Citrix XenServer ® 6.2.0 软件开发工具包

2013 年 09 月 12 日（星期四）发行
1.0 版

Citrix XenServer ® 6.2.0 软件开发工具包

版本：6.2.0

Citrix, Inc.
851 West Cypress Creek Road
Fort Lauderdale, FL 33309
United States of America

CITRIX

目录

1. 简介 1

2. 入门 2

2.1. 系统要求和准备工作 2

2.2. 下载 2

2.3. 新增功能 2

2.4. 内容图 2

3. XenServer API 概述 4

3.1. XenServer API 入门 4

3.1.1. 身份验证：获得会话引用 4

3.1.2. 获取作为新 VM 安装基础的模板列表 4

3.1.3. 基于模板安装 VM 5

3.1.4. 对 VM 执行启动/挂起/恢复/停止周期操作 5

3.1.5. 注销 5

3.1.6. 安装和启动示例：摘要 5

3.2. 对象模型概述 6

3.3. 使用 VIF 和 VBD 7

3.3.1. 创建磁盘并将它们连接到 VM 7

3.3.1.1. 创建新的空白磁盘映像 8

3.3.1.2. 将磁盘映像连接到 VM 8

3.3.1.3. 热插拔 VBD 9

3.3.2. 创建网络设备并将其连接到 VM 9

3.3.3. 网络和存储的主机配置 9

3.3.3.1. 主机存储配置：PBD 9

3.3.3.2. 主机网络配置：PIF 10

3.4. 导出和导入 VM 10

3.4.1. Xen Virtual Appliance (XVA) VM 导入格式 11

3.5. XML-RPC 说明 13

3.5.1. 日期时间 13

3.6. 其他参考来源 14

4. 使用 API 15
4.1. 剖析典型应用程序 15
4.1.1. 选择一种低级别传输 15
4.1.2. 身份验证和会话处理 15
4.1.3. 查找对有用对象的引用 16
4.1.4. 在对象上调用同步操作 16
4.1.5. 使用任务管理异步操作 16
4.1.6. 预订和侦听事件 17
4.2. 语言绑定 17
4.2.1. C 17
4.2.2. C# 18
4.2.3. Java 18
4.2.4. PowerShell 19
4.2.5. Python 20
4.2.6. 命令行接口 (CLI) 20
4.3. 完整的应用程序示例 21
4.3.1. 使用 XenMotion 同时迁移 VM 21
4.3.2. 使用 XE CLI 克隆 VM 22
5. 使用 HTTP 与 XenServer 交互 24
5.1. VM 导入和导出 24
5.2. 获取 XenServer 性能统计信息 24
6. XenServer API 扩展 26
6.1. VM 控制台转发 26
6.1.1. 使用 API 检索 VNC 控制台 26
6.1.2. 禁用 Linux VM 的 VNC 转发 27
6.2. 半虚拟化 Linux 安装 27
6.2.1. Red Hat Enterprise Linux 4.1/4.4 28
6.2.2. Red Hat Enterprise Linux 4.5/5.0 28
6.2.3. SUSE Enterprise Linux 10 SP1 28
6.2.4. CentOS 4.5 / 5.0 28

6.3. 向 VM 添加 Xenstore 条目 28
6.4. 安全功能增强 29
6.5. 网络接口的高级设置 29
6.5.1. ethtool 设置 29
6.5.2. 其他设置 30
6.6. SR 名称的国际化 30
6.7. 从 XenCenter 隐藏对象 31
7. XenCenter API 扩展 32
7.1. 池 32
7.2. 主机 32
7.3. VM 33
7.4. SR 34
7.5. VDI 34
7.6. VBD 34
7.7. 网络 35
7.8. VM_guest_metrics 35
7.9. 任务 35

第 1 章 简介

欢迎使用 XenServer 开发人员指南。本指南为您提供了解和使用 XenServer 提供的软件开发工具包 (SDK) 所需的信息。这些信息提供了一些架构性背景和思路，有助于您了解 API、已提供的工具以及如何快速入门。

CITRIX

第 2 章 入门

XenServer 包括基于 XML-RPC 的 API，可提供对丰富的 XenServer 管理功能和工具集的编程访问。可以从远程系统，也可以从XenServer 主机本地调用 XenServer API。虽然可以直接通过原始 XML-RPC 调用编写使用 XenServer 管理 API 的应用程序，但通过使用语言绑定将各 API 调用显示为目标语言中的第一类函数，可以大大简化第三方应用程序开发工作。XenServer SDK 为 C、C#、Java、Python 和 PowerShell 编程语言提供语言绑定和示例代码。

2.1. 系统要求和准备工作

要使用 SDK，首先需要安装 XenServer。可以从 http://www.citrix.com/downloads/xenserver/ 下载 Citrix XenServer 的免费版。有关如何设置开发主机的详细说明，请参阅《XenServer 安装指南》。安装完成后，请记下主机 IP 地址和主机密码。

2.2. 下载

SDK 打包为 ZIP 文件，并且从 http://www.citrix.com/downloads/xenserver/ 免费下载。

2.3. 新增功能

XenServer 6.2.0 SDK 为 C、C#、Java、PowerShell 和 Python 语言绑定提供 API。此版本包含以下多个变更和增强功能：

- C、C#、Java 和 PowerShell 绑定
  已删除对 XenServer 4.0 的支持。
- Powershell 绑定
  此版本包含两个版本的 Powershell 语言绑定：之前发布的 XenServer Powershell v1.0 Snapin 和经过重新设计的全新 XenServer Powershell v2.0 Snapin，前者包含多个缺陷修复。有关详细信息，请参阅《XenServer 发行说明》。

2.4. 内容图

SDK ZIP 文件的解压缩内容位于 XenServer-SDK 目录中。下面是其结构的概览。如有必要，各子目录可具有各自的自述文件。请注意，提供的示例在所有语言绑定中并不是相同的，因此，如果您想要使用一个绑定，建议您还要浏览其他语言绑定中的可用示例代码。

