註解類型可以加上方法的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value={METHOD,CONSTRUCTOR})
public @interface ScriptingFunction
```

`ScriptingFunction` 註解類型有下列值：

```
public abstract java.lang.String value
```

ScriptingParameter 註解類型

ScriptingParameter 註解類型可以加上參數的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value=PARAMETER)
public @interface ScriptingParameter
```

ScriptingParameter 註解類型有下列值：

```
public abstract java.lang.String value
```

vso.xml 外掛程式定義檔案元素

vso.xml 檔案包含一組標準元素。有些元素為強制，有些則為選用。每個元素都有屬性，可為您要對應至 Orchestrator 物件與作業的物件與作業定義值。

此外，元素可以沒有子系元素，也可以有多個子系元素。子系元素可進一步定義父系元素。同一個子系元素可以出現在多個父系元素中。例如，description 元素沒有子系元素，但顯示為許多父系元素 (module、example、trigger、gauge、finder、constructor、method、object 及 enumeration) 的子系元素。

之後的每個元素都會列出其屬性、父系與子系元素。

module 元素

模組說明提供予 Orchestrator 的一組外掛程式物件。

模組包含如何將外掛技術的資料對應至 Java 類別、版本設定、如何部署模組，以及外掛程式如何在 Orchestrator 詳細目錄中出現的資訊。

<module> 元素為選用。<module> 元素擁有下列屬性：

屬性	值	說明
name	字串	定義外掛程式中所有 <finder> 元素的類型。強制屬性。
version	數字	外掛程式版本號碼，在新版外掛程式中重新載入套件時使用。強制屬性。
build-number	數字	外掛程式組建編號，在新版外掛程式中重新載入套件時使用。強制屬性。
image	映像檔	在 Orchestrator 詳細目錄中顯示的圖示。強制屬性。
display-name	字串	在 Orchestrator 詳細目錄中出現的名稱。選用屬性。
interface-mapping-allowed	true 或 false	VMware 強烈建議不進行介面對應。選用屬性。

表 1-7. 元素記錄

父系元素	子系元素
無	■ <description> ■ <installation> ■ <configuration> ■ <finder-datasources> ■ <inventory> ■ <finders> ■ <scripting-objects> ■ <enumerations>

description 元素

<description> 元素提供 API Explorer 說明文件中所顯示之外掛程式的元素說明。

您新增的文字出現在 <description> 和 </description> 標籤之間的 API Explorer 說明文件。

<description>元素為選用。<description>元素沒有屬性。

表 1-8. 元素記錄

父系元素	子系元素
■ <module> ■ <example> ■ <trigger> ■ <gauge> ■ <finder> ■ <constructor> ■ <method> ■ <object> ■ <enumeration>	無

取代元素

<deprecated> 元素標記在 API Explorer 說明文件中取代的物件及方法。

您新增的文字出現在 <deprecated> 和 </deprecated> 標籤之間的 API Explorer 說明文件。

<deprecated>元素為選用。<deprecated>元素沒有屬性。

表 1-9. 元素記錄

父系元素	子系元素
■ <method> ■ <object>	無

url 元素

<url> 元素提供的 URL 指向物件或列舉的外部說明文件。

您在 <url> 與 </url> 標記之間提供 URL。

<url>元素為選用。<url>元素沒有屬性。

表 1-10. 元素記錄

父系元素	子系元素
■ <enumeration> ■ <object>	無

installation 元素

<installation> 元素可讓您在伺服器啟動時安裝套件或執行指令碼。

<installation>元素為選用。<installation> 元素擁有下列屬性：

屬性	值	說明
mode	always、never 或 version	設定 mode 值會導致 Orchestrator 伺服器啟動時發生下列行為。 ■ 動作 always 會執行 ■ 動作 never 會執行 ■ 動作會在伺服器偵測到新版的外掛程式時執行 強制屬性。

表 1-11. 元素記錄

父系元素	子系元素
<module>	<action>

action 元素

<action> 元素會指定 Orchestrator 伺服器啟動時執行的動作。

<action> 元素屬性會提供 Orchestrator 套件的路徑，或提供定義外掛程式啟動時之行為的指令碼。

<action>元素為選用。外掛程式可擁有無限數量的 <action> 元素。<action>元素擁有下列屬性：

屬性	值	說明
resource	字串	dar 檔案根目錄中 Java 套件或指令碼的路徑強制屬性。
type	install-package 或 execute-script	在 Orchestrator 伺服器中安裝指定的 Orchestrator 套件，或執行指定的指令碼。強制屬性。

表 1-12. 元素記錄

父系元素	子系元素
<installation>	無

Finder-datasources 元素

<finder-datasources> 元素是 <finder-datasource> 元素的 Container。

<finder-datasources>元素為選用。<finder-datasources>元素沒有屬性。

表 1-13. 元素記錄

父系元素	子系元素
<module>	<finder-datasource>

finder-datasource 元素

<finder-datasource> 元素指出您為外掛程式建立 IPluginAdaptor 執行的 Java 類別檔案。

您設定 Orchestrator 如何存取在 <finder-datasource> 元素中的外掛技術物件。<finder-datasource>元素識別您建立外掛程式介面卡的 Java 類別。外掛程式介面卡類別執行個體化您建立的外掛程式原廠。外掛程式原廠定義尋找外掛技術物件的方法。您可在 <finder-datasource> 元素中設定原廠執行 finder 方法呼叫逾時。不同的逾時可套用至 IPluginFactory 介面的不同 finder 方法。

<finder-datasource>元素為選用。外掛程式可擁有無限數量的 <finder-datasources> 元素。<finder-datasource>元素擁有下列屬性：

屬性	值	說明
name	字串	識別 <finder> 元素 datasource 屬性中的資料來源。等價於 XML id。強制屬性。
adaptor-class	Java 類別	指出您定義的 IPluginAdaptor 執行方式，建立外掛程式介面卡，如 com.vmware.plugins.sample.Adaptor。強制屬性。
concurrent-call	true (預設) 或 false	允許多名使用者同時存取介面卡。若外掛程式不支援同時呼叫，您必須設定 concurrent-call 為 false。選用屬性。
invoker-mode	direct (預設) 或 timeout	設定 finder 函式上的逾時。若設為 direct，finder 函式呼叫永不逾時。若設為 timeout，Orchestrator 伺服器套用對應 finder 方法的逾時時間。選用屬性。
anonymous-login-mode	never (預設) 或 always	傳遞或不傳遞使用者的使用者名稱及密碼至外掛程式。選用屬性。
timeout-fetch-relation	數字；預設 30 秒	套用至 findRelation() 的呼叫。選用屬性。
timeout-find-all	數字；預設 60 秒	套用至 findAll() 的呼叫。選用屬性。
timeout-find	數字；預設 60 秒	套用至 find() 的呼叫。選用屬性。
timeout-has-children-in-relation	數字；預設 2 秒	套用至 findChildrenInRelation() 的呼叫。選用屬性。
timeout-execute-plugin-command	數字；預設 30 秒	套用至 executePluginCommand() 的呼叫。選用屬性。

表 1-14. 元素記錄

父系元素	子系元素
<finder-datasources>	無

inventory 元素

對於 Orchestrator 用戶端**詳細目錄**視圖和物件選項對話方塊中出現的外掛程式，<inventory> 元素定義階層清單的根階層。

<inventory> 元素並非代表外掛程式應用程式中的物件，而代表在 Orchestrator 指令碼 API 中做為物件的外掛程式本身。

<inventory> 元素為選用。<inventory>元素擁有下列屬性。

屬性	值	說明
type	Orchestrator 物件類型	代表物件根階層的 <finder> 元素類型。強制屬性。

表 1-15. 元素記錄

父系元素	子系元素
<module>	無

Finders 元素

<finders> 元素是所有 <finder> 元素的 Container。

<finders>元素為選用。<finders>元素沒有屬性。

表 1-16. 元素記錄

父系元素	子系元素
<module>	<finder>

Finder 元素

<finder> 元素在 Orchestrator 用戶端中代表透過外掛程式找到的物件類型。

<finder> 元素可識別能定義物件 Finder 所代表之物件的 Java 類別。<finder> 元素可定義物件在 Orchestrator 用戶端介面中顯示的方式。也會識別 Orchestrator 指令碼 API 所定義用來代表此物件的指令碼物件。

Finder 將做為不同類型外掛技術所使用之各種物件格式的共同介面。

<finder>元素為選用。外掛程式可擁有無限數量的 <finder> 元素。<finder> 元素可定義下列屬性：

屬性	值	說明
type	Orchestrator 物件類型	Finder 代表的物件類型。強制屬性。
datasource	<finder-datasource name> 屬性	使用資料來源 refid 識別可定義物件的 Java 類別。強制屬性。

屬性	值	說明
dynamic-finder	Java 方法	定義您在 IDynamicFinder 執行個體中實作的自訂 Finder 方法，藉此以程式設計的方式傳回 Finder 的識別碼與內容，來取代在 vso.xml 檔案中進行定義這個方法。選用屬性。
hidden	true 或 false (預設)	若為 true，將會在 Orchestrator 用戶端中隱藏 Finder。選用屬性。
image	圖形檔案的路徑	16x16 圖示，在 Orchestrator 用戶端的階層清單中代表 Finder。選用屬性。
java-class	Java 類別的名稱	此 Java 類別可定義 Finder 尋找並對應至指令碼物件的物件。選用屬性。
script-object	<scripting-object type> 屬性	此 Finder 要對應的 <scripting-object> 類型 (若有)。選用屬性。

表 1-17. 元素記錄

父系元素	子系元素
<finders>	■ <id> ■ <description> ■ <properties> ■ <default-sorting> ■ <inventory-children> ■ <relations> ■ <inventory-tabs> ■ <events>

properties 元素

<properties> 元素是 <finder><property> 元素的 Container。

<properties>元素為選用。<properties>元素沒有屬性。

表 1-18. 元素記錄

父系元素	子系元素
<finder>	<property>

property 元素

<property> 元素會將找到的物件本身的內容對應至 Java 內容或方法呼叫。

您實作外掛程式處理站來取得程序的外掛程式處理站實作內容時，可以呼叫 SDKFinderProperty 類別的方法。

您可以在 Orchestrator 用戶端的視圖中顯示或隱藏物件內容。您也可以使用列舉來定義物件內容。

<property>元素為選用。外掛程式可擁有無限數量的 <property> 元素。<property>元素擁有下列屬性：

屬性	值	說明
name	Finder 名稱	FinderResult 用來儲存元素的名稱。強制屬性。
display-name	Finder 名稱	顯示的內容名稱。選用屬性。
bean-property	內容名稱	您可以使用 get 和 set 作業，藉由 bean-property 屬性識別要取得的內容。如果您識別名稱為 MyProperty 的內容，外掛程式將定義 getMyProperty 和 setMyProperty 作業。 您可以設定一個或另一個 bean-property 或 property-accessor，但是不可同時設定兩個。選用屬性。
property-accessor	從物件取得內容值的方法	property-accessor 屬性可讓您定義 OGNL 運算式以驗證物件的內容。 您可以設定一個或另一個 bean-property 或 property-accessor，但是不可同時設定兩個。選用屬性。
show-in-column	true (預設) 或 false	若是 true，此內容將出現於 Orchestrator 用戶端結果資料表。選用屬性。
show-in-description	true (預設) 或 false	若是 true，此內容將出現於物件說明。選用屬性。
hidden	true 或 false (預設)	若是 true，此內容將一律隱藏。選用屬性。
linked-enumeration	列舉名稱	將 Finder 內容連結至列舉。選用屬性。

表 1-19. 元素記錄

父系元素	子系元素
<properties>	子系元素

relations 元素

<relations> 元素是 <finder><relation> 元素的 Container。

<relations>元素為選用。<relations>元素沒有屬性。

表 1-20. 元素記錄

父系元素	子系元素
<finder>	<relation>

relation 元素

<relation> 元素將定義物件與其他物件之間的關係。

您可以在 <relation> 元素中定義關係名稱。

<relation>元素為選用。外掛程式可擁有無限數量的 <relation> 元素。<relation>元素擁有下列屬性：

屬性	值	說明
name	關係名稱。	此關係的名稱。強制屬性。
type	Orchestrator 物件類型	藉由此關係與其他物件相關聯的物件類型。強制屬性。
cardinality	to-one 或 to-many	將物件之間的關係定義為一對一或一對多。選用屬性。

表 1-21. 元素記錄

父系元素	子系元素
<relations>	無

id 元素

<id> 元素會定義一個方法，來取得 Finder 用來識別物件的唯一識別碼。

<id>元素為選用。<id>元素擁有下列屬性：

屬性	值	說明
accessor	方法名稱	accessor 屬性可讓您定義 OGNL 運算式以驗證物件的內容。強制屬性。

表 1-22. 元素記錄

父系元素	子系元素
<finder>	無

inventory-children 元素

<inventory-children> 元素定義顯示 Orchestrator 用戶端**詳細目錄**視圖和物件選項方塊之中物件的清單階層。

<inventory-children>元素為選用。<inventory-children>元素沒有屬性。

表 1-23. 元素記錄

父系元素	子系元素
<finder>	<relation-link>

relation-link 元素

<relation-link> 元素定義**詳細目錄**索引標籤中的父物件與子物件之間的階層。

<relation-link>元素為選用。外掛程式可擁有無限數量的 <relation-link> 元素。<relation-link>元素擁有下列屬性。

類型	值	說明
name	關係名稱。	關係名稱的 refid。強制屬性。

表 1-24. 元素記錄

父系元素	子系元素
<inventory-children>	無

events 元素

<events> 元素是 <trigger> 與 <gauge> 元素的 Container。

<events> 元素中可包含不限數目的觸發器或量表。

<events>元素為選用。<events>元素沒有屬性。

表 1-25. 元素記錄

父系元素	子系元素
<finder>	■ <trigger> ■ <gauge>

觸發器元素

<trigger> 元素宣告您可用於此 finder 的觸發器。您必須執行 IPluginAdaptor 的 registerEventPublisher() 和 unregisterEventPublisher() 方法，才能設定觸發器。

<trigger> 元素為選用。<trigger>元素擁有下列屬性。

類型	值	說明
name	觸發器名稱	此觸發器的名稱。強制屬性。

表 1-26. 元素記錄

父系元素	子系元素
<events>	■ <description> ■ <trigger-properties>

trigger-property 元素

<trigger-properties> 元素是 <trigger-property> 元素的 Container。

<trigger-properties>元素為選用。<trigger-properties>元素沒有屬性。

表 1-27. 元素記錄

父系元素	子系元素
<trigger>	<trigger-property>

trigger-property 元素

<trigger-property> 元素會定義識別觸發器物件的內容。

<trigger-property>元素為選用。外掛程式可擁有無限數量的 <trigger-property> 元素。<trigger-property>元素擁有下列屬性：

類型	值	說明
name	觸發器名稱	觸發器的名稱。選用屬性。
display-name	觸發器名稱	在 Orchestrator 用戶端中顯示的名稱。選用屬性。
type	觸發器類型	定義觸發器的物件類型。強制屬性。

表 1-28. 元素記錄

父系元素	子系元素
<trigger-properties>	無

gauge 元素

<gauge> 元素定義您可用於此 Finder 的量表。您必須實作 IPluginAdaptor 的 registerEventPublisher() 和 unregisterEventPublisher() 方法，才能設定量表。

<gauge> 元素為選用。外掛程式可擁有無限數量的 <gauge> 元素。<gauge>元素擁有下列屬性：

類型	值	說明
name	量表名稱	量表的名稱。強制屬性。
min-value	數字	臨界值下限。選用屬性。
max-value	數字	臨界值上限。選用屬性。
unit	物件類型	定義量表的物件類型。強制屬性。
format	字串	受監控值的格式。選用屬性。

表 1-29. 元素記錄

父系元素	子系元素
<events>	<description>

scripting-objects 元素

<scripting-objects> 元素是 <object> 元素的 Container。

<scripting-objects>元素為選用。<scripting-objects>元素沒有屬性。

表 1-30. 元素記錄

父系元素	子系元素
<module>	<object>

object 元素

<object> 元素會將外掛技術的建構函數、屬性和方法對應至 Orchestrator 指令碼 API 公開的 JavaScript 物件類型。

關於物件命名慣例，請參閱命名外掛程式物件。

<object>元素為選用。外掛程式可擁有無限數量的 <object> 元素。<object>元素擁有下列屬性：

類型	值	說明
script-name	JavaScript 名稱	類別的指令碼名稱。必須是全域唯一。強制屬性。
java-class	Java 類別	此 JavaScript 類別包裝的 Java 類別。強制屬性。
create	true (預設) 或 false	若是 true，您可以建立此類別的新執行個體。選用屬性。
strict	true 或 false (預設)	若要 true，您僅能呼叫您在 vso.xml 檔中註解或宣告的方法。選用屬性。
is-deprecated	true 或 false (預設)	若是 true，物件將對應已過時的 Java 類別。選用屬性。
since-version	字串	Java 類別過時以後的版本。選用屬性。

表 1-31. 元素記錄

父系元素	子系元素
<scripting-objects>	■ <description> ■ <deprecated> ■ <url> ■ <constructors> ■ <attributes> ■ <methods> ■ <singleton>

constructors 元素

<constructors> 元素是 <object><constructor> 元素的 Container。

<constructors>元素為選用。<constructors>元素沒有屬性。

表 1-32. 元素記錄

父系元素	子系元素
<object>	<constructor>

constructor 元素

<constructor> 元素定義建構函數方法。<constructor> 方法會在 API Explorer 中產生說明文件。

<constructor> 元素為選用。外掛程式可擁有無限數量的 <constructor> 元素。<constructor>元素沒有屬性。

表 1-33. 元素記錄

父系元素	子系元素
<constructors>	■ <description> ■ <parameters>

Constructor parameters 元素

<parameters> 元素是 <constructor><parameter> 元素的 Container。

<parameters>元素為選用。<parameters>元素沒有屬性。

表 1-34. 元素記錄

父系元素	子系元素
<constructor>	<parameter>

Constructor parameter 元素

<parameter> 元素定義建構函數的參數。

<parameter>元素為選用。外掛程式可擁有無限數量的 <parameter> 元素。<parameter>元素擁有下列屬性：

類型	值	說明
name	字串	在 API 說明文件中使用的參數名稱。強制屬性。
type	Orchestrator 參數類型	在 API 說明文件中使用的參數類型。強制屬性。
is-optional	true 或 false	如果為 true，則此值會為空值。選用屬性。
since-version	字串	方法版本。選用屬性。

表 1-35. 元素記錄

父系元素	子系元素
<parameters>	無

attributes 元素

<attributes> 元素是 <object><attribute> 元素的 Container。

<attributes>元素為選用。<attributes>元素沒有屬性。

表 1-36. 元素記錄

父系元素	子系元素
<object>	<attribute>

attribute 元素

<attribute> 元素會將 Java 類別的屬性從外掛的技術，對應至 Orchestrator JavaScript 引擎使其可用的 JavaScript 屬性。

<attribute>元素為選用。外掛程式可擁有無限數量的 <attribute> 元素。<attribute>元素擁有下列屬性：

類型	值	說明
java-name	Java 屬性	Java 屬性名稱。強制屬性。
script-name	JavaScript 物件	對應 JavaScript 物件的名稱。強制屬性。
return-type	字串	此屬性傳回的物件類型。顯示在 API Explorer 說明文件中。選用屬性。 **備註** 如果 JavaScript 傳回類型為 Properties，則支援的基礎 Java 實作為 java.util.HashMap 與 java.util.Hashtable。
read-only	true 或 false	如果為 true，您便無法修改此屬性。選用屬性。
is-optional	true 或 false	如果為 true，則此欄位會為空值。選用屬性。
show-in-api	true 或 false	如果為 false，則此屬性不會在 API 說明文件中顯示。選用屬性。
is-deprecated	true 或 false	如果為 true，則物件會對應過時的屬性。選用屬性。
since-version	數字	屬性為過時的版本。選用屬性。

表 1-37. 元素記錄

父系元素	子系元素
<attributes>	無

methods 元素

<methods> 元素是 <object><method> 元素的 Container。

<methods>元素為選用。<methods>元素沒有屬性。

表 1-38. 元素記錄

父系元素	子系元素
<object>	<method>

method 元素

<method> 元素將外掛技術中的 Java 方法對應至 Orchestrator JavaScript 引擎公開的 JavaScript 方法。

<method>元素為選用。外掛程式可擁有無限數量的 <method> 元素。<method>元素擁有下列屬性：

類型	值	說明
java-name	Java 方法	Java 方法簽章的名稱，有引數類型在括號中，例如 getVms(DataStore)。強制屬性。
script-name	JavaScript 方法	對應 JavaScript 方法的名稱。強制屬性。

類型	值	說明
return-type	Java 物件類型	此方法取得的類型。選用屬性。 **備註** 如果 JavaScript 傳回類型為 Properties，則支援的基礎 Java 實作為 java.util.HashMap 與 java.util.Hashtable。
static	true 或 false	若是 true，此方法為靜態。選用屬性。
show-in-api	true 或 false	若是 false，此方法不會出現在 API 說明文件中。選用屬性。
is-deprecated	true 或 false	若是 true，物件將對應已過時的方法。選用屬性。
since-version	數字	已過時方法的版本。選用屬性。

表 1-39. 元素記錄

父系元素	子系元素
<methods>	■ <deprecated> ■ <description> ■ <example> ■ <parameters>

元素範例

<example> 元素可讓您新增代碼範例至 API Explorer 說明文件中顯示的 Javascript 法。

<example>元素為選用。<example>元素沒有屬性。

表 1-40. 元素記錄

父系元素	子系元素
<method>	■ <code> ■ <description>

代碼元素

<code> 元素提供出現在 API Explorer 說明文件中的範例代碼。

您在 <code> 與 </code> 標籤之間提供代碼範例。<code>元素為選用。<code>元素沒有屬性。

表 1-41. 元素記錄

父系元素	子系元素
<example>	無

Method parameters 元素

<parameters> 元素是 <method><parameter> 元素的 Container。

<parameters>元素為選用。<parameters>元素沒有屬性。

表 1-42.

父系元素	子系元素
<method>	<parameter>

Method parameter 元素

<parameter> 元素定義方法的輸入參數。

<parameter>元素為選用。外掛程式可擁有無限數量的 <parameter> 元素。<parameter>元素擁有下列屬性：

類型	值	說明
name	字串	參數名稱。強制屬性。
type	Orchestrator 參數類型	參數類型。強制屬性。
is-optional	true 或 false	如果為 true，則此值會為空值。選用屬性。
since-version	字串	方法版本。選用屬性。

表 1-43. 元素記錄

父系元素	子系元素
<parameters>	無

單一元素

<singleton> 元素會將 JavaScript 指令碼物件當成單一執行個體建立。

單一物件的行為方式與靜態 Java 類別相同。單一物件定義外掛程式使用的一般物件，而非 Orchestrator 採用外掛技術存取的定義物件指定執行個體。例如：您可使用單一物件建立與外掛技術的連線。

<singleton>元素為選用。<singleton>元素擁有下列屬性：

類型	值	說明
script-name	JavaScript 物件	對應 JavaScript 物件的名稱。強制屬性。
datasource	Java 物件	用於此 JavaScript 物件的來源 Java 物件。強制屬性。

表 1-44. 元素記錄

父系元素	子系元素
<object>	無

enumerations 元素

<enumerations> 元素是 <enumeration> 元素的 Container。

<enumerations>元素為選用。<enumerations>元素沒有屬性。

表 1-45. 元素記錄

父系元素	子系元素
<module>	<enumeration>

enumeration 元素

<enumeration> 元素可定義適用於特定類型的所有物件的通用值。

若特定類型的所有物件需要特定屬性，且該屬性的值範圍受到限制，您可以定義不同的值做為列舉項目。例如，若物件的類型需要 color 屬性，且可用色彩只有紅色、藍色、綠色，您便可以定義三個列舉項目來定義這三個色彩值。您可以將項目定義為 enumeration 元素的子系元素。

<enumeration>元素為選用。外掛程式可擁有無限數量的 <enumeration> 元素。<enumeration>元素擁有下列屬性。

類型	值	說明
type	Orchestrator 物件類型	列舉類型。強制屬性。

表 1-46. 元素記錄

父系元素	子系元素
<enumerations>	■ <url> ■ <description> ■ <entries>

entries 元素

<entries> 元素是 <enumeration><entry> 元素的 Container。

<entries>元素為選用。<entries>元素沒有屬性。

表 1-47. 元素記錄

父系元素	子系元素
<enumeration>	<entry>

entry 元素

<entry> 元素提供列舉屬性的值。

<entry>元素為選用。外掛程式可擁有無限數量的 <entry> 元素。<entry>元素擁有下列屬性：

類型	值	說明
id	Text	物件用來將列舉項目設為屬性的識別碼。強制屬性。
name	Text	項目名稱。強制屬性。

表 1-48. 元素記錄

父系元素	子系元素
`<entries>`	無

Orchestrator 外掛程式開發最佳做法

您可深入瞭解外掛程式結構與內容，以及瞭解避免特定問題的方式，片面改善您開發的 Orchestrator 外掛程式。

- 建置 Building Orchestrator 外掛程式的方式

  您可利用不同的方式建置 Orchestrator 外掛程式。您可開始逐層建置外掛程式，或同時開始建置所有外掛程式的圖層。

- Orchestrator 外掛程式型別

  您可透過使用外掛程式整合一般用途的程式庫或公用程式，如 XML 或 SSH，以及整部系統，如搭載 Orchestrator 的 vCloud Director。視您整合 Orchestrator 的技術而定，外掛程式可分類為服務外掛程式或一般用途的外掛程式和系統外掛程式。

- 外掛程式執行

  您可在組織外掛程式結構、執行所需 Java 類別及 JavaScript 物件、開發外掛程式工作流程與活動，以及提供工作流程簡報時使用特定實用的作法與技術。

- Orchestrator 外掛程式開發建議事項

  在開發 Orchestrator 外掛程式的不同元件時，落實特定作法有助於您改善外掛程式的品質。

- 記錄外掛程式使用者介面字串與 API

  當您撰寫適用於 Orchestrator 外掛程式與相關 API 說明文件的使用者介面 (UI) 字串時，請依照樣式與格式的公認規則。

建置 Building Orchestrator 外掛程式的方式

您可利用不同的方式建置 Orchestrator 外掛程式。您可開始逐層建置外掛程式，或同時開始建置所有外掛程式的圖層。

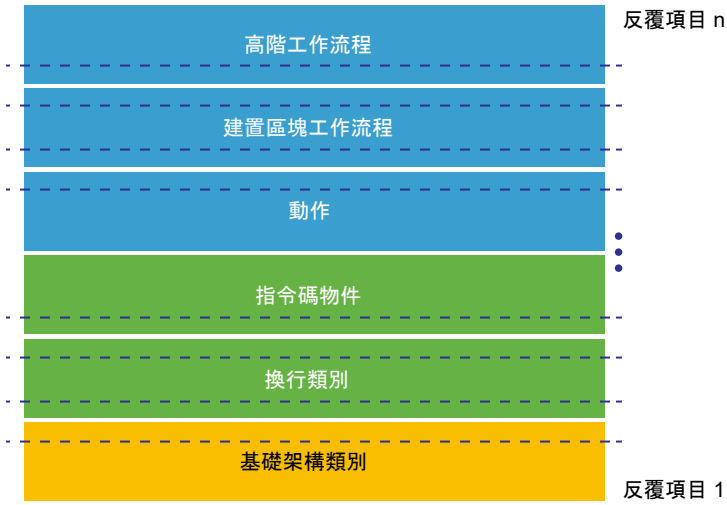
如需有關外掛程式圖層的資訊，請參閱 Orchestrator 外掛程式的結構。

由下而上開發外掛程式

外掛程式可利用由下而上的開發方式逐層建置。

由下而上開發方式可從低層級開始持續往高層級逐層建置外掛程式。當此方式混合互動式及反覆性開發方式，然後為各反覆項目交付部分或整層的外掛程式。在反覆項目 N 結束時將徹底完成外掛程式。

圖 1-5. 由下而上開發外掛程式



由下而上開發外掛程式的優點為：可在一段時間專注於單層開發。

由下而上開發外掛程式則須考慮下列的缺點。

- 在完成部分插入項目前，難以明確顯示外掛程式開發的進度。
- 無法完美整合至敏捷式開發作法中。

對含有縮減或不存在的換行類別、指令碼物件、動作或工作流程組合的小型外掛程式而言，由下而上的開發程序即足以適用。

自上而下外掛程式開發

外掛程式可使用自上而下開發法，透過配量為自上而下功能來建置。

自上而下法混合敏捷式開發程序後，各反覆項目將出現新功能。最後，在反覆項目 N 結束時將徹底執行外掛程式。

圖 1-6. 自上而下外掛程式開發

自上而下外掛程式開發法含有下列優點。

- 由於各反覆項目的新功能已完成且可釋出外掛程式，並在各反覆項目後使用，因此外掛程式的進度可在第一個反覆項目中輕易顯示。
- 完成功能的垂直配量可非常明確定義成功標準與定義已完成項目，以及改善開發人員、產品管理和品管 (QA) 工程師之間的通訊。
- 允許 QA 工程師從開發程序開始時開始測試及自動化。此法可產生寶貴的意見反應並減少整體專案交付時間範圍。

自上而下外掛程式開發法的缺點為：需同時在不同層進行開發。

您的大部分外掛程式皆應套用自上而下外掛程式開發程序。適合有動態需求的外掛程式。

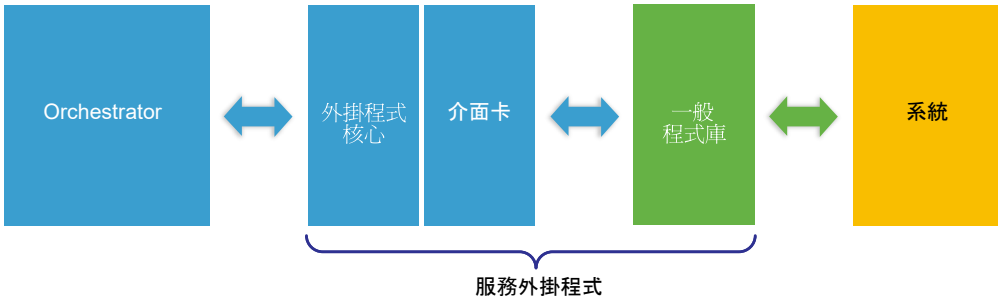
Orchestrator 外掛程式型別

您可透過使用外掛程式整合一般用途的程式庫或公用程式，如 XML 或 SSH，以及整部系統，如搭載 Orchestrator 的 vCloud Director。視您整合 Orchestrator 的技術而定，外掛程式可分類為服務外掛程式或一般用途的外掛程式和系統外掛程式。

服務外掛程式

服務外掛程式或一般用途外掛程式提供的功能可視為 Orchestrator 內的服務。

圖 1-7. 服務外掛程式架構



服務外掛程式會公開一般程式庫或公用程式至 Orchestrator，如 XML、SSH 或 SOAP。例如：下列可在 Orchestrator 中使用的外掛程式皆為服務外掛程式。

JDBC 外掛程式 讓您在工作流程內使用任何資料庫。

郵件外掛程式 讓您在工作流程內傳送電子郵件。

SSH 外掛程式 讓您在工作流程內開啟 SSH 連線及執行命令。

XML 外掛程式 讓您在工作流程內管理 XML 文件。

服務外掛程式具有下列特性。

複雜性 服務外掛程式擁有低至中級的複雜性。服務外掛程式在 Orchestrator 內部公開特定程式庫或部分程式庫，以提供具體功能。例如：XML 外掛程式會新增執行的文件物件模型 (DOM) XML 剖析器至 Orchestrator JavaScript API。

大小 服務外掛程式的大小皆相對要小。這些程式需要與所有外掛程式相同的基本類別組合，以及其他提供新指令碼物件以新增功能的類別。

詳細目錄 服務外掛程式需要小型物件詳細目錄才能運作，或者完全不需要詳細目錄。服務外掛程式含有一般及小型物件模型，因此不需要在 Orchestrator 詳細目錄內顯示此模型。

系統外掛程式

系統外掛程式連接 Orchestrator 工作流程引擎至外部系統，以便您可以協調外部系統。

下列為系統外掛程式的範例。

vCenter Server 外掛程式 讓您使用工作流程管理 vCenter Server 執行個體。

vCloud Director 外掛程式 讓您在工作流程內與 vCloud Director 安裝互動。

Cisco UCSM 外掛程式 讓您在工作流程內與 Cisco 實體互動。

下列為系統外掛程式的主要特色。

複雜性 系統外掛程式擁有比泛用外掛程式更高程度的複雜性，因為他們公開的技術相對複雜。系統外掛程式必須代表 Orchestrator 內所有外部系統的元素，並

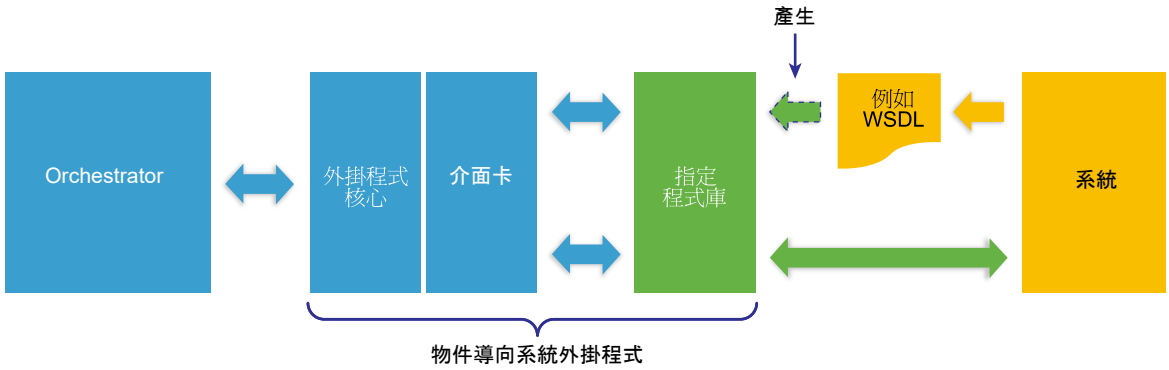
	與外部系統互動同時在 Orchestrator 中提供功能。若外部系統提供整合機制，則您可於 Orchestrator 中更輕鬆的公開系統功能。不過，除了代表 Orchestrator 外部系統元素外，系統外掛程式也可能需要提供高擴充性、快取機制、處理事件及通知等。
大小	系統外掛程式的大小為中至大型。系統外掛程式因通常需提供大量指令碼物件，所以需要除基本類別組合以外的類別。系統外掛程式可能需要一些其他協助程式和與其互動的輔助類別。
詳細目錄	通常系統外掛程式擁有大量的物件，而您必須在詳細目錄中正確公開這些物件，以便在 Orchestrator 中輕鬆找到並搭配執行。由於系統外掛程式需要公開大量物件，因此您應建置輔助工具或程序，為外掛程式盡可能自動產生更多代碼。例如：vCenter Server 外掛程式提供的工具。

物件導向系統外掛程式

物件導向系統提供的互動機制是根據物件及 RPC 為主。

適合物件導向系統最廣泛使用的模型為使用 SOAP 的網路服務模型。模型中物件擁有的屬性組合與物件狀態有關，提供在目標系統端叫用的遠端方法組合。

圖 1-8. 物件導向系統外掛程式



執行物件導向系統的外掛程式時，您可考慮下列項目。

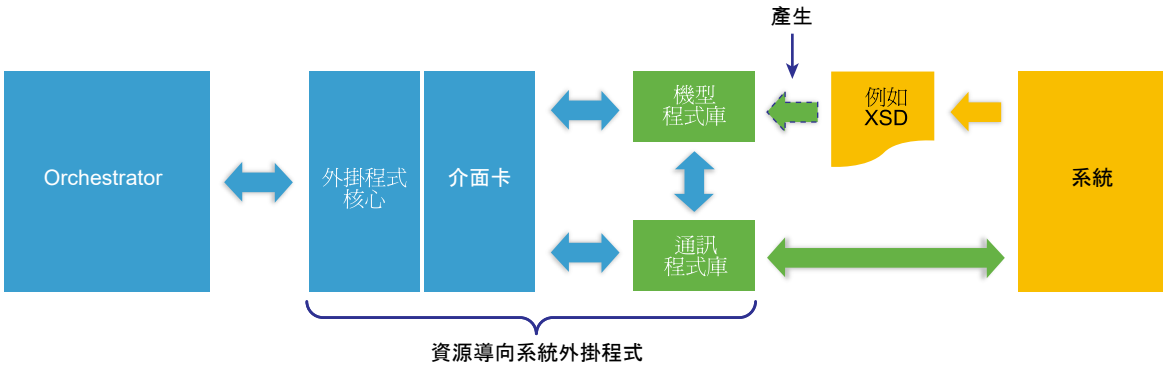
- 若您使用 SOAP，您可使用 WSDL 檔案產生結合物件模型及通訊機制的類別組合。
- 此物件模型幾乎是您必須在 Orchestrator 內公開的所有項目。

資源導向系統外掛程式

資源導向系統提供的互動機制是以使用 HTTP 方法的資源及簡易作業為基礎。

最具代表性的資源導向系統模型是 REST 模型，並與含 XML 的範例結合。此模型內的物件含有與其狀態相關的屬性組合。若要叫用目標系統上的方法 (通訊機制)，您必須使用標準 HTTP 方法，如 `GET`、`POST`、`PUT` 等，並遵循某些慣例。

圖 1-9. 資源導向系統外掛程式



開發資源導向系統的外掛程式時，您可考慮下列項目。

- 若您使用 REST 或只有含 XML 的 HTTP，您可獲得一或多個 XML 配置檔案以供讀取及寫入訊息。在這些配置中，您可產生定義物件模型的類別組合。此類別組合只定義物件狀態，因為以 HTTP 方法隱含定義作業，例如 vCloud Director 外掛程式中的定義，或明確含部分指定 XML 訊息，如 Cisco UCSM 外掛程式。
- 您需在其他類別組合中執行通訊機制。此類別組合定義與原始物件模型互動的新物件模型。通訊機制的物件模型只包含物件與方法。
- 您可公開 Orchestrator 內通訊機制的原始物件及物件模型。視公開物件模型的方式，以及您是否合併兩方相關物件（用於模擬物件導向系統）或維持分離，這可能會增加一些複雜性。

外掛程式執行

您可在組織外掛程式結構、執行所需 Java 類別及 JavaScript 物件、開發外掛程式工作流程與活動，以及提供工作流程簡報時使用特定實用的作法與技術。

- 專案結構

  您可套用用於 Orchestrator 外掛程式專案的標準結構。
- 專案內部

  您可在執行外掛程式時套用特定方式，例如快取物件、讓物件進入背景、複製物件等。您可透過此方式改善外掛程式的效能、避免發生並行處理問題，以及改善 Orchestrator 用戶端回應性。
- 工作流程內部

  您可執行工作流程監控您 Orchestrator 外掛程式執行的長時間作業。
- 工作流程及動作

  若要減輕工作流程開發及使用，您可使用特定的作法。
- 工作流程簡報

  當您建立工作流程簡報後，應套用特定結構與規則。

專案結構

您可套用用於 Orchestrator 外掛程式專案的標準結構。

您可使用含模組的標準 Maven 結構，讓您的外掛程式專案明白瞭解所有功能所在的位置。

表 1-49. 外掛程式專案結構

模組	說明
/myAwesomePlugin-plugin	外掛程式專案的根。
/o11nplugin-myAwesomePlugin	構成最終外掛程式 DAR 檔案的模組。
/o11nplugin-myAwesomePlugin-config	包含外掛程式組態網路應用程式的模組。會產生標準 WAR 檔案。
/o11nplugin-myAwesomePlugin-core	此模組包含所有執行任何標準 Orchestrator 外掛程式介面的類別及其使用的其他輔助類別。會產生標準 JAR 檔案。
/o11nplugin-myAwesomePlugin-model	此模組包含所有有助於您透過外掛程式整合第三方技術和 Orchestrator 的類別。這些類別不得包含任何標準 Orchestrator 外掛程式 API 的直接參照。
/o11nplugin-myAwesomePlugin-package	此模組匯入含動作與工作流程的外部 Orchestrator 套件檔案，並在最終外掛程式 DAR 檔案內加入。此模組為選用。

專案內部

您可在執行外掛程式時套用特定方式，例如快取物件、讓物件進入背景、複製物件等。您可透過此方式改善外掛程式的效能、避免發生並行處理問題，以及改善 Orchestrator 用戶端回應性。

快取物件

您的外掛程式會與遠端服務互動，並由本機物件提供此代表服務端遠端物件的互動。若要達到良好的外掛程式效能以及出色的 Orchestrator UI 回應性，您可快取本機物件，而非每次從遠端服務取得。您可考慮快取範圍，例如一個代表所有外掛程式用戶端的快取，每外掛程式使用者一個快取，以及每第三方服務使用者一個快取。在執行時，您的快取機制會整合外掛程式介面，以尋找物件及讓物件失效。

讓物件進入背景

若您必須在外掛程式詳細目錄中顯示大量物件清單，且沒有擷取這些物件的快速方式，您可讓物件進入背景。您可讓物件進入背景，例如讓物件擁有兩個狀態：`fake` 和 `loaded`。假設 `fake` 物件很容易建立及提供您必須在詳細目錄中顯示的最少資訊，如名稱與識別碼。接著將可隨時傳回 `fake` 物件，並在真正需要所有資訊（真實物件）時，使用實體或外掛程式可自動叫用方法 `load` 以取得真實物件。您甚至可設定載入物件的程序，在傳回假物件後自動開始，以評估使用實體的動作。

複製物件避免發生並行問題

若您針對外掛程式使用快取，您必須複製物件。對每個實體使用隨時傳回相同物件執行個體的快取，以要求可擁有不需要的效果。例如：實體 A 要求物件 O，而該實體會檢視詳細目錄中的物件及其所有屬性。在此同時，實體 B 也會要求物件 O，而實體 A 則會執行工作流程，開始變更物件 O 的屬性。在執行結束時，工作流程會叫用物件的 `update` 方法以更新伺服器端的物件。若實體 A 和實體 B 取得物件 O 的相同執行個體，實體 A 會檢視詳細目錄中實體 B 執行的所有變更，甚至包括伺服器端認可變更前的項目。若執行順利，則沒有問題，但若執行失敗，則不還原實體 A 物件 O 的屬性。在此情況下，若快取（外掛程式 `find` 作業）傳回物件複製品，而非一直相同的執行個體，實體都會在每次使用時檢視與修改本身的複本，至少在 Orchestrator 中避免發生並行問題。

通知其他人變更

當您同時使用快取及複製品物件時，可能會發生問題。最大的問題在於該物件使用的實體檢視，可能不是物件可用的最新版本。例如：若實體顯示詳細目錄，將載入一次物件，但在此同時，若其他實體變更部分物件，第一個實體不會檢視變更。若要避免此問題，您可使用 Orchestrator 外掛程式 API 的 `PluginWatcher` 和 `IPluginPublisher` 方法，告知已變更為允許其他 Orchestrator 用戶端執行個體查看變更。此方式也可在從詳細目錄單一物件變更影響其他詳細目錄物件時，以及需要告知時，套用至 Orchestrator 用戶端唯一執行個體。這些物件或物件部分內容顯示在詳細目錄時，方便使用通知的作業會新增、更新及刪除物件。

隨時啟用尋找任何物件

您必須執行 `IPluginFactory` 介面的 `find` 方法，按型別和識別碼尋找物件。`find` 方法可在重新啟動 Orchestrator 並繼續執行工作流程後直接叫用。

若您沒有查詢服務，請模擬

例如在特定狀況中，Orchestrator 用戶端會需要查詢部分物件或顯示為清單或表單而非樹狀結構。這代表您的外掛程式必須能隨時查詢部分物件組合。若第三方技術提供查詢服務，您需採用並利用此服務。否則，儘管此方式的複雜性較高或解決方案效能較低，您應能模擬查詢服務。

尋找方法不得傳回執行階段例外狀況

執行外掛程式內部搜尋的 `IPluginFactory` 介面方法不得擲出受控或不受控的執行階段例外狀況。這可能是在執行工作流程時奇怪驗證錯誤失敗的原因。例如：若第一個節點的輸出為第二個節點的輸入，則在工作流程的兩個節點之間叫用 `find` 方法。在此時，若因任何執行階段例外狀況導致找不到物件，您只能得到 Orchestrator 用戶端驗證錯誤的資訊。之後，將視外掛程式記錄例外狀況的方式，在記錄檔案內收到更多或更少的資訊。

工作流程內部

您可執行工作流程監控您 Orchestrator 外掛程式執行的長時間作業。

您可執行工作流程監控長時間執行作業，如工作監控。此工作流程可根據 Orchestrator 觸發器和等待事件為基準。您必須考慮等待工作的封鎖工作流程可於 Orchestrator 伺服器啟動時立即恢復。外掛程式必能取得所有必要資訊，以正確恢復監控程序。

監控工作流程或可在內部使用的工作皆應提供機制，以指定輪詢率和可能的逾時。

偵錯工作流程內部的一段指令碼程序並不容易，特別是該代碼未叫用任何 Java 代碼。由於此緣故，有時唯一的選擇是使用預設 Orchestrator 指令碼物件的的記錄方法。

工作流程及動作

若要減輕工作流程開發及使用，您可使用特定的作法。

開始將工作流程當成建置組塊開發

建置組塊可以是需要少數輸入參數及傳回簡易輸出的簡易工作流程。若您擁有豐富的建置組塊組合，可輕鬆建立更高階的工作流程，並可在組成複雜工作流程中提供更出色的工具組合。

根據小型元件建立高階工作流程

若您必須開發含多個輸入與內部步驟的複雜工作流程，您可分割為更小及更簡單的建置組塊工作流程和動作。

請盡可能建立動作

您可建立動作以便在開發工作流程時達到額外彈性。

- 建立複雜物件或參數以輕鬆執行指令碼方法
- 避免始終重複通用代碼片段
- 執行 UI 驗證

工作流程應在可行時叫用動作

可隨工作流程配置內的節點直接叫用動作。可確保工作流程配置簡化，因為您不需要新增指令碼代碼區塊叫用單一動作。

填入所需的資訊

提供工作流程或動作的各元素資訊。

- 提供工作流程或動作的描述。
- 提供輸入參數的描述。
- 提供輸出的描述。
- 提供工作流程屬性的描述。

維持最新的版本資訊

當您執行外掛程式的版本設定時，請新增含外掛程式主要更新、重要執行詳細資訊等有意義的註解。

工作流程簡報

當您建立工作流程簡報後，應套用特定結構與規則。

使用下列內容在工作流程簡報中進行工作流程輸入。

表 1-50. 工作流程輸入內容

內容	使用方式
Show in Inventory	使用此內容有助於使用者執行詳細目錄視圖中的工作流程。
Specify a root object to be shown in the chooser	使用此內容有助於使用者選擇輸入。若可在簡報中重新整理的根物件為屬性，或透過物件方法擷取，您需建立或設定適當的動作才能重新整理簡報中的物件。
Maximum string length	使用此用於長字串的內容，如名稱、描述、檔案路徑等。
Minimum string length	使用此內容可避開測試工具中的空字串。
Custom validation	執行含動作的非範本驗證。

利用步驟及顯示群組組織輸入。此組織有助於使用者識別及區分工作流程中的所有輸入參數。

Orchestrator 外掛程式開發建議事項

在開發 Orchestrator 外掛程式的不同元件時，落實特定作法有助於您改善外掛程式的品質。

表 1-51. 外掛程式執行的實用作法

元件	項目	說明
一般	存取第三方 API	外掛程式應盡可能提供簡化的第三方 API 存取方式。
	介面	外掛程式應提供一致且標準的介面供使用者使用，即使 API 無法提供。
動作	指令碼物件	您應為各建立、修改、刪除和所有其他指令碼物件的可用方式建立動作。
	說明	動作描述應描述動作內容而非運作方式。
	執行指令碼	當您使用指令碼取得物件屬性或方式時，可檢查物件數值是否與 `null` 或 `undefined` 不同。
	取代	若動作遭取代，`comment` 或 `throw` 陳述式應指出取代動作，或該動作應呼叫新的取代動作，以便取代的動作版本解決方案不會無效。
工作流程	採協調技術的使用者介面作業	您應為協調技術的使用者介面中的所有作業建立工作流程。
	說明	工作流程描述應描述工作流程內容而非運作方式。
	簡報內容 `mandatory input`	您必須為所有強制工作流程輸入設定 `mandatory input` 內容。
	簡報內容 `default value`	若您開發的工作流程設定實體，則該工作流程簡報應載入此實體的預設組態數值。例如：若您開發名為主機組態的工作流程，此工作流程簡報必須載入主機組態的預設數值。
	簡報內容 `Show in inventory`	您必須設定 `Show in inventory` 內容才能擁有詳細目錄物件的內容工作流程。
	簡報內容 `specify a root parameter`	您應在無須瀏覽樹狀根目錄的詳細目錄時，使用工作流程中的此內容。
	工作流程驗證	您必須驗證工作流程及修正所有錯誤。
	物件建立	所有建立新物件的工作流程皆應將新物件當成輸出參數傳回。
	取代	若工作流程遭取代，`comment` 或 `throw` 陳述式應指出取代工作流程，或該工作流程應呼叫新的取代工作流程，以確保根據舊版工作流程建立的解決方案不會無效。
詳細目錄	主機中斷連線	若您的詳細目錄包含主機連線，則此主機將無法使用，而您應指出中斷連接的主機。您可利用 vCloud Director 外掛程式相同的方式，藉由附加 `– disconnected` 重新命名根物件或移除此物件下的物件樹狀結構，完成此動作。
	`Select value as list` 內容	詳細目錄物件必須可當成 `treeview` 或 `list` 選擇。
	主機管理員	若外掛程式執行目標系統的 `host` 物件，則父系 `hostmanager` 根物件應含有用於新增、移除或編輯主機的內容。
	取得或更新物件	若正使用協調技術執行查詢服務，您應用於取得多重物件。
	子系探索	若您需要另外擷取子系物件，擷取的程序必須為多執行緒且在單一錯誤上未封鎖。
	Orchestrator 物件變更	詳細目錄中所有可變更元素狀態的工作流程都必須更新詳細目錄，以避免讓物件脫離同步化。

表 1-51. 外掛程式執行的實用作法 (續)

元件	項目	說明
	外部物件變更	您可使用通知機制來告知因執行作業超出 Orchestrator 範圍而導致協調技術中發生變更。若此作業造成協調技術中的物件移除，您必須重新整理詳細目錄，以避免故障或資料遺失。例如：若刪除 vCenter Server 中的虛擬機器，vCenter Server 外掛程式會更新詳細目錄，以移除已移除虛擬機器的物件。
	Finder 物件	Finder 物件應含有可用於區分物件的內容。這些通常是存在於使用者介面中的內容。
指令碼物件	執行	必須執行 `equals` 方式才能確保在相同物件上進行的 `==` 作業與部分狀況下可能擁有兩個執行個體的物件相同。
	外掛程式物件內容	含父系物件的物件應執行 `parent` 內容。
	外掛程式物件內容	含子系物件的物件應執行 `GET` 方式以傳回大量子系物件。
	詳細目錄物件	詳細目錄物件應可透過 `Server.find` 搜尋。
		所有詳細目錄物件應可序列化，以便在工作流程中當成輸入或輸出屬性使用。
	建構函式與方法	在大多數情況下，可編寫指令碼物件應含有建構函式或由其他物件屬性或方式傳回。
	物件識別碼	從外部系統發出擁有識別碼的物件應使用內部識別碼，藉此確保您在協調多部伺服器時不會出現重複的識別碼。
	搜尋物件	`search` 或 `find` 方式應執行篩選，以便找出指定名稱或識別碼，而非所有物件。例如：Orchestrator 伺服器的 `Server.FindForId` 方式可透過外掛程式物件的識別碼找到外掛程式。若要搜尋，必須針對外掛程式中各個可找到的物件執行此方式。
	觸發	如有可能，應可觸發變更物件，讓 Orchestrator 能在各種事件時觸發原則。例如：要決定新增虛擬機器、開機、關機等時間，Orchestrator 可監控 `Datacenter` 物件上 vCenter 外掛程式中的觸發或事件。
	物件內容	其他外掛程式中的物件應有可輕易在外掛程式間轉換的內容。例如：虛擬機器物件需有 `moref`（受管理的物件參考識別碼）。
	工作階段管理員	若您連接至可用有不同工作階段的遠端伺服器，該外掛程式應執行共用工作階段與各使用者的工作階段。
觸發	觸發	所有長時間作業與封鎖方式皆應能在傳回工作時非同步開始，並於完成時產生觸發事件。
列舉	列舉	列舉指定型別應含有能於列舉中選擇不同數值的詳細目錄物件。
記錄	記錄	方法應執行不同的記錄層級。
版本設定	外掛程式版本	外掛程式版本應遵循標準並隨外掛程式更新進行更新。
API 說明文件	方法	在 API 說明文件中描述的方法不得在物件上擲回例外狀況 `no xyz method / property`。相反的，方法應在沒有可用內容時傳回 `null` 並詳細記錄。
	`vso.xml`	所有物件、方法及內容都必須記錄在 `vso.xml` 中。

記錄外掛程式使用者介面字串與 API

當您撰寫適用於 Orchestrator 外掛程式與相關 API 說明文件的使用者介面 (UI) 字串時，請依照樣式與格式的公認規則。

一般建議

- 在外掛程式中使用提及的 VMware 產品的正式名稱。例如：使用下列產品的正式名稱及 VMware 詞彙。

正確術語	請勿使用
vCenter Server	VC 或 vCenter
vCloud Director	vCloud

- 以句點結束所有工作流程的描述。例如：`Creates a new Organization.` 為工作流程描述。
- 使用含拼字檢查程式的文字編輯器寫入描述，然後移至外掛程式中。
- 確保外掛程式的名稱與相關聯並經審核的第三方產品名稱完全相符。

工作流程及動作

- 寫入資訊性描述。大多數的動作和工作流程只需一或兩句描述即已足夠。
- 更高階層的工作流程可能包括更廣泛的描述與註解。
- 請以動詞作為描述的開頭，例如 `Creates…`。請勿使用如 `This workflow creates` 等自我參照的語言。
- 在描述結尾加上句點，完成該句的描述。
- 描述工作流程或動作的作用，而非執行的方式。
- 工作流程與動作通常包括在資料夾與套件中。同樣針對這些資料夾與套件包含簡短描述。例如：工作流程資料夾可擁有類似 `Set of workflows related to vApp Template management` 的描述。

工作流程與動作的參數

- 例如：用描述性的名詞片語 `Name of` 作為工作流程與動作描述的開頭。請勿使用如 `It's the name of` 的片語。
- 請勿在參數結尾及動作描述後加上句點。這些皆非完整句子。
- 工作流程的輸入參數必須在呈現視圖中指定含有適當名稱的標籤。在多數情況下，您可在顯示群組中組合相關的輸入。例如：在未輸入兩個含有組織名稱與組織全名標籤的情形下，您可建立含有標籤組織的顯示群組並在組織群組中放上輸入名稱及全名。
- 在步驟與顯示群組方面，新增描述或註解也會出現在工作流程簡報中。

外掛程式 API

- API 的說明文件意指 `vso.xml` 檔案及 Java 來源檔案中的所有說明文件。
- 如為 `vso.xml` 檔案，請使用相同規則來描述 finder 物件和指令碼物件，搭配您用於工作流程和動作的方法。物件屬性及方式參數的描述皆使用與工作流程及動作參數相同的規則。

- 請避免在 `vso.xml` 檔案中使用特殊字元以及在 `<![CDATA[insert your description here!]]>` 標籤中加入描述。
- Java 來源檔案請使用標準 Javadoc 樣式。

工作流程開始時取得使用者的輸入參數

如果工作流程需要輸入參數，它將開啟對話方塊，以供使用者輸入執行時所需的輸入參數值。在使用者編輯器的**呈現**索引標籤中，整理此對話方塊的內容和配置或呈現方式。

您在**呈現**索引標籤中整理參數的方式，將在工作流程執行時轉換為輸入參數對話方塊。

呈現索引標籤也可讓您新增輸入參數的說明，以協助使用者提供輸入參數。您也可以在**呈現**索引標籤中設定參數的優先順序和限制，以限制使用者提供的參數。如果使用者提供的參數不符合您在**呈現**索引標籤中設定的限制，工作流程將不會執行。

- 在呈現索引標籤中建立輸入參數對話方塊

  您可以在工作流程編輯器的**呈現**索引標籤中定義對話方塊的配置，以供使用者在執行工作流程時提供輸入參數。

- 設定參數內容

  Orchestrator 可讓您定義內容，使得使用者執行工作流程時所提供的輸入參數值合格。您定義的參數內容會限制使用者提供的輸入參數類型和值。

在呈現索引標籤中建立輸入參數對話方塊

您可以在工作流程編輯器的**呈現**索引標籤中定義對話方塊的配置，以供使用者在執行工作流程時提供輸入參數。

呈現索引標籤可讓您將輸入參數分組成不同的類別，然後定義這些類別在輸入參數對話方塊中顯示的順序。

呈現說明

您可以為每個參數或參數群組新增相關的說明，此說明會在輸入參數對話方塊中顯示。說明中的資訊可協助使用者提供正確的輸入參數。您可以使用 HTML 格式設定來增強說明文字的配置。

定義呈現輸入步驟

依預設，輸入參數對話方塊會以單一清單列出所有所需的輸入參數。若要幫助使用者輸入輸入參數，您可以在 [呈現] 索引標籤中定義節點，亦即輸入步驟。輸入步驟會將本質類似的輸入參數分成一組。當執行工作流程時，輸入步驟下的輸入參數會顯示在 [輸入參數] 對話方塊上的個別區段中。

定義呈現顯示群組

每個輸入步驟都有自己的節點，亦即顯示群組。顯示群組會定義參數輸入文字方塊在輸入參數對話方塊中其各自區段中顯示的順序。您可以獨立於輸入步驟之外定義顯示群組。

建立輸入參數對話方塊的呈現

您可以建立對話方塊呈現，以供使用者在工作流程編輯器的**呈現**索引標籤中執行工作流程時提供輸入參數。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 確認工作流程含有輸入參數的定義清單。

程序

1. 在工作流程編輯器中，按一下**呈現**索引標籤。

   依預設，所有的工作流程參數會依照您建立參數的順序顯示在主要**呈現**節點下方。

2. 在**呈現**節點上按一下滑鼠右鍵，然後選取**建立新步驟**。

   新步驟節點會在**呈現**節點下方顯示。

3. 為該步驟提供適當的名稱，然後按 Enter。

   此名稱在工作流程執行時會顯示為輸入參數對話方塊中的區段標頭。

4. 按一下輸入步驟，並在**呈現**索引標籤下半部的**一般**索引標籤中新增說明。

   此說明會在輸入參數對話方塊中顯示，以提供資訊協助使用者提供正確的輸入參數。您可以使用 HTML 格式設定來增強說明文字的配置。

5. 在您建立的輸入步驟上按一下滑鼠右鍵，然後選取**建立顯示群組**。

   新群組節點會在輸入步驟節點下方顯示。

6. 為顯示群組提供適當的名稱，然後按 Enter。

   此名稱在工作流程執行時會顯示為輸入參數對話方塊中的子區段標頭。

7. 按一下顯示群組，並在**呈現**索引標籤下半部的**一般**索引標籤中新增說明。

   此說明顯示在輸入參數對話方塊中。您可以使用 HTML 格式設定來增強說明文字的配置。您可以使用 OGNL 陳述式，將參數值新增至群組說明，例如 `${#param}`。

8. 重複前述步驟，直到您建立所有的輸入步驟，並顯示當工作流程執行時出現在輸入參數對話方塊中的群組為止。

9. 將參數從**呈現**節點下方拖曳至您選擇的步驟與群組。

您已建立輸入參數對話方塊的配置，以供使用者在執行工作流程時提供輸入參數值。

後續步驟

您必須設定參數內容。

設定參數內容

Orchestrator 可讓您定義內容，使得使用者執行工作流程時所提供的輸入參數值合格。您定義的參數內容會限制使用者提供的輸入參數類型和值。

每個參數都有多個內容。對於**呈現**索引標籤中的指定參數，您可以在**內容**索引標籤中定義輸入參數的內容。

參數內容會驗證輸入參數，並修改文字方塊在輸入參數對話方塊中的顯示方式。部分參數內容可建立參數之間的相依性。

靜態與動態參數內容值

參數內容值可以是靜態，也可以是動態。靜態內容值會保持不變。如果您將內容值設定為靜態，可從工作流程編輯器按照參數類型產生的清單中設定或選取內容的值。

動態內容值取決於其他參數或屬性的值。您可以使用對象圖導航語言 (OGNL) 運算式，定義動態內容取得值所用的函數。如果動態參數內容值取決於另一個參數內容值的值，而另一個參數內容值變更，則 OGNL 運算式將重新計算並變更動態內容值。

設定參數內容

工作流程啟動時，會依據您指定的任何參數內容驗證使用者的輸入參數值。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 確認工作流程含有輸入參數的定義清單。

程序

1. 在工作流程編輯器中，按一下**呈現**索引標籤。
2. 按一下**呈現**索引標籤中的參數。

   參數的**一般**和**內容**索引標籤會出現在**呈現**索引標籤的底端。
3. 按一下參數的**內容**索引標籤。
4. 在**內容**索引標籤上按一下滑鼠右鍵，並選取**新增內容**。

   對話方塊隨即開啟，其中列出所選類型之參數的可能內容。
5. 從對話方塊中的清單選取內容，然後按一下**確定**。

   內容隨即出現在**內容**索引標籤中。
6. 在**值**下，選取下拉式功能表中相對應的符號，將內容值設定為靜態或動態。

選項	說明
	靜態內容
	動態內容

7. 如果您將內容值設定為靜態，即可根據您設定內容的參數類型選取內容值。

8 如果您將內容值設定為動態，即可使用 OGNL 運算式來定義函數以取得參數內容值。

工作流程編輯器提供撰寫 OGNL 運算式的說明。

a 按一下 圖示，可用來列出此運算式可呼叫的工作流程所定義的所有屬性和參數。

b 按一下 圖示，可用來列出 Orchestrator API 中對於您定義的內容傳回類型之輸出參數的所有動作。

按一下參數和動作的建議清單中的項目，即可將項目新增至 OGNL 運算式。

9 按一下工作流程編輯器底端的**儲存**。

此時您即已定義工作流程的輸入參數內容。

後續步驟

驗證和偵錯工作流程。

Object Missing

This object is not available in the repository.

OGNL 運算式的預先定義常數值

您建立 OGNL 運算式取得動態參數內容值時，可以使用預先定義常數。

Orchestrator 定義用於 OGNL 運算式的下列常數。

表 1-52. 預先定義 OGNL 常數值

常數值	說明
`${#__current}`	自訂驗證內容或相符運算式內容的目前值
`${#__username}`	啟動工作流程的使用者名稱。
`${#__userdisplayname}`	啟動工作流程的使用者顯示名稱。
`${#__serverurl}`	URL，包含使用者啟動工作流程的伺服器 IP 位址。URL 包含伺服器 IP 位址和查詢連接埠： `{ServerIP}:{lookupPort}`
`${#__datetime}`	目前日期和時間
`${#__date}`	目前日期，時間設定為 00:00:00
`${#__timezone}`	目前時區

工作流程執行時要求使用者互動

工作流程有時會在執行時需要外部來源的其他輸入參數。這些輸入參數可來自於其他應用程式或工作流程，或者使用者可直接提供。

例如：若在執行工作流程時發生特定事件，工作流程可要求人為互動以決定要採取的動作過程。工作流程會先等待，再繼續執行，直到使用者回應要求的資訊，或直到等待時間超過可能的逾時時間。若等待時間超過逾時時間，工作流程會傳回例外狀況。

用於使用者互動的預設屬性為 `security.group` 及 `timeout.date`。當您設定 `security.group` 屬性為指定 LDAP 使用者群組時，您會限制該使用者群組成員的使用者互動要求回應權限。

當您設定 `timeout.date` 屬性時，您會設定工作流程在收到使用者資訊之前等待的時間與日期。您可設定絕對日期，或者可建立執行指令的工作流程元素，以計算相對於目前時間的時間。

程序

1 (選擇性) 新增使用者互動至工作流程

您在工作流程執行期間透過將**使用者互動**架構元素新增至工作流程，向使用者要求輸入參數。當工作流程碰到**使用者互動**元素時會暫停執行，並等待使用者提供所需的資料。

2 (選擇性) 設定使用者互動安全性群組屬性

使用者互動元素的 `security.group` 屬性可設定使用者或群組使用者擁有回應使用者互動的權限。

3 (選擇性) 設定逾時日期屬性為絕對日期

設定用於使用者互動的 `timeout.date` 屬性，可設定工作流程等待使用者回應使用者互動的時間長度。

4 (選擇性) 計算使用者互動的相對逾時

您可在 `Date` 物件中，於使用者互動逾時時計算相對時間與日期。

5 (選擇性) 設定逾時日期屬性為相對日期

您可繫結 **使用者互動** 元素的 `timeout.date` 屬性至 `Date` 物件，設定為相對時間與日期。您可在執行指令函式中定義物件。

6 (選擇性) 定義使用者互動的外部輸入

您指定使用者必須在工作流程執行期間提供的資訊，作為使用者互動的輸入參數。

7 (選擇性) 定義使用者互動例外狀況行為

若使用者不在逾時時間內提供輸入參數，使用者互動會傳回例外狀況。您可在執行指令函式中定義例外狀況行為。

8 (選擇性) 建立使用者互動的輸入參數對話方塊

使用者在工作流程執行期間，於輸入參數對話方塊中提供輸入參數，並透過相同方式在工作流程首次啟動時提供輸入參數。

9 (選擇性) 回應使用者互動要求

工作流程需要在執行期間進行使用者互動，在使用者提供所需資訊或工作流程逾時之前會一直暫停。

新增使用者互動至工作流程

您在工作流程執行期間透過將**使用者互動**架構元素新增至工作流程，向使用者要求輸入參數。當工作流程碰到**使用者互動**元素時會暫停執行，並等待使用者提供所需的資料。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。

- 新增一些元素至工作流程配置。

程序

1. 將**使用者互動**元素拖曳至工作流程架構中的適當位置。
2. 按一下**使用者互動**元素的**編輯**圖示 (✏)。
3. 在**資訊**索引標籤中提供使用者互動的名稱與說明，然後按一下**關閉**。
4. 按一下**儲存**。

您已將使用者互動元素新增至工作流程。工作流程接觸到此元素時，會等待使用者提供資訊後再繼續執行。

後續步驟

設定使用者互動的 `security.group` 屬性，將回應使用者互動的權限限制為使用者或使用者群組。請參閱設定使用者互動安全性群組屬性。

設定使用者互動安全性群組屬性

使用者互動元素的 `security.group` 屬性可設定使用者或群組使用者擁有回應使用者互動的權限。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增部分元素及使用者互動至工作流程配置。
- 識別 LDAP 使用者群組以回應使用者互動要求。

程序

1. 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 (✏)。
2. 按一下使用者互動的 **屬性** 索引標籤。
3. 按一下 `security.group` 來源參數的 **不設定** ，設定可回應使用者互動的使用者。
4. (選擇性) 選擇 **NULL** 讓所有使用者都能回應使用者互動的要求。
5. 若要限制回應指定使用者或使用者群組的權限，請按一下**在工作流程中建立參數 / 屬性**。

   參數資訊 對話方塊開啟。
6. 為參數命名。
7. 選擇 **建立含相同名稱的工作流程屬性** ，在工作流程中建立 `LdapGroup` 屬性。
8. 按一下參數數值的 **不設定** ，開啟 **LdapGroup** 選項方塊。
9. 在 **篩選器** 文字方塊中鍵入 LDAP 使用者群組的名稱。

10 在清單中選擇 LDAP 使用者群組並按一下 **選擇**。

例如：選擇 **管理員** 群組代表只有該群組中的成員能回應此使用者互動要求。

您已限制回應使用者互動要求的權限。

11 按一下 **確定** 關閉 **參數資訊** 對話方塊。

您已設定使用者互動的 `security.group` 屬性。

後續步驟

設定 `timer.date` 屬性，設定使用者互動的逾時時間。

- 若要設定逾時為絕對日期與時間，請參閱 設定逾時日期屬性為絕對日期。
- 若要建立與目前日期與時間相關的計算逾時功能，請參閱 計算使用者互動的相對逾時。

設定逾時日期屬性為絕對日期

設定用於使用者互動的 `timeout.date` 屬性，可設定工作流程等待使用者回應使用者互動的時間長度。

您可在 `Date` 物件中，設定絕對時間與日期。到達指定日期的時間後，等待使用者互動逾時的工作流程將在 `Failed` 狀態下結束。例如：您可設定使用者互動在 2 月 12 日中午逾時。若要計算與目前日期和時間相關的逾時，請參閱 計算使用者互動的相對逾時。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增使用者互動元素至工作流程配置。
- 設定使用者互動的 `security.group` 屬性。

程序

1 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 (✏)。

2 按一下使用者互動的 **屬性** 索引標籤。

3 按一下 `timeout.date` 來源參數的 **不設定**，設定逾時參數數值。

4 (選擇性) 選擇 **NULL** 可讓使用者互動設定工作流程一直等待直到使用者回應使用者互動。

5 按一下 **在工作流程中建立參數 / 屬性** ，設定工作流程在逾時過後失敗。

參數資訊 對話方塊開啟。

6 為參數命名。

7 選擇 **建立含相同名稱的工作流程屬性** ，在工作流程中建立 `Date` 屬性。

8 按一下參數 **數值** 的 **不設定**。

9 使用行事曆選擇工作流程等待使用者回應的絕對日期與時間。

10 按一下**確定**關閉行事曆。

11 按一下 **確定** 關閉 **參數資訊** 對話方塊。

設定絕對日期的 timeout.date 屬性。若使用者在此時間與日期之前未回應使用者介面，則工作流程逾時。

後續步驟

定義使用者互動需從使用者獲得的外部輸入參數。請參閱定義使用者互動的外部輸入。

計算使用者互動的相對逾時

您可在 Date 物件中，於使用者互動逾時時計算相對時間與日期。

您可在 Date 物件中設定絕對時間與日期。到達指定日期的時間後，使用者互動的要求會逾時。此外，您可依您定義的函式建立計算及產生相對 Date 物件的工作流程元素。例如：您可建立相對 Date 物件，以新增 24 小時至目前時間。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增使用者互動元素至工作流程配置。
- 設定使用者互動的 security.group 屬性。

程序

1. 先從**一般**功能表將**可編寫指令碼工作**元素拖曳至工作流程的架構，放在需要其 timeout.date 屬性之相關 Date 物件的元素之前。
2. 在工作流程架構中，按一下**可編寫指令碼工作**元素的**編輯**圖示 ()。
3. 在**資訊**內容索引標籤中，提供指令碼式工作流程元素的名稱和說明。
4. 按一下**輸出**內容索引標籤，然後按一下**繫結至工作流程參數/屬性** 圖示 ()。
5. 按一下**在工作流程中建立參數/屬性**，建立工作流程屬性。
   a. 將屬性命名為 timerDate。
   b. 從屬性類型的清單中選取 Date。
   c. 選取**建立相同名稱的工作流程屬性**。
   d. 將屬性值維持設定為**未設定**，因為指令碼式函數將提供此值。
   e. 按一下**確定**。
6. 按一下指令碼式工作流程元素的**指令碼**索引標籤。
7. 在**指令碼**索引標籤的指令碼輸入台中，定義函數以計算並產生名為 timerDate 的 Date 物件。

   例如，您可以實作下列 JavaScript 函數來建立 Date 物件，其中的逾時時間是相對延遲毫秒數。

```
timerDate = new Date();
System.log( "Current date : '" + timerDate + "'" );
timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
System.log( "Timer will expire at '" + timerDate + "'" );
```

前一個範例 JavaScript 函數使用 `getTime` 方法來定義包含目前日期與時間的 `Date` 物件，並新增 86,400,000 毫秒，也就是 24 小時。**可編寫指令碼工作**元素將產生此值作為其輸出參數。

8 按一下**關閉**。

9 按一下**儲存**。

您已建立計算相對目前時間與日期，並產生 `Date` 物件的時間與日期函式。**使用者互動** 元素可將此 `Date` 物件當成輸入參數接收，以設定逾時時間直到等到使用者輸入為止。工作流程到達 **使用者互動** 元素時，將暫停執行並等待直到使用者提供必要資訊，或在逾時之前等待 24 小時。

後續步驟

您必須繫結 `Date` 物件至 **使用者互動** 元素的 `timeout.date` 參數。請參閱設定逾時日期屬性為相對日期。

設定逾時日期屬性為相對日期

您可繫結 **使用者互動** 元素的 `timeout.date` 屬性至 `Date` 物件，設定為相對時間與日期。您可在執行指令函式中定義物件。

若您在執行指令函式中建立相對 `Date` 物件，您可繫結使用者互動的 `timeout.date` 屬性至此 `Date` 物件。例如：若您繫結 `timeout.date` 屬性至 `Date` 物件新增 24 小時至目前時間，使用者互動會在等待 24 小時後逾時。

必要條件

- Add a user interaction element to the workflow schema.
- Set the `security.group` attribute for the user interaction.
- 建立執行指令函式以計算相對時間與日期，並封裝至工作流程中的 `Date` 物件。請參閱計算使用者互動的相對逾時。

程序

1 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 (✏)。

2 按一下使用者互動的 **屬性** 索引標籤。

3 按一下 `timeout.date` 來源參數的 **不設定**，設定逾時參數數值。

4 選擇您在執行指令函式中定義並封存相對時間與日期的 `Date` 物件，並按一下 **選擇**。

您設定 `timeout.date` 屬性為執行指令函式計算的相對日期與時間。

後續步驟

定義使用者互動需從使用者獲得的外部輸入參數。請參閱定義使用者互動的外部輸入。

定義使用者互動的外部輸入

您指定使用者必須在工作流程執行期間提供的資訊，作為使用者互動的輸入參數。

工作流程到達使用者互動元素時，會等待直到使用者提供資訊作為所需的輸入參數為止。

必要條件

- 新增使用者互動元素至工作流程配置。
- Set the `security.group` attribute for the user interaction.
- 設定使用者互動的 `timer.date` 屬性

程序

1. 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 ( )。
2. 按一下 **外部輸入** 索引標籤。
3. 按一下 **繫結至工作流程參數 / 屬性** 圖示 ( )，以定義使用者必須在使用者互動中提供的參數。
4. (選擇性) 若您已定義工作流程中的輸入參數，請由建議清單中選擇參數。
5. 按一下 **在工作流程中建立參數 / 屬性**，建立工作流程屬性並繫結至使用者提供的輸入參數。
6. 賦予參數適當的名稱。
7. 透過搜尋 **篩選器** 方塊中的物件型別，在型別清單中選擇輸入參數型別。

   例如：若使用者互動需要使用者提供虛擬機器作為輸入參數，請選擇 `VC:VirtualMachine`。
8. 選擇 **建立相同名稱的工作流程屬性**，繫結使用者提供為工作流程中新屬性的輸入參數。
9. 維持輸入參數數值設定為 **不設定**。

   使用者在工作流程執行期間回應使用者互動時，將提供此數值。
10. 按一下 **確定** 關閉 **參數資訊** 對話方塊。

您已定義使用者在使用者互動期間提供的輸入參數。

後續步驟

若使用者互動遭遇錯誤，將定義例外狀況行為。請參閱定義使用者互動例外狀況行為。

定義使用者互動例外狀況行為

若使用者不在逾時時間內提供輸入參數，使用者互動會傳回例外狀況。您可在執行指令函式中定義例外狀況行為。

若您未定義工作流程在使用者互動逾時採取的動作，工作流程會以 `Failed` 狀態結束。定義例外狀況行為是一項良好的工作流程開發作法。

必要條件

- 新增使用者互動元素至工作流程配置。
- 設定使用者互動的 `security.group` 及 `timer.date` 屬性。
- 定義使用者互動的外部輸入參數。

程序

1 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 (✏)。

2 按一下**例外狀況**索引標籤。

3 按一下輸出例外狀況繫結的 **不設定**。

4 按一下 **在工作流程中建立參數 / 屬性** ，建立例外狀況屬性並繫結使用者互動。

參數資訊 對話方塊開啟。

5 建立 `errorCode` 屬性。

`errorCode` 屬性含有下列預設內容：

- 名稱：**errorCode**
- 型別：字串
- 建立：**建立相同名稱的工作流程屬性**
- 數值：輸入適當的錯誤訊息。

6 按一下 **確定** 關閉 **參數資訊** 對話方塊。

7 在工作流程配置中的使用者介面元素上拖曳可編寫指令碼的工作元素。

代表例外狀況連結的紅色虛線箭頭出現在兩個元素之間。可編寫指令碼工作元素自動繫結使用者互動中的 `errorCode` 屬性。

8 按兩下可編寫指令碼的工作元素並提供適當的名稱。

例如 **記錄逾時**。

9 在可編寫指令碼工作元素的 **指令碼** 索引標籤中，寫入 JavaScript 函式處理例外狀況。

例如：要記錄 Orchestrator 記錄中的逾時時，請寫入下列函式：

```
System.log("No response from user. Timed out.");
```

10 連結並繫結可編寫指令碼工作元素，處理工作流程中元素的例外狀況。

例如：連結並繫結至可編寫指令碼工作元素至 **擲回例外狀況** 元素，以結束有錯誤的工作流程。

若使用者互動逾時，您已定義例外狀況行為。

後續步驟

建立使用者提供輸入參數的對話方塊。請參閱建立使用者互動的輸入參數對話方塊。

建立使用者互動的輸入參數對話方塊

使用者在工作流程執行期間，於輸入參數對話方塊中提供輸入參數，並透過相同方式在工作流程首次啟動時提供輸入參數。

您在使用者互動元素的**呈現**索引標籤中建立對話方塊配置，而不是在整個工作流程的**呈現**索引標籤中。整個工作流程的 **簡報** 索引標籤會建立輸入參數對話方塊圖層，並在您啟動工作流程時出現。使用者互動元素的 **簡報** 索引標籤會建立輸入參數對話方塊圖層，並於工作流程到達使用者互動元素時開啟。

必要條件

- 新增使用者互動元素至工作流程配置。
- 設定使用者互動的 `security.group` 及 `timer.date` 屬性。
- 定義使用者互動的外部輸入參數。
- 定義例外狀況行為。

程序

1. 在工作流程配置中按一下 **使用者互動** 元素的 **編輯** 圖示 (✏)。
2. 按一下使用者互動元素的 **簡報** 索引標籤。

   簡報 索引標籤顯示您為使用者互動建立的外部輸入參數。
3. (選擇性) 在 **簡報** 索引標籤中的 **簡報** 節點按一下滑鼠右鍵，然後選擇 **建立新步驟**。

   步驟可讓您在對話方塊中建立區段、描述與標頭，讓您可在此組合管理輸入的參數。
4. (選擇性) 在 **簡報** 索引標籤中的 **簡報** 節點按一下滑鼠右鍵，然後選擇 **建立顯示群組**。

   顯示群組可讓您排序步驟中出現輸入參數的順序，並允許您在對話方塊中新增子標頭和指示。
5. 按一下清單中的輸入參數，並在**一般**索引標籤中為該參數新增輸入參數描述。

   您鍵入的描述文字會作為標籤顯示在輸入參數對話方塊中，告知使用者必須於回應使用者互動時提供資訊。
6. 定義輸入參數內容。

   輸入參數內容允許您限定使用者可提供的輸入參數數值，以及透過使用 OGNL 運算式。
7. 按一下**儲存並關閉**，關閉工作流程編輯器。

您已建立供使用者提供輸入參數的輸入參數對話方塊，以便在工作流程執行期間回應使用者互動。

後續步驟

如需有關建立簡報步驟及群組和設定輸入參數內容的資訊，請參閱 在呈現索引標籤中建立輸入參數對話方塊。

回應使用者互動要求

工作流程需要在執行期間進行使用者互動，在使用者提供所需資訊或工作流程逾時之前會一直暫停。

工作流程需要使用者互動定義，讓使用者能提供必要資訊並引導互動要求。

必要條件

確認至少一個工作流程為等待使用者互動狀態。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**執行**。

2 在 Orchestrator 用戶端中按一下**我的 Orchestrator** 視圖。

3 按一下 **等待輸入** 索引標籤。

等待輸入 索引標籤列出等待您或您使用者群組其他有權限的成員使用者輸入的工作流程。

4 按兩下等待輸入的工作流程。

工作流程 Token 正在等待出現在 **工作流程** 階層式清單含有下列符號的輸入：  。

5 在工作流程 Token 上按一下滑鼠右鍵，然後選擇**回應**。

6 依照輸入參數對話方塊中的指示，提供工作流程所需的資訊。

您已提供資訊至執行期間等待使用者輸入的工作流程。

呼叫工作流程內的工作流程

工作流程會在執行期間呼叫其他工作流程。工作流程會因為需要其他工作流程的結果作為自身執行的輸入參數，而啟動另一個工作流程，或是會啟動一個工作流程而繼續獨立運作。工作流程也會在未來的指定時間啟動工作流程，或同時啟動多個工作流程。

- 呼叫工作流程的工作流程元素

  有四個方法可從工作流程中呼叫其他工作流程。每個呼叫一個或多個工作流程的方法均由不同的工作流程架構元素表示。

- 同步呼叫工作流程

  若同步呼叫一個工作流程，則會在執行呼叫工作流程作業期間執行呼叫的工作流程。呼叫工作流程在執行其後續的架構元素時，會使用已呼叫工作流程的輸出參數作為輸入參數。

- 非同步呼叫工作流程

  若以非同步方式呼叫工作流程，則會獨立於呼叫工作流程之外執行呼叫的工作流程。呼叫工作流程會繼續獨自執行，而不會等待完成呼叫的工作流程。

- 排程工作流程

  您可以從工作流程呼叫工作流程，並排程工作流程在稍後的時間和日期啟動。

- 從其他工作流程內部呼叫遠端工作流程的必要條件

  若您開發的工作流程呼叫其他遠端 Orchestrator 伺服器的工作流程，則必須滿足特定必要條件以便順利執行遠端工作流程。

- 同時呼叫數個工作流程

  若同時呼叫數個工作流程，則會在執行呼叫工作流程作業期間同步執行呼叫的工作流程。呼叫工作流程會等待系統完成所有呼叫的工作流程後，再繼續執行。呼叫工作流程在執行其後續的架構元素時，會使用已呼叫工作流程的結果作為輸入參數。

呼叫工作流程的工作流程元素

有四個方法可從工作流程中呼叫其他工作流程。每個呼叫一個或多個工作流程的方法均由不同的工作流程架構元素表示。

同步工作流程

可同步啟動其他工作流程的工作流程。被呼叫的工作流程將與呼叫的工作流程一起執行，並且在呼叫的工作流程所在的同一個記憶體空間中執行。呼叫的工作流程將啟動其他工作流程，然後等待被呼叫的工作流程執行結束，再啟動其架構中的下一個元素。通常，您可同步呼叫工作流程，因為呼叫的工作流程需要被呼叫的工作流程輸出為後續架構元素的輸入參數。例如，工作流程可以呼叫 [啟動虛擬機器並等待] 工作流程以啟動虛擬機器，然後取得此虛擬機器的 IP 位址，以電子郵件傳送給其他元素或使用者。

非同步工作流程

可非同步啟動工作流程的工作流程。呼叫的工作流程會啟動其他工作流程，但是呼叫的工作流程會立即繼續執行其架構中的下一個元素，而不等待被呼叫之工作流程的結果。被呼叫的工作流程會以呼叫的工作流程所定義的輸入參數來執行，但是被呼叫的工作流程生命週期與呼叫的工作流程生命週期無關。非同步工作流程能夠讓您建立一連串的工作流程，將輸入參數從一個工作流程傳遞到下一個工作流程。例如，工作流程可在執行時建立各種物件。工作流程接著即可啟動非同步工作流程，使用這些物件作為本身所執行的輸入參數。原始工作流程啟動全部所需的工作流程並執行剩餘的元素時，就會結束。不過，它啟動的非同步工作流程會繼續執行，而與啟動這些工作流程的工作流程無關。

若要使呼叫的工作流程等待被呼叫之工作流程的結果，可使用巢狀工作流程，或建立可編寫指令碼工作擷取被呼叫之工作流程本身的工作流程 Token 狀態，並於工作流程完成時擷取其結果。

排定的工作流程

工作流程可呼叫工作流程，但是會在稍後的時間和日期前延遲啟動該工作流程。呼叫的工作流程接著將繼續執行，直到結束為止。呼叫排定的工作流程會建立工作，在指定的時間和日期啟動該工作流程。執行呼叫的工作流程時，您可在 Orchestrator 用戶端的**排程器**和**我的 Orchestrator** 視圖中檢視排定的工作流程。

排定的工作流程只會執行一次。您可以在同步工作流程中呼叫可編寫指令碼工作元素中的 `Workflow.scheduleRecurrently` 方法，排程工作流程定期執行。

巢狀工作流程

將多個工作流程放入單一架構元素中成為巢狀時，工作流程即可同時啟動多個工作流程。呼叫的工作流程到達其架構中的巢狀工作流程元素時，巢狀工作流程元素中列出的所有工作流程會同時啟動。尤其是，每個巢狀工作流程會在與呼叫的工作流程所在的不同記憶體空間中啟動。呼叫的工作流程會等待所有巢狀工作流程完成執行，才會啟動其架構中的下一個元素。因此，呼叫的工作流程執行剩餘的元素時，可使用巢狀工作流程的結果作為輸入參數。

將工作流程變更傳播至其他工作流程

如果您呼叫其他工作流程的工作流程，則您將工作流程元素新增至架構時，Orchestrator 將在父系工作流程中匯入子系工作流程的輸入參數。

您將子系工作流程新增至其他工作流程後，如果修改子系工作流程，父系工作流程將呼叫新版本的子系工作流程，但是不會匯入任何新的輸入參數。為了避免工作流程變更而影響呼叫這些工作流程的其他工作流程行為，Orchestrator 不會自動將新的輸入參數傳播至呼叫工作流程。

若要將一個工作流程的參數傳播至呼叫該工作流程的其他工作流程，您必須找出呼叫該工作流程的工作流程，並手動同步化工作流程。

必要條件

確定您有其他一個或多個工作流程呼叫的工作流程。

程序

1 修改並儲存其他工作流程呼叫的工作流程。

2 關閉工作流程編輯器。

3 在 Orchestrator 用戶端**工作流程**視圖的階層清單中，瀏覽至您變更的工作流程。

4 在工作流程上按一下滑鼠右鍵，並選取**參照 > 尋找使用此元素的元素**。

呼叫此工作流程的工作流程清單隨即出現。

5 按兩下清單中的工作流程，在 Orchestrator 用戶端的**工作流程**視圖中反白顯示該工作流程。

6 在工作流程上按一下滑鼠右鍵，然後選取**編輯**。

工作流程編輯器隨即開啟。

7 按一下工作流程編輯器中的**架構**索引標籤。

8 從工作流程架構中，對於變更的工作流程，在工作流程元素上按一下滑鼠右鍵， 並選取**同步化 > 同步化參數**。

9 選取確認對話方塊中的**繼續**。

10 儲存並關閉工作流程編輯器。

11 針對使用修改後工作流程的所有工作流程，重複 步驟 5 至 步驟 10。

您已將變更後的工作流程傳播至呼叫該工作流程的其他工作流程。

傳播子系工作流程的輸入參數與簡報至父系工作流程

若您開發呼叫其他工作流程的工作流程，可傳播子系工作流程的輸入參數與簡報至父系工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**執行**。

2 在要修改的工作流程上按右鍵，然後選擇 **編輯**。

工作流程編輯器開啟。

3 選擇**配置**索引標籤。

4 在要傳播至父系工作流程的輸入參數與簡報子系工作流程元素上按一下右鍵，然後選擇 **同步化 > 同步化簡報**。

5 在確認對話方塊中選擇 **確定**。

6 (選擇性) 在所有要傳播輸入參數與簡報至父系工作流程的子系工作流程上重複 步驟 4 和 步驟 5。

子系工作流程的輸入參數新增至父系工作流程的輸入參數。父系工作流程的簡報透過子系工作流程的簡報擴充。

同步呼叫工作流程

若同步呼叫一個工作流程，則會在執行呼叫工作流程作業期間執行呼叫的工作流程。呼叫工作流程在執行其後續的架構元素時，會使用已呼叫工作流程的輸出參數作為輸入參數。

您可以使用**工作流程**元素，同步呼叫另一個工作流程的工作流程。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 將**工作流程**元素從**一般**功能表拖曳至工作流程架構的適當位置。

選擇工作流程選擇對話方塊出現。

2 搜尋並選取需要的工作流程，然後按一下**確定**。

如果搜尋只傳回一部分的結果，請從用戶端的**工具 > 使用者喜好設定**功能表中，縮小搜尋準則或增加搜尋結果數目。

3 按一下**工作流程**元素，在**架構**索引標籤的下半部顯示元素的內容索引標籤。

4 在工作流程架構中，按一下**工作流程**元素的**編輯**圖示 ( )。

5 在工作流程架構元素的**輸入**索引標籤中，將所需的輸入參數繫結到工作流程。

6 在工作流程架構元素的**輸出**索引標籤中，將所需的輸出參數繫結到工作流程。

7 在**例外狀況**索引標籤中定義工作流程的例外狀況行為。

8 按一下**關閉**。

9 按一下工作流程編輯器底端的**儲存**。

您已同步呼叫另一個工作流程中的工作流程。當工作流程在執行期間接觸到同步工作流程時，同步工作流程會啟動，且初始工作流程會等待其完成後才繼續執行。

後續步驟

您可透過非同步方式，呼叫另一個工作流程中的工作流程。

非同步呼叫工作流程

若以非同步方式呼叫工作流程，則會獨立於呼叫工作流程之外執行呼叫的工作流程。呼叫工作流程會繼續獨自執行，而不會等待完成呼叫的工作流程。

您可以使用**非同步工作流程**元素，以非同步方式呼叫另一個工作流程的工作流程。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1. 將**非同步工作流程**元素從**一般**功能表拖曳至工作流程架構的適當位置。

   選擇工作流程選擇對話方塊出現。
2. 搜尋所需的工作流程並從清單中選取它，然後按一下**確定**。
3. 在工作流程架構中，按一下**非同步工作流程**元素的**編輯**圖示 ( )。
4. 在非同步工作流程元素的**輸入**索引標籤中，將所需的輸入參數繫結到工作流程。
5. 在非同步工作流程元素的**輸出**索引標籤中，繫結所需的輸出參數。

   您可以將輸出參數繫結到呼叫的工作流程，或繫結到該工作流程的結果。
   - 繫結到呼叫的工作流程會傳回該工作流程，以作為輸出參數
   - 繫結到已呼叫工作流程的工作流程 Token，會傳回執行中已呼叫工作流程的結果。
6. 在**例外狀況**索引標籤中，定義非同步工作流程元素的例外狀況行為。
7. 按一下**關閉**。
8. 按一下工作流程編輯器底端的**儲存**。

您已透過非同步方式，呼叫另一個工作流程中的工作流程。當工作流程在執行期間接觸到非同步工作流程時，非同步工作流程便會啟動，初始工作流程會繼續執行而不等待非同步工作流程完成。

後續步驟

您可以排程工作流程在稍後的時間與日期啟動。

排程工作流程

您可以從工作流程呼叫工作流程，並排程工作流程在稍後的時間和日期啟動。

您可以使用**排程工作流程**元素，在其他工作流程中排程工作流程。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 從**一般**功能表將**排程工作流程**元素拖曳至工作流程架構中的適當位置。

2 在文字方塊中輸入工作流程名稱的一部分，搜尋要呼叫的工作流程。

3 從清單中選取工作流程，然後按一下**確定**。

4 在工作流程架構中按一下**排程工作流程**元素的**編輯**圖示 ( )。

5 按一下**輸入**內容索引標籤。

名為 `workflowScheduleDate` 的參數將出現在要定義的內容清單中，其中也有呼叫工作流程的輸入參數。

6 按一下 `workflowScheduleDate` 參數的**未設定**來設定參數。

7 按一下**在工作流程中建立參數/屬性**來建立參數並設定參數值。

8 按一下**值**的**未設定**來設定參數值。

9 使用出現的行事曆來設定啟動排定之工作流程的日期和時間，然後按一下**確定**。

10 在排定之工作流程元素的**輸入**索引標籤中，將其餘輸入參數繫結至排定的工作流程。

11 在排定之工作流程元素的**輸出**索引標籤中，將所需的輸出參數繫結至 `Task` 物件。

12 在**例外狀況**索引標籤中，定義排定之工作流程元素的例外狀況行為。

13 按一下**關閉**。

14 按一下工作流程編輯器底端的**儲存**。

您已從其他工作流程排定工作流程在指定時間和日期啟動。

後續步驟

您可以從工作流程同時呼叫多個工作流程。

從其他工作流程內部呼叫遠端工作流程的必要條件

若您開發的工作流程呼叫其他遠端 Orchestrator 伺服器的工作流程，則必須滿足特定必要條件以便順利執行遠端工作流程。

- 遠端工作流程的所有輸入參數都必須可在遠端 Orchestrator 伺服器上解析。
- 遠端工作流程的所有輸出參數都必須可在本機 Orchestrator 伺服器上解析。