- XenServer-SDK
  - libxenserver
    适用于 C 的 XenServer SDK。
    - bin
      libxenserver 二进制文件。
    - src
      libxenserver 源代码和示例，以及构建这些源代码和示例的 Makefile。
  - XenServer.NET
    适用于 C#.NET 的 XenServer SDK。
    - bin
      XenServer.NET 二进制文件。
    - 样例
      XenServer.NET 示例。
    - src
      XenServer.NET 源代码。

- XenServerJava
  适用于 Java 的 XenServer SDK
  - bin
    Java 二进制文件。
  - javadoc
    Java 文档。
  - 样例
    Java 示例。
  - src
    Java 源代码以及构建代码和示例的 Makefile。
- XenServerPSSnapIn
  经过重新设计的适用于 PowerShell 的全新 XenServer SDK。此目录包含 Xenserver PowerShell Windows 安装程序 XenServerPSSnapIn.msi。
  - 样例
    PowerShell 示例脚本。
  - src
    Xenserver PowerShell 源代码。
- XenServerPSSnapIn_old
  之前发布的适用于 PowerShell 的 XenServer SDK。此目录包含 Xenserver PowerShell Windows 安装程序 XenServerPSSnapIn.msi。
  - 样例
    PowerShell 示例脚本。
  - src
    Xenserver PowerShell 源代码。
- XenServerPython
  此目录包含 XenServer Python 模块 XenAPI.py。库 provision.py 由示例使用。
  - 样例
    使用 Python 的 XenServer API 示例。

CITRIX

第 3 章 XenServer API 概述

在本章中，我们将介绍 XenServer API（以下简称“API”）及其相关的对象模型。该 API 具有以下重要特征：

- 全面管理 XenServer 主机的各个方面
  通过该 API，可以管理 VM、存储、网络、主机配置和池。还可以从该 API 查询性能和状态指标。

- 静态对象模型
  所有会产生其他影响的操作（例如创建、删除对象和修改参数）的结果都将永久保留在由 XenServer 安装管理的服务器端数据库中。

- 事件机制
  通过该 API，客户端可以进行注册，以在静态（服务器端）对象被修改时得到通知。这可使应用程序跟踪由当前执行客户端对数据模型进行的修改。

- 同步和异步调用
  可以同步调用所有 API 调用（直到完成才中断）；也可以异步调用可能长时间运行的任何 API 调用。异步调用可立即返回对任务对象的引用。通过该 API，可以查询此任务对象的进程和状态信息。异步调用操作完成后，可从任务对象获得结果（或错误代码）。

- 可远程和跨平台访问
  发出 API 调用的客户端不必位于要管理的主机上；也不必通过 ssh 连接到主机来执行该 API。API 调用使用 XML-RPC 协议通过网络传输请求和响应。

- 安全的、经过身份验证的访问
  在主机上执行的 XML-RPC API 服务器接受安全套接字连接。这样，客户端将可以通过 http 协议执行 API。此外，所有 API 调用都在登录会话上下文中执行，登录会话是通过在服务器端进行用户名和密码验证生成的。这样，将可以对 XenServer 安装进行安全的、经过身份验证的访问。

3.1. XenServer API 入门

在该 API 教程中，我们首先介绍在 XenServer 安装中创建新 VM 所需的调用，然后完成启动/挂起/恢复/停止周期的整个过程。无需引用任何特定语言的代码即可完成此任务；在此阶段，我们仅介绍完成“安装并启动”任务的 RPC 调用的非正式序列。

3.1.1. 身份验证：获得会话引用

第一步是调用 Session.login_with_password(***<username>***, ***<password>***, ***<client_API_version>***)。该 API 基于会话，因此，在执行其他调用之前，需要使用服务器进行身份验证。如果用户名和密码正确通过了身份验证，则此调用的结果为一个会话引用。后续 API 调用使用该会话引用作为参数。通过此方式，我们可确保只有正确通过授权的 API 用户可以在 XenServer 安装中执行操作。

3.1.2. 获取作为新 VM 安装基础的模板列表

第二步是查询主机上的“模板”列表。模板是具有特殊标记的 VM 对象，用于为各种受支持的来宾系统类型指定适当的默认参数。（如果您希望系统在 XenServer 安装期间快速枚举模板，可以执行 xe template-list CLI 命令。）要通过该 API 获取模板列表，需要在服务器上查找 is_a_template 字段设置为 true 的 VM 对象。执行此操作的一种方法是调用 VM.get_all_records(session)，其中 session 参数是先前从 Session.login_with_password 调用中获得的引用。此调用将查询服务器，返回包含所有 VM 对象引用及其字段值的快照，该快照是在调用时获取的。

（请记住，在这一阶段，我们不关注可通过任何特定客户端语言处理返回的对象引用和字段值的特殊机制：这一细节通过特定于语言的 API 绑定进行处理，将在后面的章节中具体描述。现在只需假定存在一个抽象机制，用于读取和处理由 API 调用返回的对象和字段值。）

现在客户端应用程序内存中已经拥有了所有 VM 对象字段值的快照，只需迭代浏览这些对象，并查找“is_a_template”设置为 true 的对象。在此阶段，我们假定示例应用程序将进一步迭代浏览模板对象，并记住“name_label”设置为“Debian Etch 4.0”（XenServer 提供的一个默认 Linux 模板）的对象所对应的引用。

3.1.3. 基于模板安装 VM

继续我们的示例。现在，必须基于所选模板安装新的 VM。安装过程需要 2 个 API 调用：

- 首先，必须立即调用 API 调用 VM.clone(session, t_ref, "my first VM")。该调用将通知服务器克隆 t_ref 所引用的 VM 对象，以生成新的 VM 对象。此调用将返回与新创建的 VM 对应的 VM 引用。我们称其为 new_vm_ref。
- 在此阶段，new_vm_ref 引用的对象仍然是模板（类似于 t_ref 所引用的用于克隆的 VM 对象）。要使 new_vm_ref 成为 VM 对象，需要调用 VM.provision(session, new_vm_ref)。当此调用返回结果后，new_vm_ref 对象的 is_a_template 字段会设置为 false，说明 new_vm_ref 现在引用一个可以启动的常规 VM。

请注意，置备操作可能需要几分钟，这是因为在此调用期间，还需要创建模板的磁盘映像。如果是 Debian 模板，则在此阶段，将使用 Debian 根文件系统实际填充新创建的磁盘。