若要確保遠端工作流程的參數可以解析，工作流程使用的詳細目錄物件必須可在遠端和本機 Orchestrator 伺服器上使用。假使遠端工作流程使用來自外掛程式的物件，相同的外掛程式也必須可在 Orchestrator 伺服器上使用。遠端及本機外掛程式的詳細目錄必須完全相同。假使遠端工作流程在 Orchestrator 中使用系統物件，如工作流程和動作，則相同的工作流程與動作必須存在於遠端及本機 Orchestrator 伺服器的詳細目錄上。

例如：假設您在開發測試工作流程的巢狀工作流程元素中插入重新命名虛擬機器工作流程。您想要在遠端 Orchestrator 伺服器中執行重新命名虛擬機器工作流程。當您執行測試工作流程時，將呼叫測試工作流程內的重新命名虛擬機器工作流程。您在本機 Orchestrator 伺服器的詳細目錄中指定重新命名的虛擬機器。由於重新命名虛擬機器工作流程在遠端 Orchestrator 伺服器上執行，所以必須可在該伺服器的詳細目錄中使用相同的虛擬機器。否則，重新命名虛擬機器工作流程無法解析其 `vm` 輸入參數。因此，本機和遠端 Orchestrator 伺服器上的 vCenter Server 外掛程式必須連接至相同的 vCenter Server 執行個體。

同時呼叫數個工作流程

若同時呼叫數個工作流程，則會在執行呼叫工作流程作業期間同步執行呼叫的工作流程。呼叫工作流程會等待系統完成所有呼叫的工作流程後，再繼續執行。呼叫工作流程在執行其後續的架構元素時，會使用已呼叫工作流程的結果作為輸入參數。

您可以使用**巢狀工作流程**元素，同時呼叫另一個工作流程的數個工作流程。您可以使用巢狀工作流程，透過與呼叫工作流程使用者認證不同的使用者認證，來執行工作流程。

必要條件

- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1. 將**巢狀工作流程**元素從**動作與工作流程**功能表拖曳至工作流程架構的適當位置。

   選擇工作流程選擇對話方塊出現。

2. 搜尋並選取要啟動的工作流程，然後按一下**確定**。

3. 在工作流程架構中按一下**巢狀工作流程**元素的**編輯**圖示 (✎)。

4. 按一下**工作流程**索引標籤。

   您在 步驟 2 中選取的工作流程會顯示在索引標籤中。

5. 在**工作流程**架構元素內容其右面板的**輸入**與**輸出**索引標籤中，設定此工作流程的輸入與輸出繫結。

6. 在**工作流程**架構元素內容索引標籤的右面板中，按一下**連線資訊**索引標籤。

   連線資訊索引標籤可讓您使用適當的認證，存取在不同伺服器中儲存到本機伺服器的工作流程。

7. 若要存取遠端伺服器上的工作流程，請選取**遠端**，然後按一下**未設定**，以提供遠端伺服器的主機名稱或 IP 位址。

   備註 您可以使用 vRealize Orchestrator Multi-Node 外掛程式呼叫遠端伺服器上的工作流程。

8. 定義用來存取遠端伺服器的認證。

   - 選取**繼承**以使用與執行呼叫工作流程的使用者相同的認證。
   - 選取**動態**，然後按一下**未設定**以選取 `credentials` 類型的參數在工作流程其他位置中定義的一組動態認證。
   - 選取**靜態**，然後按一下**未設定**直接輸入認證。

9 按一下**工作流程**索引標籤中的**新增工作流程**按鈕，以選取多個要新增至巢狀工作流程元素的工作流程。

10 重複 步驟 2 到 步驟 8，為您新增的每個工作流程定義設定。

11 在工作流程架構中按一下巢狀工作流程元素。

嵌套在元素中的工作流程數目，會顯示成巢狀工作流程元素上的數字。

您已同時呼叫工作流程中的數個工作流程。

後續步驟

您可以定義長時間執行的工作流程。

在物件選項上執行工作流程

您可在物件選項上執行工作流程，自動化重複工作。例如：您可建立在虛擬機器資料夾中擷取所有虛擬機器快照的工作流程，或您可建立關閉指定主機上的所有虛擬機器電源的工作流程。

您可使用下列方式之一，在物件選項上執行工作流程。

- 執行 **程式庫 > vCenter > 批次 > 在物件選項上執行工作流程** 工作流程
- 建立呼叫 **程式庫 > Orchestrator > 依序開始工作流程** 或 **同時開始工作流程的** 工作流程。
- 建立取得大量物件的工作流程，並在工作流程元素迴路陣列中執行各物件的工作流程。
- 在工作流程的執行指令元素呼叫 `For` 迴路中的 `Workflow.execute()` 方式，執行 JavaScript 中的工作流程。

您選擇在物件選項上執行工作流程的方式視執行的工作流程而定，並會影響工作流程的效能。例如：執行在物件選項上執行工作流程的工作流程是在多物件上執行工作流程最簡單的方式，不需要任何工作流程開發，但它只會執行取得單一輸入參數的工作流程。

建立呼叫依序開始工作流程或同時開始工作流程的工作流程，能讓您在取得超過一組輸入參數的多物件工作流程上執行。呼叫工作流程必須建立內容陣列，以傳遞輸入參數至依序開始工作流程或同時開始工作流程。這些工作流程只適用於其他工作流程。請勿直接執行。

在執行指令元素中的 `For` 迴路執行速度比在工作流程元素迴路中執行工作流程還快，但彈性較低且會限制重複使用的可能性。最重要的是，在執行指令迴路中執行工作流程會在工作流程中開始各元素時損失 Orchestrator 執行的檢查點檢查。最後，若在執行指令迴路期間 Orchestrator 伺服器於伺服器重新啟動時停止，則工作流程將在開始執行指令元素時恢復，並重複執行整個迴路。若 Orchestrator 伺服器在執行含有工作流程元素迴路的工作流程時停止，則該工作流程將在伺服器停止時於執行特定的迴路元素中恢復。

如需有關批次工作流程的詳細資訊，請《參閱 使用 VMware vRealize Orchestrator 外掛程式》。

在 開發複雜工作流程 中將示範如何於工作流程元素迴路中，建立可在物件陣列上執行工作流程的工作流程。

在 工作流程指令碼範例 中將展示如何於執行指令 `For` 迴路中執行工作流程。

執行依序開始工作流程及同時開始工作流程

您可使用依序開始工作流程與同時開始工作流程，針對選擇物件執行工作流程。

您無法直接執行依序開始工作流程及同時開始工作流程。您必須在建立的其他工作流程中加入。若要使用依序開始工作流程和同時開始工作流程執行選擇物件的工作流程，您必須取得執行工作流程的物件。您可傳遞這些工作流程需要的物件以及其他輸入參數至工作流程作為內容陣列。依序開始工作流程及同時開始工作流程會產生執行選擇物件工作流程的結果並當成 `WorkflowToken` 物件陣列。

您以相同方式執行依序開始工作流程及同時開始工作流程。依序開始工作流程會逐一執行各物件的工作流程。同時開始工作流程會同時執行所有物件的工作流程。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在工作流程配置中，新增可編寫指令碼工作元素或動作，取得執行工作流程的物件清單。

例如：若要在虛擬機器資料夾中執行所有虛擬機器的工作流程，您可新增 `getAllVirtualMachinesByFolder` 動作至工作流程。

2 連結執行指令元素或動作，並繫結執行指令元素或動作的輸入與輸出至工作流程輸入或屬性。

例如：您可繫結 `getAllVirtualMachinesByFolder` 動作的 `vmFolder` 輸入至工作流程輸入參數，並將 `actionResult` 輸出繫結至呼叫工作流程中的工作流程屬性。

3 新增可編寫指令碼工作元素，以轉換物件清單至內容陣列。

例如：若執行工作流程的物件屬於由 `getAllVirtualMachinesByFolder` 動作 `actionResult` 輸出傳回的虛擬機器陣列 `allVMs`，則您可撰寫下列指令碼，將物件轉換為內容陣列。

```
propsArray = new Array();

for each (var vm in allVMs) {
 var prop = new Properties();
 prop.put("vm", vm);
 propsArray.push(prop);
}
```

4 繫結可編寫指令碼工作元素的輸入與輸出至工作流程屬性。

在 步驟 3 的可編寫指令碼工作元素範例中，您需繫結輸入至虛擬機器的 `allVMs` 陣列及建立 `propsArray` 輸出屬性作為 `Properties` 物件的陣列。

5 新增工作流程元素至工作流程配置。

6 選擇依序開始工作流程或同時開始工作流程，並連結工作流程元素至其他元素。

7 繫結依序開始工作流程或同時開始工作流程的 `wf` 輸入至工作流程，以執行物件。

例如：要使用 `getAllVirtualMachinesByFolder` 動作移除所有虛擬機器傳回的任何快照，請選擇移除所有快照工作流程。

8 繫結依序開始工作流程或同時開始工作流程的 `parameters` 輸入至包含執行工作流程物件的大量 `Properties` 物件。

例如：繫結 `parameters` 輸入至 步驟 4 中定義的 `propsArray` 屬性。

9 (選擇性) 繫結依序開始工作流程或同時開始工作流程的 `workflowTokens` 輸出至工作流程中的屬性。

10 (選擇性) 繼續新增更多使用執行依序開始工作流程或同時開始工作流程的元素。

您已建立使用依序開始工作流程或同時開始工作流程的工作流程，以執行選擇物件的工作流程。

開發長時間執行的工作流程

處於等待狀態的工作流程會耗用系統資源，因為它會不斷輪詢需要進行回應的物件。如果您知道工作流程可能要等待很長的時間才會收到所需的回應，則您可將長時間執行的工作流程元素新增至工作流程。

每個執行中工作流程都會耗用系統執行緒。當工作流程接觸到長時間執行的工作流程元素時，長時間執行的工作流程會將工作流程設定為被動狀態。長時間執行的工作流程元素接著會將工作流程資訊傳遞至單一執行緒，以輪詢系統中所有正在在伺服器中執行之長時間執行的工作流程元素。不是每個長時間執行的工作流程元素都會不斷嘗試從系統擷取資訊，而是長時間執行的工作流程元素在設定的持續期間會保持為被動，在此同時，長時間執行的工作流程執行緒會代表該元素來輪詢系統。

您可以使用下列其中一種方式設定等待的持續期間：

- 設定封裝在 `Date` 物件中的計時器，其在達到某特定時間與日期之前將暫停工作流程。您透過在架構中包含**等待計時器**元素的方式，來實作以計時器為基礎之長時間執行的工作流程元素。
- 定義封裝在 `Trigger` 物件中的觸發器事件，藉此在發生觸發器事件後重新啟動工作流程。您透過在架構中新增**等待事件**元素或**使用者互動**元素的方式，來實作以觸發器為基礎之長時間執行的工作流程元素。

設定以計時器為基礎之工作流程的相對時間與日期

您可藉由將等待計時器元素繫結至 `Date` 物件，來設定其 `timer.date` 屬性的相對時間與日期。您可在執行指令碼式函數中定義 `Date` 物件。

到達指定日期的時間時，長時間執行且以計時器為基礎的工作流程將重新啟動並繼續執行。例如，您可以設定工作流程在 2 月 12 日中午重新啟動。此外，您可依您定義的函式建立計算及產生相對 `Date` 物件的工作流程元素。例如：您可建立相對 `Date` 物件，以新增 24 小時至目前時間。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 先從**一般**功能表將**可編寫指令碼工作**元素拖曳至工作流程的架構，放在需要其 `timeout.date` 屬性之相關 `Date` 物件的元素之前。

2 在工作流程架構中，按一下**可編寫指令碼工作**元素的**編輯**圖示 ( )。

3 在**資訊**內容索引標籤中，提供指令碼式工作流程元素的名稱和說明。

4 按一下**輸出**內容索引標籤，然後按一下**繫結至工作流程參數/屬性** 圖示 ( )。

5 按一下**在工作流程中建立參數/屬性**，建立工作流程屬性。

   a 將屬性命名為 `timerDate`。

   b 從屬性類型的清單中選取 `Date`。

   c 選取**建立相同名稱的工作流程屬性**。

   d 將屬性值維持設定為**未設定**，因為指令碼式函數將提供此值。

   e 按一下**確定**。

6 按一下指令碼式工作流程元素的**指令碼**索引標籤。

7 在**指令碼**索引標籤的指令碼輸入台中，定義函數以計算並產生名為 `timerDate` 的 `Date` 物件。

   例如，您可以實作下列 JavaScript 函數來建立 `Date` 物件，其中的逾時時間是相對延遲毫秒數。

```
timerDate = new Date();
System.log( "Current date : '" + timerDate + "'" );
timerDate.setTime( timerDate.getTime() + (86400 * 1000) );
System.log( "Timer will expire at '" + timerDate + "'" );
```

   前一個範例 JavaScript 函數使用 `getTime` 方法來定義包含目前日期與時間的 `Date` 物件，並新增 86,400,000 毫秒，也就是 24 小時。**可編寫指令碼工作**元素將產生此值作為其輸出參數。

8 按一下**關閉**。

9 按一下**儲存**。

您已建立計算和產生 `Date` 物件的函數。**等待計時器**元素可接收此 `Date` 物件作為輸入參數，以暫停長時間執行的工作流程，直到封存在此物件中的日期為止。工作流程到達**等待計時器**元素時，會暫停執行並等待 24 小時再繼續。

後續步驟

您必須將**等待計時器**元素新增至工作流程，才能實作長時間執行且以計時器為基礎的工作流程。

建立以計時器為基礎之長時間執行的工作流程

如果您知道工作流程將必須等待外部來源的回應達可預測時間，您便可將它實作為以計時器為基礎之長時間執行的工作流程。以計時器為基礎之長時間執行的工作流程會等待達到指定的時間與日期之後才繼續執行。

您可使用**等待計時器**元素，將工作流程實作為以計時器為基礎之長時間執行的工作流程。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1 將**等待計時器**元素從**一般**功能表拖曳至工作流程架構中您要暫停工作流程執行的位置。

如果您實作可編寫指令碼工作以計算時間與日期，此元素必須在**等待計時器**元素之前。

2 在工作流程架構中，按一下**等待計時器**元素的**編輯**圖示 ( )。

3 在**資訊**內容索引標籤中說明實作計時器的原因。

4 按一下**屬性**內容索引標籤。

`timer.date` 參數會出現在屬性清單中。

5 按一下 `timer.date` 參數的**未設定**按鈕，將參數繫結至適當的 `Date` 物件。

等待計時器選取對話方塊隨即開啟，顯示可能繫結的清單。

- 從建議清單中選取預先定義的 `Date` 物件，例如由**可編寫指令碼工作**元素在工作流程其他處定義的物件。
- 或者，建立 `Date` 物件，用以設定工作流程要等待的特定日期與時間。

6 (選擇性) 建立 `Date` 物件，用以設定工作流程要等待的特定日期與時間。

a 在**等待計時器**選取對話方塊中按一下**在工作流程中建立參數/屬性**。

參數資訊對話方塊隨即顯示。

b 賦予參數適當的名稱。

c 讓類型設為 `Date`。

d 按一下**建立相同名稱的工作流程屬性**。

e 按一下**值**內容的**未設定**按鈕，以設定參數值。

行事曆隨即顯示。

f 使用行事曆設定重新啟動工作流程的日期與時間。

g 按一下**確定**。

7 按一下**關閉**。

8 按一下工作流程編輯器底端的**儲存**。

您已定義計時器，其在到達設定的時間與日期之前將暫停以計時器為基礎之長時間執行的工作流程。

後續步驟

您可以建立長時間執行的工作流程等待觸發器事件，再繼續進行。

建立觸發器物件

觸發器物件會監視外掛程式定義的事件觸發器。例如，vCenter Server 外掛程式會將這些事件定義為 `Task` 物件。工作結束時，觸發器會將訊息傳送至以等待觸發器為基礎之長時間執行的工作流程元素，以重新啟動工作流程。

以觸發器為基礎之長時間執行的工作流程所等待的耗時事件，必須傳回 VC:Task 物件。例如，啟動虛擬機器的 startVM 動作會傳回 VC:Task 物件，讓工作流程中的後續元素監視其進度。以觸發器為基礎之長時間執行工作流程的觸發器事件，需要此 VC:Task 物件作為輸入參數。

您在**可編寫指令碼工作**元素中以 JavaScript 函數建立 Trigger 物件。此**可編寫指令碼工作**元素屬於會等待觸發器事件的以觸發器為基礎之長時間執行的工作流程。或者，屬於將輸入參數提供給以觸發器為基礎之長時間執行的工作流程的不同工作流程。觸發器函數必須從 Orchestrator API 實作 createEndOfTaskTrigger() 方法。

重要 您必須為所有觸發器定義逾時時間，否則工作流程會無限期等待。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。
- 在工作流程中，將 VC:Task 物件宣告為屬性或輸入參數，例如一或多個啟動或複製虛擬機器之工作流程元素中的 VC:Task 物件。

程序

1 將**可編寫指令碼工作**元素從**一般**功能表拖曳至工作流程的架構。

在**可編寫指令碼工作**之前的其中一個元素必須產生 VC:Task 物件作為其輸出參數。

2 在工作流程架構中，按一下**可編寫指令碼工作**元素的**編輯**圖示 ( )。

3 在**資訊**內容索引標籤中提供觸發器的名稱與說明。

4 按一下**輸入**內容索引標籤。

5 按一下**繫結至工作流程參數/屬性**圖示 ( )。

輸入參數選取對話方塊隨即開啟。

6 選取或建立 **VC:Task** 類型的輸入參數。

此 VC:Task 物件代表另一個工作流程或元素啟動的耗時事件。

7 (選擇性) 選取或建立「數字」類型的輸入參數，以定義逾時時間秒數。

8 按一下**輸出**內容索引標籤。

9 按一下**繫結至工作流程參數/屬性**圖示 ( )。

輸出參數選取對話方塊隨即開啟。

10 建立一個內容如下的輸出參數。

a 建立值為 trigger 的「名稱」內容。

b 建立值為 Trigger 的「類型」內容。

c 按一下**建立相同名稱的屬性**以建立屬性。

d 將值保留為**未設定**。

11 在**例外狀況**內容索引標籤中定義任何例外狀況行為。

12 在**指令碼**索引標籤中定義要產生 `Trigger` 物件的函數。

例如，您可以實作下列 JavaScript 函數來建立 `Trigger` 物件。

```
trigger = task.createEndOfTaskTrigger(timeout);
```

`createEndOfTaskTrigger()` 方法會傳回 `Trigger` 物件以監視名為 `task` 的 `VC:Task` 物件。

13 按一下**關閉**。

14 按一下工作流程編輯器底端的**儲存**。

您定義了一個工作流程元素，此元素可為以觸發器為基礎之長時間執行的工作流程建立觸發器事件。觸發器元素會產生 `Trigger` 物件作為其輸出參數，**等待事件**元素可繫結至該參數。

後續步驟

您必須在以觸發器為基礎之長時間執行的工作流程中，將此觸發器事件繫結至**等待事件**元素。

建立以觸發器為基礎之長時間執行的工作流程

如果您知道工作流程在執行期間將必須等待外部來源的回應，但不知道等待會持續多久的時間，您可以將工作流程實作為以觸發器為基礎之長時間執行的工作流程。以觸發器為基礎之長時間執行的工作流程會等待所定義的觸發器事件發生後，才會繼續進行。

您可使用**等待事件**元素，將工作流程實作為以觸發器為基礎之長時間執行的工作流程。當以觸發器為基礎之長時間執行的工作流程到達**等待事件**元素時，會暫停執行並在被動狀態中等待，直到收到來自觸發器的訊息為止。等待期間，被動工作流程不會耗用執行緒，但長時間執行的工作流程元素會將工作流程資訊，傳遞至監視伺服器中所有長時間執行之工作流程的單一執行緒。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。
- 定義封裝在 `Trigger` 物件中的觸發器事件。

程序

1 將**等待事件**元素從**一般**功能表拖曳至工作流程架構中您要暫停執行工作流程的位置。

宣告觸發器的可編寫指令碼工作必須緊接在**等待事件**元素之前。

2 在工作流程架構中，按一下**等待事件**元素的**編輯**圖示 (✏)。

3 在**資訊**內容索引標籤中說明等待的原因。

4 按一下**屬性**內容索引標籤。

`trigger.ref` 參數會出現在屬性清單中。

5 按一下 `trigger.ref` 參數的**未設定**連結，將參數繫結至適當的 `Trigger` 物件。

等待事件選取對話方塊隨即開啟，顯示所要繫結之可能參數的清單。

6 從建議清單中選取預先定義的 `Trigger` 物件。

此 `Trigger` 物件代表另一個工作流程或工作流程元素定義的觸發器事件。

7 在**例外狀況**內容索引標籤中定義任何例外狀況行為。

8 按一下**關閉**。

9 按一下工作流程編輯器底端的**儲存**。

您已定義一個工作流程元素，可暫停以觸發器為基礎之長時間執行的工作流程，此工作流程會先等待特定觸發器事件，再重新啟動。

後續步驟

您可以執行工作流程。

組態元素

組態元素是一種屬性清單，包含可用來在整個 Orchestrator 伺服器部署間設定常數的屬性。

在特定 Orchestrator 伺服器中執行的所有工作流程、動作與原則，皆會使用您在組態元素中設定的屬性。在組態元素中設定屬性，可提供相同屬性值給在 Orchestrator 伺服器執行的所有工作流程、動作與原則使用。

如果您建立一個套件，且其中包含的工作流程、動作或原則使用某個組態元素中的屬性，則 Orchestrator 會自動在套件中包含該組態元素。如果您將包含組態元素的套件匯入到另一個 Orchestrator 伺服器，您也可以匯入組態元素屬性。例如，如果您建立一個需要屬性值的工作流程，且該值取決於執行工作流程所在的 Orchestrator 伺服器，則在組態元素中設定這些屬性可讓您匯出該工作流程讓另一個 Orchestrator 伺服器使用。因此組態元素可讓您更輕鬆地在伺服器之間交換工作流程、動作與原則。

備註 您無法從自 Orchestrator 5.1 或更舊版本匯出的組態元素，匯入組態元素屬性的值。

建立組態元素

組態元素允許您針對整個 Orchestrator 伺服器之間設定一般屬性。在伺服器中執行的所有元素會呼叫您在組態元素中設定的屬性。建立組態元素讓您僅須在在伺服器中定義一次一般屬性，而無須在每個元素中個別定義。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**組態**視圖。

3 在資料夾階層清單中的資料夾上按一下滑鼠右鍵，然後選取**新增資料夾**建立新的資料夾。

4 提供資料夾名稱，然後按一下**確定**。

5 在您建立的資料夾上按一下滑鼠右鍵，然後選取**新增元素**。

6 提供組態元素名稱，然後按一下**確定**。

組態元素編輯器隨即開啟。

7 按一下**一般**索引標籤中的版本數字並提供版本註解，以增加版本號碼。

8 在**一般**索引標籤的**說明**文字方塊中，提供組態元素的說明。

9 按一下**屬性**索引標籤。

10 按一下**新增屬性**圖示 () 以建立新屬性。

11 按一下**名稱**、**類型**、**值**與**說明**下方的屬性值，以設定屬性名稱、類型、值與說明。

12 按一下**權限**索引標籤。

13 按一下**新增存取權限**圖示 ()，將可存取此組態元素的權限授與使用者群組。

14 在**篩選器**文字方塊中搜尋使用者群組，然後從建議清單中選取相關的使用者群組。

15 選取適當的核取方塊，為選取的使用者群組設定存取權。

您可以在組態元素上設定下列權限。

權限	說明
檢視	使用者可以檢視組態元素，但無法檢視架構或指令碼。
檢查	使用者可以檢視組態元素，包含架構與指令碼。
管理員	使用者可在組態元素中的元素上設定權限，並具有所有其他權限。
執行	使用者可執行組態元素中的元素。
編輯	使用者可編輯組態元素的元素。

16 按一下**選取**。

17 按一下**儲存並關閉**結束組態元素編輯器。

您已定義一個組態元素，可針對整個 Orchestrator 伺服器設定一般屬性。

後續步驟

您可使用組態元素，以提供屬性給工作流程或動作。

工作流程使用者權限

Orchestrator 會定義您可套用至群組的權限層級，以便允許或拒絕群組存取工作流程。

檢視 使用者可檢視工作流程中的元素，但是無法檢視架構或指令碼。

檢查 使用者可檢視工作流程中的元素，包括架構和指令碼。

執行	使用者可執行工作流程。
編輯	使用者可編輯工作流程。
管理員	使用者可設定工作流程的權限，並擁有所有其他權限。

管理員權限包括**檢視**、**檢查**、**編輯**和**執行**權限。所有權限都需要**檢視**權限。

如果您沒有設定工作流程的任何權限，工作流程將繼承包含工作流程之資料夾本身的權限。如果您設定工作流程的權限，即使資料夾的權限更具限制，這些權限也會覆寫包含工作流程之資料夾本身的權限。

設定工作流程的使用者權限

您可以設定工作流程的權限層級，以便限制使用者群組對於該工作流程所擁有的存取權。

您可以從 Orchestrator LDAP 伺服器選取要設定權限的使用者和使用者群組。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 新增一些元素至工作流程配置。

程序

1. 按一下**權限**索引標籤。
2. 按一下**新增存取權限**圖示 () 即可定義新使用者群組的權限。
3. 搜尋使用者群組。

   搜尋結果將包含 Orchestrator LDAP 伺服器中符合搜尋的所有使用者群組。
4. 選取使用者群組，並選取適當的核取方塊來設定此使用者群組的權限層級。

   若要允許此使用者群組的使用者檢視工作流程、檢查架構和指令碼、執行和編輯工作流程，並變更權限，您必須選取所有的核取方塊。
5. 按一下**選取**。

   使用者群組將出現在權限清單中。
6. 按一下**儲存並關閉**結束編輯器。

驗證工作流程

Orchestrator 提供工作流程驗證工具。驗證工作流程有助於識別工作流程中的錯誤，並檢查資料是否從一個元素正確傳送到另一個元素。

您驗證工作流程時，驗證工具會建立任何錯誤或警告的清單。檢查清單中的錯誤可找出包含錯誤的工作流程元素。

如果您在工作流程編輯器中執行驗證工具，工具將針對偵測到的錯誤提供建議的快速修正。部分快速修正會要求您提供額外資訊或輸入參數。其他快速修正則會為您解決錯誤。

工作流程驗證會檢查資料繫結和元素之間的連線。工作流程驗證不會檢查工作流程中的每個元素所執行的資料處理。因此，有效的工作流程會錯誤執行，而在架構元素中的函數不正確時產生錯誤的結果。

依預設，Orchestrator 一律在您執行工作流程時執行工作流程驗證。您可在 Orchestrator 用戶端中變更預設驗證行為。請參閱在開發期間測試工作流程。例如，在工作流程開發期間，您可能會想要執行您知道的無效工作流程來進行測試。

驗證工作流程與修正驗證錯誤

您必須先驗證工作流程，才能執行工作流程。您可以在 Orchestrator 用戶端或工作流程編輯器中驗證工作流程。不過，只有在您開啟工作流程以在工作流程編輯器中編輯時，才能修正驗證錯誤。

必要條件

確定您有可驗證的完整工作流程，其中已連結架構元素，而且已定義繫結。

程序

1 按一下**工作流程**視圖。

2 瀏覽至**工作流程**階層清單中的工作流程。

3 (選擇性) 在工作流程上按一下滑鼠右鍵，然後選取**驗證工作流程**。

如果工作流程有效，將出現確認訊息。如果工作流程無效，則會出現錯誤的清單。

4 (選擇性) 關閉 [工作流程驗證] 對話方塊。

5 在工作流程上按一下滑鼠右鍵，然後選取**編輯**開啟工作流程編輯器。

6 按一下**架構**索引標籤。

7 按一下**架構**索引標籤工具列中的**驗證**按鈕。

如果工作流程有效，將出現確認訊息。如果工作流程無效，則會出現錯誤的清單。

8 針對無效的工作流程，請按一下錯誤訊息。

驗證工具會為工作流程新增紅色圖示，指出發生錯誤的架構元素。驗證工具會盡可能顯示快速修正動作。

- 如果您同意建議的快速修正動作，請按一下工作流程來執行該動作。
- 如果您不同意建議快速修正動作，請關閉 [工作流程驗證] 對話方塊並手動修正架構元素。

重要 務必檢查 Orchestrator 建議的修正是否適當。

例如，建議的動作可能是刪除未使用的屬性，不過，實際上並未將屬性正確繫結。

9 重複前述步驟，直到您消除所有的驗證錯誤為止。

此時您即已驗證工作流程並修正驗證錯誤。

後續步驟

您可以執行工作流程。

偵錯工作流程

Orchestrator 提供工作流程偵錯工具。您可偵錯工作流程以在任何活動開始時，於工作流程執行期間的編輯模式檢查輸入與輸出參數和屬性、取代參數或屬性數值，以及從上次失敗的活動中恢復工作流程。

您可偵錯標準工作流程程式庫及自訂工作流程中的工作流程。您可同時偵錯自訂工作流程，並在工作流程編輯器中開發。

偵錯工作流程

您可透過新增中斷點至工作流程配置中的元素，偵錯工作流程元素。

到達中斷點後，您有多個選項可選擇繼續偵錯程序。在偵錯工作流程配置中的元素時，您可檢視有關工作流程執行、修改工作流程變數及檢視記錄訊息的一般資訊。

必要條件

以可執行工作流程的使用者身分登入 Orchestrator 用戶端。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 從工作流程程式庫中選擇工作流程，然後按一下 **配置** 索引標籤。

4 若要新增中斷點至您要偵錯的配置元素，請在工作流程元素上按一下滑鼠右鍵，然後選擇 **切換中斷點**。

   您可啟用或停用切換的中斷點。

5 按一下 **偵錯工作流程** 圖示 ( )。

   若工作流程需要輸入參數，您必須提供。

6 到達中斷點後暫停工作流程執行時，請選擇其中一個可用選項。

選項	說明
繼續執行	恢復工作流程執行，直到到達其他中斷點。
逐步執行	讓您逐步執行工作流程元素。 **備註** 您無法在偵錯工作流程編輯器中的工作流程時，逐步執行巢狀工作流程元素。
跨步執行	跨步執行配置中目前的元素，並暫停下一個元素的工作流程執行。
返回執行	結束您已逐步執行的工作流程元素。

7 (選擇性) 在 **中斷點** 索引標籤中，修改中斷點。

   您可啟用、停用或移除現有中斷點。

8 (選擇性) 在 **變數** 索引標籤中檢閱變數。

您可在偵錯過程期間修改部分變數的數值。

工作流程偵錯範例

您可由標準工作流程程式庫中偵錯工作流程。

例如：若您提供不正確的收件人地址，可在偵錯範例與電子郵件工作流程互動時修正數值。

必要條件

以可執行郵件工作流程的使用者身分登入 Orchestrator 用戶端。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 在工作流程階層式清單中，開啟 **程式庫 > 郵件**。

4 選擇與電子郵件工作流程範例互動，然後按一下 **配置** 索引標籤。

5 在 **傳送電子郵件 (互動)** 工作流程元素上按右鍵，然後選擇 **切換中斷點**。

6 按一下 **偵錯工作流程** 圖示 ( )。

7 提供必要資訊。

a 在 **目的地地址** 文字方塊中，鍵入不完整的收件人地址。

例如：*名稱@公司.c*。

b 選擇獲回答查詢授權的使用者 LDAP 群組。

c 按一下**提交**。

8 到達中斷點後，按一下 **逐步執行** 圖示 ( )。

9 在 **變數** 索引標籤上確認數值。

10 在 **toAddress** 文字方塊中鍵入正確的收件人地址數值。

例如：*名稱@公司.com*。

11 按一下 **恢復** 圖示 ( ) ，繼續執行工作流程。

在偵錯過程期間的工作流程使用您提供的數值，並持續執行工作流程。

正在執行工作流程

Orchestrator 工作流程會按照事件的邏輯流程執行。

您執行工作流程時，工作流程中的每個架構元素將按照下列順序執行。

1 工作流程會將工作流程 Token 屬性和輸入參數繫結至架構元素的輸入參數。

2 架構元素隨即執行。

3 架構元素的輸出參數將被複製到工作流程 Token 屬性和工作流程輸出參數。

4 工作流程 Token 屬性和輸出參數將儲存於資料庫。

5 下一個架構元素將開始執行。

針對各個架構元素重複此順序，直到工作流程結束為止。

工作流程 Token 檢查點

執行工作流程時，各個架構元素是檢查點。執行每個架構元素後，Orchestrator 會將工作流程 Token 屬性儲存於資料庫，接著下一個架構元素將開始執行。如果工作流程意外停止，則下次 Orchestrator 伺服器重新啟動時，目前作用中的架構元素將再次執行，而工作流程將從中斷時執行的架構元素開始繼續執行。不過，Orchestrator 不會實作交易管理或復原函數。

工作流程結束

如果目前作用中的架構元素是結束元素，工作流程將結束。工作流程到達結束元素後，其他工作流程或應用程式可使用工作流程的輸出參數。

在工作流程編輯器中執行工作流程

您可在執行工作流程的同時開發。

在工作流程編輯器中執行工作流程，能讓您確認該工作流程是否能在不中斷開發程序的情況下正確執行。您可檢視提供有關工作流程執行資訊的記錄訊息。若工作流程傳回非預期結果，您可再次修改工作流程並於不關閉工作流程編輯器的情況下重新執行。

必要條件

- 建立工作流程。
- 在工作流程編輯器中開啟工作流程以進行編輯。
- 驗證工作流程。

程序

1 按一下**配置**索引標籤。

2 按一下**執行**。

3 (選擇性) 檢閱 **記錄** 索引標籤中的訊息。

執行工作流程

您可以從標準程式庫或您建立的工作流程執行工作流程，以便執行 vCenter Server 中的自動作業。

例如，您可以執行 [建立簡單虛擬機器] 工作流程，以建立虛擬機器。

必要條件

確定您已經設定 vCenter Server 外掛程式。如需詳細資料，請參閱《安裝和設定 VMware vCenter Orchestrator》。

程序

1 從 Orchestrator 用戶端的下拉式功能表中，選取**執行**或**設計**。

2 按一下**工作流程**視圖。

3 在工作流程階層清單中，開啟**程式庫 > vCenter > 虛擬機器管理 > 基本** 以瀏覽至 [建立簡單虛擬機器] 工作流程。

4 在 [建立簡單虛擬機器] 工作流程上按一下滑鼠右鍵，然後選取**啟動工作流程**。

5 在**啟動工作流程**輸入參數對話方塊中提供下列資訊，以在連線到 Orchestrator 的 vCenter Server 中建立虛擬機器。

選項	動作
虛擬機器名稱	將虛擬機器命名為 `orchestrator-test`。
虛擬機器資料夾	a 針對**虛擬機器資料夾**值，按一下**未設定**。 b 從詳細目錄中選取虛擬機器資料夾。 **選取**按鈕將停用，直到您選取正確類型的物件為止，在此案例中為 `VC:VmFolder`。
新磁碟的大小 (GB)	輸入適當的數值。
記憶體大小 (MB)	輸入適當的數值。
虛擬 CPU 數	從**虛擬 CPU 數**下拉式功能表中，選取適當的 CPU 數。
虛擬機器客體作業系統	按一下**未設定**連結，並且從清單中選取客體作業系統。
將建立虛擬機器的主機	針對**將建立虛擬機器的主機**值，按一下**未設定**，並瀏覽 vCenter Server 基礎結構階層找出主機。
資源集區	針對**資源集區**值，按一下**未設定**，並瀏覽 vCenter Server 基礎結構階層找出資源集區。
要連線的網路	針對**要連線的網路**值，按一下**未設定**，並選取網路。 在**篩選器**文字方塊中按 Enter，查看所有可用的網路。
要儲存虛擬機器檔案的資料存放區	針對**要儲存虛擬機器的資料存放區**值，按一下**未設定**，並瀏覽 vCenter Server 基礎結構階層找出資料存放區。

6 按一下**提交**執行工作流程。

工作流程 Token 將出現在 [建立簡單虛擬機器] 工作流程下，顯示工作流程執行圖示。

7 按一下工作流程 Token 以檢視執行的工作流程狀態。

8 按一下工作流程 Token 視圖中的**事件**索引標籤，以追循工作流程 Token 的進度，直到完成為止。

9 按一下**詳細目錄**視圖。

10 瀏覽 vCenter Server 基礎結構階層找出您定義的資源集區。

如果虛擬機器未出現在清單中，按一下重新整理按鈕即可重新載入詳細目錄。

`orchestrator-test` 虛擬機器將出現在資源集區中。

11 (選擇性) 在**詳細目錄**視圖中的 `orchestrator-test` 虛擬機器上按一下滑鼠右鍵，查看內容清單，其中列出您可以在 `orchestrator-test` 虛擬機器上執行的工作流程。

[建立簡單虛擬機器] 工作流程已成功執行。

後續步驟

您可以登入 vSphere Client 並管理新的虛擬機器。

恢復失敗工作流程執行

若工作流程失敗，Orchestrator 提供一個從上次失敗活動中恢復工作流程執行的選項。

您可變更工作流程的參數並嘗試恢復，或保留參數並變更影響工作流程執行的外部元件。例如：若工作流程執行因第三方系統發生問題而失敗，您可變更系統並從失敗的活動中恢復工作流程執行，而不必變更工作流程參數且無須重複成功的活動。

設定恢復失敗工作流程執行的行為

您可設定恢復各自訂工作流程失敗執行的行為。程式庫中的預設工作流程使用預設系統設定恢復失敗的工作流程執行。

您可透過修改組態檔案，變更預設系統行為。請參閱設定恢復失敗工作流程執行的自訂內容。

必要條件

確認您有編輯工作流程的權限。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 展開工作流程階層式清單，瀏覽您要設定行為的工作流程。

4 在工作流程上按一下滑鼠右鍵，然後選取**編輯**。

工作流程編輯器開啟。

5 在**一般**索引標籤上選擇**恢復失敗行為**下拉式功能表中的選項。

選項	說明
系統預設值	依預設行為。
已啟用	若工作流程執行失敗，快顯視窗會顯示選項以供恢復工作流程執行。
已停用	若工作流程執行失敗，將無法恢復。

6 按一下**儲存並關閉**。

設定恢復失敗工作流程執行的自訂內容

依預設，Orchestrator 未設定恢復失敗工作流程執行。您可啟用 Orchestrator 恢復失敗工作流程執行，並設定無法恢復失敗工作流程執行後的自訂逾時時間。

程序

1 在 Orchestrator 伺服器系統上，導覽至 `/etc/vco/app-server/`。

2 在文字編輯器中開啟 `vmo.properties` 組態檔。

3 在 `vmo.properties` 檔案中編輯下列行，設定 Orchestrator 恢復失敗的工作流程執行。

```
com.vmware.vco.engine.execute.resume-from-failed=true
```

4 在 `vmo.properties` 檔案中編輯下列行，設定恢復失敗工作流程執行的自訂逾時時間。

```
com.vmware.vco.engine.execute.resume-from-failed.timeout-sec=<秒>
```

您設定越控預設逾時設定 86400 秒的數值。

5 儲存 `vmo.properties` 檔案。

6 重新啟動 Orchestrator 伺服器。

恢復失敗工作流程執行

若啟用恢復失敗工作流程執行，您可從上次失敗的活動中恢復工作流程執行。

啟用恢復失敗工作流程執行的選項後，您可變更工作流程參數並嘗試利用工作流程失敗後出現快顯視窗中的選項恢復。您也可保留參數並變更影響工作流程執行的外部元件。若您不選擇選項，工作流程執行會逾時且無法恢復。如需修改逾時時間，請參閱 設定恢復失敗工作流程執行的自訂內容。

程序

1 在快顯視窗中的下拉式功能表中，選擇**恢復**並按一下**下一步**。

若您選擇 **取消**，工作流程執行將無法在之後恢復。

2 (選擇性) 修改工作流程參數。

3 按一下**提交**。

產生工作流程說明文件

您可隨時以 PDF 格式匯出選擇的工作流程或工作流程資料夾說明文件。

匯出的文件包含有關所選工作流程或資料夾中工作流程的詳細資訊。有關各工作流程的資訊，包括名稱、工作流程版本紀錄、屬性、參數簡報、工作流程配置及工作流程動作。另外，說明文件也提供來源程式碼供動作使用。

程序

1 從 Orchestrator 用戶端的下拉式功能表中，選取**執行**或**設計**。

2 按一下**工作流程**視圖。

3 瀏覽至您要產生說明文件的工作流程或工作流程資料夾，然後按一下滑鼠右鍵。

4 選擇 **產生說明文件**。

5 瀏覽找出儲存 PDF 檔案的資料夾，提供檔案名稱，然後按一下 **儲存**。

包含有關所選工作流程的資訊，或資料夾中工作流程資訊的 PDF 檔案已儲存在您的系統上。

使用工作流程版本紀錄

您可使用版本紀錄還原工作流程為上一個版本。您可還原工作流程狀態為先前或之後的工作流程版本。您也可比較目前工作流程狀態與儲存工作流程版本間的差異。

當您增加並儲存工作流程版本後，Orchestrator 隨即會為各工作流程建立新版本紀錄項目。工作流程的後續變更不會改變目前儲存的版本。例如：當您建立工作流程版本 1.0.0 並儲存後，將在版本紀錄中儲存工作流程狀態。若您對工作流程進行任何變更，可在 Orchestrator 用戶端儲存工作流程狀態，但無法套用變更至工作流程版本 1.0.0。若要在版本紀錄中儲存變更，您必須建立後續工作流程版本並儲存。版本紀錄將連同工作流程本身保存在資料庫中。

當您刪除工作流程後，Orchestrator 會在不刪除資料庫元素版本紀錄下，將資料庫中的元素標記為刪除。您可利用此方式還原刪除的工作流程。請參閱還原刪除的工作流程。

必要條件

在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的 **一般** 索引標籤，然後按一下 **顯示版本紀錄**。

2 選擇工作流程版本並按一下 **目前差異** 比較差異。

視窗顯示目前工作流程版本與選取工作流程版本間的差異。

3 選擇工作流程版本並按一下 **還原** ，還原工作流程狀態。

> 注意 若您未儲存目前的工作流程版本，將從版本紀錄中刪除，且您無法還原回目前的版本。

工作流程狀態還原為選取版本的狀態。

還原刪除的工作流程

您可還原已從工作流程程式庫中刪除的工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中，選取**執行**或**設計**。

2 按一下**工作流程**視圖。

3 瀏覽至您要還原刪除工作流程的資料夾。

4 在資料夾上按一下滑鼠右鍵，然後選擇 **還原刪除的工作流程**。

5 選擇工作流程或您想要還原的工作流程，然後按一下 **還原**。

工作流程出現在選取的資料夾中。

開發簡單範例工作流程

開發簡單範例工作流程以示範工作流程開發程序中最常見的步驟。

您即將建立的範例工作流程會啟動 vCenter Server 中現有的虛擬機器，並寄給管理員一封電子郵件確認虛擬機器已啟動。

範例工作流程會執行下列工作：

1 提示使用者選取要啟動的虛擬機器。

2 提示使用者提供接收通知的電子郵件地址。

3 檢查所選虛擬機器的電源是否已開啟。

4 將要求傳送給 vCenter Server 執行個體以啟動虛擬機器。

5 等待 vCenter Server 啟動虛擬機器，如果虛擬機器無法啟動，或啟動虛擬機器所需的時間過長，則傳回錯誤。

6 等待 vCenter Server 啟動虛擬機器上的 VMware Tools，如果虛擬機器無法啟動，或啟動 VMware Tools 所需的時間過長，則傳回錯誤。

7 確認虛擬機器正在執行中。

8 將通知寄到所提供的電子郵件地址，告知機器已啟動或發生錯誤。

您可從 Orchestrator 說明文件登陸頁面下載 Orchestrator 範例 ZIP 檔案，其中包含完整版的 [啟動虛擬機器並傳送電子郵件] 工作流程。

開發範例工作流程的程序包含數個工作。

必要條件

在嘗試開發簡單範例工作流程前，請先閱讀工作流程的重要概念。

程序

1 建立簡單工作流程範例

您必須在 Orchestrator 用戶端中建立工作流程，以開始工作流程開發程序。

2 建立簡單工作流程範例架構

您可以在工作流程編輯器中建立工作流程的架構。工作流程架構包含工作流程執行的元素，並決定工作流程的邏輯流程。

3 (選擇性) 建立簡單工作流程範例區域

您可以新增不同色彩的工作流程備註，來強調工作流程中的不同區域。建立不同的工作流程區域有助於更容易看懂與瞭解複雜的工作流程架構。

4 定義簡單工作流程範例參數

在工作流程開發的這個階段中，您定義工作流程執行所需的輸入參數。以範例工作流程為例，您需要可開啟虛擬機器電源的輸入參數，及要獲得作業結果通知的人員其電子郵件地址參數。使用者執行工作流程時，必須指定要開啟電源的虛擬機器和電子郵件地址。

5 定義簡單工作流程範例決定繫結

您在工作流程編輯器的**架構**索引標籤中，將工作流程的元素繫結在一起。決定繫結會定義決定元素如何將收到的輸入參數與決定元素進行比較，然後根據輸入參數是否符合決定陳述式來產生輸出參數。

6 繫結簡單工作流程範例的動作元素

您可以在工作流程編輯器中將工作流程元素繫結在一起。繫結會定義動作元素如何處理輸入參數和產生輸出參數。

7 繫結簡單工作流程範例指令碼式工作元素

您在工作流程編輯器的**架構**索引標籤中，將工作流程的元素繫結在一起。繫結會定義指令碼式工作元素如何處理輸入參數和產生輸出參數。您也可以將可編寫指令碼工作元素繫結到其 JavaScript 函數。

8 定義簡單工作流程範例例外狀況繫結

您在工作流程編輯器的**架構**索引標籤中定義例外狀況繫結。例外狀況繫結會定義項目處理錯誤的方式。

9 設定簡單工作流程範例屬性的讀寫內容

您可以定義參數和屬性是唯讀常數或可寫入變數。您也可以限制使用者可針對輸入參數提供的值。

10 設定簡單工作流程範例參數內容

您可以在工作流程編輯器中設定參數內容。設定參數內容會影響參數的行為，並且會限制該參數的可能值。

11 設定 [簡單工作流程範例輸入參數] 對話方塊的配置

您可以在工作流程編輯器中建立輸入參數對話方塊的配置或呈現。當使用者執行的工作流程需要輸入參數才能執行時，輸入參數對話方塊將開啟。

12 驗證和執行簡單工作流程範例

您建立工作流程後，可驗證工作流程來發現任何可能的錯誤。如果工作流程沒有任何錯誤，您便可以執行工作流程。

建立簡單工作流程範例

您必須在 Orchestrator 用戶端中建立工作流程，以開始工作流程開發程序。

必要條件

確認系統上已安裝並設定下列元件。

- vCenter Server，可控制一些虛擬機器，其中至少有一部虛擬機器的電源為關閉

- SMTP 伺服器的存取權
- 有效的電子郵件地址

如需安裝與設定 vCenter Server 的相關資訊，請參閱《vSphere 安裝和設定》說明文件。如需如何設定 Orchestrator 使用 SMTP 伺服器的相關資訊，請參閱《安裝和設定 VMware vRealize Orchestrator》。