3.1.4. 对 VM 执行启动/挂起/恢复/停止周期操作

现在我们有了代表新创建的 VM 的对象引用。接下来对它执行生命周期操作就很简单了：

- 要启用 VM，我们只需调用 VM.start(session, new_vm_ref)。
- VM 运行后，可以通过调用 VM.suspend(session, new_vm_ref) 将该 VM 挂起，
- 挂起 VM 后，可以通过 VM.resume(session, new_vm_ref) 进行恢复。
- 调用 VM.shutdown(session, new_vm_ref) 可彻底关闭该 VM。

3.1.5. 注销

当应用程序完成与 XenServer 主机的交互后，最好调用 Session.logout(session)。这将使会话引用无效（因此，该会话引用无法用于后续 API 调用），同时会释放用于存储会话对象的服务器端内存。

虽然非活动状态的会话最终将超时，但服务器仍有最多允许 200 个并发会话的硬编码限制。达到此限制后，新登录将收回最早的会话对象，从而导致这些对象的关联会话引用失效。因此，如果您希望应用程序能够与并发访问服务器的其他应用程序正常交互，最佳策略是在当日开始时创建单一会话，在所有这些应用程序中使用该会话（请注意，可以跨多个独立的客户端服务器网络连接使用该会话），然后在必要时显式注销。

3.1.6. 安装和启动示例：摘要

我们已经了解如何使用该 API 安装基于 XenServer 模板的 VM 并对它执行一系列的生命周期操作。您会注意到，我们为影响这些操作而执行的调用的数量很少：

- 使用一个调用获取会话：Session.login_with_password()
- 使用一个调用查询在 XenServer 安装期间显示的 VM（和模板）对象：VM.get_all_records()。请回忆一下，我们曾使用该调用所返回的信息选择了一个合适的模板来完成安装。
- 使用两个调用基于所选模板安装 VM：先调用 VM.clone()，再调用 VM.provision()。
- 使用一个调用启动生成的 VM：VM.start()（以及与之类似的分别用来挂起、恢复和关闭生成的 VM 的其他单个调用）
- 此外，还有一个调用执行注销：Session.logout()

此处的要点是，虽然整个 API 功能齐全且复杂，但常见任务（例如，VM 上的创建和执行生命周期操作）非常易于执行，只需少量的简单 API 调用。在学习下一部分时，请记得您在初读时可能会感到有些困难。

3.2. 对象模型概述

本部分从较高层面概述了 API 的对象模型。《XenServer API 参考》文档中更详细地介绍了此处所述每个类的参数和方法。

我们首先简要概括一下组成该 API 的一些核心类。（如果这些定义初看起来有些抽象，请不要担心；后面有更详尽的文字说明，下一章中详细演示的代码示例也将有助于使这些概念具体化。）

VM	VM 对象表示 XenServer 主机或资源池上的特定虚拟机实例。示例方法包括 start、suspend 和 pool_migrate；示例参数包括 *power_state*、*memory_static_max* 和 *name_label*。（在上一部分中，我们已了解了如何使用 VM 类来表示模板和常规 VM。）
主机	主机对象表示 XenServer 池中的物理主机。示例方法包括 reboot 和 shutdown。示例参数包括 *software_version*、*hostname* 和 [IP] *address*。
VDI	VDI 对象表示虚拟磁盘映像。可以将虚拟磁盘映像连接到 VM，这样，将在 VM 内部显示块设备，通过该设备可以读取和写入虚拟磁盘映像封装的位。VDI 类的示例方法包括“resize”和“clone”。示例字段包括“virtual_size”和“sharable”。（在前面的示例中，当在 VM 模板上调用 VM.provision 后，一些表示新创建的磁盘的 VDI 对象便自动创建并连接到 VM 对象。）
SR	SR（存储库）聚合 VDI 的集合并封装 VDI 位所在的物理存储的属性。示例参数包括 *type*（确定 XenServer 安装用来读/写 SR VDI 的存储专用驱动程序）和 *physical_utilisation*；示例方法包括 scan（调用存储专用驱动程序以获得 SR 中的 VDI 列表以及这些 VDI 的属性）和 create（初始化物理存储块，使其为存储 VDI 做好准备）。
网络	网络对象表示存在于 XenServer 主机实例所在的环境中的第二层网络。XenServer 不直接管理网络，因此，这是一个次要类，仅用于对物理和虚拟网络拓扑进行建模。连接到特定网络对象（通过 VIF 和 PIF 实例，如下所述）的 VM 和主机对象可以彼此发送网络数据包。

此时，读者如果认为这样列举类过于简单，可以跳到下一章提供的代码部分：该部分提供了大量有用的应用程序，仅使用已经介绍的一些类就可以编写这些程序！读者如果希望继续概括地了解类，请继续阅读。

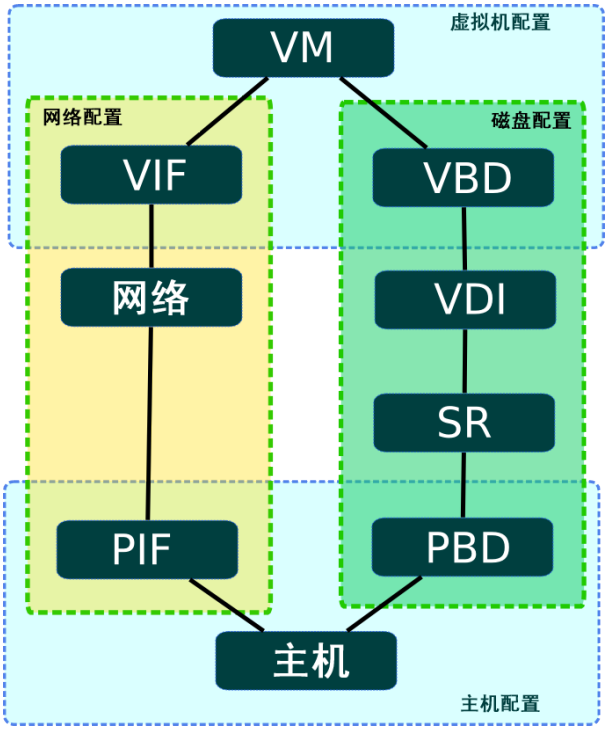
在上面列出的类的顶部，有另外 4 个用作连接器的类，用于指定 VM 和主机、存储和网络之间的关系。我们将介绍其中的前 2 个类（VBD 和 VIF）确定如何将 VM 分别连接到虚拟磁盘和网络对象：