若要撰寫工作流程，您必須具有 Orchestrator 使用者帳戶，且該帳戶必須至少有伺服器或目前您所使用工作流程資料夾的**檢視**、**執行**、**檢查**、**編輯**權限，最好有**管理員**權限。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 在工作流程清單的根目錄按一下滑鼠右鍵，並選取**新增資料夾**。

4 將新資料夾命名為**工作流程範例**，然後按一下**確定**。

5 在**工作流程範例**資料夾上按一下滑鼠右鍵，然後選取**新增工作流程**。

6 將新的工作流程命名為**啟動虛擬機器並傳送電子郵件**，然後按一下**確定**。

工作流程編輯器隨即開啟。

7 在**一般**索引標籤上按一下版本號碼數字，以增加版本號碼。

因為這是初次建立工作流程，請將版本設定為 **0.0.1**。

8 按一下**一般**索引標籤中的**伺服器重新啟動行為**值，以設定工作流程在伺服器重新啟動後是否繼續執行。

9 在**一般**索引標籤的**說明**文字方塊中輸入工作流程用途的說明。

例如，您可以新增下列說明。

此工作流程會啟動虛擬機器，並將確認電子郵件傳送給 Orchestrator 管理員。

10 按一下**一般**索引標籤底端的**儲存**。

您已建立一個名為 [啟動虛擬機器並傳送電子郵件] 工作流程，但您尚未定義其函數。

後續步驟

建立工作流程的架構。

建立簡單工作流程範例架構

您可以在工作流程編輯器中建立工作流程的架構。工作流程架構包含工作流程執行的元素，並決定工作流程的邏輯流程。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**架構**索引標籤。

2 從**一般**功能表，將決定元素拖曳至連結架構中開始元素與 End 元素的箭頭。

3 按兩下決定元素，將其名稱變更為 **VM powered on?**。

此決定元素會對應到檢查虛擬機器的電源是否已開啟的布林值函數。

4 從**一般**功能表中，將動作元素拖曳至連結決定元素和 End 元素的紅色箭頭。

隨即顯示可供選取動作的對話方塊。

5 在**篩選器**文字方塊中輸入 **start**，然後從經篩選的動作清單中選取 **startVM** 動作，然後按一下**選取**。

6 將下列動作元素一個接著一個拖曳至將 startVM 動作元素連結至 End 元素的藍色箭頭。

vim3WaitTaskEnd	暫停工作流程執行，並定時對進行中的 vCenter Server 工作執行 Ping 動作，直到工作完成為止。startVM 動作會啟動虛擬機器，vim3WaitTaskEnd 動作則會讓工作流程等待，直到虛擬機器啟動為止。虛擬機器啟動後，vim3WaitTaskEnd 會讓工作流程繼續進行。
vim3WaitToolsStarted	暫停執行工作流程，並等待直到 VMware Tools 在目標虛擬機器上啟動為止。

7 從**一般**功能表中，將可編寫指令碼工作元素拖曳至將 vim3WaitToolsStarted 動作元素連結至 End 元素的藍色箭頭。

8 按兩下可編寫指令碼工作元素，並將其重新命名為 **OK**。

9 將另一個可編寫指令碼工作元素拖曳至將 VM powered on? 決定元素連結至 End 元素的綠色箭頭，並將此可編寫指令碼工作元素命名為 **Already started**。

10 修改 Already started 可編寫指令碼工作元素的連結。

a 將 Already started 可編寫指令碼工作元素拖曳至 startVM 動作元素的左側。

b 刪除將 Already started 可編寫指令碼工作元素連接至 End 元素的藍色箭頭。

c 使用藍色箭頭將 Already started 可編寫指令碼工作元素連結至 vim3WaitToolsStarted 動作元素。

11 從**一般**功能表，將下列可編寫指令碼工作元素拖曳至架構。

- 將某個可編寫指令碼工作元素拖曳至 startVM 動作元素，並將該可編寫指令碼工作元素命名為 **Start VM Failed**。
- 將某個可編寫指令碼工作元素拖曳至 vim3WaitTaskEnd 動作元素，並將該可編寫指令碼工作元素命名為 **Timeout 1**。
- 將某個可編寫指令碼工作元素拖曳至 vim3WaitToolsStarted 動作元素，並將該可編寫指令碼工作元素命名為 **Timeout 2**。

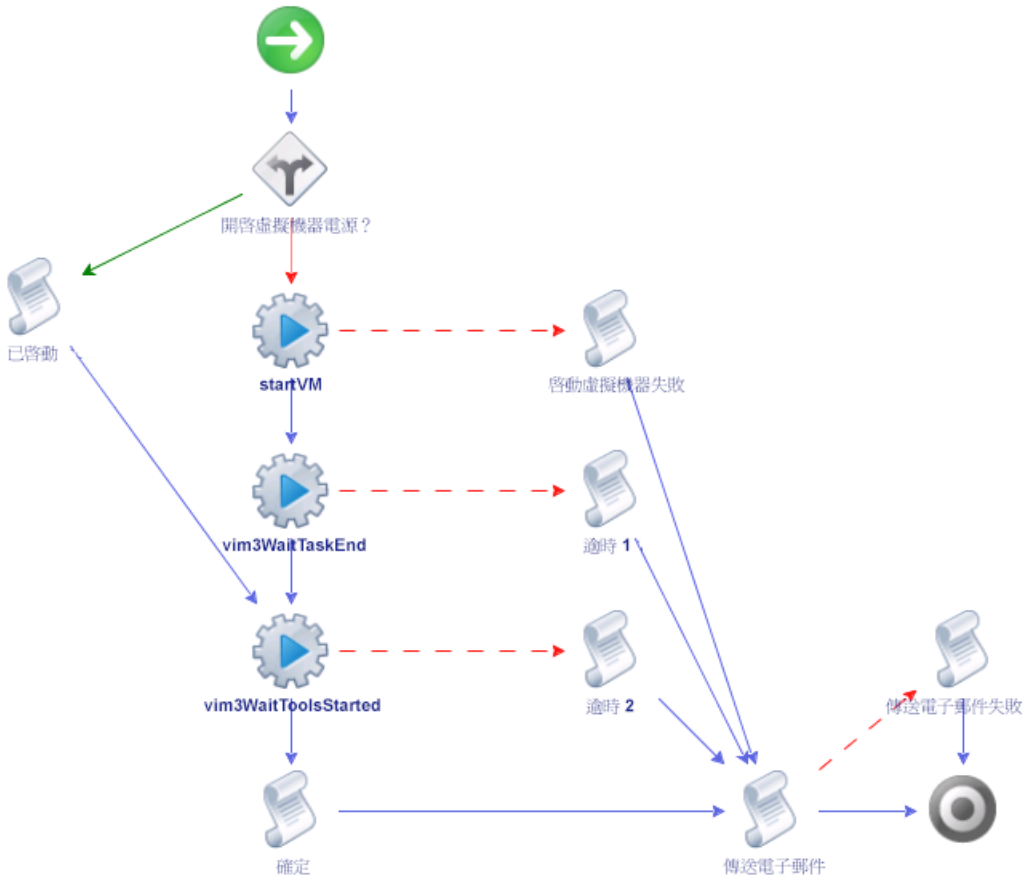
- 將某個可編寫指令碼工作元素拖曳至將 OK 可編寫指令碼工作元素連結至 End 元素的藍色箭頭，並將新的可編寫指令碼工作元素命名為**傳送電子郵件**，然後將其拖曳到 OK 可編寫指令碼工作元素的右側。
- 使用藍色箭頭將 Start VM Failed、Timeout 1 和 Timeout 2 可編寫指令碼工作元素連結至 Send Email 可編寫指令碼工作元素。
- 將某個可編寫指令碼工作元素拖曳至 Send Email 可編寫指令碼工作元素，並將新的可編寫指令碼工作元素命名為 **Send Email Failed**，然後將其拖曳到 Timeout 2 可編寫指令碼工作元素的右側，再使用藍色箭頭將它連結至 End 元素。

12 將 End 元素拖曳至 Send Email 可編寫指令碼工作元素的右側。

13 按一下**架構**索引標籤底端的**儲存**。

下圖顯示 [啟動虛擬機器並傳送電子郵件] 工作流程架構元素的配置。

圖 1-10. 連結啟動虛擬機器並傳送電子郵件範例工作流程的元素



後續步驟

您可以反白工作流程中的不同區域。

建立簡單工作流程範例區域

您可以新增不同色彩的工作流程備註，來強調工作流程中的不同區域。建立不同的工作流程區域有助於更容易看懂與瞭解複雜的工作流程架構。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 在工作流程編輯器中開啟工作流程以進行編輯。

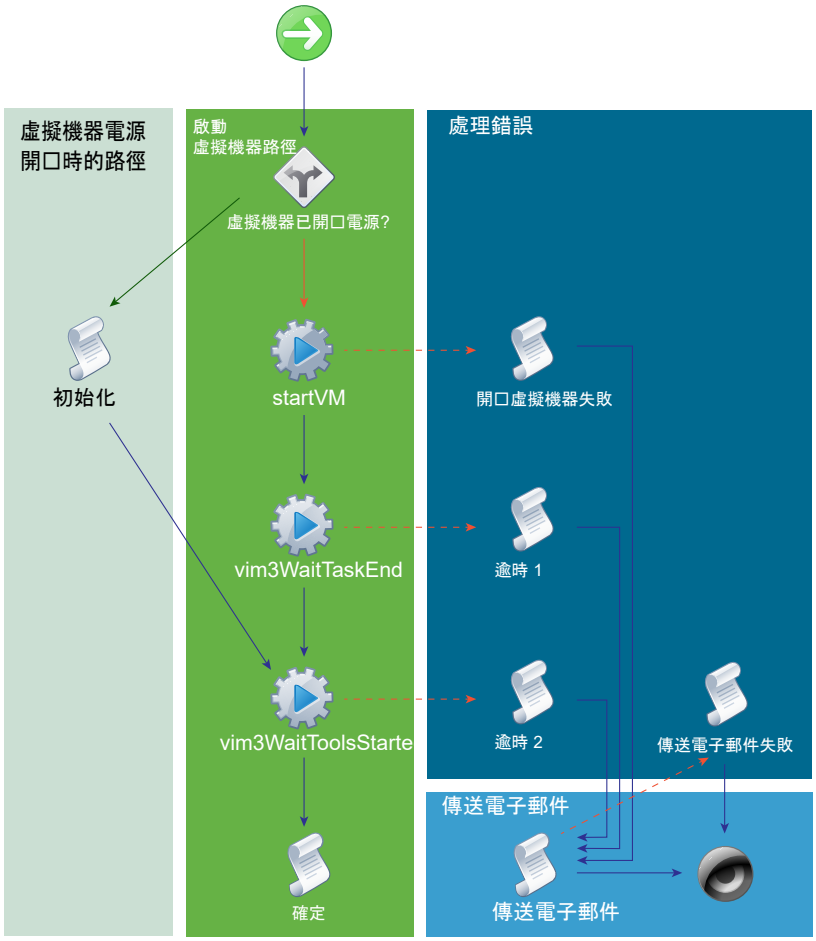
程序

1. 將**一般**功能表中的工作流程備註元素拖曳到工作流程編輯器。
2. 將工作流程備註放置在 `Already started` 可編寫指令碼工作流程元素上方。
3. 拖曳工作流程備註的邊緣以調整大小，讓它圍繞在 `Already started` 可編寫指令碼工作元素的四周。
4. 按兩下文字，然後新增說明。

   例如，**如果虛擬機器電源已開啟的路徑**。
5. 按 Ctrl+E 選取背景色彩。
6. 重複前述步驟以反白工作流程中的其他區域。
   - 將備註放置在從 `VM powered on?` 決定元素到 `OK` 元素之間垂直順序的元素上。新增以下說明：**啟動虛擬機器路徑**。
   - 將備註放置在 `startVM failed` 上，及 `Timeout` 可編寫指令碼工作元素與 `Send Email Failed` 可編寫指令碼工作元素上。新增以下說明：**處理錯誤**。
   - 將備註放置在 `Send Email` 可編寫指令碼工作流程元素上方。新增以下說明：**傳送電子郵件**。

範例工作流程區域如下圖所示。

圖 1-11. 啟動虛擬機器並傳送電子郵件範例工作流程區域



後續步驟

您必須定義工作流程的屬性和輸入與輸出參數。

定義簡單工作流程範例參數

在工作流程開發的這個階段中，您定義工作流程執行所需的輸入參數。以範例工作流程為例，您需要可開啟虛擬機器電源的輸入參數，及要獲得作業結果通知的人員其電子郵件地址參數。使用者執行工作流程時，必須指定要開啟電源的虛擬機器和電子郵件地址。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**輸入**索引標籤。

2 在**輸入**索引標籤內按一下滑鼠右鍵，然後選取**新增參數**。

名為 `arg_in_0` 的參數隨即出現在**輸入**索引標籤中。

3 按一下 **`arg_in_0`**。

4 在 [選擇屬性名稱] 對話方塊中輸入 **`vm`** 作為名稱，然後按一下**確定**。

5 按一下**類型**文字方塊，然後在參數類型對話方塊的文字方塊中輸入 **`vc:virtualm`**。

6 從參數類型建議清單中選取 **`VC:VirtualMachine`**，然後按一下**接受**。

7 在**說明**文字方塊中，新增參數的說明。

例如，輸入**要開啟電源的虛擬機器**。

8 重複 步驟 2 到 步驟 7，以下列值建立第二個輸入參數。

- 名稱：`toAddress`
- 類型：字串
- 說明：**此工作流程的結果寄到這個電子郵件地址**

9 按一下**輸入**索引標籤底端的**儲存**。

您已定義工作流程的輸入參數。

後續步驟

定義元素參數之間的繫結。

定義簡單工作流程範例決定繫結

您在工作流程編輯器的**架構**索引標籤中，將工作流程的元素繫結在一起。決定繫結會定義決定元素如何將收到的輸入參數與決定元素進行比較，然後根據輸入參數是否符合決定陳述式來產生輸出參數。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在**架構**索引標籤上，按一下 **`VM Powered On?`** 決定元素的**編輯**圖示 (✏)。

2 在**決定**索引標籤上，按一下**未設定 (空值)** 按鈕，然後從建議的參數清單中選取 `vm` 作為決定元素的輸入參數。

3 從下拉式功能表建議的決定陳述式清單中，選取**電源狀態等於**陳述式。

未設定按鈕會出現在值文字方塊中，提供給您有限制的可能值選項。

4 選取 **poweredOn**。

5 在工作流程編輯器之**架構**索引標籤的底端，按一下**儲存**。

您已定義決定元素用來比較所收到輸入參數值的 true 或 false 陳述式。

後續步驟

您必須定義工作流程中其他元素的繫結。

繫結簡單工作流程範例的動作元素

您可以在工作流程編輯器中將工作流程元素繫結在一起。繫結會定義動作元素如何處理輸入參數和產生輸出參數。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在**架構**索引標籤上，按一下 `startVM` 動作元素的**編輯**圖示 (✏)。

2 在**資訊**索引標籤上設定下列一般資訊。

選項	動作
互動	選取**無外部互動**。
業務狀態	選取該核取方塊，然後新增以下文字：**傳送開始虛擬機器**。
說明	留下「開始/繼續虛擬機器」文字。傳回開始工作。

3 按一下**輸入**索引標籤。

輸入索引標籤會顯示兩個可供 `startVM` 動作使用的可能輸入參數，分別為 `vm` 與 `host`。

Orchestrator 會自動將 `vm` 參數繫結到 `vm[in-parameter]`，因為 `startVM` 動作只會讓 `VC:VirtualMachine` 作為輸入參數。Orchestrator 會偵測您在設定工作流程輸入參數時定義的 `vm` 參數，並因此自動將該參數繫結到動作。

4 將 `host` 設定為 **NULL**。

這是選用的參數，因此您可以將它設為空值。但如果您將它保持設為**未設定**，則工作流程會無法驗證。

5 按一下**輸出**索引標籤。

隨即顯示所有動作產生的預設輸出參數 `actionResult`。

6 對 `actionResult` 參數按一下**未設定**。

7 按一下**在工作流程中建立參數/屬性**。

[參數資訊] 對話方塊會顯示您為此輸出參數設定的值。`startVM` 動作的輸出參數類型為 `VC:Task` 物件。

8 將參數命名為 **powerOnTask**，並提供說明。

例如，**包含將虛擬機器電源開啟的結果**。

9 按一下**建立相同名稱的工作流程屬性**，然後按一下**確定**結束 [參數資訊] 對話方塊。

10 重複前述步驟，將輸入與輸出參數繫結到 `vim3WaitTaskEnd` 和 `vim3WaitToolsStarted` 動作元素。

簡單工作流程範例動作元素繫結 列出 `vim3WaitTaskEnd` 與 `vim3WaitToolsStarted` 動作元素的繫結。

11 在工作流程編輯器之**架構**索引標籤的底端，按一下**儲存**。

動作元素的輸入與輸出參數隨即繫結到適當的參數類型與值。

後續步驟

繫結可編寫指令碼工作元素，並定義其函數。

簡單工作流程範例動作元素繫結

繫結定義簡單工作流程範例的動作元素程序如何處理輸入與輸出參數。

定義繫結時，Orchestrator 會將您在工作流程中定義的參數呈現為繫結的候選。如果您尚未在工作流程中定義所需的參數，唯一的參數選擇是 `NULL`。按一下**在工作流程中建立參數/屬性**建立新參數。

vim3WaitTaskEnd 動作

`vim3WaitTaskEnd` 動作元素會宣告常數來追蹤工作進度和輪詢速率。下表顯示 `vim3WaitTaskEnd` 動作需要的輸入和輸出參數繫結。

表 1-53. vim3WaitTaskEnd 動作的繫結值

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
`task`	輸入	繫結	■ 本機參數：`powerOnTask` ■ 來源參數：`task[attribute]` ■ 類型：`VC:Task` ■ 說明：**包含開啟虛擬機器電源的結果。**
`progress`	輸入	建立	■ 本機參數：`progress` ■ 來源參數：`progress[attribute]` ■ 類型：布林值 ■ 值：無 (false) ■ 說明：**記錄等待 vCenter Server 工作完成的進度。**

表 1-53. vim3WaitTaskEnd 動作的繫結值 (續)

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
pollRate	輸入	建立	■ 本機參數：pollRate ■ 來源參數：pollRate[attribute] ■ 類型：數字 ■ 值：**2** ■ 說明：**vim3WaitTaskEnd 檢查 vCenter Server 工作進展的輪詢速率秒數。**
actionResult	輸出	建立	■ 本機參數：actionResult[attribute] ■ 來源參數：returnedManagedObject[attribute] ■ 類型：任何 ■ 說明：**從 waitTaskEnd 動作傳回的受管理物件。**

vim3WaitToolsStarted 動作

vim3WaitToolsStarted 動作元素會等待 VMware Tools 在虛擬機器上安裝完畢，並定義輪詢速率和逾時時間。下表顯示 vim3WaitToolsStarted 動作需要的輸入參數繫結。

vim3WaitToolsStarted 動作元素沒有輸出，因此不需要輸出繫結。

表 1-54. vim3WaitToolsStarted 動作的繫結值

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	自動繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 值：不可編輯，變數不是工作流程屬性。 ■ 說明：**要啟動的虛擬機器。**
pollingRate	輸入	繫結	■ 本機參數：pollRate ■ 來源參數：pollRate[attribute] ■ 類型：數字 ■ 說明：**vim3WaitTaskEnd 檢查 vCenter Server 工作進展的輪詢速率秒數。**
timeout	輸入	建立	■ 本機參數：timeout ■ 來源參數：timeout[attribute] ■ 類型：數字 ■ 值：**10** ■ 說明：**vim3WaitToolsStarted 在擲回例外狀況前等待的逾時限制。**

繫結簡單工作流程範例指令碼式工作元素

您在工作流程編輯器的**架構**索引標籤中，將工作流程的元素繫結在一起。繫結會定義指令碼式工作元素如何處理輸入參數和產生輸出參數。您也可以將可編寫指令碼工作元素繫結到其 JavaScript 函數。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在**架構**索引標籤上，按一下 `Already Started` 可編寫指令碼工作元素的**編輯**圖示 ( )。

2 在**資訊**索引標籤中設定下列一般資訊。

選項	動作
互動	選取**無外部互動**。
業務狀態	選取該核取方塊，然後新增以下文字：**虛擬機器電源已開啟**。
說明	留下「虛擬機器電源已開啟」文字，略過 startVM 與 waitTaskEnd，以檢查虛擬機器工具是否已啟動且正在執行中。

3 按一下**輸入**索引標籤。

由於這是自訂可編寫指令碼工作元素，因此沒有已預先定義的內容。

4 按一下**繫結至工作流程參數/屬性**圖示 ( )。

5 從建議參數清單中選取 `vm`。

6 將**輸出**與**例外狀況**索引標籤保留空白。

此元素未產生輸出參數或例外狀況。

7 按一下**指令碼**索引標籤。

8 新增下列 JavaScript 函數。

```
//Writes the following event in the Orchestrator database
Server.log("VM '"+ vm.name +"' already started");
```

9 重複先前的步驟，將剩餘的輸入參數繫結到其他可編寫指令碼工作元素。

簡單工作流程範例之可編寫指令碼工作元素的繫結 會列出 `Start VM failed`，及 `Timeout` 或 `Error`、`Send Email Failed` 和 `OK` 可編寫指令碼工作元素。

10 在工作流程編輯器之**架構**索引標籤的底端，按一下**儲存**。

您已將可編寫指令碼工作元素繫結到其輸入與輸出參數，並已提供定義其函數的指令碼。

後續步驟

您必須定義例外狀況處理。

簡單工作流程範例之可編寫指令碼工作元素的繫結

繫結會定義簡單工作流程範例的可編寫指令碼工作元素應如何處理輸入參數。您也可以將可編寫指令碼工作元素繫結到其 JavaScript 函數。

定義繫結時，Orchestrator 會將您在工作流程中定義的參數呈現為繫結的候選。如果您尚未在工作流程中定義所需的參數，唯一的參數選擇是 `NULL`。按一下**在工作流程中建立參數/屬性**建立新參數。

Start VM failed 可編寫指令碼工作

Start VM failed 可編寫指令碼工作元素會處理 `startVM` 動作擲回的任何例外狀況，方法是設定關於無法啟動虛擬機器的電子郵件通知內容，並在 Orchestrator 記錄中寫入事件。

下表顯示 Start VM failed 可編寫指令碼工作元素需要的輸入和輸出參數繫結。

表 1-55. Start VM failed 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
`vm`	輸入	繫結	■ 本機參數：`vm` ■ 來源參數：`vm[in-parameter]` ■ 類型：`VC:VirtualMachine` ■ 說明：**要開啟電源的虛擬機器。**
`errorCode`	輸入	建立	■ 本機參數：`errorCode` ■ 來源參數：`errorCode[attribute]` ■ 類型：字串 ■ 說明：**捕捉開啟虛擬機器電源時發生的任何例外狀況。**
`body`	輸出	建立	■ 本機參數：`body` ■ 來源參數：`body[attribute]` ■ 類型：字串 ■ 說明：**電子郵件內文**

Start VM Failed 可編寫指令碼工作元素會執行下列指令碼式函數。

```
body = "Unable to execute powerOnVM_Task() on VM '"+vm.name+"', exception found: "+errorCode;
//Writes the following event  in the Orchestrator database
Server.error("Unable to execute powerOnVM_Task() on VM '"+vm.name+"', exception found: "+errorCode);
```

Timeout 1 可編寫指令碼工作元素

Timeout 1 可編寫指令碼工作元素會處理 `vim3WaitTaskEnd` 動作擲回的任何例外狀況，方法是設定關於工作失敗的電子郵件通知內容，並在 Orchestrator 記錄中寫入事件。

下表顯示 Timeout 1 可編寫指令碼工作元素需要的輸入和輸出參數繫結。

表 1-56. Timeout 1 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 說明：**要啟動的虛擬機器。**
errorCode	輸入	繫結	■ 本機參數：errorCode ■ 來源參數：errorCode[attribute] ■ 類型：字串 ■ 說明：**捕捉開啟虛擬機器電源時發生的任何例外狀況。**
body	輸出	繫結	■ 本機參數：body ■ 來源參數：body[attribute] ■ 類型：字串 ■ 說明：**電子郵件內文**

Timeout 1 可編寫指令碼工作元素需要下列指令碼式函數。

```
body = "Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name+"', exception found:
"+errorCode;
//Writes the following event in the Orchestrator database
Server.error("Error while waiting for poweredOnVM_Task() to complete on VM '"+vm.name+"', exception
found: "+errorCode);
```

Timeout 2 可編寫指令碼工作元素

Timeout 2 可編寫指令碼工作元素會處理 vim3WaitToolsStarted 動作擲回的任何例外狀況，方法是設定關於工作失敗的電子郵件通知內容，並在 Orchestrator 記錄中寫入事件。

下表顯示 Timeout 2 可編寫指令碼工作元素需要的輸入和輸出參數繫結。

表 1-57. Timeout 2 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 說明：**要開啟電源的虛擬機器。**
errorCode	輸入	繫結	■ 本機參數：errorCode ■ 來源參數：errorCode[attribute] ■ 類型：字串 ■ 說明：**捕捉開啟虛擬機器電源時發生的任何例外狀況。**
body	輸出	繫結	■ 本機參數：body ■ 來源參數：body[attribute] ■ 類型：字串 ■ 說明：**電子郵件內文**

Timeout 2 可編寫指令碼工作元素需要下列指令碼式函數。

```
body = "Error while waiting for VMware tools to be up on VM '"+vm.name+"', exception found:
"+errorCode;
//Writes the following event in the Orchestrator database
Server.error("Error while waiting for VMware tools to be up on VM '"+vm.name+"', exception found:
"+errorCode);
```

OK 可編寫指令碼工作元素

OK 可編寫指令碼工作元素會接收虛擬機器已成功啟動的通知，並設定虛擬機器成功啟動的電子郵件通知內容，然後在 Orchestrator 記錄中寫入事件。

下表顯示 OK 可編寫指令碼工作元素需要的輸入和輸出參數繫結。

表 1-58. OK 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 說明：**要開啟電源的虛擬機器。**
body	輸出	繫結	■ 本機參數：body ■ 來源參數：body[attribute] ■ 類型：字串 ■ 說明：**電子郵件內文**

OK 可編寫指令碼工作元素需要下列指令碼式函數。

```
body = "The VM '"+vm.name+"' has started successfully and is ready for use";
//Writes the following event in the Orchestrator database
Server.log(body);
```

Send Email Failed 可編寫指令碼工作元素

Send Email Failed 可編寫指令碼工作元素會接收傳送電子郵件失敗的通知，然後在 Orchestrator 記錄中寫入事件。

下表顯示 Send Email Failed 可編寫指令碼工作元素需要的輸入參數繫結。

表 1-59. Send Email Failed 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 說明：**要開啟電源的虛擬機器。**
toAddress	輸入	繫結	■ 本機參數：toAddress ■ 來源參數：toAddress[in-parameter] ■ 類型：字串 ■ 說明：**通知此工作流程結果的收件人電子郵件地址**
emailErrorCode	輸入	建立	■ 本機參數：emailErrorCode ■ 來源參數：emailErrorCode[attrbute] ■ 類型：字串 ■ 說明：**捕捉傳送電子郵件時發生的任何例外狀況**

Send Email Failed 可編寫指令碼工作元素需要下列指令碼式函數。

```
//Writes  the following event in the Orchestrator database
Server.error("Couldn't send result email to '"+toAddress+"' for VM '"+vm.name+"', exception found:
"+emailErrorCode);
```

Send Email 可編寫指令碼工作元素

[啟動虛擬機器並傳送電子郵件] 工作流程的目的是在啟動虛擬機器時通知管理員。若要這麼做，您必須定義傳送電子郵件的可編寫指令碼工作。若要傳送電子郵件，Send Email 可編寫指令碼工作元素需要 SMTP 伺服器、電子郵件寄件人和收件人的地址、電子郵件主旨，以及電子郵件內容。

下表顯示 Send Email 可編寫指令碼工作元素需要的輸入和輸出參數繫結。

表 1-60. Send Email 可編寫指令碼工作元素的繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[in-parameter] ■ 類型：VC:VirtualMachine ■ 說明：**要開啟電源的虛擬機器。**
toAddress	輸入	繫結	■ 本機參數：toAddress ■ 來源參數：toAddress[in-parameter] ■ 類型：字串 ■ 說明：**通知此工作流程結果的收件人電子郵件地址**
body	輸入	繫結	■ 本機參數：body ■ 來源參數：body[attribute] ■ 類型：字串 ■ 說明：**電子郵件內文**
smtpHost	輸入	建立	■ 本機參數：smtpHost ■ 來源參數：smtpHost[attribute] ■ 類型：字串 ■ 說明：**電子郵件 SMTP 伺服器**
fromAddress	輸入	建立	■ 本機參數：fromAddress ■ 來源參數：fromAddress[attribute] ■ 類型：字串 ■ 說明：**寄件人的電子郵件地址**
subject	輸入	建立	■ 本機參數：subject ■ 來源參數：subject[attribute] ■ 類型：字串 ■ 說明：**電子郵件主旨**

Send Email 可編寫指令碼工作元素需要下列指令碼式函數。

```
//Create an instance of EmailMessage
var myEmailMessage = new EmailMessage() ;

//Apply methods on this instance that populate the email message
myEmailMessage.smtpHost = smtpHost;
myEmailMessage.fromAddress = fromAddress;
myEmailMessage.toAddress = toAddress;
myEmailMessage.subject = subject;
myEmailMessage.addMimePart(body , "text/html");

//Apply the method that sends the email message
myEmailMessage.sendMessage();
System.log("Sent email to '"+toAddress+"'");
```

定義簡單工作流程範例例外狀況繫結

您在工作流程編輯器的**架構**索引標籤中定義例外狀況繫結。例外狀況繫結會定義項目處理錯誤的方式。

工作流程中的下列元素傳回例外狀況：`startVM`、`vim3WaitTaskEnd`、`Send Email` 與 `vim3WaitToolsStarted`。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 繫結簡單工作流程範例的動作元素。
- 繫結簡單工作流程範例指令碼式工作元素。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1. 在**架構**索引標籤上，按一下 **startVM** 動作元素的**編輯**圖示 (✏)。
2. 按一下**例外狀況**索引標籤。
3. 按一下**未設定**按鈕。
4. 從建議的清單中選取 **errorCode**。
5. 重複前述步驟，將 `vim3WaitTaskEnd` 與 `vim3WaitToolsStarted` 的例外狀況繫結設定為 `errorCode`。
6. 按一下 **Send Email** 可編寫指令碼工作元素的**編輯**圖示 (✏)。
7. 按一下**例外狀況**索引標籤。
8. 按一下**未設定**按鈕。
9. 從建議的清單中選取 **emailErrorCode**。
10. 在工作流程編輯器之**架構**索引標籤的底端，按一下**儲存**。

您已為傳回例外狀況的元素定義例外狀況繫結。

後續步驟

您必須設定屬性與參數的讀取與寫入內容。

設定簡單工作流程範例屬性的讀寫內容

您可以定義參數和屬性是唯讀常數或可寫入變數。您也可以限制使用者可針對輸入參數提供的值。

將特定參數設定為唯讀能夠讓其他開發人員改寫工作流程，或修改工作流程而不中斷工作流程的核心功能。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 繫結簡單工作流程範例的動作元素。
- 繫結簡單工作流程範例指令碼式工作元素。
- 定義簡單工作流程範例例外狀況繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器頂端的**一般**索引標籤。

屬性下會列出所有定義的屬性，每個屬性旁邊都有核取方塊。您選取這些核取方塊時，即可將屬性設定為唯讀。

2 選取核取方塊會將下列屬性設定為唯讀常數：

- progress
- pollRate
- timeout
- smtpHost
- fromAddress
- subject

您已經定義哪些工作流程的屬性是常數和變數。

後續步驟

設定參數內容，並限制該參數的可能值。

設定簡單工作流程範例參數內容

您可以在工作流程編輯器中設定參數內容。設定參數內容會影響參數的行為，並且會限制該參數的可能值。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 繫結簡單工作流程範例的動作元素。
- 繫結簡單工作流程範例指令碼式工作元素。
- 定義簡單工作流程範例例外狀況繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**呈現**索引標籤。

列出您對於此工作流程定義的兩個輸入參數。

2 按一下 **(VC:VirtualMachine)vm** 參數。

3 在畫面下半部的**一般**索引標籤中新增說明。

例如，輸入**將啟動的虛擬機器**。

4 按一下畫面下半部的**內容**索引標籤。

在此索引標籤上，您可以設定 (VC:VirtualMachine)vm 參數的內容。

5 按一下**新增內容**圖示 ( )。

6 從建議內容的清單中，選取**強制輸入**內容，並按一下**確定**，然後將其值設定為**是**。

您啟用此內容時，若使用者不提供要啟動的虛擬機器，則無法執行 [啟動虛擬機器並傳送電子郵件] 工作流程。

7 按一下**新增內容**圖示 ( )。

8 從建議內容的清單中，選取**選取值作為**，並按一下**確定**，然後從可能值的清單中選取**清單**。

設定此內容時，您可以設定使用者如何選取 (VC:VirtualMachine)vm 輸入參數的值。

9 按一下**呈現**索引標籤上半部的 **(string)toAddress** 參數。

10 在畫面下半部的**說明**索引標籤中新增說明。

例如，輸入**被通知人的電子郵件地址**。

11 按一下 (string)toAddress 的**內容**索引標籤，然後按一下**新增內容**圖示 ( )。

12 從建議內容的清單中，選取**強制輸入**內容，並按一下**確定**，然後將其值設定為**是**。

13 按一下**新增內容**圖示 ( )。

14 從建議內容的清單中，選取**比對規則運算式**，並按一下**確定**。

此內容可讓您限制使用者可提供什麼作為輸入。

15 按一下**比對規則運算式**的**值**文字方塊，並將限制設定為 `[a-zA-Z0-9_%-+.]+@[a-zA-Z0-9-.]+\.[a-zA-Z]{2,4}`。

設定這些限制會限制使用者輸入對於電子郵件地址而言適當的字元。使用者啟動工作流程時，如果使用者嘗試對收件人的電子郵件地址輸入其他任何字元，工作流程將不會啟動。

您已經將兩個參數設定為強制參數，並定義使用者能夠如何選取要啟動的虛擬機器，而且限制可針對收件人電子郵件地址輸入的字元。

後續步驟

您必須建立輸入參數對話方塊的配置或呈現，以便使用者在執行工作流程時指定工作流程的輸入參數值。

設定 [簡單工作流程範例輸入參數] 對話方塊的配置

您可以在工作流程編輯器中建立輸入參數對話方塊的配置或呈現。當使用者執行的工作流程需要輸入參數才能執行時，輸入參數對話方塊將開啟。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 繫結簡單工作流程範例的動作元素。
- 繫結簡單工作流程範例指令碼式工作元素。
- 定義簡單工作流程範例例外狀況繫結。
- 設定簡單工作流程範例屬性的讀寫內容。
- 設定簡單工作流程範例參數內容。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**呈現**索引標籤。

2 在呈現階層清單的**呈現**節點上按一下滑鼠右鍵，然後選取**建立顯示群組**。

新步驟節點和**新群組**子節點將出現在**呈現**節點下。

3 在**新步驟**上按一下滑鼠右鍵，然後選取**刪除**。

由於此工作流程只有兩個參數，因此輸入參數對話方塊不需要多層顯示區段。

4 按兩下**新群組**以編輯群組名稱，然後按 Enter。

例如，將顯示群組命名為**虛擬機器**。

您在此處輸入的文字將成為使用者啟動工作流程時的輸入參數對話方塊標題。

5 在**呈現**索引標籤底端的**一般**索引標籤**說明**文字方塊中，提供新顯示群組的說明。

例如，輸入**選取將啟動的虛擬機器**。

您在此處輸入的文字將成為使用者啟動工作流程時的輸入參數對話方塊提示。

6 將 **(VC:VirtualMachine)vm** 參數拖曳至**虛擬機器**顯示群組之下。

在輸入參數對話方塊中，使用者輸入虛擬機器名稱的文字方塊將出現在虛擬機器標題之下。

7 重複進行前述的步驟來建立 `toAddress` 參數的顯示群組，並設定下列內容：

a 建立顯示群組，並將它命名為**收件人的電子郵件地址**。

b 新增顯示群組的說明，例如，**輸入此虛擬機器開啟電源收到通知的被通知人電子郵件地址**。

c 將 **toAddress** 參數拖曳至**收件人的電子郵件地址**顯示群組之下。

此時您即已設定使用者執行工作流程時出現的輸入參數對話方塊配置。

後續步驟

您現在已完成簡單工作流程範例的開發。現在您可以驗證與執行工作流程。

驗證和執行簡單工作流程範例

您建立工作流程後，可驗證工作流程來發現任何可能的錯誤。如果工作流程沒有任何錯誤，您便可以執行工作流程。

必要條件

請完成以下工作。

- 建立簡單工作流程範例。
- 建立簡單工作流程範例架構。
- 定義簡單工作流程範例參數。
- 定義簡單工作流程範例決定繫結。
- 繫結簡單工作流程範例的動作元素。
- 繫結簡單工作流程範例指令碼式工作元素。
- 定義簡單工作流程範例例外狀況繫結。
- 設定簡單工作流程範例屬性的讀寫內容。
- 設定簡單工作流程範例參數內容。
- 設定 [簡單工作流程範例輸入參數] 對話方塊的配置。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 在工作流程編輯器的**架構**索引標籤中，按一下**驗證**。

驗證工具會找出工作流程定義中的任何錯誤。

2 消除任何錯誤後，請按一下工作流程編輯器底端的**儲存並關閉**。

隨即返回 Orchestrator 用戶端。

3 按一下**工作流程**視圖。

4 選取工作流程階層清單中的**工作流程範例 > 啟動虛擬機器並傳送電子郵件**。

5 在**啟動虛擬機器並傳送電子郵件**工作流程上按一下滑鼠右鍵，然後選取**啟動工作流程**。

輸入參數對話方塊隨即開啟，並提示您將啟動的虛擬機器，以及將收到通知的電子郵件地址。

6 從 vCenter Server 詳細目錄選取將啟動的虛擬機器。

7 輸入將收到電子郵件通知的電子郵件地址。

8 按一下**提交**啟動工作流程。

工作流程 Token 隨即出現在 [啟動虛擬機器並傳送電子郵件] 工作流程之下。

9 按一下工作流程 Token 以追蹤工作流程執行的進度。

如果工作流程成功執行，您選取的虛擬機器將進入開啟電源狀態，且您定義的電子郵件收件人將收到確認電子郵件。

後續步驟

您可產生文件來檢視工作流程的資訊。請參閱產生工作流程說明文件。

開發複雜工作流程

開發複雜範例工作流程以示範工作流程開發程序中最常見的步驟，以及更進階的案例，例如建立自訂決定與迴圈。

在複雜工作流程練習中，您會開發一個工作流程，以拍攝指定資源集區中所包含之所有虛擬機器的快照。您建立的工作流程將執行下列工作：

1 提示使用者指定一個資源集區，其中包含要拍攝快照的虛擬機器。

2 決定資源集區是否包含執行中的虛擬機器。

3 決定資源包含多少部執行中虛擬機器。

4 確認在集區中執行的個別虛擬機器是否符合所要拍攝快照的特定準則。

5 拍攝虛擬機器的快照。

6 判定集區中是否有更多要拍攝快照的虛擬機器。

7 重複驗證與快照程序，直到工作流程已拍攝資源集區中所有合格虛擬機器的快照。

您可從 Orchestrator 說明文件登陸頁面下載 Orchestrator 範例 ZIP 檔案，其中包含完整版的 [拍攝資源集區中所有虛擬機器快照] 工作流程。

必要條件

在您嘗試開發此複雜工作流程前，請遵照 開發簡單範例工作流程 中的練習進行。開發複雜工作流程的程序會提供豐富的開發程序步驟，但不像簡單工作流程練習那樣詳細。

程序

1 建立複雜工作流程範例

您必須在 Orchestrator 用戶端中建立工作流程，以開始工作流程開發程序。

2 為複雜工作流程建立自訂動作範例

`Check VM` 可編寫指令碼元素會呼叫 Orchestrator API 中不存在的動作。您必須建立 `getVMDiskModes` 動作。

3 建立複雜工作流程範例架構

您可以在工作流程編輯器中建立工作流程的架構。工作流程架構包含工作流程執行的元素，並決定工作流程的邏輯流程。

4 (選擇性) 建立複雜工作流程範例區域

(選擇性) 您可以透過新增工作流程備註反白工作流程的不同區域。建立不同的工作流程區域有助於更容易看懂與瞭解複雜的工作流程架構。

5 定義複雜工作流程參數的範例

您可在工作流程編輯器中定義工作流程參數。輸入參數會為工作流程提供要處理的資料。輸出參數是工作流程執行完成時所傳回的資料。

6 定義複雜工作流程範例繫結

您可以在工作流程編輯器中將工作流程元素繫結在一起。繫結可定義工作流程的資料流程。您也可以將可編寫指令碼工作元素繫結到其 JavaScript 函數。

7 設定複雜工作流程範例屬性內容

您可以在工作流程編輯器的**一般**索引標籤中設定屬性內容。

8 建立複雜工作流程範例輸入參數配置

您可以在工作流程編輯器的**呈現**索引標籤中，建立輸入參數對話方塊的配置，亦即呈現方式。輸入參數對話方塊會在使用者執行工作流程時開啟，可供使用者用來輸入工作流程執行時使用的輸入參數。

9 驗證和執行複雜工作流程的範例

您建立工作流程後，可驗證工作流程來偵測任何可能的錯誤。如果工作流程沒有任何錯誤，您便可以執行工作流程。

建立複雜工作流程範例

您必須在 Orchestrator 用戶端中建立工作流程，以開始工作流程開發程序。

如需安裝與設定 vCenter Server 的相關資訊，請參閱《vSphere 安裝和設定》說明文件。如需如何設定 Orchestrator 的相關資訊，請參閱《安裝和設定 VMware vRealize Orchestrator》。

必要條件

確認系統上已安裝並設定下列元件。

- vCenter Server，控制包含某些虛擬機器的資源集區
- 工作流程階層清單中的工作流程範例資料夾 (您於 建立簡單工作流程範例 中建立)。

程序

1. 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。
2. 選取**工作流程 > 工作流程範例**。
3. 在**工作流程範例**資料夾上按一下滑鼠右鍵，然後選取**新增工作流程**。
4. 將新的工作流程命名為**拍攝資源集區中所有虛擬機器快照**，然後按一下**確定**。

   工作流程編輯器隨即開啟。
5. 在工作流程編輯器的**一般**索引標籤上按一下版本號碼數字，以增加版本號碼。

   若為初次建立工作流程，請將版本設定為 **0.0.1**。
6. 按一下**伺服器重新啟動行為**值，以設定工作流程在伺服器重新啟動後是否繼續進行。
7. 在**說明**文字方塊中輸入工作流程用途的說明。
8. 按一下**一般**索引標籤底端的**儲存**。

您已建立 [拍攝資源集區中所有虛擬機器快照] 工作流程。

後續步驟

您必須建立自訂動作。

為複雜工作流程建立自訂動作範例

`Check VM` 可編寫指令碼元素會呼叫 Orchestrator API 中不存在的動作。您必須建立 `getVMDiskModes` 動作。

如需建立動作的詳細資料，請參閱第 3 章 開發動作。

必要條件

建立 [拍攝資源集區中所有虛擬機器快照] 工作流程。請參閱建立複雜工作流程範例。

程序

1. 按一下**儲存並關閉**以關閉工作流程編輯器。
2. 在 Orchestrator 用戶端中按一下**動作**視圖。
3. 在動作階層清單的根目錄按一下滑鼠右鍵，並選取**新增模組**。
4. 將新模組命名為 **`com.vmware.example`**。
5. 在 **com.vmware.example** 模組上按一下滑鼠右鍵，然後選取**新增動作**。

6 建立一個名為 getVMDiskModes 的動作。

7 在動作編輯器的**一般**索引標籤中按一下版本數字，以增加版本號碼。

8 在**一般**索引標籤中新增以下的動作說明。

```
This action returns an array containing the disk modes of all disks on a VM.
The elements in the array each have one of the following string values:
- persistent
- independent-persistent
- nonpersistent
- independent-nonpersistent
Legacy values:
- undoable
- append
```

9 按一下**指令碼**索引標籤。

10 在**指令碼**索引標籤的上方窗格中按一下滑鼠右鍵，然後選取**新增參數**以建立下列輸入參數。

- 名稱：vm
- 類型：VC:VirtualMachine
- 說明：**傳回磁碟模式的虛擬機器**

11 在**指令碼**索引標籤底端新增下列指令碼。

下列程式碼會傳回虛擬機器磁碟的磁碟模式陣列。

```
var devicesArray = vm.config.hardware.device;
var retArray = new Array();
if (devicesArray!=null && devicesArray.length!=0) {
    for (i in devicesArray) {
        if (devicesArray[i] instanceof VcVirtualDisk) {
            retArray.push(devicesArray[i].backing.diskMode);
        }
    }
}
return retArray;
```

12 按一下**儲存並關閉**以結束**動作**選擇區。

您已定義 [拍攝資源集區中所有虛擬機器快照] 工作流程需要的自訂動作。

後續步驟

建立工作流程的架構。

建立複雜工作流程範例架構

您可以在工作流程編輯器中建立工作流程的架構。工作流程架構包含工作流程執行的元素，並決定工作流程的邏輯流程。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。
- 為複雜工作流程建立自訂動作範例。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**架構**索引標籤。

2 新增下列架構元素至工作流程架構。

元素類型	元素名稱	架構中的位置
可編寫指令碼工作	**Initializing**	在 Start 元素下方
決定	**VMs to Process?**	在 Initializing 可編寫指令碼工作元素下方
可編寫指令碼工作	**Pool Has No VMs**	在 VMs to Process? 自訂決定元素下方，透過紅色箭頭連結
自訂決定	**Remaining VMs?**	在 VMs to Process? 自訂決定元素右方，透過綠色箭頭連結
動作	getVMDiskModes	在 Remaining VMs? 自訂決定元素右方，透過綠色箭頭連結
自訂決定	**Create Snapshot?**	在 getVMDiskModes 動作元素右方，透過藍色箭頭連結
工作流程	Create a snapshot	在 Create Snapshot? 自訂決定元素上方，透過綠色箭頭連結
可編寫指令碼工作	**VM Snapshots**	在 Create a snapshot 工作流程左方，透過藍色箭頭連結
可編寫指令碼工作	**Increment**	在 VM Snapshots 可編寫指令碼工作元素左方，透過藍色箭頭連結
可編寫指令碼工作	**Set Output**	在 Pool Has No VMs 可編寫指令碼工作元素右方，透過藍色箭頭連結

3 新增 Log Exception 可編寫指令碼工作元素。

a 建立從 [建立快照] 工作流程到 End 元素的例外狀況處理連結。

b 將可編寫指令碼工作元素拖曳至將 [建立快照] 工作流程連結到 End 元素的紅色虛線箭頭。

c 按兩下可編寫指令碼工作元素，並將其重新命名為 **Log Exception**。

d 將 Log Exception 可編寫指令碼工作元素移至 VM Snapshots 可編寫指令碼工作元素上方。

4 取消連結所有 End 元素，但 Set Output 可編寫指令碼工作元素右側的 End 元素除外。

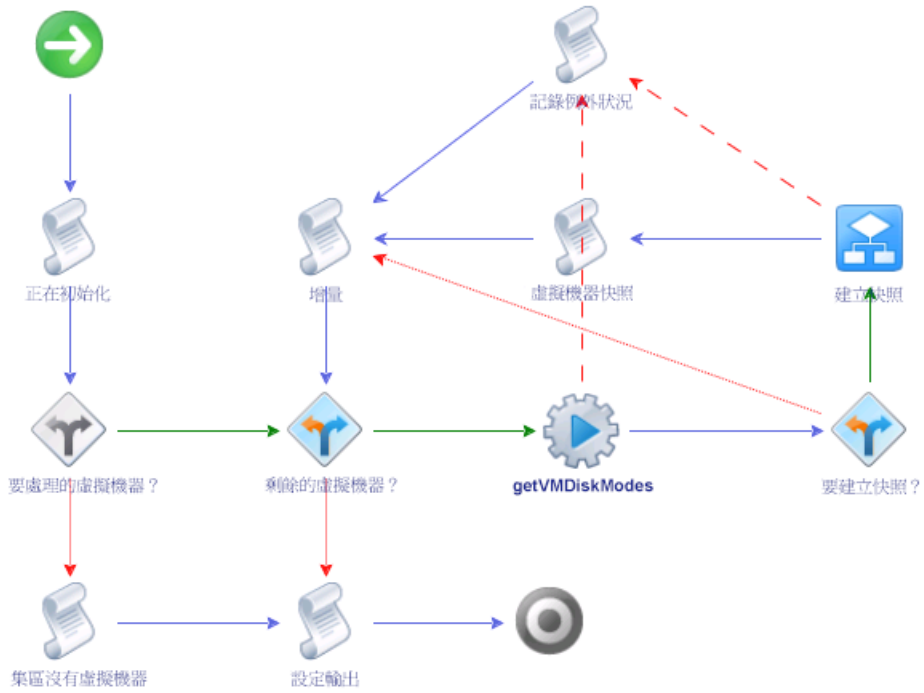
5 如下表所述連結剩餘的元素。

元素	連結至	箭頭類型	說明
getVMDiskModes 動作元素	Log Exception 可編寫指令碼工作元素	紅色虛線	例外狀況處理
Create Snapshot? 自訂決定元素	Increment 可編寫指令碼工作元素	紅色	False 結果
Log Exception 可編寫指令碼工作元素	Increment 可編寫指令碼工作元素	藍色	一般工作流程進展
Increment 可編寫指令碼工作元素	Remaining VMs? 自訂決定元素	藍色	一般工作流程進展
Remaining VMs? 自訂決定元素	Set Output 可編寫指令碼工作元素	紅色	False 結果

6 按一下**架構**索引標籤底端的**儲存**。

[拍攝資源集區中所有虛擬機器快照] 工作流程的已連結元素應如下圖所示。

圖 1-12. 拍攝資源集區中所有虛擬機器快照之範例工作流程的連結



後續步驟

您可以使用工作流程備註選擇性定義工作流程區域。

建立複雜工作流程範例區域

(選擇性) 您可以透過新增工作流程備註反白工作流程的不同區域。建立不同的工作流程區域有助於更容易看懂與瞭解複雜的工作流程架構。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。
- 建立複雜工作流程範例架構。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 使用工作流程備註建立下列工作流程區域。

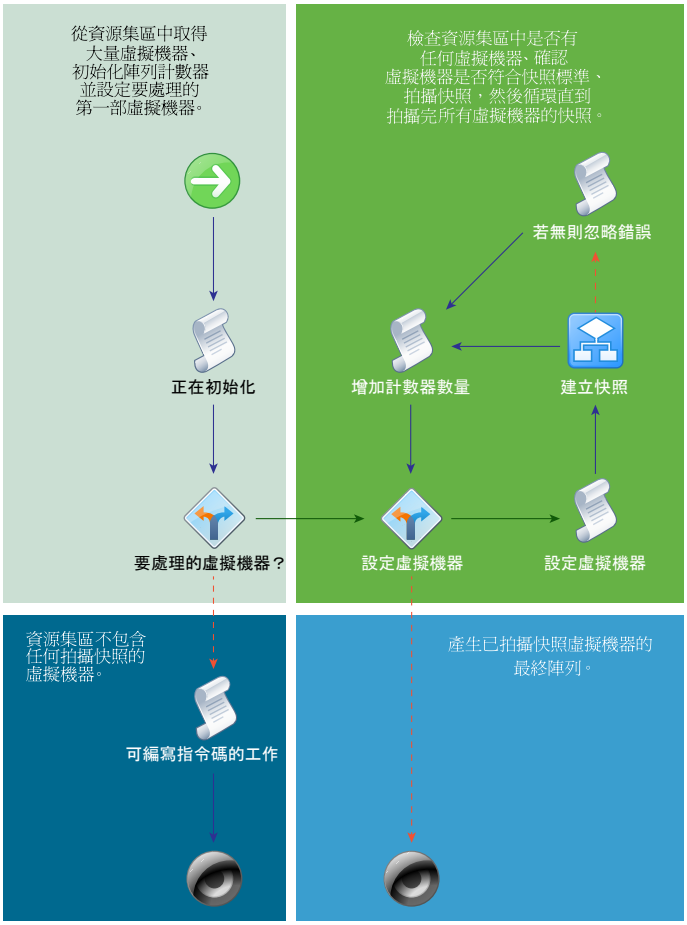
區域中的元素	說明
開始元素；初始化可編寫指令碼工作；要處理的虛擬機器？自訂決定	從資源集區中取得大量虛擬機器、初始化陣列計數器並設定要處理的第一部虛擬機器。
集區沒有虛擬機器可編寫指令碼工作。	資源集區不包含任何拍攝快照的虛擬機器。
有其餘的虛擬機器？自訂決定；getVMDisksModes 動作，建立快照？決定；建立快照工作流程；虛擬機器快照可編寫指令碼工作；增量可編寫指令碼工作；記錄例外狀況可編寫指令碼工作	檢查資源集區中是否有任何虛擬機器、確認虛擬機器是否符合快照標準、拍攝快照，然後循環直到拍攝完所有虛擬機器的快照。
設定輸出可編寫指令碼工作元素；結束元素	產生已拍攝快照虛擬機器的最終陣列。

2 選取工作流程備註，並按下 Ctrl+E 選取背景色彩。

3 按一下工作流程編輯器之**架構**索引標籤底端的**儲存**。

您的工作流程區域應如下圖所示。

圖 1-13. 拍攝資源集區中所有虛擬機器快照之範例工作流程的架構圖



後續步驟

您必須定義工作流程的輸入與輸出參數。

定義複雜工作流程參數的範例

您可在工作流程編輯器中定義工作流程參數。輸入參數會為工作流程提供要處理的資料。輸出參數是工作流程執行完成時所傳回的資料。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。
- 建立複雜工作流程範例架構。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1. 按一下工作流程編輯器中的**輸入**索引標籤。
2. 定義下列輸入參數。
   - 名稱：`resourcePool`
   - 類型：`VC:ResourcePool`
   - 說明：**包含要拍攝其快照之虛擬機器的資源集區。**
3. 按一下工作流程編輯器中的**輸出**索引標籤。
4. 定義下列輸出參數。
   - 名稱：`snapshotVmArrayOut`
   - 類型：`Array/VC:VirtualMachine`
   - 說明：**產生已拍攝快照之虛擬機器的陣列。**

您已定義工作流程的輸入與輸出參數。

後續步驟

您必須定義元素參數之間的繫結。

定義複雜工作流程範例繫結

您可以在工作流程編輯器中將工作流程元素繫結在一起。繫結可定義工作流程的資料流程。您也可以將可編寫指令碼工作元素繫結到其 JavaScript 函數。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。

- 建立複雜工作流程範例架構
- 定義複雜工作流程參數的範例
- 檢閱您必須定義的繫結。請參閱複雜工作流程範例繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**架構**索引標籤。

2 定義繫結。

3 按一下**架構**索引標籤底端的**儲存**。

元素的所有輸入與輸出參數均繫結到適當的參數類型與值。

後續步驟

設定屬性內容。

複雜工作流程範例繫結

繫結定義簡單工作流程範例的動作元素程序如何處理輸入與輸出參數。

「資源集區」工作流程中的「拍攝所有虛擬機器的快照」需要下列輸入與輸出參數繫結。您也會為可編寫指令碼工作元素定義 JavaScript 函數。

當您繫結到現有參數時，繫結會繼承原始參數的類型與說明值。

Initializing 可編寫指令碼工作

Initializing 可編寫指令碼工作元素會初始化工作流程的屬性。下表顯示 Initializing 可編寫指令碼工作元素需要的輸入與輸出參數繫結。

表 1-61. Initializing 可編寫指令碼工作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
`resourcePool`	輸入	繫結	■ 本機參數：`resourcePool` ■ 來源參數：`resourcePool[in-parameter]` ■ 類型：`VC:ResourcePool` ■ 說明：**包含要拍攝其快照之虛擬機器的資源集區**
`allVMs`	輸出	建立	■ 本機參數：`allVMs` ■ 來源參數：`allVMs[attribute]` ■ 類型：`Array/VC:VirtualMachine` ■ 說明：**資源集區中的虛擬機器。**

表 1-61. Initializing 可編寫指令碼工作元素繫結 (續)

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
numberOfVMs	輸出	建立	■ 本機參數：numberOfVMs ■ 來源參數：numberOfVMs[attribute] ■ 類型：數字 ■ 說明：**resourcePool 中的虛擬機器數目**
vmCounter	輸出	建立	■ 本機參數：vmCounter ■ 來源參數：vmCounter[attribute] ■ 類型：數字 ■ 說明：**陣列內虛擬機器的計數器**
vm	輸出	建立	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**已拍攝快照的目前虛擬機器**
snapshotVmArray	輸出	建立	■ 本機參數：snapshotVmArray ■ 來源參數：snapshotVmArray[attribute] ■ 類型：Array/VC:VirtualMachine ■ 說明：**已拍攝快照之虛擬機器的陣列**

Initializing 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Retrieve an array of virtual machines contained in the specified Resource Pool
allVMs = resourcePool.vm;
//Initialize the size of the Array and the first VM to snapshot
if (allVMs!=null && allVMs.length!=0) {
    numberOfVms = allVMs.length;
    vm = allVMs[0];
} else {
    numberOfVms = 0;
}
//Initialize the VM counter
vmCounter = 0;
//Initializing the array of VM snapshots
snapshotVmArray = new Array();
```

VMs to Process? 決定元素

VMs to Process? 決定元素會判斷資源集區中是否有任何要拍攝其快照的虛擬機器存在。下表顯示 VMs to Process? 決定元素需要的繫結。

表 1-62. VMs to Process? 繫結決定元素

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
numberOfVMs	決定	繫結	■ 來源參數：numberOfVMs[attribute] ■ 決定陳述式：大於 ■ 值：0.0 ■ 說明：**resourcePool 中的虛擬機器數目**

Pool Has No VMs 可編寫指令碼工作元素

Pool Has No VMs 可編寫指令碼工作元素記錄以下事實：資源集區未包含 Orchestrator 資料庫中適合的虛擬機器。下表顯示 Pool Has No VMs 可編寫指令碼工作元素需要的繫結。

表 1-63. Pool Has No VMs 可編寫指令碼工作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
resourcePool	輸入	繫結	■ 本機參數：resourcePool ■ 來源參數：resourcePool[in-parameter] ■ 類型：VC:ResourcePool ■ 說明：**包含要拍攝其快照之虛擬機器的資源集區。**

Pool Has No VMs 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Writes the following event in the Orchestrator database
Server.warn("The specified ResourcePool "+resourcePool.name+" does not contain any VMs.");
```

Remaining VMs?自訂決定元素

Remaining VMs?自訂決定元素會判斷資源集區中是否有任何要拍攝其快照的虛擬機器存留。下表顯示 Remaining VMs? 自訂決定元素需要的繫結。

表 1-64. Remaining VMs? 繫結自訂決定元素

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
numberOfVMs	輸入	繫結	■ 來源參數：numberOfVMs[attribute] ■ 決定陳述式：大於 ■ 值：0.0 ■ 說明：**resourcePool 中的虛擬機器數目**
vmCounter	輸入	繫結	■ 本機參數：vmCounter ■ 來源參數：vmCounter[attribute] ■ 類型：數字 ■ 說明：**陣列內虛擬機器的計數器**

Remaining VMs?自訂決定元素執行下列指令碼式函數。

```
//Checks if the workflow has reached the end of the array of VMs
if (vmCounter < numberOfVms) {
    return true;
} else {
    return false;
}
```

getVMDisksModes 動作元素

getVMDisksModes 動作元素包含在虛擬機器中執行的磁碟模式。下表顯示 getVMDisksModes 動作元素需要的繫結。

表 1-65. getVMDisksModes 動作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**已拍攝快照的目前虛擬機器**
actionResult	輸出	建立	■ 本機參數：actionResult ■ 來源參數：vmDisksModes[attribute] ■ 類型：陣列/字串 ■ 說明：**虛擬機器的目前磁碟模式**
errorCode	例外狀況	建立	本機參數：errorCode

Create Snapshot? 自訂決定元素

Create Snapshot? 自訂決定元素會決定是否根據虛擬機器的磁碟模式拍攝虛擬機器的快照。下表顯示 Create Snapshot? 自訂決定元素需要的繫結。

表 1-66. Create Snapshot? 繫結決定元素

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vmDisksMode	輸入	繫結	■ 本機參數：vmDisksMode ■ 來源參數：vmDisksMode[attribute] ■ 類型：陣列/字串 ■ 說明：**虛擬機器的目前磁碟模式**
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**已拍攝快照的目前虛擬機器**

Create Snapshot?自訂決定元素執行下列指令碼式函數。

```
//A snapshot cannot be taken if one of its disks is in independent mode
// (independent-persistent or independent-nonpersistent)
var containsIndependentDisks = false;
if (vmDisksModes!=null && vmDisksModes.length>0) {
    for (i in vmDisksModes) {
        if (vmDisksModes[i].charAt(0)=="i") {
            containsIndependentDisks = true;
        }
    }
} else {
    //if no disk found no need to try to snapshot the VM
    System.warn("Won't snapshot '"+vm.name+"', no disks found");
    return false;
}
if (containsIndependentDisks) {
    System.warn("Won't snapshot '"+vm.name+"', independent disk(s) found");
    return false;
} else {
    System.log("Snapshoting '"+vm.name+"'");
    return true;
}
```

Create a snapshot 工作流程元素

Create a snapshot 工作流程元素會拍攝虛擬機器的快照。下表顯示 Create a snapshot 工作流程元素需要的繫結。

表 1-67. Create a snapshot 工作流程元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**要拍攝其快照的作用中虛擬機器。**
name	輸入	建立	■ 本機參數：name ■ 來源參數：snapshotName[attribute] ■ 類型：字串 ■ 說明：**此快照的名稱。此虛擬機器的名稱不必是唯一的。**
description	輸入	建立	■ 本機參數：description ■ 來源參數：snapshotDescription[attribute] ■ 類型：字串 ■ 說明：**此快照的說明。**

表 1-67. Create a snapshot 工作流程元素繫結 (續)

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
memory	輸入	建立	■ 本機參數：memory ■ 來源參數：snapshotMemory[attribute] ■ 類型：布林值 ■ 值：否 ■ 說明：**如果為 TRUE，則快照中會包含虛擬機器內部狀態的傾印 (記憶體傾印)。**
quiesce	輸入	建立	■ 本機參數：quiesce ■ 來源參數：snapshotQuiesce[attribute] ■ 類型：布林值 ■ 值：是 ■ 說明：**如果為 TRUE 且在拍攝快照時虛擬機器的電源是開啟的，則會使用 VMware Tools 來靜止虛擬機器中的檔案系統。**
snapshot	輸出	建立	■ 本機參數：snapshot ■ 來源參數：NULL ■ 類型：VC:VirtualMachineSnapshot ■ 說明：**拍攝的快照。**
errorCode	例外狀況	建立	本機參數：errorCode

虛擬機器快照可編寫指令碼工作元素

虛擬機器快照可編寫指令碼工作元素會將快照新增至陣列。下表顯示「虛擬機器快照」可編寫指令碼工作元素需要的繫結。

表 1-68.「虛擬機器快照可編寫指令碼工作元素」繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**要拍攝其快照的作用中虛擬機器。**
snapshotVmArray	輸入	繫結	■ 本機參數：snapshotVmArray ■ 來源參數：snapshotVmArray[attribute] ■ 類型：Array/VC:VirtualMachine ■ 說明：**已拍攝快照之虛擬機器的陣列**
snapshotVmArray	輸出	繫結	■ 本機參數：snapshotVmArray ■ 來源參數：snapshotVmArray[attribute] ■ 類型：Array/VC:VirtualMachine ■ 說明：**已拍攝快照之虛擬機器的陣列**

VM Snapshots 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Writes the following event in the Orchestrator database
Server.log("Successfully took snapshot of the VM '"+vm.name);
//Inserts the VM snapshot in an array
snapshotVmArray.push(vm);
```

Increment 可編寫指令碼工作元素

Increment 可編寫指令碼工作元素會增加計算陣列中虛擬機器數目的計數器計數。下表顯示 Increment 可編寫指令碼工作元素需要的繫結。

表 1-69. Increment 可編寫指令碼工作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vmCounter	輸入	繫結	■ 本機參數：vmCounter ■ 來源參數：vmCounter[attribute] ■ 類型：數字 ■ 說明：**陣列內虛擬機器的計數器**
allVMs	輸入	繫結	■ 本機參數：allVMs ■ 來源參數：allVMs[attribute] ■ 類型：Array/VC:VirtualMachine ■ 說明：**資源集區中的虛擬機器。**

表 1-69. Increment 可編寫指令碼工作元素繫結 (續)

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vmCounter	輸出	繫結	■ 本機參數：vmCounter ■ 來源參數：vmCounter[attribute] ■ 類型：數字 ■ 說明：**陣列內虛擬機器的計數器**
vm	輸出	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**已拍攝快照的目前虛擬機器**

Increment 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Increases the array VM counter
vmCounter++;
//Sets the next VM to be snapshot in the attribute vm
vm = allVMs[vmCounter];
```

Log Exception 可編寫指令碼工作元素

Log Exception 可編寫指令碼工作元素會處理工作流程與動作元素中的例外狀況。下表顯示 Log Exception 可編寫指令碼工作元素需要的繫結。

表 1-70. Log Exception 工作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
vm	輸入	繫結	■ 本機參數：vm ■ 來源參數：vm[attribute] ■ 類型：VC:VirtualMachine ■ 說明：**已拍攝快照的目前虛擬機器**
errorCode	輸入	繫結	■ 本機參數：errorCode ■ 來源參數：errorCode[attribute] ■ 類型：字串 ■ 說明：**拍攝虛擬機器快照時捕捉到的例外狀況**

Log Exception 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Writes  the following event in the Orchestrator database
Server.error("Coudln't snapshot the VM '"+vm.name+"', exception: "+errorCode);
```

Set Output 可編寫指令碼工作元素

Set Output 可編寫指令碼會產生工作流程的輸出參數，其中包含已拍攝其快照的虛擬機器陣列。下表顯示 Set Output 可編寫指令碼工作元素需要的繫結。

表 1-71. Set Output 工作元素繫結

參數名稱	繫結類型	繫結到現有參數或建立參數?	繫結值
snapshotVmArray	輸入	繫結	■ 本機參數：snapshotVmArray ■ 來源參數：snapshotVmArray[attribute] ■ 類型：Array/VC:VirtualMachine ■ 說明：**已拍攝快照之虛擬機器的陣列**
snapshotVmArrayOut	輸出	繫結	■ 本機參數：snapshotVmArrayOut ■ 來源參數：snapshotVmArrayOut[out-parameter] ■ 類型：Array/VC:VirtualMachine ■ 說明：**產生已拍攝快照之虛擬機器的陣列**

Set Output 可編寫指令碼工作元素執行下列指令碼式函數。

```
//Passes the value of the internal attribute to a workflow output parameter
snapshotVmArrayOut = snapshotVmArray;
```

設定複雜工作流程範例屬性內容

您可以在工作流程編輯器的**一般**索引標籤中設定屬性內容。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。
- 建立複雜工作流程範例架構。
- 定義複雜工作流程範例繫結。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下**一般**索引標籤。

2 選取下列屬性的唯讀核取方塊，使這些成為唯讀常數：

- snapshotName
- snapshotDescription
- snapshotMemory
- snapshotQuiesce

您已經定義哪些工作流程的屬性是常數和變數。

後續步驟

您必須建立工作流程呈現，才能建立輸入參數對話方塊的配置，以便使用者在執行工作流程時指定工作流程的輸入參數值。

建立複雜工作流程範例輸入參數配置

您可以在工作流程編輯器的**呈現**索引標籤中，建立輸入參數對話方塊的配置，亦即呈現方式。輸入參數對話方塊會在使用者執行工作流程時開啟，可供使用者用來輸入工作流程執行時使用的輸入參數。

必要條件

請完成以下工作。

- 建立複雜工作流程範例。
- 建立複雜工作流程範例架構。
- 定義複雜工作流程參數的範例。
- 定義複雜工作流程範例繫結。
- 設定複雜工作流程範例屬性內容。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1 按一下工作流程編輯器中的**呈現**索引標籤。

[拍攝資源集區中所有虛擬機器快照] 工作流程只有一個輸入參數，因此能夠簡單直接地建立呈現。

2 在呈現階層清單的**呈現**節點上按一下滑鼠右鍵，然後選取**建立顯示群組**。

3 刪除出現在**新群組**元素上方的**新步驟**元素。

4 按兩下**新群組**元素，並將群組名稱變更為**資源集區**。

5 在**呈現**索引標籤底端**一般**索引標籤的**說明**文字方塊中，輸入**資源集區**顯示群組的說明。

例如，**輸入包含要拍攝快照之虛擬機器的資源集區名稱。**

6 按一下 `(VC:ResourcePool)resourcePool` 參數。

7 按一下 `(VC:ResourcePool)resourcePool` 的**內容**索引標籤。

8 在**內容**索引標籤內按一下滑鼠右鍵，然後選取**新增內容 > 強制輸入**。

9 在**內容**索引標籤內按一下滑鼠右鍵，然後選取**新增內容 > 選取值作為**。

設定此內容時，您可以設定使用者如何選取 `(VC:ResourcePool)resourcePool` 輸入參數的值。

10 將 `(VC:ResourcePool)resourcePool` 參數拖曳至**資源集區**顯示群組下方。

您已建立當使用者執行工作流程時顯示的對話方塊配置。

後續步驟

您已完成複雜工作流程範例的開發。現在您可以驗證與執行工作流程。

驗證和執行複雜工作流程的範例

您建立工作流程後，可驗證工作流程來偵測任何可能的錯誤。如果工作流程沒有任何錯誤，您便可以執行工作流程。

必要條件

建立工作流程、配置其架構、定義連結和繫結、定義參數內容，並建立輸入參數對話方塊的呈現。

請完成以下工作。

- 建立複雜工作流程範例。
- 為複雜工作流程建立自訂動作範例。
- 建立複雜工作流程範例架構。
- 定義複雜工作流程參數的範例。
- 定義複雜工作流程範例繫結。
- 設定複雜工作流程範例屬性內容。
- 建立複雜工作流程範例輸入參數配置。
- 在工作流程編輯器中開啟工作流程以進行編輯。

程序

1. 在工作流程編輯器的**架構**索引標籤中，按一下**驗證**。

   驗證工具會偵測工作流程之定義中的任何錯誤。
2. 消除任何錯誤後，請按一下工作流程編輯器底端的**儲存並關閉**。

   隨即返回 Orchestrator 用戶端。
3. 按一下**工作流程**視圖。
4. 在工作流程階層清單中，選取**工作流程範例 > 拍攝資源集區中全部虛擬機器的快照**。
5. 在**拍攝資源集區中全部虛擬機器的快照**工作流程上按一下滑鼠右鍵，並選取**啟動工作流程**。

   輸入參數對話方塊隨即開啟，並提示包含您要拍攝快照之虛擬機器的資源集區。
6. 按一下**提交**執行工作流程。

   工作流程 Token 隨即出現在 [拍攝資源集區中全部虛擬機器的快照] 工作流程之下。
7. 按一下工作流程 Token 以追蹤工作流程執行的進度。

如果工作流程成功執行，工作流程將拍攝所選資源集區中全部虛擬機器的快照。

後續步驟

您可產生文件來檢視工作流程的資訊。請參閱產生工作流程說明文件。

指令碼

2

Orchestrator 使用 JavaScript 建立建置區塊，以供您建立動作、工作流程元素和原則；這些原則可以存取您外掛至 Orchestrator 之技術的 API。

Orchestrator 使用 Mozilla Rhino 1.7R4 JavaScript 引擎做為指令碼引擎。此指令碼引擎可進行變數類型檢查、命名空間管理、自動完成和例外狀況處理。

Orchestrator 工作流程引擎可讓您使用基本 JavaScript 語言功能，例如 if、迴圈、陣列和字串。您可以使用 Orchestrator API 提供之指令碼中的物件，或其他任何 API (您透過外掛程式匯入 Orchestrator，並對應至 JavaScript 物件的 API) 中的物件。如需 Rhino 的相關資訊，請參閱 Mozilla Rhino 網站。

本章節討論下列主題：

需要指令碼的 Orchestrator 元素

並非所有 Orchestrator 元素都需要您撰寫指令碼。若要使您的應用程式達到最大的彈性，您可以新增 JavaScript 函數來自訂特定元素。

您可以在下列 Orchestrator 元素中新增指令碼。

動作

動作是指令碼式函數。您可以將您對於動作撰寫的指令碼侷限於單一作業，以便動作更有機會由其他元素 (例如其他工作流程) 重複使用。或者，動作可包含許多作業，藉以限制工作流程的複雜度，不過，這也會降低重複使用動作的能力。

原則

您可以使用監看觸發器事件的指令碼來設定原則。觸發器事件發生時，原則將啟動您在指令碼中定義的協調作業。

工作流程

[可編寫指令碼工作] 工作流程元素可讓您撰寫能在工作流程中使用的自訂指令碼式作業或作業序列。對於指令碼的自訂決定元素，您也可以定義傳回 `true` 或 `false` 的布林值決定陳述式。

Orchestrator 中的 Mozilla Rhino 執行限制

Orchestrator 使用 Mozilla Rhino 1.7R4 JavaScript 引擎。不過在 Orchestrator 中執行的 Rhino 會出現部分限制。

寫入工作流程指令碼時，您必須考慮下列 Orchestrator 的 Mozilla Rhino 執行限制。

- 執行工作流程時，在工作流程元素間傳遞的不屬於 JavaScript 物件。在元素間傳遞的內容為擁有 JavaScript 映像的序列化 Java 物件。因此，您無法使用整個 JavaScript 語言，只能使用出現在 API Explorer 中的類別。您無法在工作流程元素間傳遞函式物件。
- Orchestrator 在不屬於 Rhino 根內容的可編寫指令碼工作元素中執行代碼。Orchestrator 明確換行可編寫指令碼工作元素及動作至 JavaScript 函式，然後再一起執行。包含 `System.log(this);` 的可編寫指令碼工作元素不會以和標準 Rhino 執行相同的方式顯示全域物件 `this`。
- 您只能呼叫從指令碼傳回非序列化物件的動作，而非從工作流程傳回的物件。若要呼叫傳回非序列化物件的動作，您必須使用 `System.getModule`*`ModuleName`*`.action()` 法呼叫動作，以寫入可編寫指令碼工作元素。
- 工作流程驗證不會檢查工作流程屬性型別是否與動作或子工作流程的輸入型別不同。若您變更工作流程輸入參數的型別，例如從 `VIM3:VirtualMachine` 變更為 `VC:VirtualMachine`，但您卻未更新任何使用原始輸入型別的可編寫指令碼工作或動作，則該工作流程會驗證但不執行。

使用 Orchestrator 指令碼 API

Orchestrator API 會公開技術的所有物件和功能，而 Orchestrator 則會以 JavaScript 物件的形式和各種方法，透過其外掛程式存取這些物件和功能。

例如，您可以透過 Orchestrator API 存取 vCenter Server API 的 JavaScript 實作，在您建立的指令碼式元素中加入 vCenter 作業。您也可以從您在 Orchestrator 伺服器中安裝的其他所有外掛程式來存取物件的 JavaScript 實作。如果您建立第三方應用程式的自訂外掛程式，即可將其 API 的物件對應至 Orchestrator API 公開的 JavaScript 物件。

程序

1 從工作流程編輯器存取指令碼引擎

Orchestrator 指令碼引擎使用 Mozilla Rhino 1.7R4 JavaScript 引擎幫助您為工作流程的指令碼式元素撰寫指令碼。您可以從工作流程編輯器的**指令碼**索引標籤，存取指令碼式工作流程元素的指令碼引擎。

2 從動作或原則編輯器存取指令碼引擎

Orchestrator 指令碼引擎使用 Mozilla Rhino JavaScript 引擎幫助您撰寫動作或原則的指令碼。您可以從動作與原則編輯器的**指令碼**索引標籤，存取動作與原則的指令碼引擎。

3 存取 Orchestrator API Explorer

Orchestrator 提供 API Explorer，可供您用於搜尋 Orchestrator API，並參閱您可在指令碼式元素中所使用 JavaScript 物件的說明文件。

4 使用 Orchestrator API Explorer 尋找物件

Orchestrator API 會公開所有外掛技術的 API，包括整個 vCenter Server API。Orchestrator API Explorer 可供您尋找您需要新增至指令碼的物件。

5 撰寫指令碼

Orchestrator 指令碼引擎可幫助您撰寫指令碼。自動插入函數及自動完成指令碼行可加速指令碼程序，並減少撰寫指令碼時發生錯誤的可能性。

6 新增參數至指令碼

Orchestrator 指令碼引擎可幫助您將可用參數匯入至指令碼。

7 從 JavaScript 及工作流程存取 Orchestrator 伺服器檔案系統

Orchestrator 限制從 JavaScript 及工作流程指定目錄存取 Orchestrator 伺服器檔案系統。

8 從 JavaScript 存取 Java 類別

依預設，Orchestrator 會限制 JavaScript 存取有限的 Java 類別。若您需要 JavaScript 存取更廣泛的 Java 類別，您必須正確設定 Orchestrator 系統，以允許此存取。

9 從 JavaScript 存取作業系統命令

Orchestrator API 提供的指令碼類別 `Command` 可執行 Orchestrator 伺服器主機作業系統中的命令。若要防止未授權存取 Orchestrator 伺服器主機，依預設 Orchestrator 應用程式沒有執行 `Command` 類別的權限。

從工作流程編輯器存取指令碼引擎

Orchestrator 指令碼引擎使用 Mozilla Rhino 1.7R4 JavaScript 引擎幫助您為工作流程的指令碼式元素撰寫指令碼。您可以從工作流程編輯器的**指令碼**索引標籤，存取指令碼式工作流程元素的指令碼引擎。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 在 Orchestrator 用戶端**工作流程**視圖的工作流程按一下滑鼠右鍵，然後選取**編輯**。

3 按一下工作流程編輯器中的**架構**索引標籤。

4 將可編寫指令碼工作元素或自訂決定元素新增至工作流程架構。

5 按一下可編寫指令碼元素的**指令碼**索引標籤。

您已存取指令碼引擎定義工作流程元素的指令碼式函數。**指令碼**索引標籤可讓您瀏覽 API、參閱物件的相關說明文件、搜尋物件及撰寫 JavaScript。

後續步驟

使用 API Explorer 搜尋 Orchestrator API。

從動作或原則編輯器存取指令碼引擎

Orchestrator 指令碼引擎使用 Mozilla Rhino JavaScript 引擎幫助您撰寫動作或原則的指令碼。您可以從動作與原則編輯器的**指令碼**索引標籤，存取動作與原則的指令碼引擎。

程序

1 視您要編輯其指令碼的元素類型而定，從 Orchestrator 用戶端的下拉式功能表選取選項。

選項	說明
設計	選取此選項可編輯動作元素的指令碼。
執行	選取此選項可編輯原則的指令碼。

2 在**動作**或**原則**視圖中的動作或原則按一下滑鼠右鍵，然後選取**編輯**。

3 按一下動作或原則編輯器中的**指令碼**索引標籤。

您已存取指令碼引擎定義動作或原則元素的指令碼式函數。**指令碼**索引標籤可讓您瀏覽 API、參閱物件的相關說明文件、搜尋物件及撰寫 JavaScript。

後續步驟

使用 API Explorer 搜尋 Orchestrator API。

存取 Orchestrator API Explorer

Orchestrator 提供 API Explorer，可供您用於搜尋 Orchestrator API，並參閱您可在指令碼式元素中所使用 JavaScript 物件的說明文件。

您可以在 Orchestrator 說明文件首頁查詢線上版的指令碼 API 瞭解 vCenter Server 外掛程式。

程序

1 登入 Orchestrator 用戶端。

2 選取**工具 > API Explorer**。

API Explorer 顯示。您可以用於搜尋 Orchestrator API 的所有物件與函數。

後續步驟

使用 API Explorer 為可編寫指令碼的元素撰寫指令碼。

使用 Orchestrator API Explorer 尋找物件

Orchestrator API 會公開所有外掛技術的 API，包括整個 vCenter Server API。Orchestrator API Explorer 可供您尋找您需要新增至指令碼的物件。

必要條件

開啟 API Explorer。

程序

1 在 API Explorer **搜尋**文字方塊中輸入物件的名稱或一部分名稱，然後按一下**搜尋**。

若要將您的搜尋限制於特定物件類型，請取消勾選或勾選**指令碼類別**、**屬性及方法**與**類型及列舉**核取方塊。

2 按兩下建議清單中的元素。

物件將在左邊的階層清單中反白顯示。階層清單下的說明文件窗格將顯示物件的資訊。

後續步驟

在指令碼中使用您找到的物件。

API Explorer 中的 JavaScript 物件

Orchestrator API Explorer 會識別不同類型的 JavaScript 物件，並在**指令碼**索引標籤或 API Explorer 對話方塊左邊的階層樹狀結構中將這些物件分組。API Explorer 使用圖示協助您識別不同類型的物件。

下表說明 Orchestrator API 的物件，並顯示其圖示。

表 2-1. Orchestrator API 中的 JavaScript 物件

Object	階層清單中的圖示	說明
類型		類型
函數集合		包含靜態方法的內部類型
原始		原始類型
Object		標準 Orchestrator 指令碼物件
屬性		JavaScript 屬性
方法		JavaScript 方法
建構函數		JavaScript 建構函數
列舉		JavaScript 列舉
字串集合		字串集合、預設值
模組		動作的集合
外掛程式	外掛程式定義的映像	外掛程式向 Orchestrator 公開的 API

撰寫指令碼

Orchestrator 指令碼引擎可幫助您撰寫指令碼。自動插入函數及自動完成指令碼行可加速指令碼程序，並減少撰寫指令碼時發生錯誤的可能性。

必要條件

開啟要編輯的指令碼式元素，然後按一下其**指令碼**索引標籤。

程序

1 瀏覽**指令碼**索引標籤左側的物件階層清單，或使用 API Explorer 搜尋函數，以選取要新增至指令碼的類型、類別或方法。

2 在類型、類別或方法上按一下滑鼠右鍵，然後選取**複製**。

如果指令碼引擎不允許您複製所選的元素，則此物件不可能在指令碼內容中。

3 在指令碼板中按一下滑鼠右鍵，然後將複製的元素貼到指令碼中適當的位置上。

指令碼引擎會將該元素輸入到指令碼中，再補上其建構函數與執行個體名稱加以完成。

例如，如果您複製 Date 物件，則指令碼引擎會將下列程式碼貼到指令碼中。

```
var myDate = new Date();
```

4 複製並貼上要新增至指令碼中的方法。

指令碼引擎完成方法呼叫，並新增必要屬性。

例如，如果您從 com.vmware.library.vc.vm 模組複製 cloneVM() 方法，則指令碼引擎會將下列程式碼貼到指令碼中。

```
System.getModule("com.vmware.library.vc.vm").cloneVM(vm,folder,name,spec)
```

指令碼引擎會反白顯示您已在元素中定義的參數。任何未定義的參數仍保持為未反白顯示。

5 將游標放置在您貼到指令碼中元素的結尾，然後按下 Ctrl+space 從該物件會呼叫的可能方法與屬性內容清單中選取。

備註 自動完成目前為實驗性功能。

您已將物件與函數新增至指令碼中。

後續步驟

新增參數至指令碼。

指令碼關鍵字色彩編碼

當您在執行指令工作流程元素的 **指令碼** 索引標籤上新增指令碼時，特定型別的關鍵字會出現不同的色彩強化該代碼的可讀性。

除非另外聲明，否則所有指令碼皆顯示為標準黑色字型。

表 2-2. 指令碼關鍵字色彩編碼

關鍵字型別	指令碼索引標籤中的文字色彩
標準 JavaScript 關鍵字，例如 if、else、for 和 new	粗體黑色
變數宣告，意即 var	綠色
迴路中的輔助按鍵，例如 in	紅色
空值變數	紫色
非空值變數	綠色
色彩註解	斜體灰色
Orchestrator 外掛程式物件型別，例如 VC:VirtualMachine 或 VC:Host	綠色
輸出文字	綠色
工作流程屬性	粉紅色

表 2-2. 指令碼關鍵字色彩編碼 (續)

關鍵字型別	指令碼索引標籤中的文字色彩
工作流程輸入	粉紅色
工作流程輸出	粉紅色

新增參數至指令碼

Orchestrator 指令碼引擎可幫助您將可用參數匯入至指令碼。

如果您已定義正在編輯元素的參數，這些參數會顯示成**指令碼**索引標籤工具列中的連結。

必要條件

已開啟指令碼式元素以供編輯，其**指令碼**索引標籤也已開啟。

程序

1 在**指令碼**索引標籤的指令碼板上，將游標移到指令碼中的適當位置。

2 按一下**指令碼**索引標籤工具列中的參數連結。

Orchestrator 會將參數插入游標的位置。

3 將含空值的參數插入指令碼中。

如果您將空值傳遞給原始類型，例如整數、布林值與字串，則 Orchestrator 指令碼 API 會自動設定此引數的預設值。

您已將參數新增至指令碼中。

後續步驟

在指令碼中新增 Java 類別的存取權。

從 JavaScript 及工作流程存取 Orchestrator 伺服器檔案系統

Orchestrator 限制從 JavaScript 及工作流程指定目錄存取 Orchestrator 伺服器檔案系統。

在永久目錄 `c:\orchestrator` 中，JavaScript 函式及工作流程僅擁有讀取、寫入及執行權限。

Orchestrator 管理員可透過設定系統內容，修改 JavaScript 函式和工作流程擁有讀取、寫入及執行存取權限的資料夾。如需有關設定系統內容的資訊，請參閱安裝及設定 VMware vRealize Orchestrator。

JavaScript 函式及工作流程在伺服器系統預設暫存 I/O 資料夾中也擁有讀取、寫入和執行權限。寫入預設暫存 I/O 資料夾是以完整權限存取檔案系統唯一可攜、保證及組態獨立的方式。然而，您寫入暫存 I/O 資料夾的檔案會在您重新啟動伺服器時遺失。

您可呼叫 JavaScript 函式中的 `System.getTempDirectory` 法取得預設暫存 I/O 資料夾。

使用 System.getTempDirectory 法存取伺服器檔案系統

作為寫入 Orchestrator 伺服器系統資料夾的替代方案中，管理員已設定適當的權限，您可寫入預設暫存 I/O 資料夾。

依預設，在預設暫存 I/O 資料夾中 Orchestrator 擁有完整讀取、寫入及執行權限。您可使用 JavaScript 函式中的 `System.getTempDirectory` 法取得預設暫存 I/O 資料夾

程序

- 在 JavaScript 函式中加入下列代碼行存取 `java.io.temp-dir` 資料夾。

```
var tempDir = System.getTempDirectory()
```

從 JavaScript 存取 Java 類別

依預設，Orchestrator 會限制 JavaScript 存取有限的 Java 類別。若您需要 JavaScript 存取更廣泛的 Java 類別，您必須正確設定 Orchestrator 系統，以允許此存取。

依預設，Orchestrator JavaScript 引擎只能存取 `java.util.*` 套件中的類別。

Orchestrator 管理員可允許透過正確設定系統，存取 JavaScript 函式中的其他 Java 類別。如需有關設定系統內容的資訊，請參閱安裝及設定 VMware vRealize Orchestrator。

從 JavaScript 存取作業系統命令

Orchestrator API 提供的指令碼類別 `Command` 可執行 Orchestrator 伺服器主機作業系統中的命令。若要防止未授權存取 Orchestrator 伺服器主機，依預設 Orchestrator 應用程式沒有執行 `Command` 類別的權限。

Orchestrator 管理員可允許透過正確設定 `com.vmware.js.allow-local-process=true` 系統，存取 `Command` 指令碼類別。

如需設定系統內容的相關資訊，請參閱 安裝和設定 VMware vCenter Orchestrator。

如需設定系統內容的相關資訊，請參閱 安裝和設定 VMware vCenter Orchestrator。

搭配 vCenter Server 外掛程式使用 XPath 運算式

您可以在 vCenter Server 外掛程式中使用 finder 方法查詢 vCenter Server 詳細目錄物件。您可以使用 XPath 運算式定義搜尋參數。

vCenter Server 外掛程式包含物件 finder 方法的集合，例如 `getAllDatastores()`、`getAllResourcePools()`、`findAllForType()`。您可以使用這些方法存取將連接 Orchestrator 伺服器的 vCenter Server 執行個體列出的詳細目錄，並依照識別碼、名稱或其他內容搜尋物件。

由於效能因素，finder 方法不會傳回已查詢物件的任何內容，除非您在搜尋查詢中指定一組內容。

您可以在 Orchestrator 說明文件首頁查詢線上版的指令碼 API 瞭解 vCenter Server 外掛程式。

重要 以 XPath 運算式為基礎的查詢可能影響 Orchestrator 效能，因為 finder 方法會傳回 vCenter Server 端指定類型的所有物件，而且查詢篩選器將套用於 vCenter Server 外掛程式端。

搭配 vCenter Server 外掛程式使用 XPath 運算式

您叫用 finder 方法時，可以使用以 XPath 查詢語言為基礎的運算式。搜尋會傳回符合 XPath 運算式的所有詳細目錄物件。如果您要查詢任何內容，可以透過字串陣列的形式將這些內容加入於搜尋指令碼。

下列 JavaScript 範例使用 `VcPlugin` 指令碼物件和 XPath 運算式傳回屬於 vCenter Server 受管理物件一部分而且名稱包含字串 **ds** 的所有資料存放區物件名稱。

```
var datastores = VcPlugin.getAllDatastores(null, "xpath:name[contains(.,'ds')]");
for each (datastore in datastores){
     System.log(datastore.name);
 }
```

使用 `Server` 指令碼物件和 `findAllForType` finder 方法可叫用同一個 XPath 運算式。

```
var datastores = Server.findAllForType("VC:Datastore", "xpath:name[contains(.,'ds')]");
for each (datastore in datastores){
     System.log(datastore.name);
 }
```

下列指令碼範例會傳回識別碼開頭為數字 **1** 的所有主機系統物件。

```
var hosts = VcPlugin.getAllHostSystems(null, "xpath:id[starts-with(.,'1')]");
for each (host in hosts){
     System.log(host.name);
}
```

下列指令碼會傳回名稱包含大寫或小寫字母字串 **DC** 的所有資料中心物件名稱和識別碼。指令碼也會擷取**標記**內容。

```
var datacenters = VcPlugin.getAllDatacenters(['tag'], "xpath:name[contains(translate(., 'DC', 'dc'),
'dc')]");
for each (datacenter in datacenters){
     System.log(datacenter.name + " "  +  datacenter.id);
}
```

例外狀況處理準則

Mozilla Rhino JavaScript 引擎的 Orchestrator 實作支援例外狀況處理，讓您能夠處理錯誤。您在指令碼中撰寫例外狀況處理常式時，必須採用下列準則。

- 使用下列歐洲電腦製造商協會(ECMA) 的錯誤類型。使用 `Error` 做為外掛程式函數傳回的一般例外狀況，並使用下列特定錯誤類型。
  - `TypeError`
  - `RangeError`
  - `EvalError`
  - `ReferenceError`
  - `URIError`
  - `SyntaxError`

下列範例會顯示 URIError 定義。

```
try {
     ...
        throw new URIError("VirtualMachine with ID 'vm-0056'
                       not found on 'vcenter-test-1'") ;
     ...
} catch ( e if e instanceof URIError ) {

}
```

- 指令碼未偵測到的所有例外狀況，一定是格式為 <type>:SPACE<human readable message> 的簡單字串物件，如下例所示。

```
throw "ValidationError: The input parameter 'myParam' of type 'string' is too short."
```

- 盡量清楚地寫出人類可理解的訊息。
- 簡單字串例外狀況類型檢查必須使用下列模式。

```
try {
      throw "VMwareNoSpaceLeftOnDatastore: Datastore 'myDatastore' has no space left" ;
} catch ( e if (typeof(e)=="string" && e.indexOf("VMwareNoSpaceLeftOnDatastore:") == 0) ) {
      System.log("No space left on device") ;
      // Do something useful here
}
```

- 簡單字串例外狀況類型檢查，必須在工作流程裡的指令碼式元素中採用下列模式。