VBD	VBD（虚拟块设备）对象表示 VM 和 VDI 之间的连接。引导 VM 时，将查询它的 VBD 对象以确定应该连接哪些磁盘映像 (VDI)。VBD 类的示例方法包括“plug”（将磁盘设备热插入到运行中的 VM，从而使指定的 VDI 可访问）和“unplug”（从运行中的来宾系统中热拔出磁盘设备）；示例字段包括“device”（确定来宾系统内部的设备名称，通过该设备下可访问指定的 VDI）。
VIF	VIF（虚拟网络接口）对象表示 VM 和网络对象之间的连接。引导 VM 时，将查询它的 VIF 对象以确定应创建哪些网络设备。VIF 类的示例方法包括“plug”（将网络设备热插入到运行中的 VM）和“unplug”（从运行中的来宾系统中热弹出网络设备）。

我们要介绍的第二组连接器类确定如何将主机连接到网络和存储。

PIF	PIF（物理接口）对象表示主机和网络对象之间的连接。如果将主机连接到网络（通过 PIF），则指定的主机提供的数据包可以由对应的主机传输/接收。PIF 类的示例字段包括“device”（指定 PIF 对应的设备名称，例如 eth0）和“MAC”（指定 PIF 表示的基础 NIC 的 MAC 地址）。请注意，PIF 将物理接口和 VLAN（后者由“VLAN”字段中的正整数标识）抽象化。
PBD	PBD（物理块设备）对象表示主机和 SR（存储库）对象之间的连接。字段包括“currently-attached”和“device_config”，前者指定由指定的 SR 对象表示的存储区块当前是否对主机可用，后者指定可用来确定如何在指定的主机上配置低级存储设备的特定于存储驱动程序的参数，例如，对于在 NFS 文件管理器上呈现的 SR，device_config 可以指定该文件管理器的主机名和 SR 文件所在的文件管理器的路径。

图 3.1. 常见 API 类



管理 VM、主机、存储和网络的 API 类的图表概述

图 3.1 “常见 API 类”显示了用于管理 VM、主机、存储和网络的 API 类的图表概述。从此图表中，可以清晰地看出存储和网络配置之间以及虚拟机和主机配置之间的对称性。

3.3. 使用 VIF 和 VBD

本部分将详细演示几个复杂方案，并将非正式性地介绍如何使用 API 完成与虚拟存储和网络设备相关的多种任务。

3.3.1. 创建磁盘并将它们连接到 VM

首先，考虑如何生成新的空白磁盘映像并将其连接到正在运行的 VM。假设我们已经通过一些前期工作获得了一个运行中的 VM，并且知道其相应的 API 对象引用（例如，可能已使用上一部分介绍的过程创建了此 VM，并且服务器已将其引用返回。）同时假设已针对 XenServer 安装进行身份验证，并且具有相应的会话引用。为简要起见，在本章的其余部分中将不再提及会话。

3.3.1.1. 创建新的空白磁盘映像

第一步是实例化物理存储中的磁盘映像。通过调用 VDI.create() 来执行此操作。VDI.create 调用将使用多个参数，包括：

- name_label 和 name_description：磁盘的可读名称/说明（例如，为了便于在 UI 中显示等）。如果需要，这些字段可以保留空白。
- SR：存储库的对象引用，代表将放置 VDI 的位的物理存储。
- read_only：将此字段设置为 true 表示 VDI 只能以只读方式连接到 VM。（尝试以读/写方式连接 read_only 字段设置为 true 的 VDI 会引发错误。）

如果调用 VDI.create 调用，XenServer 安装过程将在物理存储中创建空白磁盘映像、创建相关联的 VDI 对象（表示物理存储中的磁盘映像的数据模型实例）并返回对此新建 VDI 对象的引用。

物理存储中表示磁盘映像的方式取决于创建的 VDI 所在 SR 的类型。例如，如果 SR 为“lvm”类型，则新的磁盘映像将呈现为 LVM 卷；如果 SR 为“nfs”类型，则新的磁盘映像将呈现为在 NFS 文件管理器中创建的稀疏 VHD 文件。（通过 API 使用 SR.get_type() 调用可以查询 SR 类型。）

> 注意：
>
> 某些 SR 类型可能会对 *virtual-size* 值进行舍入处理，以使此值可被已配置块的大小整除。

3.3.1.2. 将磁盘映像连接到 VM

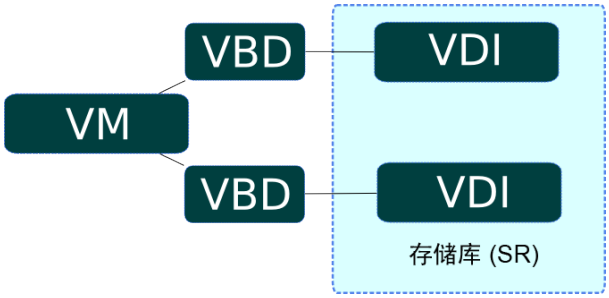
至此，我们有了运行中的 VM（在示例开头已假设存在）和刚创建的新 VDI。现在，XenServer 主机中已有两个独立对象，但没有什么东西将二者链接起来。因此，下一步要创建一个这样的链接，将 VDI 与 VM 关联起来。

通过创建称为 VBD（虚拟块设备）的“连接器”对象建立连接。要创建 VBD，需要调用 VBD.create() 调用。VBD.create() 调用使用多个参数，包括：

- VM - 连接 VDI 的 VM 的对象引用。
- VDI - 要连接的 VDI 的对象引用。
- mode - 指定 VDI 是以只读方式还是读写方式连接。
- userdevice - 指定来宾系统中的块设备，通过该设备正在 VM 中运行的应用程序将能读/写 VDI 的位。
- type - 指定 VDI 是应当作为常规磁盘还是作为 CD 显示在 VM 中。（请注意，此特定字段在 Windows VM 中的含义比在 Linux VM 中更丰富，但在本章中，我们不研究此级别的详细内容。）