```
if (typeof(errorCode)=="string"
      && errorCode.indexOf("VMwareNoSpaceLeftOnDatastore:")
      == 0) {
      // Do something useful here
}
```

Orchestrator JavaScript 範例

您可以剪下、貼上和改寫 Orchestrator JavaScript 範例來協助您撰寫一般協調工作的 JavaScript。

- 基本指令碼範例

  Workflow 指令碼式元素、動作及原則需要一般工作的基本指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- 電子郵件指令碼範例

  Workflow 指令碼式元素可能包含一般電子郵件相關工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- 檔案系統指令碼範例

  Workflow 指令碼式元素、動作及原則需要一般檔案系統工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- LDAP 指令碼範例

  Workflow 指令碼式元素、動作和原則需要一般 LDAP 工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- 記錄指令碼範例

  Workflow 指令碼式元素、動作和原則需要一般記錄工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- 網路指令碼範例

  Workflow 指令碼式元素、動作和原則需要一般網路工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

- 工作流程指令碼範例

  Workflow 指令碼式元素、動作和原則需要一般工作流程工作的指令碼範例。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

基本指令碼範例

Workflow 指令碼式元素、動作及原則需要一般工作的基本指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

存取 XML 文件

下列 JavaScript 範例可讓您使用 Orchestrator JavaScript API 中的 ECMAScript for XML (E4X) 實作，以存取 JavaScript 中的 XML 文件。

備註 除了在 JavaScript API 中實作 E4X 外，Orchestrator 也會在 XML 外掛程式中提供文件物件模型 (DOM) XML 實作。如需 XML 外掛程式及其範例工作流程的相關資訊，請參閱《使用 vRealize Orchestrator 外掛程式》。

```
var people = <people>
                <person id="1">
                    <name>Moe</name>
                </person>
                <person id="2">
                    <name>Larry</name>
                </person>
             </people>;

System.log("'people' = " + people);

// built-in XML type
System.log("'people' is of type : " + typeof(people));

// list-like interface System.log("which contains a list of " +
people.person.length() + " persons");
System.log("whose first element is : " + people.person[0]);

// attribute 'id' is mapped to field '@id'
people.person[0].@id='47';
```

```
// change Moe's id to 47
// also supports search by constraints
System.log("Moe's id is now : " + people.person.(name=='Moe').@id);

// suppress Moe from the list
delete people.person[0];
System.log("Moe is now removed.");

// new (sub-)document can be built from a string
people.person[1] = new XML("<person id=\"3\"><name>James</name></person>");
System.log("Added James to the list, which is now :");
for each(var person in people..person)

for each(var person in people..person){
    System.log("- " + person.name + " (id=" + person.@id + ")");
}
```

從 Hashtable 設定與取得內容

下列 JavaScript 範例會在 Hashtable 中設定內容，並從 Hashtable 取得內容。在下列範例中，金鑰一律為字串，值為物件、數字、布林值或字串。

```
var table = new Properties() ;
table.put("myKey",new Date()) ;
// get the object back
var myDate= table.get("myKey") ;
System.log("Date is : "+myDate) ;
```

取代字串內容

下列 JavaScript 範例會使用新的內容取代字串內容。

```
var str1 = "'hello'" ;
var reg = new RegExp("(')", "g");
var str2 = str1.replace(reg,"\\'") ;
System.log(""+str2) ; // result : \'hello\'
```

比較類型

下列 JavaScript 範例會檢查物件是否符合指定的物件類型。

```
var path = 'myurl/test';
if(typeof(path, string)){
  throw("string");
else {
  throw("other");
}
```

在 Orchestrator 伺服器中執行命令

下列 JavaScript 範例允許您在 Orchestrator 伺服器上執行命令列。使用與用來啟動伺服器之認證相同的認證。

備註 依預設，檔案系統的存取受到限制。

```
var cmd = new Command("ls -al") ;
cmd.execute(true) ;
System.log(cmd.output) ;
```

電子郵件指令碼範例

Workflow 指令碼式元素可能包含一般電子郵件相關工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

當您執行郵件工作流程時，它會使用您在設定郵件工作流程中設定的預設郵件伺服器組態。您可以使用輸入參數，或在 workflow 指令碼式元素中定義自訂值，來覆寫預設值。

取得電子郵件地址

下列 JavaScript 範例可取得執行中指令碼的目前擁有者的電子郵件地址。

```
var emailAddress = Server.getRunningUser().emailAddress ;
```

傳送電子郵件

下列 JavaScript 範例會透過 SMTP 伺服器，將含已定義內容的電子郵件傳送給已定義的收件者。

```
var message = new EmailMessage() ;
message.smtpHost = "smtpHost" ;
message.subject= "my subject" ;
message.toAddress = "receiver@vmware.com" ;
message.fromAddress = "sender@vmware.com" ;
message.addMimePart("This is a simple message","text/html") ;
message.sendMessage() ;
```

擷取電子郵件訊息

下列 JavaScript 範例會使用 `MailClient` 類別提供的指令碼 API 來擷取電子郵件帳戶中的訊息，但不會刪除訊息。

```
var myMailClient = new MailClient();

myMailClient.setProtocol(mailProtocol);
if(useSSL){
 myMailClient.enableSSL();
}

myMailClient.connect( mailServer, mailPort, mailUsername, mailPassword);
System.log("Successfully login!");
```

```
try {
    myMailClient.openFolder("Inbox");

    var messages = myMailClient.getMessages();
    System.log("Reading messages...!");
    if ( messages != null && messages.length > 0 ) {
      System.log( "You have " + messages.length + " email(s) in your inbox" );
     for (i = 0; i < messages.length; i++) {
      System.log("");
      System.log("-----MSG-------");
      System.log("Headers: ");
      var headerProp = messages[i].getHeaders();
      for each(key in headerProp.keys){
       System.log(key+": "+headerProp.get(key));
      }
      System.log("");

      System.log( "Message["+ i +"] with from: " + messages[i].from + " to: " + messages[i].to);
      System.log( "Message["+ i +"] with subject: " + messages[i].subject);
      var content = messages[i].getContent();
      System.log("Msg content as string: " + content);
     }
    } else {
      System.warn( "No messages found" );
    }
} finally {
    myMailClient.closeFolder();
    myMailClient.close();
}
```

檔案系統指令碼範例

Workflow 指令碼式元素、動作及原則需要一般檔案系統工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

新增內容至簡單文字檔案

下列 JavaScript 範例會將內容新增至文字檔案。

```
var tempDir = System.getTempDirectory() ;
var fileWriter = new FileWriter(tempDir + "/readme.txt") ;
fileWriter.open() ;
fileWriter.writeLine("File written at : "+new Date()) ;
fileWriter.writeLine("Another line") ;
fileWriter.close() ;
```

取得檔案的內容

下列 JavaScript 範例會從 Orchestrator 伺服器主機機器中取得檔案的內容。

```
var tempDir = System.getTempDirectory() ;
var fileReader = new FileReader(tempDir + "/readme.txt") ;
fileReader.open() ;
```

```
var fileContentAsString = fileReader.readAll();
fileReader.close() ;
```

LDAP 指令碼範例

Workflow 指令碼式元素、動作和原則需要一般 LDAP 工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

將 LDAP 物件轉換為 Active Directory 物件

下列 JavaScript 範例將 LDAP 群組元素轉換為 Active Directory 使用者群組物件，並將 Active Directory 使用者群組物件轉換為 LDAP 群組元素。

```
var ldapGroup ;
// convert from ldap element to Microsoft:UserGroup object
var adGroup = ActiveDirectory.search("UserGroup",ldapGroup.commonName) ;
// convert back to LdapGroup element
var ldapElement = Server.getLdapElement(adGroup.distinguishedName) ;
```

記錄指令碼範例

Workflow 指令碼式元素、動作和原則需要一般記錄工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

持續型記錄

下列 JavaScript 範例建立持續型記錄項目。

```
Server.log("This is a persistant message", "enter a long description here");
Server.warn("This is a persistant warning", "enter a long description here");
Server.error("This is a persistant error", "enter a long description here");
```

非持續型記錄

下列 JavaScript 範例建立非持續型記錄項目。

```
System.log("This is a non-persistant log message");
System.warn("This is a non-persistant log warning");
System.error("This is a non-persistant log error");
```

網路指令碼範例

Workflow 指令碼式元素、動作和原則需要一般網路工作的指令碼。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

取得 URL 的文字

下列 JavaScript 範例存取 URL，並取得文字，然後將文字轉換為字串。

```
var url = new URL("http://www.vmware.com") ;
var htmlContentAsString = url.getContent() ;
```

工作流程指令碼範例

Workflow 指令碼式元素、動作和原則需要一般工作流程工作的指令碼範例。您可以將這些範例剪下並貼上至您的指令碼式元素中，然後調整成您需要的內容。

傳回目前使用者執行的所有工作流程

下列 JavaScript 範例包含從伺服器執行的所有工作流程，並檢查這些工作流程是否屬於目前使用者。

```
var allTokens = Server.findAllForType('WorkflowToken');
var currentUser = Server.getCredential().username;
var res = [];
for(var i = 0; i<res.length; i++){
    if(allTokens[i].runningUserName == currentUser){
      res.push(allTokens[i]);
    }
}
return res;
```

存取目前工作流程 Token

您可使用 `workflow` 變數來存取目前工作流程 Token。這是提供執行目前工作流程之存取權的 `WorkflowToken` 類型物件。下列 JavaScript 範例會取得工作流程 Token 的識別碼及其開始日期。

```
System.log("Current workflow run ID: " + workflow.id);
System.log("Current workflow run start date: "+workflow.startDate);
```

排程工作流程

下列 JavaScript 範例會以指定的內容集啟動工作流程，然後將工作流程排程在一小時後啟動。

```
var workflowToLaunch = myWorkflow ;
// create parameters
var workflowParameters = new Properties() ;
workflowParameters.put("name","John Doe") ;
// change the task name
workflowParameters.put("__taskName","Workflow for John Doe") ;

// create scheduling date one hour in the future
var workflowScheduleDate = new Date() ;
var time = workflowScheduleDate.getTime() + (60*60*1000) ;
workflowScheduleDate.setTime(time) ; var scheduledTask =
workflowToLaunch.schedule(workflowParameters,workflowScheduleDate);
```

針對在迴圈中選取的物件執行工作流程

下列 JavaScript 範例使用虛擬機器的陣列，並在 `For` 迴圈中針對每個虛擬機器分別執行工作流程。`VMs` 和 `workflowToRun` 均為工作流程輸入。

```
var len=VMs.length;
for (var i=0; i < len; i++ )
{
```

```
 var VM = VMs[i];
 //var workflowToLaunch = Server.getWorkflowWithId("workflowId");
 var workflowToLaunch = workflowToRun;
 if (workflowToLaunch == null) {
 throw "Workflow not found";
 }
var workflowParameters = new Properties();
workflowParameters.put("vm",VM);
var wfToken = workflowToLaunch.execute(workflowParameters);
System.log ("Ran workflow on " +VM.name);
}
```

開發動作

3

Orchestrator 提供預先定義動作的程式庫。動作代表您在工作流程與指令碼中作為建置區塊的個別函數。

動作是 JavaScript 函數。動作可取得多個輸入參數並擁有單一傳回數值。動作可呼叫 Orchestrator API 中的任何物件，或任何您使用外掛程式匯入到 Orchestrator 之任何 API 中的物件。

執行工作流程時，動作會從工作流程的屬性取得輸入參數。這些屬性可以是工作流程的初始輸入參數，或是工作流程中的其他元素在執行時設定的屬性。

本章節討論下列主題：

重複使用動作

當您將個別函數定義為動作時，您會在程式庫中公開該函數，而非直接以程式將該函數編寫為可編寫指令碼工作工作流程元素。動作在程式庫中出現時，其他工作流程可以使用該動作。

您定義動作時，如果動作獨立於呼叫動作的工作流程之外，您可以更容易更新或最佳化動作。定義個別動作也能夠讓其他工作流程重複使用動作。執行工作流程時，Orchestrator 只會在工作流程執行動作時快取每個動作。如此，Orchestrator 即可重複使用快取的動作。對於工作流程中的遞迴呼叫或快速迴圈，快取動作相當實用。

您可以複製動作、將動作匯出至其他工作流程或套件，或將動作移動至動作階層清單中的不同模組。

存取動作視圖

Orchestrator 用戶端介面上有**動作**視圖，可供存取 Orchestrator 伺服器的動作程式庫。

Orchestrator 用戶端介面的**動作**視圖，提供 Orchestrator 伺服器中所有可用動作的階層清單。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**動作**視圖。

3 展開動作階層清單的節點即可瀏覽動作程式庫。

您可以使用**動作**視圖檢視程式庫中動作的相關資訊，也可以建立與編輯動作。

動作視圖元件

當您按一下動作階層清單中的動作時，該動作的相關資訊即會顯示在 Orchestrator 用戶端的右窗格。

動作視圖有四個索引標籤。

一般	顯示動作相關的一般資訊，包括其名稱、版本號碼、權限及說明。
指令碼	顯示動作的傳回類型、輸入參數，以及定義動作函數的 JavaScript 指令碼。
事件	顯示此動作已碰到或觸發的所有事件。
權限	顯示有權限存取此動作的使用者與使用者群組。

建立動作

您可以定義個別的函數作為其他元素 (例如工作流程) 可以使用的動作。動作是 JavaScript 函數，含已定義的輸入與輸出參數和權限。

- 建立動作

  當您將個別函數定義為動作，而非直接將函數編碼至可編寫指令碼工作工作流程元素中時，您可以在程式庫中公開該函數以供其他工作流程使用。

- 尋找實作動作的元素

  若您編輯動作並變更其行為，可能會不小心損壞實作該動作的工作流程或應用程式。Orchestrator 提供一項功能，可尋找實作指定元素的所有動作、工作流程或套件。您可以確認修改元素是否會影響到其他元素的作業。

- 動作編碼準則

  若要最佳化工作流程的效能並盡可能重複使用動作，您在建立動作時應遵循某些基本編碼準則。

建立動作

當您將個別函數定義為動作，而非直接將函數編碼至可編寫指令碼工作工作流程元素中時，您可以在程式庫中公開該函數以供其他工作流程使用。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**動作**視圖。

3 展開動作階層清單的根目錄，並瀏覽至您要建立動作的模組。

4 在模組上按一下滑鼠右鍵，然後選取**新增動作**。

5 在文字方塊中輸入動作名稱，然後按一下**確定**。

您的自訂動作隨即新增至動作的程式庫。

6 在該動作上按一下滑鼠右鍵，然後選取**編輯**。

7 按一下**指令碼**索引標籤。

8 若要變更預設的傳回類型，請按一下**作廢**連結。

9 按一下箭頭圖示新增動作輸入參數。

10 撰寫動作指令碼。

11 設定動作權限。

12 按一下**儲存並關閉**。

您已建立自訂動作，並新增了動作輸入參數。

後續步驟

您可以在工作流程中使用新的自訂動作。

尋找實作動作的元素

若您編輯動作並變更其行為，可能會不小心損壞實作該動作的工作流程或應用程式。Orchestrator 提供一項功能，可尋找實作指定元素的所有動作、工作流程或套件。您可以確認修改元素是否會影響到其他元素的作業。

重要 **尋找使用此元素的元素**功能會檢查所有的套件、工作流程及原則，但不會檢查指令碼。因此，修改動作可能會影響到在指令碼 (**尋找使用此元素的元素**功能未識別出的指令碼) 中呼叫此動作的元素。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**動作**視圖。

3 展開動作階層清單的節點，瀏覽至指定的動作。

4 在動作上按一下滑鼠右鍵，然後選取**尋找使用此元素的元素**。

隨即出現一個對話方塊，列出實作此動作的所有元素，例如工作流程或套件。

5 按兩下結果清單中的元素，可在 Orchestrator 用戶端中顯示該元素。

您找到實作某動作的所有元素了。

後續步驟

您可以確認修改此元素是否會影響到其他元素。

動作編碼準則

若要最佳化工作流程的效能並盡可能重複使用動作，您在建立動作時應遵循某些基本編碼準則。

基本動作準則

當您建立動作時，您必須使用基本準則。

- 每個動作都必須包含其角色與函數的說明。
- 撰寫簡短的基礎動作，並將這些動作合併到工作流程中。
- 避免撰寫會執行多個函數的動作，因為這會限制重複使用該動作的可能性。
- 避免動作的執行時間過長。而是在工作流程中建立迴圈，並在動作元素之後包含等待事件或等待計時器元素。
- 不要在動作中寫入檢查點。工作流程會在每個元素執行的起點與終點設定檢查點。
- 避免在動作中寫入迴圈。而是在工作流程中建立迴圈。如果伺服器重新啟動，則執行中的工作流程會從其最後的檢查點 (位在元素的起點) 繼續執行。如果您在動作內部撰寫迴圈，且伺服器在工作流程正在執行該動作時重新啟動，則工作流程會從該動作開頭處的檢查點繼續執行，且迴圈會從頭開始再次啟動。

動作命名準則

使用基本準則為動作命名。

- 寫入的動作名稱須為英文。
- 動作名稱的開頭須為小寫字母。名稱中每個連結文字的開頭字母須為大寫。例如，`myAction`。
- 讓動作名稱盡可能一目了然，使動作的函數清楚明白。例如，`backupAllVMsInPool`。
- 讓模組名稱盡可能一目了然。
- 讓模組名稱為唯一。
- 使用反向網際網路位址格式作為模組名稱。例如，`com.vmware.myactions.myAction`。

動作參數準則

使用基本準則撰寫動作參數定義。

- 寫入的參數名稱須為英文。
- 參數名稱的開頭須為小寫字母。
- 讓參數名稱盡可能一目了然。
- 建議將參數名稱限制為一個字。如果名稱必須包含多個字，請讓名稱中每個聯結文字的開頭為大寫字母。例如，`myParameter`。
- 讓代表物件陣列的參數為複數格式。
- 讓變數名稱清楚明白，例如 `displayName`。
- 包含每個參數的說明以說明其用途。
- 在單一動作中請勿使用過多的參數。

使用動作版本紀錄

您可使用版本紀錄還原動作為上一個版本。您可還原動作狀態為先前或之後的動作版本。您也可比較目前動作狀態與儲存動作版本間的差異。

當您增加並儲存動作版本後，Orchestrator 隨即會為各動作建立新版本紀錄項目。動作的後續變更不會改變目前的版本項目。例如：當您建立動作版本 1.0.0 並儲存後，將在資料庫中儲存動作狀態。若您對動作進行任何變更，可在 Orchestrator 用戶端儲存動作狀態，但無法套用變更至動作版本 1.0.0。若要在資料庫中儲存變更，您必須建立後續動作版本並儲存。版本紀錄將連同動作本身保存在資料庫中。

當您刪除動作後，Orchestrator 會在不刪除資料庫元素版本紀錄下，將資料庫中的元素標記為刪除。您可利用此方式還原刪除的動作。請參閱還原刪除的動作。

必要條件

開啟動作進行編輯。

程序

1 按一下動作編輯器中的**一般**索引標籤。

2 按一下**顯示版本紀錄**。

出現版本紀錄視窗。

3 選擇動作版本並按一下 **目前差異** 比較差異。

視窗顯示目前動作版本與選取動作版本間的差異。

4 選擇動作版本並按一下 **還原** ，還原動作狀態。

> 注意 若您未儲存目前的動作版本，將從版本紀錄中刪除，且您無法還原回目前的版本。

動作狀態還原為選取版本的狀態。

還原刪除的動作

您可還原已從程式庫中刪除的動作。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**動作**視圖。

3 瀏覽至您要還原刪除動作的資料夾。

4 在資料夾上按一下滑鼠右鍵，然後選擇 **還原刪除的動作**。

5 選擇動作或您想要還原的動作，然後按一下 **還原**。

動作出現在選取的資料夾中。

建立資源元素

4

工作流程可能需要您在 Orchestrator 以外獨立建立的物件，當成屬性使用。若要使用外部物件作為工作流程中的屬性，您需當成資源元素匯入至 Orchestrator 伺服器。

工作流程可當成資源元素使用的物件包括映像檔案、指令碼、XML 範本、HTML 檔案等。在 Orchestrator 伺服器中執行的任何工作流程皆可使用您匯入至 Orchestrator 的所有資源元素。

當成資源元素匯入物件至 Orchestrator，可讓您變更單一位置中的物件，並自動傳播變更至所有使用此資源元素的工作流程。

您可在資料夾中組織資源元素。資源元素的最大大小為 16MB。

本章節討論下列主題：

檢視資源元素

您可檢視 Orchestrator 用戶端中的現有資源元素，以檢驗其內容並探索何種工作流程使用此資源元素。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**資源**視圖。

3 展開階層式樹狀檢視器，瀏覽資源元素。

4 按一下資源元素，顯示在右窗格中的有關資訊。

5 按一下 **檢視器** 索引標籤，顯示資源元素的內容。

6 在資源元素上按一下右鍵並選擇 **尋找使用此元素的元素**。

Orchestrator 列出使用此資源元素的所有工作流程。

後續步驟

匯入和編輯資源元素。

匯入外部物件作為資源元素使用

工作流程會需要您在 Orchestrator 以外獨立建立的物件，以當成屬性使用。若要使用外部物件作為工作流程中的屬性，您必須當成資源元素匯入至 Orchestrator 伺服器。

必要條件

確認您有映像檔、指令碼、XML 範本、HTML 檔案或其他匯入的物件型別。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**資源**視圖。

3 在階層式清單或根的資源資料夾上按一下右鍵，選擇 **新資料夾** 建立儲存資源元素的資料夾。

4 在要匯入資源元素的資源資料夾上按一下右鍵，選擇 **匯入資源**。

5 選擇要匯入的資源，然後按一下 **開啟**。

Orchestrator 新增資源元素至您選取的資料夾。

您已匯入資源元素至 Orchestrator 伺服器。

後續步驟

編輯資源元素的一般資訊並設定使用者存取權限。

編輯資源元素資訊及存取權限

將物件當成資源元素匯入至 Orchestrator 伺服器後，您可編輯資源元素詳細資訊及權限。

必要條件

請確認您已將映像、指令碼、XML 或 HTML 檔案，或任何其他類型的物件做為資源元素匯入 Orchestrator 中。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**資源**視圖。

3 在資源元素上按一下滑鼠右鍵，然後選取**編輯**。

4 按一下 **一般** 索引標籤並設定資源元素名稱、版本及描述。

5 按一下 **權限** 索引標籤並按一下 **新增存取權限** 圖示 () 以定義使用者群組的權限。

6 在 **篩選器** 文字方塊中鍵入使用者群組名稱。

7 選取使用者群組，然後按一下**確定**。

8 在使用者群組上按一下滑鼠右鍵，然後選擇 **新增存取權限**。

9 勾選適當的核取方塊設定此使用者群組的權限等級，然後按一下 **確定**。

權限非累計性。若要讓使用者檢視資源元素，請在其工作流程中使用及變更權限，您必須勾選所有核取方塊。

10 按一下**儲存並關閉**結束編輯器。

您已編輯有關資源元素的一般資訊並設定使用者存取權限。

後續步驟

儲存資源元素至檔案進行更新，或新增資源元素至工作流程。

儲存資源元素至檔案

您可儲存資源元素至您本機系統上的檔案。另存資源元素為讓您編輯的檔案。

您無法編輯 Orchestrator 用戶端中的資源元素。例如：若資源元素為 XML 組態檔案或指令碼，您必須儲存在本機才能修改。

必要條件

確認 Orchestrator 伺服器包含可儲存至檔案的資源元素。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**資源**視圖。

3 在資源元素上按右鍵並選擇 **儲存至檔案**。

4 在檔案中進行必要修改。

您已儲存資源元素至檔案。

後續步驟

更新 Orchestrator 伺服器中的資源元素。

更新資源元素

若您想要更新資源元素，您必須匯出至檔案系統、利用適當工具編輯匯出的檔案，並匯入編輯的檔案以更新資源元素。

必要條件

請確認您已將映像、指令碼、XML 或 HTML 檔案，或任何其他類型的物件做為資源元素匯入 Orchestrator 中。

程序

1 在您的本機系統中，修改資源元素的來源檔案。

2 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

3 按一下**資源**視圖。

4 瀏覽至您已更新資源元素的階層式清單。

5 在資源元素上按一下滑鼠右鍵，然後選取**更新資源**。

6 (選擇性) 按一下 **檢視器** 索引標籤，確認 Orchestrator 已更新資源元素。

您已更新 Orchestrator 伺服器包含的資源元素。

新增資源元素至工作流程

資源元素屬於您可匯入至 Orchestrator 伺服器並用於工作流程的外部物件，可在執行時當成屬性使用。例如：工作流程可使用匯入的 XML 檔案定義對應轉換資料型別，或定義執行函式時的指令碼。

必要條件

請確認您的 Orchestrator 伺服器中包含以下物件：

- 做為資源元素匯入 Orchestrator 的映像、指令碼、XML 或 HTML 檔案，或任何其他類型的物件。
- 需要資源元素作為屬性的工作流程。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**設計**。

2 按一下**工作流程**視圖。

3 展開階層式樹狀檢視器，瀏覽至需要資源元素作為屬性的工作流程。

4 在工作流程上按一下滑鼠右鍵，然後選取**編輯**。

5 在 **一般** 索引標籤上的屬性窗格中，按一下 **新增屬性** 圖示 ( )。

6 按一下屬性名稱，然後輸入屬性的新名稱。

7 按一下**類型**設定屬性類型。

8 在**選取類型**對話方塊的**篩選器**文字方塊中輸入 **`resource`**，以搜尋物件類型。

選項	動作
定義單一資源元素做為屬性	從清單中選取 `ResourceElement`。
將包含多個資源元素的資料夾定義為屬性	從清單中選取 `ResourceElementCategory`。

9 按一下**值**，然後在**篩選器**文字方塊中輸入資源元素的名稱或資源元素的類別。

10 在建議清單中選擇資源元素或包含資源元素的資料夾，然後按一下 **選擇**。

11 按一下**儲存並關閉**結束編輯器。

您已新增資源元素或資源元素資料夾作為工作流程中的屬性。

5 建立套件

套件用於在不同 Orchestrator 伺服器之間散佈內容。套件可包含工作流程、動作、原則範本、組態或資源。

當您將元素新增至套件時，Orchestrator 會檢查相依性，並將任何相依的元素新增至套件。例如，如果您新增的工作流程使用動作或其他工作流程，Orchestrator 會將這些動作與工作流程新增至套件。

當您匯入套件時，伺服器會比較其內容之不同元素的版本，以比對本機元素。這樣的比較會顯示本機元素與所匯入元素之間版本的差異。管理員可決定是否匯入套件，或者可選取要匯入的特定元素。

套件會使用數位權利管理來控制接收伺服器如何使用套件的內容。Orchestrator 會簽署與加密套件，以保護資料。套件使用 X509 憑證來追蹤哪些使用者匯出與重新散佈憑證。

如需使用套件的詳細資訊，請參閱《使用 VMware vRealize Orchestrator 用戶端》。

- 建立套件

  您可以匯出套件中的工作流程、原則範本、動作、外掛程式參考、資源與組態元素。系統會自動將套件中某個元素實作的所有元素新增至套件，以確定版本之間的相容性。如果您不想要新增參考的元素，可以在套件編輯器中刪除這些元素。

- 設定套件的使用者權限

  您可以設定套件的不同權限層級，以便限制不同使用者或使用者群組對於該套件內容所擁有的存取權。

建立套件

您可以匯出套件中的工作流程、原則範本、動作、外掛程式參考、資源與組態元素。系統會自動將套件中某個元素實作的所有元素新增至套件，以確定版本之間的相容性。如果您不想要新增參考的元素，可以在套件編輯器中刪除這些元素。

必要條件

確認 Orchestrator 伺服器包含如工作流程、動作及原則範本等您可新增至套件的元素。

程序

1 從 Orchestrator 用戶端的下拉式功能表中選取**管理**。

2 按一下**套件**視圖。

3 在左邊窗格中按一下滑鼠右鍵，並選取**新增套件**。

4 輸入新套件的名稱並按一下**確定**。

套件名稱的語法為 *domain.your_company*。*資料夾*.*package_name*。

例如 `com.vmware.myfolder.mypackage`。

5 在套件上按一下滑鼠右鍵並選取**編輯**。

套件編輯器隨即開啟。

6 在**一般**索引標籤上新增套件的說明。

7 在**工作流程**索引標籤上新增工作流程到套件。

- 按一下**插入工作流程 (清單搜尋)** 搜尋並在選取對話方塊中選取工作流程。
- 按一下**插入工作流程 (樹狀瀏覽)** 以瀏覽階層清單並選取工作流程的資料夾。

8 在**原則範本**、**動作**、**組態**、**資源**及**已使用外掛程式**索引標籤上，將原則範本、動作組態元素、資源元素及外掛程式新增至套件。

9 按一下**儲存並關閉**結束編輯器。

您已建立套件，並將元素新增至該套件。

後續步驟

為此套件設定使用者權限。

設定套件的使用者權限

您可以設定套件的不同權限層級，以便限制不同使用者或使用者群組對於該套件內容所擁有的存取權。

您可以從 Orchestrator LDAP 或 vCenter Single Sign-On 伺服器中的使用者和使用者群組，選取要設定權限的不同使用者和使用者群組。Orchestrator 會定義可套用於使用者或群組的權限層級。

檢視	使用者可檢視套件中的元素，但是無法檢視架構或指令碼。
檢查	使用者可檢視套件中的元素，包括架構和指令碼。
編輯	使用者可編輯套件中的元素。
管理員	使用者可針對套件中的元素設定權限。

必要條件

建立套件，並在套件編輯器中開啟套件進行編輯，然後將所需的元素新增至套件。

程序

1 按一下套件編輯器中的**權限**索引標籤。

2 按一下**新增存取權限**圖示 () 即可定義新使用者或使用者群組的權限。

3 搜尋使用者或使用者群組。

搜尋結果將顯示與搜尋相符的所有使用者和使用者群組。

4 選取使用者或使用者群組。

5 勾選適當的核取方塊以設定此使用者的權限層級，然後按一下**選取**。

若要允許使用者檢視元素、檢查架構和指令碼、執行和編輯元素，並變更權限，您必須勾選所有的核取方塊。

6 按一下**儲存並關閉**結束編輯器。

此時您即已建立套件並設定適當的使用者權限。

開發外掛程式

6

Orchestrator 允許透過開放式外掛程式架構整合管理及系統管理解決方案。您使用 Orchestrator 用戶端執行及建立外掛程式工作流程和存取外掛程式 API。

本章節討論下列主題：

- 外掛程式概觀
- 外掛程式的內容及結構
- Orchestrator 外掛程式 API 參考
- vso.xml 外掛程式定義檔案元素
- Orchestrator 外掛程式開發最佳做法

外掛程式概觀

Orchestrator 外掛程式必須包含一組標準的元件，而且必須遵循標準架構。這些做法有助於您建立盡可能最多種外部技術都適用的外掛程式。

- Orchestrator 外掛程式的結構

  Orchestrator 外掛程式的一般結構由執行特定功能的各種分層類型組成。

- 公開外部 API 至 Orchestrator

  您透過建立 Orchestrator 外掛程式從外部產品公開 API 至 Orchestrator 平台。您可建立用於任何技術的外掛程式公開 API，並可對應至 Orchestrator 可使用的 JavaScript 物件。

- 外掛程式的元件

  外掛程式包含一組標準的元件，這些元件向 Orchestrator 平台公開外掛技術中的物件。

- vso.xml 檔案角色

  您使用 `vso.xml` 檔案對應外掛技術物件、級別、方法和屬性至 Orchestrator 詳細目錄物件、指令碼型別、指令碼級別、指令碼方法和屬性。`vso.xml` 檔案也負責定義外掛程式的組態與啟動行為。

- 外掛程式介面卡

  外掛程式介面卡是 Orchestrator 伺服器外掛程式進入點。外掛程式介面卡可作為 Orchestrator 伺服器中的外掛技術資料存放區、建立外掛程式處理站，並管理外掛技術中發生的事件。

- 外掛程式處理站的角色

  外掛程式處理站定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。

- Finder 物件角色

  Finder 物件會識別並找出外掛技術中受管理物件型別的指定執行個體。Orchestrator 可藉由執行 finder 物件上的工作流程，修改及與在外掛技術中找到的物件互動。

- 指令碼物件角色

  指令碼物件為外掛技術中物件的 JavaScript 表示法。外掛程式的指令碼物件出現在 Orchestrator Javascript API 中，您可在工作流程和動作中的執行指令元素上使用。

- 事件控點角色

  事件是指 Orchestrator 在外掛技術中發現的物件狀態變更或屬性變更。Orchestrator 會藉由執行事件控點監控事件。

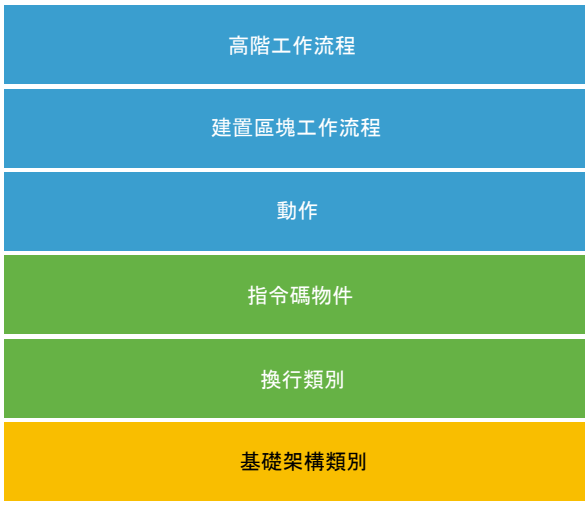
Orchestrator 外掛程式的結構

Orchestrator 外掛程式的一般結構由執行特定功能的各種分層類型組成。

Orchestrator 外掛程式的底部三圖層為基礎架構類別、換行類別及指令碼物件，皆可在外掛技術和 Orchestrator 間執行連線。

Orchestrator 外掛程式的使用者可視部分則為頂部三圖層，分別是動作、建置區塊及高階工作流程。

圖 6-1. Orchestrator 外掛程式的結構



基礎架構類別 在外掛技術與 Orchestrator 之間提供連線的類別組合。基礎架構類別包括依外掛程式定義執行的類別，如外掛工廠、外掛介面卡等。基礎架構類別也包括為一般工作和物件提供的功能類別，如協助程式、快取、詳細目錄等。

換行類別 讓外掛技術物件模型配合您要在 Orchestrator 內部公開物件模型的類別組合。

指令碼物件 在外掛技術中提供換行類別存取權限、方法及屬性的 JavaScript 物件型別。在 `vso.xml` 檔案中，您定義的外掛技術換行類別、屬性及方法，將公開至 Orchestrator。

動作 您可在工作流程及指令碼工作中，直接使用的 JavaScript 函式組合。動作可取得多個輸入參數並擁有單一傳回數值。

建置區塊工作流程 工作流程組合，包括您要搭配外掛程式提供的所有一般功能。一般而言，建置區塊工作流程代表協調技術中的使用者介面作業。建置區塊工作流程可直接使用，或可加入高階工作流程內。

高階工作流程 包括外掛程式指定功能的工作流程組合。您可提供高階工作流程以符合具體需求或顯示外掛程式用法的複雜範例。

公開外部 API 至 Orchestrator

您透過建立 Orchestrator 外掛程式從外部產品公開 API 至 Orchestrator 平台。您可建立用於任何技術的外掛程式公開 API，並可對應至 Orchestrator 可使用的 JavaScript 物件。

外掛程式對應 Java 物件及方法至新增至 Orchestrator 指令碼 API 的 JavaScript 物件。若外部技術公開 Java API，您可直接對應 API 至 Orchestrator 的 JavaScript，以便在工作流程及動作中使用。

您可透過使用 WSDL（網路服務定義語言）、REST（具象狀態傳輸）或傳訊服務，建立適用以非 Java 語言公開 API 的應用程式外掛程式，以整合公開 API 至 Java 物件。接著您可對應整合的 Java 物件至 Orchestrator 的 JavaScript 進行使用。

外掛技術獨立於 Orchestrator 之外。即使您只有二進位代碼的存取權限，您仍可建立用於外部產品的 Orchestrator 外掛程式，例如：在 Java 封存（JAR 檔案），而非原始程式碼中。

外掛程式的元件

外掛程式包含一組標準的元件，這些元件向 Orchestrator 平台公開外掛技術中的物件。

外掛程式的主要元件有外掛程式介面卡、處理站和事件實作。您可以將介面卡、處理站和事件實作中定義的物件和作業，對應至名稱為 `vso.xml` 的 XML 定義檔中出現的 Orchestrator 物件。`Vso.xml` 檔會將外掛技術中的物件和函數，對應至 Orchestrator JavaScript API 中出現的 JavaScript 指令碼物件。`vso.xml` 檔也會將外掛技術的物件類型，對應至 Orchestrator **詳細目錄**索引標籤中出現的 Finder。

外掛程式包含下列元件。

外掛程式模組 外掛程式本身 (由 Java 類別集合所定義)、`vso.xml` 檔，以及與您透過外掛程式存取之物件互動的工作流程和動作套件。外掛程式模組是必要的。

外掛程式介面卡 定義外掛技術和 Orchestrator 伺服器之間的介面。介面卡是 Orchestrator 平台外掛程式的進入點。介面卡可建立外掛程式處理站、管理外掛程式的載入和卸載，並管理針對外掛技術中物件發生的事件。外掛程式介面卡是必要的。

外掛程式處理站 定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。對於 Orchestrator 與外掛技術之間開啟的用戶端工作階段，介面卡會建立處理站。處理站可讓您在所有用戶端連線之間共用工作階段，或對於個別用戶端連線開啟一個工作階段。外掛程式處理站是必要的。

組態 Orchestrator 不會定義外掛程式儲存其組態的標準方式。您可以透過使用 Windows 登錄、靜態組態檔、將資訊儲存於資料庫，或儲存於 XML 檔的方式儲存組態資訊。在 Orchestrator 用戶端中執行組態工作流程，即可設定 Orchestrator 外掛程式。

Finder 對於 Orchestrator 如何尋找和代表外掛技術中的物件予以定義的互動規則。Finder 會從外掛技術向 Orchestrator 公開的一組物件擷取物件。您可在 `vso.xml` 檔定義物件之間的關係，以便瀏覽物件的網路。Orchestrator 代表**詳細目錄**索引標籤中的外掛技術物件模型。如果您要向 Orchestrator 公開外掛技術中的物件，Finder 是必要的。

指令碼物件 對於外掛技術中的物件、作業及屬性提供存取權限的 JavaScript 物件類型。指令碼物件定義 Orchestrator 如何透過 JavaScript 存取外掛技術的物件模型。您可以將外掛技術的類別和方法對應至 `vso.xml` 檔中的 JavaScript 物件。您可以存取 Orchestrator 指令碼 API 中的 JavaScript 物件，並將這些物件整合於 Orchestrator 指令碼式工作、動作和工作流程。如果您要將指令碼類型、類別和方法新增至 Orchestrator JavaScript API，指令碼物件是必要的。

詳細目錄 Orchestrator 使用 Finder 找到的外掛技術物件執行個體，會出現在 Orchestrator 用戶端的**詳細目錄**視圖中。您可以對詳細目錄中的物件執行工作流程，以便對這些物件執行作業。詳細目錄是選用的。您可以建立外掛程式僅將指令碼類型和類別新增至 Orchestrator JavaScript API，而不公開詳細目錄之中物件的任何執行個體。

事件 外掛技術中的物件發生的狀態變更。Orchestrator 可被動接聽外掛技術中發生的事件。Orchestrator 也可主動觸發外掛技術中的事件。事件是選用的。

vso.xml 檔案角色

您使用 `vso.xml` 檔案對應外掛技術物件、級別、方法和屬性至 Orchestrator 詳細目錄物件、指令碼型別、指令碼級別、指令碼方法和屬性。`vso.xml` 檔案也負責定義外掛程式的組態與啟動行為。

`vso.xml` 檔案執行下列主體角色。

啟動及組態行為 定義外掛程式啟動的方式，並找出外掛程式定義的所有執行組態。載入外掛程式介面卡。

詳細目錄物件 定義外掛技術中，外掛程式存取的物件型別。外掛程式原廠執行的 Finder 方法可找出這些物件的執行個體，並顯示在 Orchestrator 詳細目錄中。

指令碼型別 新增指令碼型別至 Orchestrator JavaScript API，表示詳細目錄中的不同物件型別。您可利用這些指令碼型別作為工作流程中的輸入參數。

指令碼級別 新增級別至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。

指令碼方法	新增方法至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。
指令碼屬性	新增外掛技術中的物件屬性至 Orchestrator JavaScript API，以便在工作流程、動作、原則等執行指令元素中使用。

外掛程式介面卡

外掛程式介面卡是 Orchestrator 伺服器外掛程式進入點。外掛程式介面卡可作為 Orchestrator 伺服器中的外掛技術資料存放區、建立外掛程式處理站，並管理外掛技術中發生的事件。

若要建立外掛程式介面卡，您可以建立實作 `IPluginAdaptor` 介面的 Java 類別。

您建立的外掛程式介面卡類別可管理外掛技術中的外掛程式處理站、事件和觸發器。`IPluginAdaptor` 介面提供您可用來執行這些工作的方法。

外掛程式介面卡執行下列主體角色。

建立處理站	對於從 Orchestrator 到外掛技術的每個連線而言，外掛程式介面卡的最重要角色是載入和卸載一個外掛程式處理站執行個體。外掛程式介面卡類別會呼叫 `IPluginAdaptor.createPluginFactory()` 方法，建立實作 `IPluginFactory` 介面的類別執行個體。
管理事件	外掛程式介面卡是 Orchestrator 伺服器與外掛技術之間的介面。外掛程式介面卡可管理 Orchestrator 對外掛技術中的物件執行和監看的事件。介面卡可透過事件發佈者管理事件。事件發佈者為由呼叫 `IPluginAdaptor.registerEventPublisher()` 方法所建立之介面卡的 `IPluginEventPublisher` 介面執行個體。事件發佈者會對外掛技術中的物件設定觸發器和量表，以便在物件發生特定事件或物件的值通過特定臨界值的情況下，允許 Orchestrator 啟動定義的動作。同樣地，您可以定義 `PluginTrigger` 和 `PluginWatcher` 執行個體來定義長時間執行之工作流程中的等待事件元素等待的事件。
設定外掛程式名稱	您可在 `vso.xml` 檔中提供外掛程式的名稱。外掛程式介面卡可從 `vso.xml` 檔取得此名稱，並在 Orchestrator 用戶端**詳細目錄**視圖中發佈該名稱。
安裝授權	您可以呼叫方法來安裝外掛技術在介面卡實作中需要的任何授權檔案。

如需 `IPluginAdaptor` 介面、其所有方法和外掛程式 API 其他所有類別的完整詳細資料，請參閱 Orchestrator 外掛程式 API 參考。

外掛程式處理站的角色

外掛程式處理站定義 Orchestrator 如何尋找外掛技術中的物件並對這些物件執行作業。

若要建立外掛程式處理站，您必須從 Orchestrator 外掛程式 API 實作和擴充 `IPluginFactory` 介面。您建立的外掛程式處理站類別會定義 Orchestrator 存取外掛技術中之物件所用的 Finder 函數。處理站允許 Orchestrator 伺服器按照物件的識別碼、物件與其他物件之間的關係或搜尋查詢字串來尋找物件。

外掛程式處理站執行下列主體工作。

尋找物件	您可以建立函數，按照物件名稱和類型來尋找物件。您可以使用 `IPluginFactory.find()` 方法，按照名稱和類型來尋找物件。
尋找與其他物件相關聯的物件	您可以建立函數，按照特定關係類型來尋找與特定物件相關聯的物件。您可以在 `vso.xml` 檔中定義關係。您也可以建立 Finder，按照特定關係類型尋找與所有父系相關聯的相依子物件。您可以實作 `IPluginFactory.findRelation()` 方法，按照特定關係類型尋找與特定父系物件相關聯的任何物件。您可以實作 `IPluginFactory.hasChildrenInRelation()` 方法，探索對於父系執行個體是否至少存在一個子物件。
定義查詢來按照您自己的準則尋找物件	您可以建立物件 Finder 實作您定義的查詢規則。您可以實作 `IPluginFactory.findAll()` 方法，尋找處理站呼叫此方法時與您定義的查詢規則相符合的所有物件。您將取得 `QueryResult` 物件的 `findAll()` 方法結果，其中包含與您定義的查詢規則相符合的所有物件清單。

如需 `IPluginFactory` 介面、其所有方法和外掛程式 API 其他所有類別的詳細資訊，請參閱 Orchestrator 外掛程式 API 參考。

Finder 物件角色

Finder 物件會識別並找出外掛技術中受管理物件型別的指定執行個體。Orchestrator 可藉由執行 finder 物件上的工作流程，修改及與在外掛技術中找到的物件互動。

在外掛技術中指定受管理物件型別的各執行個體必須擁有唯一識別碼，如此 Orchestrator finder 物件才能找到。外掛技術提供用於物件執行個體的唯一識別碼作為字串。工作流程執行時，Orchestrator 會設定物件的唯一識別碼並當成工作流程屬性數值尋找。需要指定類型物件作為輸入參數的工作流程會在該類型物件的特定執行個體上執行。

外掛程式新增至 Orchestrator JavaScript API 的 Finder 物件含有作為前置詞的外掛程式名稱。例如：vCenter Server API 中的 `VirtualMachine` 受管理物件型別會作為 `VC:VirtualMachine` JavaScript 型別出現在 Orchestrator 中。

例如：Orchestrator 藉由執行 finder 物件並使用虛擬機器 `id` 屬性作為其唯一識別碼，以透過 vCenter Server 外掛程式存取指定 `VC:VirtualMachine` 執行個體。您可將此物件執行個體當成屬性數值傳遞至工作流程元素。

Orchestrator 外掛程式對應外掛技術中的物件至 `vso.xml` 檔案中 `<finder>` 元素內的對等 Orchestrator finder 物件。`<finder>`元素可識別外掛技術中的方法或功能，以取得指定物件執行個體的唯一識別碼。`<finder>`元素也會定義物件間的關係，以透過和其他物件的關聯方式尋找物件。

Finder 物件出現在外掛程式下的 Orchestrator **詳細目錄** 索引標籤中。

指令碼物件角色

指令碼物件為外掛技術中物件的 JavaScript 表示法。外掛程式的指令碼物件出現在 Orchestrator Javascript API 中，您可在工作流程和動作中的執行指令元素上使用。

外掛程式中的指令碼物件作為 JavaScript 模組、類型和類別出現在 Orchestrator JavaScript API 中。大多數的 Finder 物件都有指令碼物件表示法。JavaScript 類別可新增方法和屬性至 Orchestrator JavaScript API，以代表外掛技術 API 的物件方法與屬性。外掛技術會提供 Orchestrator 獨立執行的物件、型別、級別、屬性和方法。例如：vCenter Server 外掛程式代表所有來自 vCenter Server API 中作為 Orchestrator JavaScript API 的 JavaScript 物件，並包含所有 vCenter Server API 定義的級別、方法和屬性的 JavaScript 表示法。您可使用在 Orchestrator 執行指令函式中定義的 vCenter Server 指令碼類別與方法和屬性。

例如：vCenter Server API 的 `VirtualMachine` 受管理物件型別可透過 `VC:VirtualMachine` finder 找到並作為 `VcVirtualMachine` JavaScript 級別出現在 Orchestrator JavaScript API 中。Orchestrator JavaScript API 中的 `VcVirtualMachine` JavaScript 級別負責定義所有與 vCenter Server API `VirtualMachine` 受管理物件相同的方法與屬性。

Orchestrator 外掛程式對應外掛技術中的物件、型別、級別、屬性和方法至 `vso.xml` 檔案中 `<scripting-objects>` 元素的對等 Orchestrator JavaScript 物件、型別、級別、屬性與方法。

事件控點角色

事件是指 Orchestrator 在外掛技術中發現的物件狀態變更或屬性變更。Orchestrator 會藉由執行事件控點監控事件。

Orchestrator 外掛程式允許您使用不同方式監控外掛技術中的事件。Orchestrator 外掛程式 API 允許您建立下列型別的事件控點，監控外掛技術中的事件。

接聽程式

被動監控外掛技術中的物件是否變更狀態。外掛技術或外掛程式執行可定義接聽程式監控的事件。接聽程式不會啟動事件，但會在發生事件時通知 Orchestrator。接聽程式會透過輪詢外掛技術或接收外掛技術的通知來偵測事件。發生事件時，正在等待事件的 Orchestrator 原則或工作流程會透過開始 Orchestrator 伺服器中的作業作為反應。接聽程式元件為選用。

原則

監控外掛技術中的特定事件，若發生事件則開始 Orchestrator 伺服器中的作業。原則可監控原則觸發器與原則量表。原則觸發器負責定義外掛技術中發生的事件，會造成執行中的原則開始 Orchestrator 伺服器中的作業，例如工作流程。原則量表負責定義外掛技術中物件屬性的數值範圍，如超過則會造成 Orchestrator 開始作業。原則為選用。

工作流程觸發器

若執行中的工作流程包含等待事件元素，則在到達該元素後將暫停執行並等待外掛技術中的事件發生。工作流程觸發器負責定義外掛技術中，工作流程中等待事件元素等候的事件。您可透過監看程式登錄工作流程觸發器。工作流程觸發器為選用。

監看程式

代表工作流程中等待事件元素監看外掛技術中的工作流程觸發器是否發生特定事件。發生事件時，監看程式會通知正在等待該事件的所有工作流程。監看程式為選用。

外掛程式的內容及結構

Orchestrator 外掛程式必須包含標準元件組合並符合標準檔案結構。如為符合標準檔案結構的外掛程式，必須包括指定資料夾及檔案。

若要建立 Orchestrator 外掛程式，您需定義 Orchestrator 存取及與外掛技術物件互動的方式。此外，您需對應所有物件與外掛技術的功能，以對應 `vso.xml` 檔案中的 Orchestrator 物件和功能。

`vso.xml` 檔案必須包括所有物件型別或 Orchestrator 公開作業的參考。外掛程式在外掛技術中找到的各物件必須擁有您提供的唯一識別碼。您定義在 `vso.xml` 檔案中的 `finder` 元素和物件元素的物件名稱。

外掛程式可當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但在任何情況下，檔案必須以 `.dar` 副檔名重新命名。

備註 您可使用 Orchestrator Control Center 匯入 DAR 檔案至 Orchestrator 伺服器。

- 在 vso.xml 檔案中定義應用程式對應

  您在 `vso.xml` 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。

- vso.xml 外掛程式定義檔案格式

  `vso.xml` 檔案會定義 Orchestrator 伺服器與外掛技術互動的方式。您必須在 `vso.xml` 檔案中包含要對 Orchestrator 公開的每一類型物件或作業的參照。

- 命名外掛程式物件

  對於外掛程式在外掛技術中找到的每個物件，您必須提供唯一識別碼。您定義在 `vso.xml` 檔案中的 `<finder>` 元素和 `<object>` 元素的物件名稱。

- 外掛程式物件命名慣例

  您必須在命名所有外掛程式中的物件時遵守 Java 類別命名慣例。

- 外掛程式的檔案結構

  外掛程式必須符合標準檔案結構，而且必須包括某些特定資料夾及檔案。您可以將外掛程式當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但是檔案必須以 `.dar` 副檔名重新命名。

在 vso.xml 檔案中定義應用程式對應

您在 `vso.xml` 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。

`vso.xml` 檔案會將下列資訊提供給 Orchestrator 伺服器：

- 外掛程式的版本、名稱與說明
- 外掛技術類別的參照及相關外掛程式介面卡的參照
- 當 Orchestrator 伺服器啟動時初始化外掛程式
- 代表外掛技術中物件類型的指令碼類型
- 物件類型之間定義物件在 Orchestrator 詳細目錄中如何顯示的關係

- 將外掛技術中的物件與作業對應至 Orchestrator JavaScript API 中函數與物件類型的指令碼類別
- 用於定義套用至某特定類型的所有物件之常數值清單的列舉
- Orchestrator 在外掛技術中監視的事件

`vso.xml` 檔案必須符合 Orchestrator 外掛程式的 XML 架構定義。您可以存取 VMware 支援站台的架構定義。

```
http://www.vmware.com/support/orchestrator/plugin-4-1.xsd
```

如需 `vso.xml` 檔案所有元素的說明，請參閱 vso.xml 外掛程式定義檔案元素。

vso.xml 外掛程式定義檔案格式

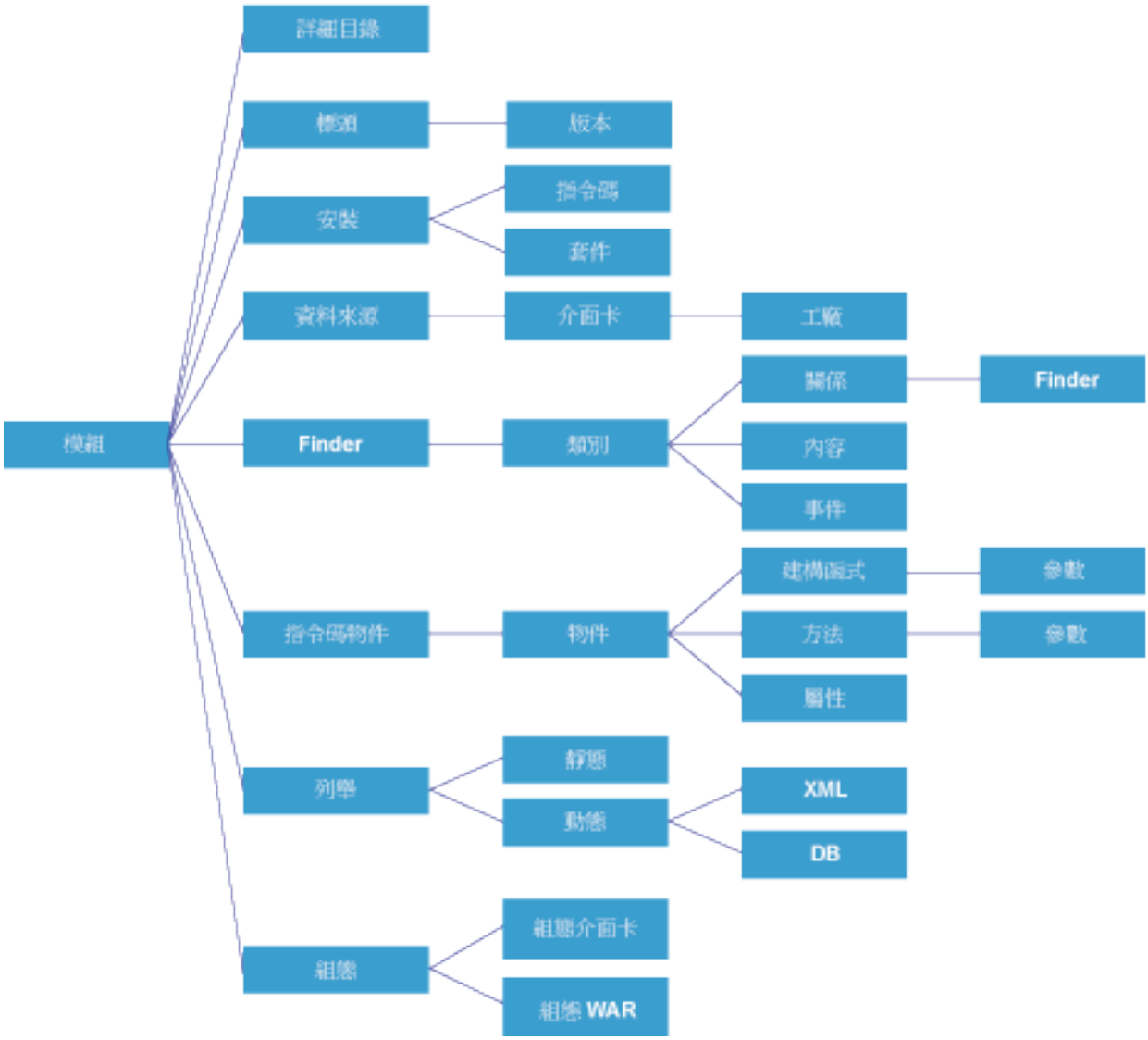
`vso.xml` 檔案會定義 Orchestrator 伺服器與外掛技術互動的方式。您必須在 `vso.xml` 檔案中包含要對 Orchestrator 公開的每一類型物件或作業的參照。

您在 `vso.xml` 檔案中包含的物件會在 Orchestrator 指令碼 API 中顯示成指令碼物件，或在 Orchestrator **詳細目錄**索引標籤中顯示成 Finder 物件。

`vso.xml` 檔案為外掛程式的開放式架構與標準化實作的一部分，因此必須採用標準格式。

下圖顯示 `vso.xml` 外掛程式定義檔案的格式，以及元素如何互相嵌套。

圖 6-2. vso.xml 外掛程式定義檔案格式



命名外掛程式物件

對於外掛程式在外掛技術中找到的每個物件，您必須提供唯一識別碼。您定義在 `vso.xml` 檔案中的 `<finder>` 元素和 `<object>` 元素的物件名稱。

您在外掛技術的處理站執行尋找物件中定義的 Finder 作業。外掛程式尋找物件時，您可以在 Orchestrator 工作流程中使用這些物件，並且在工作流程元素之間傳送這些物件。您對於物件提供的唯一識別碼，能夠讓這些物件在工作流程中的元素之間傳遞。

Orchestrator 伺服器僅儲存所處理的每個物件本身的類型和識別碼，不會儲存 Orchestrator 如何取得物件的相關資訊。您必須在外掛程式實作中以一致的方式命名物件，以便您追蹤從外掛程式取得的物件。

工作流程進行時，如果 Orchestrator 伺服器停止，則您重新啟動伺服器時，工作流程會從伺服器停止時執行的工作流程元素恢復進行。工作流程使用識別碼擷取伺服器停止時元素正在處理的物件。

外掛程式物件命名慣例

您必須在命名所有外掛程式中的物件時遵守 Java 類別命名慣例。

重要 由於工作流程引擎執行資料序列化的方式，請勿在物件名稱中使用下列字串順序。在物件識別碼中使用這些字元順序會造成工作流程引擎間接剖析工作流程，並會在您執行工作流程時導致非預期行為。

- `#;#`
- `#,#`
- `#=#`

在您命名外掛程式中的物件時使用這些方針。

- 在名稱中的各文字使用首字母大寫。
- 請勿使用空格分隔文字。
- 在字母中，請使用標準字元 `A` 至 `Z` 和 `a` 至 `z`。
- 請勿使用特殊字元，如重音符號。
- 請勿使用數字作為名稱的第一個字元。
- 如有可能，請使用少於 10 個字元。

表 6-1. 外掛程式物件命名規定 顯示套用至個別物件型別的規定。

表 6-1. 外掛程式物件命名規定

物件類型	命名規定
外掛程式	■ 在 `vso.xml` 檔案中的 `<module>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 必須唯一。您無法在 Orchestrator 伺服器中執行兩個相同名稱的外掛程式。
Finder 物件	■ 在 `vso.xml` 檔案中的 `<finder>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 在外掛程式中必須為唯一。 Orchestrator 新增外掛程式名稱及冒號至 Orchestrator 指令碼 API finder 物件類型中的 finder 物件名稱。例如：vCenter Server 外掛程式中的 `VirtualMachine` 物件型別以 `VC:VirtualMachine` 出現在 Orchestrator 指令碼 API 中。
指令碼物件	■ 在 `vso.xml` 檔案中的 `<scripting-object>` 元素中定義。 ■ 必須遵守 Java 類別命名慣例。 ■ 在 Orchestrator 伺服器中必須為唯一。 ■ 若要避免混淆指令碼物件與相同名稱的 finder 物件，或與其他外掛程式的指令碼物件混淆，請務必以外掛程式的名稱作為指令碼物件的前置詞，但不要新增分號。例如：vCenter Server 外掛程式中的 `VirtualMachine` 類別以 `VcVirtualMachine` 出現在 Orchestrator 指令碼 API 中。

外掛程式的檔案結構

外掛程式必須符合標準檔案結構，而且必須包括某些特定資料夾及檔案。您可以將外掛程式當成標準 Java 封存檔案 (JAR) 或 ZIP 檔案交付，但是檔案必須以 `.dar` 副檔名重新命名。

DAR 封存檔案的內容必須使用下列資料夾結構和命名慣例。

表 6-2. DAR 封存檔案的結構

資料夾	說明
plug-in_name\VSO-INF\	包含 vso.xml 檔，其定義外掛技術中的物件與 Orchestrator 物件之間的對應。 VSO-INF 資料夾和 vso.xml 檔是必要的。
plug-in_name\lib\	包含外掛技術二進位檔的 JAR 檔案。也包含 JAR 檔案，其包括介面卡、處理站、通知處理常式和外掛程式其他介面的實作。 lib 資料夾和 JAR 檔案是必要的。
plug-in_name\resources\	包含外掛程式需要的資源檔案。resources 資料夾可包含下列類型的元素： ■ 映像檔，代表 Orchestrator **詳細目錄**索引標籤中的外掛程式物件。 ■ 指令碼，定義外掛程式啟動時的初始化行為。 ■ Orchestrator 套件，其可包含自訂工作流程、動作，以及與您使用外掛程式存取的物件進行互動的其他資源。 您可以將資源整理到子資料夾中。例如，resources\images\、resources\scripts\ 或 resources\packages\。 resources 資料夾是選用的。

您可使用 Orchestrator Control Center 將 DAR 檔案匯入至 Orchestrator 伺服器。

Orchestrator 外掛程式 API 參考

您開發 IPluginAdaptor 和 IPluginFactory 實作來建立外掛程式時，Orchestrator 外掛程式 API 定義 Java 介面和類別進行實作和擴充。

除非另有說明，否則所有類別均包含在 ch.dunes.vso.sdk.api 套件中。

IAop 介面

IAop 介面提供使用外掛技術取得與設定物件內容的方法。

```
public interface IAop
```

IAop 介面定義下列方法：

方法	傳回	說明
get(java.lang.String propertyName, java.lang.Object object, java.lang.Object sdkObject)	java.lang.Object	從外掛程式指定的物件中取得內容。
set(java.lang.String propertyName, java.lang.String propertyValue, java.lang.Object object)	作廢	設定外掛程式指定的物件之內容。

IDynamicFinder 介面

IDynamicFinder 介面會以程式設計方式傳回 Finder 的識別碼與內容，而不會透過在 vso.xml 檔案中定義識別碼與內容的方式傳回。

IDynamicFinder 介面定義下列方法。

方法	傳回	說明
getIdAccessor(java.lang.String type)	java.lang.String	提供 OGNL 運算式，以透過程式設計方式取得物件識別碼。
getProperties(java.lang.String type)	java.util.List<SDKFinderProperty>	以程式設計方式提供物件內容清單。

IPluginAdaptor 介面

您可以實作 IPluginAdaptor 介面來管理外掛程式處理站、事件和監看程式。IPluginAdaptor 介面會定義外掛程式與 Orchestrator 伺服器之間的介面卡。

IPluginAdaptor 執行個體負責工作階段管理。IPluginAdaptor 介面定義下列方法。

方法	傳回	說明
addWatcher(PluginWatcher watcher)	作廢	新增監視特定事件的監看程式
createPluginFactory(java.lang.String sessionID, java.lang.String username, java.lang.String password, IPluginNotificationHandler notificationHandler)	IPluginFactory	建立 IPluginFactory 執行個體。Orchestrator 伺服器使用處理站，按照物件的識別碼、與其他物件之間的關係等等，從外掛技術取得物件。 工作階段識別碼 可讓您識別執行中的工作階段。例如，使用者可登入兩個不同的 Orchestrator 用戶端，並同時執行兩個工作階段。 同樣地，開始工作流程將建立與開始工作階段的用戶端無關聯的工作流程。即使您關閉 Orchestrator 用戶端，工作流程仍將持續執行。
installLicenses(PluginLicense[] licenses)	作廢	安裝 VMware 提供的標準外掛程式有關的授權資訊。
registerEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)	作廢	在詳細目錄的元素上設定觸發器和量表
removeWatcher(java.lang.String watcherId)	作廢	移除監看程式
setPluginName(java.lang.String pluginName)	作廢	從 vso.xml 檔案取得外掛程式名稱
setPluginPublisher(IPluginPublisher pluginPublisher)	作廢	設定外掛程式的發佈者

方法	傳回	說明
`uninstallPluginFactory(IPluginFactory plugin)`	作廢	解除安裝外掛程式處理站。
`unregisterEventPublisher(java.lang.String type, java.lang.String id, IPluginEventPublisher publisher)`	作廢	從詳細目錄的元素中移除觸發器和量表

IPluginEventPublisher 介面

`IPluginEventPublisher` 介面會在事件通知匯流排上發佈量表和觸發器，以便 Orchestrator 原則進行監視。

您可以直接在外掛程式介面卡實作中建立 `IPluginEventPublisher` 執行個體，也可以在個別事件產生器類別中建立這些執行個體。

您可執行 `IPluginEventPublisher` 介面，將外掛技術事件發佈至 Orchestrator 原則引擎。您可以在外掛技術中建立方法來設定物件的原則觸發器和量表，以及事件接聽程式來接聽這些物件上的事件。

原則可執行量表或觸發器，監控外掛技術中的物件。原則量表負責監控物件屬性，若物件數值超過特定限制，則在 Orchestrator 伺服器中推播事件。若物件上發生定義的事件，原則觸發器會監控物件並在 Orchestrator 伺服器中推播事件。您向 `IPluginEventPublisher` 執行個體登錄原則量表和觸發器，以便 Orchestrator 原則可以監控它們。

`IPluginEventPublisher` 介面定義下列方法。

類型	傳回	說明
`pushGauge(java.lang.String type, java.lang.String id, java.lang.String gaugeName, java.lang.String deviceName, java.lang.Double gaugeValue)`	作廢	發佈量表，以便原則進行監視。使用下列參數： ■ `type`：要監視的物件所屬的類型。 ■ `id`：要監視的物件所用的識別碼。 ■ `gaugeName`：此量表的名稱。 ■ `deviceName`：該量表監視的屬性所用的類型名稱。 ■ `gaugeValue`：量表監視物件的值。
`pushTrigger(java.lang.String type, java.lang.String id, java.lang.String triggerName, java.util.Properties additionalProperties)`	作廢	發佈觸發器，以便原則進行監視。使用下列參數： ■ `type`：要監視的物件所屬的類型。 ■ `id`：要監視的物件所用的識別碼。 ■ `triggerName`：此觸發器的名稱。 ■ `additionalProperties`：觸發器監視的其他任何內容。

IPluginFactory 介面

`IPluginAdaptor` 傳回 `IPluginFactory` 執行個體。`IPluginFactory` 執行個體在外掛程式應用程式中執行命令，並尋找執行 Orchestrator 作業的物件。

`IPluginFactory` 介面定義下列欄位：

`static final java.lang.String RELATION_CHILDREN`

`IPluginFactory` 介面定義下列方法。

方法	傳回	說明
`executePluginCommand(java.lang.String cmd)`	作廢	使用外掛程式執行命令。VMware 建議不要使用此方法。
`find(java.lang.String type, java.lang.String id)`	`java.lang.Object`	使用外掛程式尋找物件。按識別碼及類型識別物件。
`findAll(java.lang.String type, java.lang.String query)`	`QueryResult`	使用外掛程式尋找屬於特定類型且符合查詢字串的物件。您可以在外掛程式的 `IPluginFactory` 實作中定義查詢的語法。如果不定義查詢語法，`findAll()` 將傳回指定類型的所有物件。
`findRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)`	`java.util.List`	決定物件是否有子系。
`hasChildrenInRelation(java.lang.String parentType, java.lang.String parentId, java.lang.String relationName)`	`HasChildrenResult`	按特定關係尋找與指定父系相關的所有子系。
`invalidate(java.lang.String type, java.lang.String id)`	作廢	按類型及識別碼使物件失效。
`void invalidateAll()`	作廢	使快取中的所有物件失效。

IPluginNotificationHandler 介面

`IPluginNotificationHandler` 定義方法，通知 Orchestrator 有不同類型的事件在 Orchestrator 透過外掛程式存取的物件上發生。

`IPluginNotificationHandler` 介面定義下列方法。

方法	傳回	說明
`getSessionID()`	`java.lang.String`	傳回目前工作階段識別碼。
`notifyElementDeleted(java.lang.String type, java.lang.String id)`	作廢	通知系統已刪除指定類型和識別碼的物件。
`notifyElementInvalidate(java.lang.String type, java.lang.String id)`	作廢	通知系統物件的關係已變更。您可以使用 `notifyElementInvalidate()` 方法，通知 Orchestrator 物件之間的所有關係變更，而不僅止於讓物件失效的關係變更。例如，將子物件新增至父系代表兩個物件之間的關係變更。

方法	傳回	說明
`notifyElementUpdated(java.lang.String type, java.lang.String id)`	作廢	通知系統已修改物件的屬性。
`notifyMessage(ch.dunes.vso.sdk.api.ErrorLevel severity, java.lang.String type, java.lang.String id, java.lang.String message)`	作廢	發佈與目前模組相關的錯誤訊息

IPluginPublisher 介面

`IPluginPublisher` 介面在事件通知匯流排上發佈監看程式事件，以便長時間執行的工作流程等待事件元素進行監視。

工作流程觸發器開始外掛技術事件時，監看觸發器的外掛監看程式及透過 `IPluginPublisher` 執行個體登錄的監看程式會告知任何等待中的工作流程已發生事件。

`IPluginPublisher` 介面定義下列方法。

類型	值	說明
`pushWatcherEvent(java.lang.String id, java.util.Properties properties)`	作廢	在事件通知匯流排上發佈監看程式事件

WebConfigurationAdaptor 介面

`WebConfigurationAdaptor` 介面增量 `IConfigurationAdaptor` 與定義方法可在組態索引標籤中找出及安裝用於外掛程式的網路應用程式。

備註 自 Orchestrator 4.1 開始，`WebConfigurationAdaptor` 介面已過時。若要新增網路應用程式至組態，請執行 `IConfigurationAdaptor` 並使用 `vso.xml` 檔案中的 `configuration-war` 屬性來識別網路應用程式。

`WebConfigurationAdaptor` 介面定義下列方法。

方法	傳回	說明
`getWebAppContext()`	字串	找出網路應用程式的 WAR 檔案，取得組態索引標籤。從 DAR 檔案的 `/webapps` 目錄中提供名稱與路徑至 WAR 檔案作為字串。
`setWebConfiguration(boolean webConfiguration)`	布林值	判斷組態索引標籤的內容是否由網路應用程式定義。

PluginTrigger 類別

`PluginTrigger` 類別會建立觸發器模組，取得代表工作流程中的等待事件元素在外掛技術中監視的物件和事件相關的資訊。

`PluginTrigger` 類別定義方法以取得或設定監控物件型別與名稱、事件本質和逾時時間。

您建立專供工作流程中等待事件元素使用的 PluginTrigger 執行類別。您在定義事件及執行 IPluginEventPublisher.pushTrigger() 方法類別中定義用於 Orchestrator 原則的原則觸發器。

```
public class PluginTrigger
extends java.lang.Object
implements java.io.Serializable
```

PluginTrigger 類別定義下列方法：

方法	傳回	說明
getModuleName()	java.lang.String	取得觸發器模組的名稱。
getProperties()	java.util.Properties	取得觸發器內容的清單。
getSdkId()	java.lang.String	取得在外掛技術中監視的物件識別碼。
getSdkType()	java.lang.String	取得在外掛技術中監視的物件類型。
getTimeout()	完整	取得觸發器逾時時間。
setModuleName(java.lang.String moduleName)	作廢	設定觸發器模組的名稱。
setProperties(java.util.Properties properties)	作廢	設定觸發器內容的清單。
setSdkId(java.lang.String sdkId)	作廢	設定在外掛技術中監視的物件識別碼。
setSdkType(java.lang.String sdkType)	作廢	設定在外掛技術中監視的物件類型。
setTimeout(long timeout)	作廢	設定逾時時間 (以秒為單位)。負值將停用逾時。

建構函數

- PluginTrigger()
- PluginTrigger(java.lang.String moduleName, long timeout, java.lang.String sdkType, java.lang.String sdkId)

PluginWatcher 類別

PluginWatcher 類別會代表長時間執行的工作流程等待事件元素，監看外掛技術中已定義事件的觸發器模組。

PluginWatcher 類別定義建構函式，讓您可用來建立外掛監看程式執行個體。PluginWatcher 類別定義方法以取得或設定監看工作流程觸發器名稱及逾時時間。

```
public class PluginWatcher
extends java.lang.Object
implements java.io.Serializable
```

PluginWatcher 類別定義下列方法：

方法	傳回	說明
getId()	java.lang.String	取得觸發器的識別碼
getModuleName()	java.lang.String	取得觸發器模組名稱
getTimeoutDate()	完整	取得觸發器逾時日期
getTrigger()	作廢	取得觸發器
setId(java.lang.String id)	作廢	設定觸發器的識別碼
setTimeoutDate()	作廢	設定觸發器逾時日期

建構函數

PluginWatcher(PluginTrigger trigger)

QueryResult 類別

QueryResult 類別包含對於 Orchestrator 透過外掛程式存取之物件取得的 find 查詢結果。

```
public class QueryResult
extends java.lang.Object
implements java.io.Serializable
```

如果找到的結果總數大於查詢傳回的結果數，則 totalCount 值會大於 QueryResult 傳回的元素數。查詢傳回的結果數是在 vso.xml 檔的查詢語法中定義的。

QueryResult 類別定義下列方法：

方法	傳回	說明
addElement(java.lang.Object element)	作廢	將元素新增至 QueryResult
addElements(java.util.List elements)	作廢	將元素的清單新增至 QueryResult
getElements()	java.util.List	取得外掛程式應用程式中的元素
getTotalCount()	完整	取得外掛技術中出現的所有元素計數。
isPartialResult()	布林值	決定取得的結果是否完整
removeElement(java.lang.Object element)	作廢	移除外掛技術中的元素
setElements(java.util.List elements)	作廢	在外掛技術中設定元素
setTotalCount(long totalCount)	作廢	設定外掛技術中出現的元素總數

建構函數

- QueryResult()
- QueryResult(java.util.List ret)
- QueryResult(java.util.List elements, long totalCount)

SDKFinderProperty 類別

SDKFinderProperty 類別可定義方法來取得和設定 Orchestrator Finder 物件在外掛技術中找到的物件內容。IDynanmicFinder.getProperties 方法會傳回 SDKFinderProperty 物件。

```
public class SDKFinderProperty
extends java.lang.Object
```

SDKFinderProperty 類別定義下列方法：

方法	傳回	說明
getAttributeName()	java.lang.String	取得物件屬性名稱
getBeanProperty()	java.lang.String	取得 Java bean 的內容
getDescription()	java.lang.String	取得物件說明
getDisplayName()	java.lang.String	取得物件顯示名稱
getPossibleResultType()	java.lang.String	取得 Finder 傳回的可能結果類型
getPropertyAccessor()	java.lang.String	取得物件內容存取子
getPropertyAccessorTree()	java.lang.Object	取得物件內容存取子樹狀結構
isHidden()	布林值	顯示或隱藏物件
isShowInColumn()	布林值	顯示或隱藏資料庫資料行中的物件
isShowInDescription()	布林值	顯示或隱藏物件說明
setAttributeName(java.lang.String attributeName)	作廢	設定物件屬性名稱
setBeanProperty(java.lang.String beanProperty)	作廢	設定 Java bean 中的內容
setDescription(java.lang.String description)	作廢	設定物件說明
setDisplayName(java.lang.String displayName)	作廢	設定物件顯示名稱
setHidden(boolean hidden)	作廢	顯示或隱藏物件
setPossibleResultType(java.lang.String possibleResultType)	作廢	設定 Finder 傳回的可能結果類型
setPropertyAccessor(java.lang.String propertyAccessor)	作廢	設定物件內容存取子
setPropertyAccessorTree(java.lang.Object propertyAccessorTree)	作廢	設定物件內容存取子樹狀結構
setShowInColumn(boolean showInTable)	作廢	顯示或隱藏資料庫資料行中的物件
setShowInDescription(boolean showInDescription)	作廢	顯示或隱藏物件說明

建構函數

SDKFinderProperty(java.lang.String attributeName, java.lang.String displayName, java.lang.String beanProperty, java.lang.String propertyAccessor)

PluginExecutionException 類別

如果外掛程式執行作業時發生例外狀況，PluginExecutionException 類別將傳回錯誤訊息。

```
public class PluginExecutionException
extends java.lang.Exception
implements java.io.Serializable
```

PluginExecutionException 類別繼承 class java.lang.Throwable 的下列方法：

fillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace、printStackTrace、printStackTrace、setStackTrace、toStringfillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace

建構函數

PluginExecutionException(java.lang.String message)

PluginOperationException 類別

PluginOperationException 類別會處理外掛程式作業期間發生的錯誤。

```
public class PluginOperationException
extends java.lang.RuntimeException
implements java.io.Serializable
```

PluginOperationException 類別繼承 class java.lang.Throwable 的下列方法：

fillInStackTrace、getCause、getLocalizedMessage、getMessage、getStackTrace、initCause、printStackTrace、printStackTrace、printStackTrace、setStackTrace、toString

建構函數

PluginOperationException(java.lang.String message)

HasChildrenResult 列舉

HasChildrenResult 列舉宣告指定的父系是否有子系。IPluginFactory.hasChildrenInRelation 方法會傳回 HasChildrenResult 物件。

```
public enum HasChildrenResult
extends java.lang.Enum<HasChildrenResult>
implements java.io.Serializable
```

HasChildrenResult 列舉定義下列常數：

- public static final HasChildrenResult Yes

- `public static final HasChildrenResult No`
- `public static final HasChildrenResult Unknown`

`HasChildrenResult` 列舉定義下列方法：

方法	傳回	說明
`getValue()`	int	傳回下列其中一個值。 **1** 父系有子系 **-1** 父系沒有子系 **0** 參數未知或無效
`valueOf(java.lang.String name)`	`static HasChildrenResult`	傳回此類型的列舉常數及指定的名稱。字串必須完全符合用於宣告此類型列舉常數的識別碼。請勿在列舉名稱中使用空白字元。
`values()`	`static HasChildrenResult[]`	依照常數被宣告的順序，傳回包含此列舉類型之常數的陣列。此方法會逐一查看常數，方法如下： `for (HasChildrenResult c : HasChildrenResult.values()) System.out.println(c);`

`HasChildrenResult` 列舉會從 `class java.lang.Enum` 繼承下列方法：

`clone`、`compareTo`、`equals`、`finalize`、`getDeclaringClass`、`hashCode`、`name`、`ordinal`、`toString`、`valueOf`

ScriptingAttribute 註解類型

`ScriptingAttribute` 註解類型會在外掛技術中加上物件中屬性的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value={METHOD,FIELD})
public @interface ScriptingAttribute
```

`ScriptingAttribute` 註解類型有下列值：

```
public abstract java.lang.String value
```

ScriptingFunction 註解類型

`ScriptingFunction` 註解類型可以加上方法的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value={METHOD,CONSTRUCTOR})
public @interface ScriptingFunction
```

ScriptingFunction 註解類型有下列值：

```
public abstract java.lang.String value
```

ScriptingParameter 註解類型

ScriptingParameter 註解類型可以加上參數的註解，以做為指令碼中的內容。

```
@Retention(value=RUNTIME)
@Target(value=PARAMETER)
public @interface ScriptingParameter
```

ScriptingParameter 註解類型有下列值：