调用 VBD.create 将在 XenServer 安装中创建 VBD 对象并返回其对象引用。但是，此调用本身对运行中的 VM 没有任何其他影响（即，如果查看运行中的 VM，您会发现尚未创建块设备）。VBD 对象的 currently_attached 字段设置为 false 可反映存在 VBD 对象但来宾系统中的块设备处于非活动状态。

图 3.2. 具有两个相关联 VDI 的 VM 对象



具有两个相关联 VDI 的 VM 对象

为了进行说明，图 3.2 “具有两个相关联 VDI 的 VM 对象”提供了一个图形示例以显示 VM、VBD、VDI 和 SR 之间关系。在该实例中，一个 VM 对象连接两个 VDI：两个 VBD 对象构成 VM 对象与其 VDI 之间的连接，这些 VDI 位于同一个 SR 内。

3.3.1.3. 热插拔 VBD

如果我们在该阶段重新启动 VM，则重新启动后，将显示对应于 VBD 的块设备：引导时，XenServer 将查询 VM 的所有 VBD，并主动连接每个对应的 VDI。

重新启动 VM 的确很好，但请记住我们的目的是将新创建的空白磁盘连接到运行中的 VM。这可以通过对新创建的 VBD 对象调用 plug 方法来实现。plug 调用成功返回后，与 VBD 相关的块设备将出现在运行中的 VM 内，也就是说，从运行中的 VM 的角度来看，来宾操作系统将认为刚刚热插入了新磁盘设备。在 API 的托管环境中，这表现为 VBD 的 currently_attached 字段设置为 true。

VBD plug 方法相应地具有称为“unplug”的方法与其对应。对 VBD 对象调用 unplug 方法会导致从运行中的 VM 热拔出相关的块设备，相应地导致 VBD 对象的 currently_attached 字段设置为 false。

3.3.2. 创建网络设备并将其连接到 VM

在 VM 中配置虚拟网络接口所涉及的 API 调用与配置虚拟磁盘设备所涉及的调用在许多方面都很相似。因此，我们不通过完整的示例来说明如何使用 API 对象模型创建网络接口；而只在本部分中简要说明虚拟网络设备与虚拟存储设备配置之间的对称性。

VBD 类的网络对等对象为 VIF 类。如同 VBD 是块设备在 VM 中的 API 表示，VIF（虚拟网络接口）是网络设备在 VM 中的 API 表示。VBD 将 VM 对象与 VDI 对象相关联，而 VIF 将 VM 对象与网络对象相关联。如同 VBD 一样，VIF 也具有 currently_attached 字段，用于确定与 VIF 相关联的网络设备（来宾操作系统内部）当前是否处于活动状态。同样与 VBD 一样，在引导 VM 时，系统将查询 VM 的 VIF 并在引导的 VM 内部创建每个 VIF 的相应网络设备。同样，VIF 也具有 plug 和 unplug 方法，用于在运行中的 VM 上热插拔网络设备。

3.3.3. 网络和存储的主机配置

我们已经知道 VBD 和 VIF 类分别用于在 VM 中管理块设备和网络设备的配置。为了管理主机存储配置和主机网络配置，系统提供了两个相似的类：PBD（物理块设备）和 PIF（物理 [网络] 接口）。

3.3.3.1. 主机存储配置：PBD

从 PBD 类开始讨论。PBD_create() 调用使用多个参数，包括：

参数	说明
host	可以在上面使用 PBD 的物理机
SR	PBD 所连接的存储库
device_config	提供给主机的 SR 后端驱动程序的字符串到字符串映射，包含配置要在其上实现 SR 的物理存储设备所需的低级参数。*`device_config`* 字段的特定内容取决于 PBD 连接到的 SR 的类型。（执行 xe sm-list 将显示可能的 SR 类型的列表；此枚举中的 configuration 字段将指定每个 SR 类型所需的 *`device_config`* 参数。）

例如，假设我们具有一个类型为 nfs（表示 NFS 文件管理器上用于将 VDI 作为 VHD 文件存储的目录）的 SR 对象 s，并且我们希望主机 h 能够访问 s。在这种情况下，我们应调用 PBD.create() 来指定主机 h、SR s 和 device_config 参数的值，即如下映射：

```
("server", "my_nfs_server.example.com"), ("serverpath", "/scratch/mysrs/
sr1")
```

这会通知 XenServer 主机在主机 h 上可访问 SR s，如需进一步访问 SR s，主机需要在名为 `my_nfs_server.example.com` 的 NFS 服务器中装载 `/scratch/mysrs/sr1` 目录。

如同 VBD 对象，PBD 对象也具有名为 currently_attached 的字段。通过分别调用 PBD.plug 和 PBD.unplug 方法，可以从给定主机连接和分离存储库。

3.3.3.2. 主机网络配置：PIF

主机网络配置通过 PIF 对象指定。如果 PIF 对象将网络对象 n 连接到主机对象 h，则对应于 n 的网络将桥接到由 PIF 对象的字段指定的物理接口（或一个加了 VLAN 标记的物理接口）。

例如，假设存在将主机 h 连接到网络 n 的 PIF 对象，并且 PIF 对象的 device 字段设置为 `eth0`。这意味着，网络 n 上的所有数据包都将通过桥接传输到主机中与主机网络设备 `eth0` 对应的 NIC。

3.4. 导出和导入 VM

VM 可以导出到某个文件，然后再导入到任何 XenServer 主机。导出协议是简单的 HTTP(S) GET，如果 VM 位于池成员上，应在主节点上执行此协议。授权是标准 HTTP 基本身份验证，或者如果已获得了会话，则可以使用会话。要导出的 VM 由 UUID 或引用指定。要跟踪导出，可以创建一个任务并使用其引用传入。如果仅可以在池成员上访问 VM 的磁盘，则请求可能会导致重定向。

通过命令行传递下列参数：

参数	说明
session_id	用于进行身份验证的会话的引用；仅在不使用 HTTP 基本身份验证时需要
task_id	用于跟踪操作的任务对象的引用；是可选参数，只有在已创建任务对象来跟踪导出时，才需要此参数
ref	VM 的引用；仅在不使用 UUID 时需要
uuid	VM 的 UUID；仅在不使用引用时需要