```
public abstract java.lang.String value
```

vso.xml 外掛程式定義檔案元素

vso.xml 檔案包含一組標準元素。有些元素為強制，有些則為選用。每個元素都有屬性，可為您要對應至 Orchestrator 物件與作業的物件與作業定義值。

此外，元素可以沒有子系元素，也可以有多個子系元素。子系元素可進一步定義父系元素。同一個子系元素可以出現在多個父系元素中。例如，description 元素沒有子系元素，但顯示為許多父系元素 (module、example、trigger、gauge、finder、constructor、method、object 及 enumeration) 的子系元素。

之後的每個元素都會列出其屬性、父系與子系元素。

module 元素

模組說明提供予 Orchestrator 的一組外掛程式物件。

模組包含如何將外掛技術的資料對應至 Java 類別、版本設定、如何部署模組，以及外掛程式如何在 Orchestrator 詳細目錄中出現的資訊。

<module> 元素為選用。<module> 元素擁有下列屬性：

屬性	值	說明
name	字串	定義外掛程式中所有 <finder> 元素的類型。強制屬性。
version	數字	外掛程式版本號碼，在新版外掛程式中重新載入套件時使用。強制屬性。
build-number	數字	外掛程式組建編號，在新版外掛程式中重新載入套件時使用。強制屬性。
image	映像檔	在 Orchestrator 詳細目錄中顯示的圖示。強制屬性。

屬性	值	說明
display-name	字串	在 Orchestrator 詳細目錄中出現的名稱。選用屬性。
interface-mapping-allowed	true 或 false	VMware 強烈建議不進行介面對應。選用屬性。

表 6-3. 元素記錄

父系元素	子系元素
無	■ <description> ■ <installation> ■ <configuration> ■ <finder-datasources> ■ <inventory> ■ <finders> ■ <scripting-objects> ■ <enumerations>

description 元素

<description> 元素提供 API Explorer 說明文件中所顯示之外掛程式的元素說明。

您新增的文字出現在 <description> 和 </description> 標籤之間的 API Explorer 說明文件。

<description>元素為選用。<description>元素沒有屬性。

表 6-4. 元素記錄

父系元素	子系元素
■ <module> ■ <example> ■ <trigger> ■ <gauge> ■ <finder> ■ <constructor> ■ <method> ■ <object> ■ <enumeration>	無

取代元素

<deprecated> 元素標記在 API Explorer 說明文件中取代的物件及方法。

您新增的文字出現在 <deprecated> 和 </deprecated> 標籤之間的 API Explorer 說明文件。

<deprecated>元素為選用。<deprecated>元素沒有屬性。

表 6-5. 元素記錄

父系元素	子系元素
■ <method> ■ <object>	無

url 元素

<url> 元素提供的 URL 指向物件或列舉的外部說明文件。

您在 <url> 與 </url> 標記之間提供 URL。

<url>元素為選用。<url>元素沒有屬性。

表 6-6. 元素記錄

父系元素	子系元素
■ <enumeration> ■ <object>	無

installation 元素

<installation> 元素可讓您在伺服器啟動時安裝套件或執行指令碼。

<installation>元素為選用。<installation> 元素擁有下列屬性：

屬性	值	說明
mode	always、never 或 version	設定 mode 值會導致 Orchestrator 伺服器啟動時發生下列行為。 ■ 動作 always 會執行 ■ 動作 never 會執行 ■ 動作會在伺服器偵測到新版的外掛程式時執行 強制屬性。

表 6-7. 元素記錄

父系元素	子系元素
<module>	<action>

action 元素

<action> 元素會指定 Orchestrator 伺服器啟動時執行的動作。

<action> 元素屬性會提供 Orchestrator 套件的路徑，或提供定義外掛程式啟動時之行為的指令碼。

<action>元素為選用。外掛程式可擁有無限數量的 <action> 元素。<action>元素擁有下列屬性：

屬性	值	說明
resource	字串	dar 檔案根目錄中 Java 套件或指令碼的路徑強制屬性。
type	install-package 或 execute-script	在 Orchestrator 伺服器中安裝指定的 Orchestrator 套件，或執行指定的指令碼。強制屬性。

表 6-8. 元素記錄

父系元素	子系元素
<installation>	無

Finder-datasources 元素

<finder-datasources> 元素是 <finder-datasource> 元素的 Container。

<finder-datasources>元素為選用。<finder-datasources>元素沒有屬性。

表 6-9. 元素記錄

父系元素	子系元素
<module>	<finder-datasource>

finder-datasource 元素

<finder-datasource> 元素指出您為外掛程式建立 IPluginAdaptor 執行的 Java 類別檔案。

您設定 Orchestrator 如何存取在 <finder-datasource> 元素中的外掛技術物件。<finder-datasource>元素識別您建立外掛程式介面卡的 Java 類別。外掛程式介面卡類別執行個體化您建立的外掛程式原廠。外掛程式原廠定義尋找外掛技術物件的方法。您可在 <finder-datasource> 元素中設定原廠執行 finder 方法呼叫逾時。不同的逾時可套用至 IPluginFactory 介面的不同 finder 方法。

<finder-datasource>元素為選用。外掛程式可擁有無限數量的 <finder-datasources> 元素。<finder-datasource>元素擁有下列屬性：

屬性	值	說明
name	字串	識別 <finder> 元素 datasource 屬性中的資料來源。等價於 XML id。強制屬性。
adaptor-class	Java 類別	指出您定義的 IPluginAdaptor 執行方式，建立外掛程式介面卡，如 com.vmware.plugins.sample.Adaptor。強制屬性。
concurrent-call	true (預設) 或 false	允許多名使用者同時存取介面卡。若外掛程式不支援同時呼叫，您必須設定 concurrent-call 為 false。選用屬性。
invoker-mode	direct (預設) 或 timeout	設定 finder 函式上的逾時。若設為 direct，finder 函式呼叫永不逾時。若設為 timeout，Orchestrator 伺服器套用對應 finder 方法的逾時時間。選用屬性。

屬性	值	說明
anonymous-login-mode	never (預設) 或 always	傳遞或不傳遞使用者的使用者名稱及密碼至外掛程式。選用屬性。
timeout-fetch-relation	數字；預設 30 秒	套用至 findRelation() 的呼叫。選用屬性。
timeout-find-all	數字；預設 60 秒	套用至 findAll() 的呼叫。選用屬性。
timeout-find	數字；預設 60 秒	套用至 find() 的呼叫。選用屬性。
timeout-has-children-in-relation	數字；預設 2 秒	套用至 findChildrenInRelation() 的呼叫。選用屬性。
timeout-execute-plugin-command	數字；預設 30 秒	套用至 executePluginCommand() 的呼叫。選用屬性。

表 6-10. 元素記錄

父系元素	子系元素
<finder-datasources>	無

inventory 元素

對於 Orchestrator 用戶端**詳細目錄**視圖和物件選項對話方塊中出現的外掛程式，<inventory> 元素定義階層清單的根階層。

<inventory> 元素並非代表外掛程式應用程式中的物件，而代表在 Orchestrator 指令碼 API 中做為物件的外掛程式本身。

<inventory> 元素為選用。<inventory>元素擁有下列屬性。

屬性	值	說明
type	Orchestrator 物件類型	代表物件根階層的 <finder> 元素類型。強制屬性。

表 6-11. 元素記錄

父系元素	子系元素
<module>	無

Finders 元素

<finders> 元素是所有 <finder> 元素的 Container。

<finders>元素為選用。<finders>元素沒有屬性。

表 6-12. 元素記錄

父系元素	子系元素
<module>	<finder>

Finder 元素

<finder> 元素在 Orchestrator 用戶端中代表透過外掛程式找到的物件類型。

<finder> 元素可識別能定義物件 Finder 所代表之物件的 Java 類別。<finder> 元素可定義物件在 Orchestrator 用戶端介面中顯示的方式。也會識別 Orchestrator 指令碼 API 所定義用來代表此物件的指令碼物件。

Finder 將做為不同類型外掛技術所使用之各種物件格式的共同介面。

<finder>元素為選用。外掛程式可擁有無限數量的 <finder> 元素。<finder> 元素可定義下列屬性：

屬性	值	說明
type	Orchestrator 物件類型	Finder 代表的物件類型。強制屬性。
datasource	<finder-datasource name> 屬性	使用資料來源 refid 識別可定義物件的 Java 類別。強制屬性。
dynamic-finder	Java 方法	定義您在 IDynamicFinder 執行個體中實作的自訂 Finder 方法，藉此以程式設計的方式傳回 Finder 的識別碼與內容，來取代在 vso.xml 檔案中進行定義這個方法。選用屬性。
hidden	true 或 false (預設)	若為 true，將會在 Orchestrator 用戶端中隱藏 Finder。選用屬性。
image	圖形檔案的路徑	16x16 圖示，在 Orchestrator 用戶端的階層清單中代表 Finder。選用屬性。
java-class	Java 類別的名稱	此 Java 類別可定義 Finder 尋找並對應至指令碼物件的物件。選用屬性。
script-object	<scripting-object type> 屬性	此 Finder 要對應的 <scripting-object> 類型 (若有)。選用屬性。

表 6-13. 元素記錄

父系元素	子系元素
<finders>	■ <id> ■ <description> ■ <properties> ■ <default-sorting> ■ <inventory-children> ■ <relations> ■ <inventory-tabs> ■ <events>

properties 元素

<properties> 元素是 <finder><property> 元素的 Container。

<properties>元素為選用。<properties>元素沒有屬性。

表 6-14. 元素記錄

父系元素	子系元素
<finder>	<property>

property 元素

<property> 元素會將找到的物件本身的內容對應至 Java 內容或方法呼叫。

您實作外掛程式處理站來取得程序的外掛程式處理站實作內容時，可以呼叫 SDKFinderProperty 類別的方法。

您可以在 Orchestrator 用戶端的視圖中顯示或隱藏物件內容。您也可以使用列舉來定義物件內容。

<property>元素為選用。外掛程式可擁有無限數量的 <property> 元素。<property>元素擁有下列屬性：

屬性	值	說明
name	Finder 名稱	FinderResult 用來儲存元素的名稱。強制屬性。
display-name	Finder 名稱	顯示的內容名稱。選用屬性。
bean-property	內容名稱	您可以使用 get 和 set 作業，藉由 bean-property 屬性識別要取得的內容。如果您識別名稱為 MyProperty 的內容，外掛程式將定義 getMyProperty 和 setMyProperty 作業。 您可以設定一個或另一個 bean-property 或 property-accessor，但是不可同時設定兩個。選用屬性。
property-accessor	從物件取得內容值的方法	property-accessor 屬性可讓您定義 OGNL 運算式以驗證物件的內容。 您可以設定一個或另一個 bean-property 或 property-accessor，但是不可同時設定兩個。選用屬性。
show-in-column	true (預設) 或 false	若是 true，此內容將出現於 Orchestrator 用戶端結果資料表。選用屬性。
show-in-description	true (預設) 或 false	若是 true，此內容將出現於物件說明。選用屬性。
hidden	true 或 false (預設)	若是 true，此內容將一律隱藏。選用屬性。
linked-enumeration	列舉名稱	將 Finder 內容連結至列舉。選用屬性。

表 6-15. 元素記錄

父系元素	子系元素
<properties>	子系元素

relations 元素

<relations> 元素是 <finder><relation> 元素的 Container。

<relations>元素為選用。<relations>元素沒有屬性。

表 6-16. 元素記錄

父系元素	子系元素
<finder>	<relation>

relation 元素

<relation> 元素將定義物件與其他物件之間的關係。

您可以在 <relation> 元素中定義關係名稱。

<relation>元素為選用。外掛程式可擁有無限數量的 <relation> 元素。<relation>元素擁有下列屬性：

屬性	值	說明
name	關係名稱。	此關係的名稱。強制屬性。
type	Orchestrator 物件類型	藉由此關係與其他物件相關聯的物件類型。強制屬性。
cardinality	to-one 或 to-many	將物件之間的關係定義為一對一或一對多。選用屬性。

表 6-17. 元素記錄

父系元素	子系元素
<relations>	無

id 元素

<id> 元素會定義一個方法，來取得 Finder 用來識別物件的唯一識別碼。

<id>元素為選用。<id>元素擁有下列屬性：

屬性	值	說明
accessor	方法名稱	accessor 屬性可讓您定義 OGNL 運算式以驗證物件的內容。強制屬性。

表 6-18. 元素記錄

父系元素	子系元素
<finder>	無

inventory-children 元素

<inventory-children> 元素定義顯示 Orchestrator 用戶端**詳細目錄**視圖和物件選項方塊之中物件的清單階層。

<inventory-children>元素為選用。<inventory-children>元素沒有屬性。

表 6-19. 元素記錄

父系元素	子系元素
<finder>	<relation-link>

relation-link 元素

<relation-link> 元素定義**詳細目錄**索引標籤中的父物件與子物件之間的階層。

<relation-link>元素為選用。外掛程式可擁有無限數量的 <relation-link> 元素。<relation-link>元素擁有下列屬性。

類型	值	說明
name	關係名稱。	關係名稱的 refid。強制屬性。

表 6-20. 元素記錄

父系元素	子系元素
<inventory-children>	無

events 元素

<events> 元素是 <trigger> 與 <gauge> 元素的 Container。

<events> 元素中可包含不限數目的觸發器或量表。

<events>元素為選用。<events>元素沒有屬性。

表 6-21. 元素記錄

父系元素	子系元素
<finder>	■ <trigger> ■ <gauge>

觸發器元素

<trigger> 元素宣告您可用於此 finder 的觸發器。您必須執行 IPluginAdaptor 的 registerEventPublisher() 和 unregisterEventPublisher() 方法，才能設定觸發器。

<trigger> 元素為選用。<trigger>元素擁有下列屬性。

類型	值	說明
name	觸發器名稱	此觸發器的名稱。強制屬性。

表 6-22. 元素記錄

父系元素	子系元素
<events>	■ <description> ■ <trigger-properties>

trigger-property 元素

<trigger-properties> 元素是 <trigger-property> 元素的 Container。

<trigger-properties>元素為選用。<trigger-properties>元素沒有屬性。

表 6-23. 元素記錄

父系元素	子系元素
<trigger>	<trigger-property>

trigger-property 元素

<trigger-property> 元素會定義識別觸發器物件的內容。

<trigger-property>元素為選用。外掛程式可擁有無限數量的 <trigger-property> 元素。<trigger-property>元素擁有下列屬性：

類型	值	說明
name	觸發器名稱	觸發器的名稱。選用屬性。
display-name	觸發器名稱	在 Orchestrator 用戶端中顯示的名稱。選用屬性。
type	觸發器類型	定義觸發器的物件類型。強制屬性。

表 6-24. 元素記錄

父系元素	子系元素
<trigger-properties>	無

gauge 元素

<gauge> 元素定義您可用於此 Finder 的量表。您必須實作 IPluginAdaptor 的 registerEventPublisher() 和 unregisterEventPublisher() 方法，才能設定量表。

<gauge> 元素為選用。外掛程式可擁有無限數量的 <gauge> 元素。<gauge>元素擁有下列屬性：

類型	值	說明
name	量表名稱	量表的名稱。強制屬性。
min-value	數字	臨界值下限。選用屬性。
max-value	數字	臨界值上限。選用屬性。

類型	值	說明
unit	物件類型	定義量表的物件類型。強制屬性。
format	字串	受監控值的格式。選用屬性。

表 6-25. 元素記錄

父系元素	子系元素
<events>	<description>

scripting-objects 元素

<scripting-objects> 元素是 <object> 元素的 Container。

<scripting-objects>元素為選用。<scripting-objects>元素沒有屬性。

表 6-26. 元素記錄

父系元素	子系元素
<module>	<object>

object 元素

<object> 元素會將外掛技術的建構函數、屬性和方法對應至 Orchestrator 指令碼 API 公開的 JavaScript 物件類型。

關於物件命名慣例，請參閱命名外掛程式物件。

<object>元素為選用。外掛程式可擁有無限數量的 <object> 元素。<object>元素擁有下列屬性：

類型	值	說明
script-name	JavaScript 名稱	類別的指令碼名稱。必須是全域唯一。強制屬性。
java-class	Java 類別	此 JavaScript 類別包裝的 Java 類別。強制屬性。
create	true (預設) 或 false	若是 true，您可以建立此類別的新執行個體。選用屬性。
strict	true 或 false (預設)	若要 true，您僅能呼叫您在 vso.xml 檔中註解或宣告的方法。選用屬性。
is-deprecated	true 或 false (預設)	若是 true，物件將對應已過時的 Java 類別。選用屬性。
since-version	字串	Java 類別過時以後的版本。選用屬性。

表 6-27. 元素記錄

父系元素	子系元素
<scripting-objects>	■ <description> ■ <deprecated> ■ <url> ■ <constructors> ■ <attributes> ■ <methods> ■ <singleton>

constructors 元素

<constructors> 元素是 <object><constructor> 元素的 Container。

<constructors>元素為選用。<constructors>元素沒有屬性。

表 6-28. 元素記錄

父系元素	子系元素
<object>	<constructor>

constructor 元素

<constructor> 元素定義建構函數方法。<constructor> 方法會在 API Explorer 中產生說明文件。

<constructor> 元素為選用。外掛程式可擁有無限數量的 <constructor> 元素。<constructor>元素沒有屬性。

表 6-29. 元素記錄

父系元素	子系元素
<constructors>	■ <description> ■ <parameters>

Constructor parameters 元素

<parameters> 元素是 <constructor><parameter> 元素的 Container。

<parameters>元素為選用。<parameters>元素沒有屬性。

表 6-30. 元素記錄

父系元素	子系元素
<constructor>	<parameter>

Constructor parameter 元素

<parameter> 元素定義建構函數的參數。

<parameter>元素為選用。外掛程式可擁有無限數量的 <parameter> 元素。<parameter>元素擁有下列屬性：

類型	值	說明
name	字串	在 API 說明文件中使用的參數名稱。強制屬性。
type	Orchestrator 參數類型	在 API 說明文件中使用的參數類型。強制屬性。
is-optional	true 或 false	如果為 true，則此值會為空值。選用屬性。
since-version	字串	方法版本。選用屬性。

表 6-31. 元素記錄

父系元素	子系元素
<parameters>	無

attributes 元素

<attributes> 元素是 <object><attribute> 元素的 Container。

<attributes>元素為選用。<attributes>元素沒有屬性。

表 6-32. 元素記錄

父系元素	子系元素
<object>	<attribute>

attribute 元素

<attribute> 元素會將 Java 類別的屬性從外掛的技術，對應至 Orchestrator JavaScript 引擎使其可用的 JavaScript 屬性。

<attribute>元素為選用。外掛程式可擁有無限數量的 <attribute> 元素。<attribute>元素擁有下列屬性：

類型	值	說明
java-name	Java 屬性	Java 屬性名稱。強制屬性。
script-name	JavaScript 物件	對應 JavaScript 物件的名稱。強制屬性。
return-type	字串	此屬性傳回的物件類型。顯示在 API Explorer 說明文件中。選用屬性。 **備註** 如果 JavaScript 傳回類型為 Properties，則支援的基礎 Java 實作為 java.util.HashMap 與 java.util.Hashtable。
read-only	true 或 false	如果為 true，您便無法修改此屬性。選用屬性。

類型	值	說明
is-optional	true 或 false	如果為 true，則此欄位會為空值。選用屬性。
show-in-api	true 或 false	如果為 false，則此屬性不會在 API 說明文件中顯示。選用屬性。
is-deprecated	true 或 false	如果為 true，則物件會對應過時的屬性。選用屬性。
since-version	數字	屬性為過時的版本。選用屬性。

表 6-33. 元素記錄

父系元素	子系元素
<attributes>	無

methods 元素

<methods> 元素是 <object><method> 元素的 Container。

<methods>元素為選用。<methods>元素沒有屬性。

表 6-34. 元素記錄

父系元素	子系元素
<object>	<method>

method 元素

<method> 元素將外掛技術中的 Java 方法對應至 Orchestrator JavaScript 引擎公開的 JavaScript 方法。

<method>元素為選用。外掛程式可擁有無限數量的 <method> 元素。<method>元素擁有下列屬性：

類型	值	說明
java-name	Java 方法	Java 方法簽章的名稱，有引數類型在括號中，例如 getVms(DataStore)。強制屬性。
script-name	JavaScript 方法	對應 JavaScript 方法的名稱。強制屬性。
return-type	Java 物件類型	此方法取得的類型。選用屬性。 **備註** 如果 JavaScript 傳回類型為 Properties，則支援的基礎 Java 實作為 java.util.HashMap 與 java.util.Hashtable。
static	true 或 false	若是 true，此方法為靜態。選用屬性。
show-in-api	true 或 false	若是 false，此方法不會出現在 API 說明文件中。選用屬性。
is-deprecated	true 或 false	若是 true，物件將對應已過時的方法。選用屬性。
since-version	數字	已過時方法的版本。選用屬性。

表 6-35. 元素記錄

父系元素	子系元素
<methods>	■ <deprecated> ■ <description> ■ <example> ■ <parameters>

元素範例

<example> 元素可讓您新增代碼範例至 API Explorer 說明文件中顯示的 Javascript 法。

<example>元素為選用。<example>元素沒有屬性。

表 6-36. 元素記錄

父系元素	子系元素
<method>	■ <code> ■ <description>

代碼元素

<code> 元素提供出現在 API Explorer 說明文件中的範例代碼。

您在 <code> 與 </code> 標籤之間提供代碼範例。<code>元素為選用。<code>元素沒有屬性。

表 6-37. 元素記錄

父系元素	子系元素
<example>	無

Method parameters 元素

<parameters> 元素是 <method><parameter> 元素的 Container。

<parameters>元素為選用。<parameters>元素沒有屬性。

表 6-38.

父系元素	子系元素
<method>	<parameter>

Method parameter 元素

<parameter> 元素定義方法的輸入參數。

<parameter>元素為選用。外掛程式可擁有無限數量的 <parameter> 元素。<parameter>元素擁有下列屬性：

類型	值	說明
name	字串	參數名稱。強制屬性。
type	Orchestrator 參數類型	參數類型。強制屬性。
is-optional	true 或 false	如果為 true，則此值會為空值。選用屬性。
since-version	字串	方法版本。選用屬性。

表 6-39. 元素記錄

父系元素	子系元素
<parameters>	無

單一元素

<singleton> 元素會將 JavaScript 指令碼物件當成單一執行個體建立。

單一物件的行為方式與靜態 Java 類別相同。單一物件定義外掛程式使用的一般物件，而非 Orchestrator 採用外掛技術存取的定義物件指定執行個體。例如：您可使用單一物件建立與外掛技術的連線。

<singleton>元素為選用。<singleton>元素擁有下列屬性：

類型	值	說明
script-name	JavaScript 物件	對應 JavaScript 物件的名稱。強制屬性。
datasource	Java 物件	用於此 JavaScript 物件的來源 Java 物件。強制屬性。

表 6-40. 元素記錄

父系元素	子系元素
<object>	無

enumerations 元素

<enumerations> 元素是 <enumeration> 元素的 Container。

<enumerations>元素為選用。<enumerations>元素沒有屬性。

表 6-41. 元素記錄

父系元素	子系元素
<module>	<enumeration>

enumeration 元素

<enumeration> 元素可定義適用於特定類型的所有物件的通用值。

若特定類型的所有物件需要特定屬性，且該屬性的值範圍受到限制，您可以定義不同的值做為列舉項目。例如，若物件的類型需要 color 屬性，且可用色彩只有紅色、藍色、綠色，您便可以定義三個列舉項目來定義這三個色彩值。您可以將項目定義為 enumeration 元素的子系元素。

<enumeration>元素為選用。外掛程式可擁有無限數量的 <enumeration> 元素。<enumeration>元素擁有下列屬性。

類型	值	說明
type	Orchestrator 物件類型	列舉類型。強制屬性。

表 6-42. 元素記錄

父系元素	子系元素
<enumerations>	■ <url> ■ <description> ■ <entries>

entries 元素

<entries> 元素是 <enumeration><entry> 元素的 Container。

<entries>元素為選用。<entries>元素沒有屬性。

表 6-43. 元素記錄

父系元素	子系元素
<enumeration>	<entry>

entry 元素

<entry> 元素提供列舉屬性的值。

<entry>元素為選用。外掛程式可擁有無限數量的 <entry> 元素。<entry>元素擁有下列屬性：

類型	值	說明
id	Text	物件用來將列舉項目設為屬性的識別碼。強制屬性。
name	Text	項目名稱。強制屬性。

表 6-44. 元素記錄

父系元素	子系元素
<entries>	無

Orchestrator 外掛程式開發最佳做法

您可深入瞭解外掛程式結構與內容，以及瞭解避免特定問題的方式，片面改善您開發的 Orchestrator 外掛程式。

- 建置 Building Orchestrator 外掛程式的方式

  您可利用不同的方式建置 Orchestrator 外掛程式。您可開始逐層建置外掛程式，或同時開始建置所有外掛程式的圖層。

- Orchestrator 外掛程式型別

  您可透過使用外掛程式整合一般用途的程式庫或公用程式，如 XML 或 SSH，以及整部系統，如搭載 Orchestrator 的 vCloud Director。視您整合 Orchestrator 的技術而定，外掛程式可分類為服務外掛程式或一般用途的外掛程式和系統外掛程式。

- 外掛程式執行

  您可在組織外掛程式結構、執行所需 Java 類別及 JavaScript 物件、開發外掛程式工作流程與活動，以及提供工作流程簡報時使用特定實用的作法與技術。

- Orchestrator 外掛程式開發建議事項

  在開發 Orchestrator 外掛程式的不同元件時，落實特定作法有助於您改善外掛程式的品質。

- 記錄外掛程式使用者介面字串與 API

  當您撰寫適用於 Orchestrator 外掛程式與相關 API 說明文件的使用者介面 (UI) 字串時，請依照樣式與格式的公認規則。

建置 Building Orchestrator 外掛程式的方式

您可利用不同的方式建置 Orchestrator 外掛程式。您可開始逐層建置外掛程式，或同時開始建置所有外掛程式的圖層。

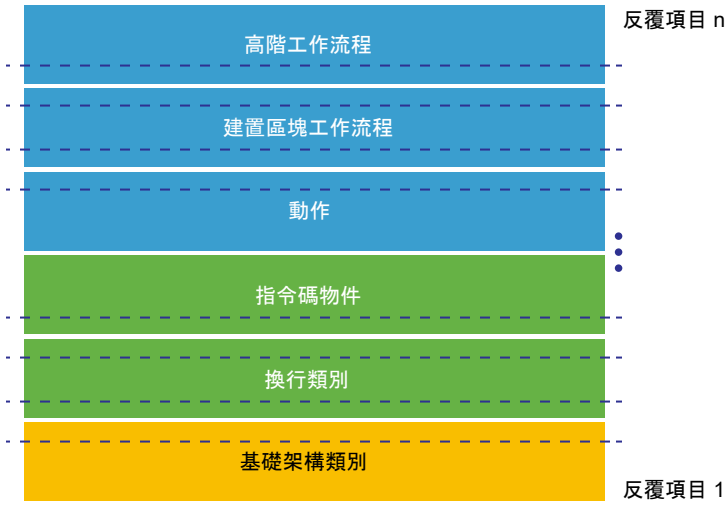
如需有關外掛程式圖層的資訊，請參閱 Orchestrator 外掛程式的結構。

由下而上開發外掛程式

外掛程式可利用由下而上的開發方式逐層建置。

由下而上開發方式可從低層級開始持續往高層級逐層建置外掛程式。當此方式混合互動式及反覆性開發方式，然後為各反覆項目交付部分或整層的外掛程式。在反覆項目 N 結束時將徹底完成外掛程式。

圖 6-3. 由下而上開發外掛程式



由下而上開發外掛程式的優點為：可在一段時間專注於單層開發。

由下而上開發外掛程式則須考慮下列的缺點。

- 在完成部分插入項目前，難以明確顯示外掛程式開發的進度。
- 無法完美整合至敏捷式開發作法中。

對含有縮減或不存在的換行類別、指令碼物件、動作或工作流程組合的小型外掛程式而言，由下而上的開發程序即足以適用。

自上而下外掛程式開發

外掛程式可使用自上而下開發法，透過配量為自上而下功能來建置。

自上而下法混合敏捷式開發程序後，各反覆項目將出現新功能。最後，在反覆項目 N 結束時將徹底執行外掛程式。

圖 6-4. 自上而下外掛程式開發

自上而下外掛程式開發法含有下列優點。

- 由於各反覆項目的新功能已完成且可釋出外掛程式，並在各反覆項目後使用，因此外掛程式的進度可在第一個反覆項目中輕易顯示。
- 完成功能的垂直配量可非常明確定義成功標準與定義已完成項目，以及改善開發人員、產品管理和品管 (QA) 工程師之間的通訊。
- 允許 QA 工程師從開發程序開始時開始測試及自動化。此法可產生寶貴的意見反應並減少整體專案交付時間範圍。

自上而下外掛程式開發法的缺點為：需同時在不同層進行開發。

您的大部分外掛程式皆應套用自上而下外掛程式開發程序。適合有動態需求的外掛程式。

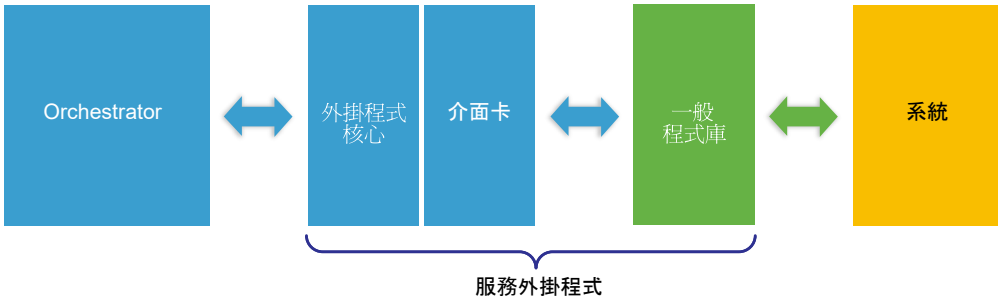
Orchestrator 外掛程式型別

您可透過使用外掛程式整合一般用途的程式庫或公用程式，如 XML 或 SSH，以及整部系統，如搭載 Orchestrator 的 vCloud Director。視您整合 Orchestrator 的技術而定，外掛程式可分類為服務外掛程式或一般用途的外掛程式和系統外掛程式。

服務外掛程式

服務外掛程式或一般用途外掛程式提供的功能可視為 Orchestrator 內的服務。

圖 6-5. 服務外掛程式架構



服務外掛程式會公開一般程式庫或公用程式至 Orchestrator，如 XML、SSH 或 SOAP。例如：下列可在 Orchestrator 中使用的外掛程式皆為服務外掛程式。

JDBC 外掛程式 讓您在工作流程內使用任何資料庫。

郵件外掛程式 讓您在工作流程內傳送電子郵件。

SSH 外掛程式 讓您在工作流程內開啟 SSH 連線及執行命令。

XML 外掛程式 讓您在工作流程內管理 XML 文件。

服務外掛程式具有下列特性。

複雜性 服務外掛程式擁有低至中級的複雜性。服務外掛程式在 Orchestrator 內部公開特定程式庫或部分程式庫，以提供具體功能。例如：XML 外掛程式會新增執行的文件物件模型 (DOM) XML 剖析器至 Orchestrator JavaScript API。

大小 服務外掛程式的大小皆相對要小。這些程式需要與所有外掛程式相同的基本類別組合，以及其他提供新指令碼物件以新增功能的類別。

詳細目錄 服務外掛程式需要小型物件詳細目錄才能運作，或者完全不需要詳細目錄。服務外掛程式含有一般及小型物件模型，因此不需要在 Orchestrator 詳細目錄內顯示此模型。

系統外掛程式

系統外掛程式連接 Orchestrator 工作流程引擎至外部系統，以便您可以協調外部系統。

下列為系統外掛程式的範例。

vCenter Server 外掛程式 讓您使用工作流程管理 vCenter Server 執行個體。

vCloud Director 外掛程式 讓您在工作流程內與 vCloud Director 安裝互動。

Cisco UCSM 外掛程式 讓您在工作流程內與 Cisco 實體互動。

下列為系統外掛程式的主要特色。

複雜性 系統外掛程式擁有比泛用外掛程式更高程度的複雜性，因為他們公開的技術相對複雜。系統外掛程式必須代表 Orchestrator 內所有外部系統的元素，並

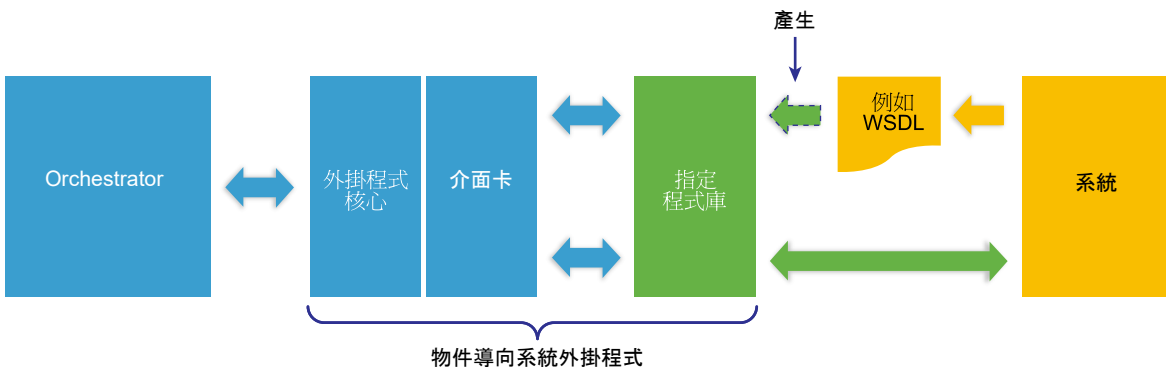
	與外部系統互動同時在 Orchestrator 中提供功能。若外部系統提供整合機制，則您可於 Orchestrator 中更輕鬆的公開系統功能。不過，除了代表 Orchestrator 外部系統元素外，系統外掛程式也可能需要提供高擴充性、快取機制、處理事件及通知等。
大小	系統外掛程式的大小為中至大型。系統外掛程式因通常需提供大量指令碼物件，所以需要除基本類別組合以外的類別。系統外掛程式可能需要一些其他協助程式和與其互動的輔助類別。
詳細目錄	通常系統外掛程式擁有大量的物件，而您必須在詳細目錄中正確公開這些物件，以便在 Orchestrator 中輕鬆找到並搭配執行。由於系統外掛程式需要公開大量物件，因此您應建置輔助工具或程序，為外掛程式盡可能自動產生更多代碼。例如：vCenter Server 外掛程式提供的工具。

物件導向系統外掛程式

物件導向系統提供的互動機制是根據物件及 RPC 為主。

適合物件導向系統最廣泛使用的模型為使用 SOAP 的網路服務模型。模型中物件擁有的屬性組合與物件狀態有關，提供在目標系統端叫用的遠端方法組合。

圖 6-6. 物件導向系統外掛程式



執行物件導向系統的外掛程式時，您可考慮下列項目。

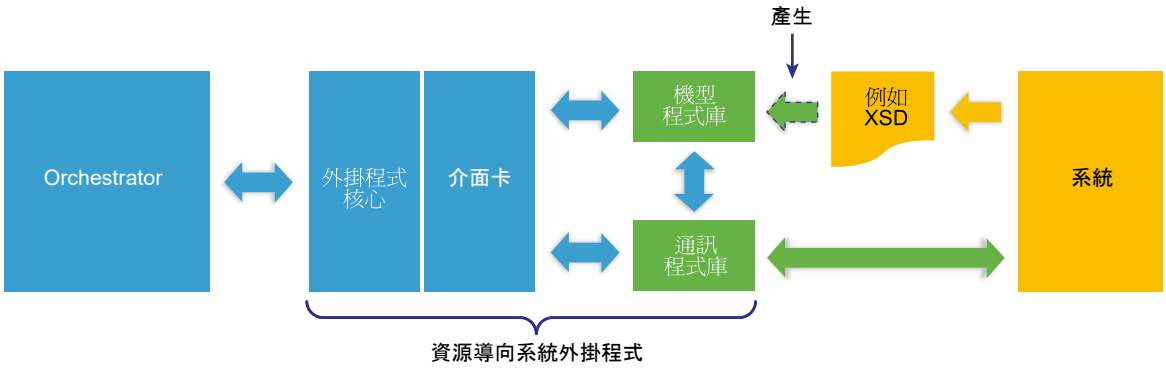
- 若您使用 SOAP，您可使用 WSDL 檔案產生結合物件模型及通訊機制的類別組合。
- 此物件模型幾乎是您必須在 Orchestrator 內公開的所有項目。

資源導向系統外掛程式

資源導向系統提供的互動機制是以使用 HTTP 方法的資源及簡易作業為基礎。

最具代表性的資源導向系統模型是 REST 模型，並與含 XML 的範例結合。此模型內的物件含有與其狀態相關的屬性組合。若要叫用目標系統上的方法 (通訊機制)，您必須使用標準 HTTP 方法，如 `GET`、`POST`、`PUT` 等，並遵循某些慣例。

圖 6-7. 資源導向系統外掛程式



開發資源導向系統的外掛程式時，您可考慮下列項目。

- 若您使用 REST 或只有含 XML 的 HTTP，您可獲得一或多個 XML 配置檔案以供讀取及寫入訊息。在這些配置中，您可產生定義物件模型的類別組合。此類別組合只定義物件狀態，因為以 HTTP 方法隱含定義作業，例如 vCloud Director 外掛程式中的定義，或明確含部分指定 XML 訊息，如 Cisco UCSM 外掛程式。
- 您需在其他類別組合中執行通訊機制。此類別組合定義與原始物件模型互動的新物件模型。通訊機制的物件模型只包含物件與方法。
- 您可公開 Orchestrator 內通訊機制的原始物件及物件模型。視公開物件模型的方式，以及您是否合併兩方相關物件（用於模擬物件導向系統）或維持分離，這可能會增加一些複雜性。

外掛程式執行

您可在組織外掛程式結構、執行所需 Java 類別及 JavaScript 物件、開發外掛程式工作流程與活動，以及提供工作流程簡報時使用特定實用的作法與技術。

- 專案結構

  您可套用用於 Orchestrator 外掛程式專案的標準結構。
- 專案內部

  您可在執行外掛程式時套用特定方式，例如快取物件、讓物件進入背景、複製物件等。您可透過此方式改善外掛程式的效能、避免發生並行處理問題，以及改善 Orchestrator 用戶端回應性。
- 工作流程內部

  您可執行工作流程監控您 Orchestrator 外掛程式執行的長時間作業。
- 工作流程及動作

  若要減輕工作流程開發及使用，您可使用特定的作法。
- 工作流程簡報

  當您建立工作流程簡報後，應套用特定結構與規則。

專案結構

您可套用用於 Orchestrator 外掛程式專案的標準結構。

您可使用含模組的標準 Maven 結構，讓您的外掛程式專案明白瞭解所有功能所在的位置。

表 6-45. 外掛程式專案結構

模組	說明
`/myAwesomePlugin-plugin`	外掛程式專案的根。
`/o11nplugin-myAwesomePlugin`	構成最終外掛程式 DAR 檔案的模組。
`/o11nplugin-myAwesomePlugin-config`	包含外掛程式組態網路應用程式的模組。會產生標準 WAR 檔案。
`/o11nplugin-myAwesomePlugin-core`	此模組包含所有執行任何標準 Orchestrator 外掛程式介面的類別及其使用的其他輔助類別。會產生標準 JAR 檔案。
`/o11nplugin-myAwesomePlugin-model`	此模組包含所有有助於您透過外掛程式整合第三方技術和 Orchestrator 的類別。這些類別不得包含任何標準 Orchestrator 外掛程式 API 的直接參照。
`/o11nplugin-myAwesomePlugin-package`	此模組匯入含動作與工作流程的外部 Orchestrator 套件檔案，並在最終外掛程式 DAR 檔案內加入。此模組為選用。

專案內部

您可在執行外掛程式時套用特定方式，例如快取物件、讓物件進入背景、複製物件等。您可透過此方式改善外掛程式的效能、避免發生並行處理問題，以及改善 Orchestrator 用戶端回應性。

快取物件

您的外掛程式會與遠端服務互動，並由本機物件提供此代表服務端遠端物件的互動。若要達到良好的外掛程式效能以及出色的 Orchestrator UI 回應性，您可快取本機物件，而非每次從遠端服務取得。您可考慮快取範圍，例如一個代表所有外掛程式用戶端的快取，每外掛程式使用者一個快取，以及每第三方服務使用者一個快取。在執行時，您的快取機制會整合外掛程式介面，以尋找物件及讓物件失效。

讓物件進入背景

若您必須在外掛程式詳細目錄中顯示大量物件清單，且沒有擷取這些物件的快速方式，您可讓物件進入背景。您可讓物件進入背景，例如讓物件擁有兩個狀態：`fake` 和 `loaded`。假設 `fake` 物件很容易建立及提供您必須在詳細目錄中顯示的最少資訊，如名稱與識別碼。接著將可隨時傳回 `fake` 物件，並在真正需要所有資訊（真實物件）時，使用實體或外掛程式可自動叫用方法 `load` 以取得真實物件。您甚至可設定載入物件的程序，在傳回假物件後自動開始，以評估使用實體的動作。

複製物件避免發生並行問題

若您針對外掛程式使用快取，您必須複製物件。對每個實體使用隨時傳回相同物件執行個體的快取，以要求可擁有不需要的效果。例如：實體 A 要求物件 O，而該實體會檢視詳細目錄中的物件及其所有屬性。在此同時，實體 B 也會要求物件 O，而實體 A 則會執行工作流程，開始變更物件 O 的屬性。在執行結束時，工作流程會叫用物件的 `update` 方法以更新伺服器端的物件。若實體 A 和實體 B 取得物件 O 的相同執行個體，實體 A 會檢視詳細目錄中實體 B 執行的所有變更，甚至包括伺服器端認可變更前的項目。若執行順利，則沒有問題，但若執行失敗，則不還原實體 A 物件 O 的屬性。在此情況下，若快取（外掛程式 `find` 作業）傳回物件複製品，而非一直相同的執行個體，實體都會在每次使用時檢視與修改本身的複本，至少在 Orchestrator 中避免發生並行問題。

通知其他人變更

當您同時使用快取及複製品物件時，可能會發生問題。最大的問題在於該物件使用的實體檢視，可能不是物件可用的最新版本。例如：若實體顯示詳細目錄，將載入一次物件，但在此同時，若其他實體變更部分物件，第一個實體不會檢視變更。若要避免此問題，您可使用 Orchestrator 外掛程式 API 的 `PluginWatcher` 和 `IPluginPublisher` 方法，告知已變更為允許其他 Orchestrator 用戶端執行個體查看變更。此方式也可在從詳細目錄單一物件變更影響其他詳細目錄物件時，以及需要告知時，套用至 Orchestrator 用戶端唯一執行個體。這些物件或物件部分內容顯示在詳細目錄時，方便使用通知的作業會新增、更新及刪除物件。

隨時啟用尋找任何物件

您必須執行 `IPluginFactory` 介面的 `find` 方法，按型別和識別碼尋找物件。`find` 方法可在重新啟動 Orchestrator 並繼續執行工作流程後直接叫用。

若您沒有查詢服務，請模擬

例如在特定狀況中，Orchestrator 用戶端會需要查詢部分物件或顯示為清單或表單而非樹狀結構。這代表您的外掛程式必須能隨時查詢部分物件組合。若第三方技術提供查詢服務，您需採用並利用此服務。否則，儘管此方式的複雜性較高或解決方案效能較低，您應能模擬查詢服務。

尋找方法不得傳回執行階段例外狀況

執行外掛程式內部搜尋的 `IPluginFactory` 介面方法不得擲出受控或不受控的執行階段例外狀況。這可能是在執行工作流程時奇怪驗證錯誤失敗的原因。例如：若第一個節點的輸出為第二個節點的輸入，則在工作流程的兩個節點之間叫用 `find` 方法。在此時，若因任何執行階段例外狀況導致找不到物件，您只能得到 Orchestrator 用戶端驗證錯誤的資訊。之後，將視外掛程式記錄例外狀況的方式，在記錄檔案內收到更多或更少的資訊。

工作流程內部

您可執行工作流程監控您 Orchestrator 外掛程式執行的長時間作業。

您可執行工作流程監控長時間執行作業，如工作監控。此工作流程可根據 Orchestrator 觸發器和等待事件為基準。您必須考慮等待工作的封鎖工作流程可於 Orchestrator 伺服器啟動時立即恢復。外掛程式必能取得所有必要資訊，以正確恢復監控程序。

監控工作流程或可在內部使用的工作皆應提供機制，以指定輪詢率和可能的逾時。

偵錯工作流程內部的一段指令碼程序並不容易，特別是該代碼未叫用任何 Java 代碼。由於此緣故，有時唯一的選擇是使用預設 Orchestrator 指令碼物件的的記錄方法。

工作流程及動作

若要減輕工作流程開發及使用，您可使用特定的作法。

開始將工作流程當成建置組塊開發

建置組塊可以是需要少數輸入參數及傳回簡易輸出的簡易工作流程。若您擁有豐富的建置組塊組合，可輕鬆建立更高階的工作流程，並可在組成複雜工作流程中提供更出色的工具組合。

根據小型元件建立高階工作流程

若您必須開發含多個輸入與內部步驟的複雜工作流程，您可分割為更小及更簡單的建置組塊工作流程和動作。

請盡可能建立動作

您可建立動作以便在開發工作流程時達到額外彈性。

- 建立複雜物件或參數以輕鬆執行指令碼方法
- 避免始終重複通用代碼片段
- 執行 UI 驗證

工作流程應在可行時叫用動作

可隨工作流程配置內的節點直接叫用動作。可確保工作流程配置簡化，因為您不需要新增指令碼代碼區塊叫用單一動作。

填入所需的資訊

提供工作流程或動作的各元素資訊。

- 提供工作流程或動作的描述。
- 提供輸入參數的描述。
- 提供輸出的描述。
- 提供工作流程屬性的描述。

維持最新的版本資訊

當您執行外掛程式的版本設定時，請新增含外掛程式主要更新、重要執行詳細資訊等有意義的註解。

工作流程簡報

當您建立工作流程簡報後，應套用特定結構與規則。

使用下列內容在工作流程簡報中進行工作流程輸入。

表 6-46. 工作流程輸入內容

內容	使用方式
`Show in Inventory`	使用此內容有助於使用者執行詳細目錄視圖中的工作流程。
`Specify a root object to be shown in the chooser`	使用此內容有助於使用者選擇輸入。若可在簡報中重新整理的根物件為屬性，或透過物件方法擷取，您需建立或設定適當的動作才能重新整理簡報中的物件。
`Maximum string length`	使用此用於長字串的內容，如名稱、描述、檔案路徑等。
`Minimum string length`	使用此內容可避開測試工具中的空字串。
`Custom validation`	執行含動作的非範本驗證。

利用步驟及顯示群組組織輸入。此組織有助於使用者識別及區分工作流程中的所有輸入參數。

Orchestrator 外掛程式開發建議事項

在開發 Orchestrator 外掛程式的不同元件時，落實特定作法有助於您改善外掛程式的品質。

表 6-47. 外掛程式執行的實用作法

元件	項目	說明
一般	存取第三方 API	外掛程式應盡可能提供簡化的第三方 API 存取方式。
	介面	外掛程式應提供一致且標準的介面供使用者使用，即使 API 無法提供。
動作	指令碼物件	您應為各建立、修改、刪除和所有其他指令碼物件的可用方式建立動作。
	說明	動作描述應描述動作內容而非運作方式。
	執行指令碼	當您使用指令碼取得物件屬性或方式時，可檢查物件數值是否與 `null` 或 `undefined` 不同。
	取代	若動作遭取代，`comment` 或 `throw` 陳述式應指出取代動作，或該動作應呼叫新的取代動作，以便取代的動作版本解決方案不會無效。
工作流程	採協調技術的使用者介面作業	您應為協調技術的使用者介面中的所有作業建立工作流程。
	說明	工作流程描述應描述工作流程內容而非運作方式。
	簡報內容 `mandatory input`	您必須為所有強制工作流程輸入設定 `mandatory input` 內容。
	簡報內容 `default value`	若您開發的工作流程設定實體，則該工作流程簡報應載入此實體的預設組態數值。例如：若您開發名為主機組態的工作流程，此工作流程簡報必須載入主機組態的預設數值。
	簡報內容 `Show in inventory`	您必須設定 `Show in inventory` 內容才能擁有詳細目錄物件的內容工作流程。
	簡報內容 `specify a root parameter`	您應在無須瀏覽樹狀根目錄的詳細目錄時，使用工作流程中的此內容。
	工作流程驗證	您必須驗證工作流程及修正所有錯誤。
	物件建立	所有建立新物件的工作流程皆應將新物件當成輸出參數傳回。
	取代	若工作流程遭取代，`comment` 或 `throw` 陳述式應指出取代工作流程，或該工作流程應呼叫新的取代工作流程，以確保根據舊版工作流程建立的解決方案不會無效。
詳細目錄	主機中斷連線	若您的詳細目錄包含主機連線，則此主機將無法使用，而您應指出中斷連接的主機。您可利用 vCloud Director 外掛程式相同的方式，藉由附加 `– disconnected` 重新命名根物件或移除此物件下的物件樹狀結構，完成此動作。
	`Select value as list` 內容	詳細目錄物件必須可當成 `treeview` 或 `list` 選擇。
	主機管理員	若外掛程式執行目標系統的 `host` 物件，則父系 `hostmanager` 根物件應含有用於新增、移除或編輯主機的內容。
	取得或更新物件	若正使用協調技術執行查詢服務，您應用於取得多重物件。
	子系探索	若您需要另外擷取子系物件，擷取的程序必須為多執行緒且在單一錯誤上未封鎖。
	Orchestrator 物件變更	詳細目錄中所有可變更元素狀態的工作流程都必須更新詳細目錄，以避免讓物件脫離同步化。

表 6-47. 外掛程式執行的實用作法 (續)

元件	項目	說明
	外部物件變更	您可使用通知機制來告知因執行作業超出 Orchestrator 範圍而導致協調技術中發生變更。若此作業造成協調技術中的物件移除，您必須重新整理詳細目錄，以避免故障或資料遺失。例如：若刪除 vCenter Server 中的虛擬機器，vCenter Server 外掛程式會更新詳細目錄，以移除已移除虛擬機器的物件。
	Finder 物件	Finder 物件應含有可用於區分物件的內容。這些通常是存在於使用者介面中的內容。
指令碼物件	執行	必須執行 `equals` 方式才能確保在相同物件上進行的 `==` 作業與部分狀況下可能擁有兩個執行個體的物件相同。
	外掛程式物件內容	含父系物件的物件應執行 `parent` 內容。
	外掛程式物件內容	含子系物件的物件應執行 `GET` 方式以傳回大量子系物件。
	詳細目錄物件	詳細目錄物件應可透過 `Server.find` 搜尋。
		所有詳細目錄物件應可序列化，以便在工作流程中當成輸入或輸出屬性使用。
	建構函式與方法	在大多數情況下，可編寫指令碼物件應含有建構函式或由其他物件屬性或方式傳回。
	物件識別碼	從外部系統發出擁有識別碼的物件應使用內部識別碼，藉此確保您在協調多部伺服器時不會出現重複的識別碼。
	搜尋物件	`search` 或 `find` 方式應執行篩選，以便找出指定名稱或識別碼，而非所有物件。例如：Orchestrator 伺服器的 `Server.FindForId` 方式可透過外掛程式物件的識別碼找到外掛程式。若要搜尋，必須針對外掛程式中各個可找到的物件執行此方式。
	觸發	如有可能，應可觸發變更物件，讓 Orchestrator 能在各種事件時觸發原則。例如：要決定新增虛擬機器、開機、關機等時間，Orchestrator 可監控 `Datacenter` 物件上 vCenter 外掛程式中的觸發或事件。
	物件內容	其他外掛程式中的物件應有可輕易在外掛程式間轉換的內容。例如：虛擬機器物件需有 `moref`（受管理的物件參考識別碼）。
	工作階段管理員	若您連接至可用有不同工作階段的遠端伺服器，該外掛程式應執行共用工作階段與各使用者的工作階段。
觸發	觸發	所有長時間作業與封鎖方式皆應能在傳回工作時非同步開始，並於完成時產生觸發事件。
列舉	列舉	列舉指定型別應含有能於列舉中選擇不同數值的詳細目錄物件。
記錄	記錄	方法應執行不同的記錄層級。
版本設定	外掛程式版本	外掛程式版本應遵循標準並隨外掛程式更新進行更新。
API 說明文件	方法	在 API 說明文件中描述的方法不得在物件上擲回例外狀況 `no xyz method / property`。相反的，方法應在沒有可用內容時傳回 `null` 並詳細記錄。
	`vso.xml`	所有物件、方法及內容都必須記錄在 `vso.xml` 中。

記錄外掛程式使用者介面字串與 API

當您撰寫適用於 Orchestrator 外掛程式與相關 API 說明文件的使用者介面 (UI) 字串時，請依照樣式與格式的公認規則。

一般建議

- 在外掛程式中使用提及的 VMware 產品的正式名稱。例如：使用下列產品的正式名稱及 VMware 詞彙。

正確術語	請勿使用
vCenter Server	VC 或 vCenter
vCloud Director	vCloud

- 以句點結束所有工作流程的描述。例如：`Creates a new Organization.` 為工作流程描述。
- 使用含拼字檢查程式的文字編輯器寫入描述，然後移至外掛程式中。
- 確保外掛程式的名稱與相關聯並經審核的第三方產品名稱完全相符。

工作流程及動作

- 寫入資訊性描述。大多數的動作和工作流程只需一或兩句描述即已足夠。
- 更高階層的工作流程可能包括更廣泛的描述與註解。
- 請以動詞作為描述的開頭，例如 `Creates…`。請勿使用如 `This workflow creates` 等自我參照的語言。
- 在描述結尾加上句點，完成該句的描述。
- 描述工作流程或動作的作用，而非執行的方式。
- 工作流程與動作通常包括在資料夾與套件中。同樣針對這些資料夾與套件包含簡短描述。例如：工作流程資料夾可擁有類似 `Set of workflows related to vApp Template management` 的描述。

工作流程與動作的參數

- 例如：用描述性的名詞片語 `Name of` 作為工作流程與動作描述的開頭。請勿使用如 `It's the name of` 的片語。
- 請勿在參數結尾及動作描述後加上句點。這些皆非完整句子。
- 工作流程的輸入參數必須在呈現視圖中指定含有適當名稱的標籤。在多數情況下，您可在顯示群組中組合相關的輸入。例如：在未輸入兩個含有組織名稱與組織全名標籤的情形下，您可建立含有標籤組織的顯示群組並在組織群組中放上輸入名稱及全名。
- 在步驟與顯示群組方面，新增描述或註解也會出現在工作流程簡報中。

外掛程式 API

- API 的說明文件意指 `vso.xml` 檔案及 Java 來源檔案中的所有說明文件。
- 如為 `vso.xml` 檔案，請使用相同規則來描述 finder 物件和指令碼物件，搭配您用於工作流程和動作的方法。物件屬性及方式參數的描述皆使用與工作流程及動作參數相同的規則。

- 請避免在 `vso.xml` 檔案中使用特殊字元以及在 `<![CDATA[insert your description here!]]>` 標籤中加入描述。
- Java 來源檔案請使用標準 Javadoc 樣式。

使用 Maven 建立外掛程式

7

Orchestrator 應用裝置提供含 Maven 成品的存放庫，讓您可用於從原型建立外掛程式專案。

如您的 Maven 版本不支援 HTTPS 通訊協定，則存放庫位在 `https://orchestrator_server:8281/vco-repo/` 或 `http://orchestrator_server:8280/vco-repo/`。此位置嵌入至標準 Orchestrator Maven 外掛程式專案的 `pom.xml` 檔案中。如果已部署 Orchestrator Appliance，則只能存取存放庫。

本章節討論下列主題：

- 使用原型中的 Maven 建立 Orchestrator 外掛程式
- Maven 原型
- Maven 架構外掛程式開發最佳作法

使用原型中的 Maven 建立 Orchestrator 外掛程式

您可透過在命令行介面中執行命令，從原型中建立標準 Orchestrator Maven 外掛程式。

必要條件

- 確認您已安裝 Orchestrator Appliance 5.5.1 以上版本。
- 確認您已安裝 Apache Maven 3.0.4 或 3.0.5。

程序

1 選擇原型，在互動式模式中建立專案。

```
mvn archetype:generate -DarchetypeCatalog=https://orchestrator_server:8281/vco-repo/archetype-catalog.xml -DrepoUrl=https://orchestrator_server:8281/vco-repo -Dmaven.repo.remote=https://orchestrator_server:8281/vco-repo -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true
```

備註 如果已部署 Orchestrator Appliance，則只能存取 Maven 存放庫。

2 (選擇性) 若您無法存取 HTTPS 的存放庫，可存取 HTTP 的存放庫。若您存取 HTTP 的存放庫或擁有有效的 SSL 憑證，則可建立不使用 `-Dmaven.wagon.http.ssl.allowall=true` 旗標的專案。

```
mvn archetype:generate -DarchetypeCatalog=http://orchestrator_server:8280/vco-repo/archetype-catalog.xml -DrepoUrl=http://orchestrator_server:8280/vco-repo -Dmaven.repo.remote=http://orchestrator_server:8280/vco-repo -Dmaven.wagon.http.ssl.insecure=true
```

3 瀏覽至專案目錄並建置外掛程式。

```
cd project_dir && mvn clean install -Dmaven.wagon.http.ssl.insecure=true -
Dmaven.wagon.http.ssl.allowall=true
```

若建置程序順利，將在 DAR 模組的 `target/` 目錄產生外掛程式 `.dar` 檔案。

Maven 原型

您可使用預先定義的 Maven 原型組合作為範本，開發 Orchestrator 外掛程式。

下表說明 Orchestrator 中可用的預設 Maven 原型。

表 7-1. 預設 Maven 原型

原型	說明
`com.vmware.o11n:o11n-plugin-archetype-simple`	`com.vmware.o11n:o11n-plugin-archetype-simple`
`com.vmware.o11n:o11n-package-archetype`	僅有內容的 Maven 專案可透過來源形式將套件保存在套件中，以提升與 RCS、diff、後續處理等項目的互動。
`com.vmware.o11n:o11n-client-archetype-rest`	簡易命令行工具可與 Orchestrator REST API 通訊及呼叫工作流程。
`com.vmware.o11n:o11n-plugin-archetype-inventory`	外掛程式可展示詳細目錄用途。外掛程式執行用於單一類型的存放庫、介面卡及處理站。詳細目錄儲存在磁碟上的檔案中。
`com.vmware.o11n:o11n-archetype-inventory-annotation`	在註解最上方產生外掛程式的 `vso.xml` 描述元。
`com.vmware.o11n:o11n-archetype-spring`	外掛程式使用 Spring-based 的 SDK 以提供具有 DI 功能的環境，且相較於標準外掛程式 API，外掛程式可新增更高層級的服務。
`com.vmware.o11n:o11n-plugin-archetype-modeldriven`	原型會產生外掛程式骨架，以建置含 ModelDriven 的外掛程式。

Maven 架構外掛程式開發最佳作法

您可執行一系列工作，改善透過 Maven 建立 Orchestrator 外掛程式的交付處理程序。

使用存放庫管理員

若您在大型組織中建立外掛程式，請使用企業存放庫管理員設定要當成 Proxy 存放庫新增的預設 Orchestrator Appliance 存放庫。使用中央存放庫可改善管理及外掛程式專案協作。在新的存放庫中完成第一個建置後，存放庫管理員會快取 Orchestrator 應用裝置存放庫中的成品，且您可關閉預設存放庫。

鎖定工作流程

確認外掛程式中所有工作流程皆如預期執行後，請鎖定以防止未授權修改。透過鎖定工作流程，您可確保外掛程式的基本功能不會減損。若使用者因指定用途而必須修改預設工作流程，可建立原始工作流程的複本並編輯該複本。

有兩種方式可透過鎖定的工作流程生成發行版本。

- 將 `-DallowedMask=vf` 參數傳遞至 Maven。

- 編輯 pom.xml 並將 allowedMask 參數的值變更為 vf。

```
<allowedMask>vf</allowedMask>
```

使用套件簽署憑證

使用自我簽署憑證或由憑證授權機構簽署的憑證，確保外掛程式的完整性及真確性。利用 JDK 中的金鑰工具匯入憑證，將其儲存在 _dunesrsa_alias_ 別名下的金鑰儲存區。

指定金鑰儲存區檔案及金鑰儲存區密碼路徑的方式有兩種。

- 定義 *MAVEN_OPTS* 變數的 –DkeystoreLocation 和 –DkeystorePassword 命令行參數。
- 編輯 pom.xml 檔案，手動插入數值。例如，

```
<keystore>路徑為金鑰儲存取檔案</keystore>
<storepass>金鑰儲存區密碼</storepass>
```

若未匯入金鑰儲存區，將利用 archetype.keystore 檔案簽署 .package 檔案。