例如，使用 Linux 命令行工具 cURL：

```
curl http://root:foo@myxenserver1/export?uuid=<vm_uuid> -o <exportfile>
```

会将指定 VM 导出到文件 `exportfile`。

若要仅导出元数据，请使用 URI `http://server/export_metadata`。

导入协议与导出协议类似，也使用 HTTP(S) PUT。`session_id` 和 `task_id` 参数与导出一样。不使用 `ref` 和 `uuid`；将为 VM 生成新引用和 UUID。还有一些其他参数：

参数	说明
restore	如果为 true，则对导入的处理方式是替换原始 VM - 这意味着在当前的情况下 VIF 上的 MAC 地址与导出完全一样，如果原始 VM 仍在运行，就会导致冲突。
force	如果为 true，将忽略所有校验和失败（默认为在检测到校验和错误时销毁 VM）。
sr_id	应将 VM 导入其中的 SR 的引用。默认行为是导入到 Pool.default_SR。

例如，再次使用 cURL：

```
curl -T <exportfile> http://root:foo@myxenserver2/import
```

会将 VM 导入到服务器上的默认 SR。

注意：

请注意，如果尚未设置默认 SR 且未指定 sr_uuid，则返回错误消息“DEFAULT_SR_NOT_FOUND”。

另一个示例：

```
curl -T <exportfile> http://root:foo@myxenserver2/import?sr_id=<opaque_ref_of_sr>
```

会将 VM 导入到服务器上的指定 SR。

若要仅导入元数据，请使用 URI `http://server/import_metadata`

3.4.1. Xen Virtual Appliance (XVA) VM 导入格式

XenServer 支持一种便于用户理解的旧的 VM 输入格式，称为 XVA。本部分介绍 XVA 的语法和结构。

XVA 由包含 XML 元数据和一组磁盘映像的目录组成。由 XVA 表示的 VM 并不是可直接执行的。XVA 包中的数据已压缩，并且可在永久性存储上归档或传输到 VM 服务器（如 XenServer 主机），可在这里进行解压缩和执行。

XVA 是一种不依赖虚拟机管理程序的打包格式；可以创建简单工具来在任何其他平台上实例化 XVA VM。XVA 不指定任何特定运行时格式；例如，可以将磁盘实例化为文件映像、LVM 卷、QCoW 映像、VMDK 或 VHD 映像。可以对 XVA VM 进行任意次数的实例化，每次实例化可以使用不同的运行时格式。

XVA 不能：

- 指定任何特定序列化或传输格式
- 提供安装时自定义 VM（或模板）的任何机制
- 解决安装后如何升级 VM
- 定义用作设备的多个 VM 如何进行通信

这些问题全部由相关的开放式虚拟设备规范解决。

XVA 是至少包含一个 `ova.xml` 文件的目录。此文件描述包含在 XVA 中的 VM，将在 3.2 部分中介绍。磁盘存储在子目录中，并通过 ova.xml 进行引用。磁盘数据的格式稍后将在 3.3 部分中介绍。

在本章的其他部分中将会使用以下术语：

- HVM：未修改的操作系统内核的一种运行模式，在这种模式下需要借助硬件中的虚拟化支持。
- 半虚拟化：经过特殊修改的“半虚拟化”内核基于虚拟机管理程序显式运行的一种模式，在这种模式下不需要对虚拟化提供硬件支持。

“ova.xml”文件包含以下元素：

```
<appliance version="0.1">
```

属性“version”中的数字表示构建 XVA 的此规范的版本；在本例中为 0.1。在 <appliance> 中，只有一个 <vm>（在 OVA 规范中，允许使用多个 <vm>）：

```
<vm name="name">
```

每个 <vm> 元素描述一个 VM。“name”属性仅供将来在内部使用，且在 ova.xml 文件中必须是唯一的。“name”属性可以是任何有效的 UTF-8 字符串。每个 <vm> 标记包含以下必需元素：

```
<label>... text ... </label>
```

显示在 UI 中的 VM 的短名称。

```
<shortdesc> ... description ... </shortdesc>
```

显示在 UI 中的 VM 的说明。请注意，对于<label> 和 <shortdesc> 内容，将忽略前导空格和后导空格。

```
<config mem_set="268435456" vcpus="1"/>
```

<config> 元素具有用于描述 VM 内存量（以字节为单位）的属性 (mem_set)，以及描述 VM 的 CPU 数量的属性 (VCPU)。

每个 <vm> 包含零个或多个表示块设备的 <vbd> 元素，如下所示：

```
<vbd device="sda" function="root" mode="w" vdi="vdi_sda"/>
```

这些属性具有以下含义：

device	向 VM 公开的物理设备的名称。对于 Linux 来宾系统，使用“sd[a-z]”；对于 Windows 来宾系统，使用“hd[a-d]”。
function	如果标记为“root”，则此磁盘将用于引导来宾系统。（需特别注意，这并不意味着存在 Linux 根目录，即 / filesystem）只应将一个设备标记为“root”。请参阅介绍“VM 引导”的 3.4 节。将忽略任何其他字符串。
mode	“w”或“ro”，分别表示设备处于读/写或只读状态。
vdi	此块设备要连接的磁盘映像（由 <vdi> 元素表示）的名称

每个 <vm> 可以包含可选的 <hacks> 部分，如：<hacks is_hvm="false" kernel_boot_cmdline="root=/dev/sda1 ro"/>。<hacks> 元素位于由 XenServer 生成的 XVA 文件中，但以后会被删除。属性“is_hvm”为“true”或“false”，具体取决于 VM 是否应在 HVM 中引导。使用 pygrub 引导来宾系统时，“kernel_boot_cmdline”包含其他内核命令行参数。

除 <vm> 元素外，<appliance> 还将包含零个或多个 <vdi> 元素，如下所示：

```
<vdi name="vdi_sda" size="5368709120" source="file://sda"
type="dir-gzipped-chunks">
```

每个 <vdi> 对应于一个磁盘映像。这些属性具有以下含义：

- name：VDI 的名称，由 <vbd> 元素的 vdi 属性引用。允许使用任何有效 UTF-8 字符串。
- size: 所需映像的大小（以字节为单位）
- source：描述映像数据位置的 URI，目前仅允许使用 file:// URI 且必须描述相对于 ova.xml 所在目录的路径。
- type: 描述磁盘数据的格式（请参阅 3.3 节）

指定了类型为“dir-gzipped-chunks”的单个磁盘映像编码：每个映像由包含文件序列的目录表示，如下所示：

```
-rw-r--r-- 1 dscott xendev 458286013    Sep 18 09:51 chunk000000000.gz
-rw-r--r-- 1 dscott xendev 422271283    Sep 18 09:52 chunk000000001.gz
-rw-r--r-- 1 dscott xendev 395914244    Sep 18 09:53 chunk000000002.gz
-rw-r--r-- 1 dscott xendev 9452401      Sep 18 09:53 chunk000000003.gz
-rw-r--r-- 1 dscott xendev 1096066      Sep 18 09:53 chunk000000004.gz
-rw-r--r-- 1 dscott xendev 971976       Sep 18 09:53 chunk000000005.gz
-rw-r--r-- 1 dscott xendev 971976       Sep 18 09:53 chunk000000006.gz
-rw-r--r-- 1 dscott xendev 971976       Sep 18 09:53 chunk000000007.gz
-rw-r--r-- 1 dscott xendev 573930       Sep 18 09:53 chunk000000008.gz
```

每个文件（名为“chunk-XXXXXXXXX.gz”）都是准确包含 1e9 字节（1 GB，不是 1 GiB）原始块数据的 gzip 压缩文件。选择较小的文件大小是为了确保不超过几种文件系统的最大文件大小限制。如果先将这些文件进行解压，然后再连接到一起，将恢复原始映像。

XenServer 提供了两种 VM 引导机制：(i) 使用通过 pygrub 提取的半虚拟化内核；(ii) 使用 HVM。当前的实现通过 <hacks> 部分中的“is_hvm”标志来确定使用哪种机制。

本部分的其余内容描述一个非常简单的 Debian VM，已打包为 XVA。此 VM 有两个磁盘：一个磁盘大小为 5120 MiB，用于根文件系统和使用 pygrub 引导来宾系统；另一个磁盘大小为 512 MiB，用于交换。此 VM 的内存量为 512 MiB 并使用一个虚拟 CPU。

在最高级别，该简单 Debian VM 由一个目录表示：

```
$ ls -l
total 4
drwxr-xr-x 3 dscott xendev 4096 Oct 24 09:42 very simple Debian VM
```

主 XVA 目录中包含两个子目录（每个磁盘一个）和一个文件 ova.xml：

```
$ ls -l very\ simple\ Debian\ VM/
total 8
-rw-r--r-- 1 dscott xendev 1016 Oct 24 09:42 ova.xml
drwxr-xr-x 2 dscott xendev 4096 Oct 24 09:42 sda
drwxr-xr-x 2 dscott xendev 4096 Oct 24 09:53 sdb
```

各个磁盘子目录中都有一组文件，其中每个文件都包含使用 gzip 压缩的 1 GB 大小的原始磁盘块数据：

```
$ ls -l very\ simple\ Debian\ VM/sda/
total 2053480
-rw-r--r-- 1 dscott xendev 202121645 Oct 24 09:43 chunk-000000000.gz
-rw-r--r-- 1 dscott xendev 332739042 Oct 24 09:45 chunk-000000001.gz
-rw-r--r-- 1 dscott xendev 401299288 Oct 24 09:48 chunk-000000002.gz
-rw-r--r-- 1 dscott xendev 389585534 Oct 24 09:50 chunk-000000003.gz
-rw-r--r-- 1 dscott xendev 624567877 Oct 24 09:53 chunk-000000004.gz
-rw-r--r-- 1 dscott xendev 150351797 Oct 24 09:54 chunk-000000005.gz

$ ls -l very\ simple\ Debian\ VM/sdb
total 516
-rw-r--r-- 1 dscott xendev 521937 Oct 24 09:54 chunk-000000000.gz
```

该简单 Debian VM 示例将包含一个 XVA 文件，如下所示：

```
<?xml version="1.0" ?>
<appliance version="0.1">
  <vm name="vm">
  <label>
  very simple Debian VM
  </label>
  <shortdesc>
  the description field can contain any valid UTF-8
  </shortdesc>
  <config mem_set="536870912" vcpus="1"/>
  <hacks is_hvm="false" kernel_boot_cmdline="root=/dev/sda1 ro ">
  <!--This section is temporary and will be ignored in future. Attribute
is_hvm ("true" or "false") indicates whether the VM will be booted in HVM mode. In
future this will be autodetected. Attribute kernel_boot_cmdline contains the kernel
commandline for the case where a proper grub menu.lst is not present. In future
booting shall only use pygrub.-->
  </hacks>
  <vbd device="sda" function="root" mode="w" vdi="vdi_sda"/>
  <vbd device="sdb" function="swap" mode="w" vdi="vdi_sdb"/>
  </vm>
  <vdi name="vdi_sda" size="5368709120" source="file://sda" type="dir-gzippedchunks"/>
  <vdi name="vdi_sdb" size="536870912" source="file://sdb" type="dir-gzippedchunks"/>
</appliance>
```

3.5. XML-RPC 说明

3.5.1. 日期时间

API 在处理日期时间方面与 XML-RPC 规范有所不同。API 将“Z”附加到日期时间字符串的末尾，以指示以 UTC 表示时间。

3.6. 其他参考来源

在本章中，我们已从较高层面简要概括了 API 及其对象模型。本部分的目的不是提供详细的 API 语义，而只是为您提供足够的背景知识，帮助您开始阅读下一章的代码示例并熟悉内容更为详尽的《XenServer API 参考》参考文档。

您可以从很多位置查找详细信息：

- 《XenServer 管理员指南》包含 xe CLI 的概述。由于许多 xe 命令都建立在 API 的基础之上，因此最好从使用 xe 开始了解本章中介绍的 API 对象模型。
- 下一章中的代码示例提供了一些 API 代码的具体实例，其中使用了多种客户端语言。
- 《XenServer API 参考》参考文档更详细地描述了 API 语义，并介绍了相关 XML/RPC 消息的格式。
- XenServer 主机 dom0 本身中存在一些使用 API 的脚本。例如，“/opt/xensource/libexec/shutdown”是一个可完全关闭 VM 的 python 程序。当关闭主机本身时调用此脚本。

CITRIX

第 4 章 使用 API

本章介绍如何在真实程序中使用 XenServer 管理 API 来管理 XenServer 主机和 VM。本章首先概述典型的客户端应用程序并说明如何使用 API 执行常见任务。提供的示例代码段使用的是 python 语法，但使用其他编程语言的等效代码看起来非常相似。之后将讨论语言绑定本身，在本章末尾介绍两个完整示例。

4.1. 剖析典型应用程序

本节介绍使用 XenServer 管理 API 的典型应用程序的结构。大多数客户端应用程序首先连接到 XenServer 主机并进行身份验证（例如，使用用户名和密码）。假定身份验证成功，服务器将创建一个“会话”对象并将引用返回给客户端。此引用将作为参数传递给所有后续 API 调用。通过身份验证后，客户端可以搜索其他有用对象（例如 XenServer 主机、VM 等）的引用并在其上调用操作。可同步或异步调用操作；特殊任务对象表示异步操作的状态和进度。下列部分详细介绍所有这些应用程序元素。

4.1.1. 选择一种低级别传输

可通过以下两种传输发出 API 调用：

- 基于 IP 网络的端口 443 (https) 上的 SSL 加密 TCP
- 基于本地 Unix 域套接字的纯文本：`/var/xapi/xapi`

SSL 加密的 TCP 传输适用于所有主机外通信，而 Unix 域套接字可用于直接在 XenServer 主机本身上运行的服务。在 SSL 加密的 TCP 传输中，所有 API 调用都应在资源池主服务器直接定向；否则，将导致出现 *HOST_IS_SLAVE* 错误，其中包含主服务器 IP 地址作为错误参数。

由于作为池主服务器的主机可以更改（特别是当在池中启用了高可用性时），因此客户端必须执行下列步骤以检测主服务器主机更改并根据需要连接到新主服务器：

处理池主服务器更改

1. 订阅主机服务器列表中的更新并维护池中当前主机的列表
2. 如果与池主服务器的连接无响应，则尝试与列表中的所有主机进行连接，直到其中一个主机发出响应
3. 发出响应的第一个主机将返回 `HOST_IS_SLAVE` 错误消息，其中包含新池主服务器的标识（除非该主机为新主节点）
4. 连接到新主节点

> 注意：
>
> 作为特殊情况，通过 Unix 域套接字发送的所有消息将透明地转发给正确节点。

4.1.2. 身份验证和会话处理

绝大多数 API 调用采用会话引用作为其第一个参数；如果不能提供有效引用，会导致返回 *SESSION_INVALID* 错误。通过向 login_with_password 函数提供用户名和密码获取会话引用。

> 注意：
>
> 作为特殊情况，如果此调用通过本地 Unix 域套接字执行，则会忽略用户名和密码，并且此调用始终都会成功。

每个会话都包含一个关联的“last active”时间戳，该时间戳会在每次执行 API 调用时进行更新。服务器软件当前具有 200 个活动会话的内置限制，如果超过此限制，将删除“last active”字段最早的会话。此外，还将删除“last active”字段早于 24 小时的所有会话。因此，请务必：

- 记得注销活动会话以避免将其泄露；
- 在遇到 *SESSION_INVALID* 错误时准备再次登录到服务器。

在以下段中，建立了通过 Unix 域套接字的连接且创建了一个会话：

```
import XenAPI

    session = XenAPI.xapi_local()
    try:
        session.xenapi.login_with_password("root", "")
        ...
    finally:
        session.xenapi.session.logout()
```

4.1.3. 查找对有用对象的引用

对应用程序进行身份验证后，下一步是获取对对象的引用，以便查询其状态或在其上调用操作。所有对象都包括一组“隐式”消息，这些消息包含以下内容：

- get_by_name_label：返回特定类具有特定标签的所有对象列表；
- get_by_uuid：返回由其 UUID 指定的单个对象；
- get_all：返回一组对特定类的所有对象的引用；
- get_all_records：返回对特定类的每个对象的记录的引用映射。

例如，列出所有主机：

```
hosts = session.xenapi.host.get_all()
```

查找名为“my first VM”的所有 VM：

```
vms = session.xenapi.VM.get_by_name_label('my first VM')
```

> 注意：
>
> 无法保证对象 *name_label* 字段是唯一的，因此 get_by_name_label API 调用会返回一组引用而不是一个引用。

除了前面介绍的查找对象的方法以外，大多数对象在字段中还包含对其他对象的引用。例如，可以通过调用以下内容查找在特殊主机上运行的 VM 组：

```
vms = session.xenapi.host.get_resident_VMs(host)
```

4.1.4. 在对象上调用同步操作

获取对象引用后，可以在其上调用操作。例如，启动 VM：

```
session.xenapi.VM.start(vm, False, False)
```

默认情况下，所有 API 调用都是同步的，在操作完成或失败之前，不会返回。例如，在使用 VM.start 的情况下，在 VM 开始引导之前，该调用不会返回。

> 注意：
>
> 在 VM.start 调用返回时，VM 将进行引导。要确定引导何时完成，请等待来宾系统内的代理通过 VM_guest_metrics 对象报告内部统计数据。

4.1.5. 使用任务管理异步操作

为了简化非常耗时的管理操作（如 VM.clone 和 VM.copy），系统提供了两种形式的函数：同步函数（默认）和异步函数。每个异步函数都会返回任务对象的引用，其中包括与正在执行的操作有关的信息，包括：

- 操作是否挂起
- 操作成功还是失败
- 操作的进度（范围是 0-1）

- 操作返回的结果或错误代码

以下是一个要跟踪 VM.clone 操作进度并显示进度条的应用程序所包含的代码：

```
vm = session.xenapi.VM.get_by_name_label('my vm')
task = session.xenapi.Async.VM.clone(vm)
while session.xenapi.task.get_status(task) == "pending":
        progress = session.xenapi.task.get_progress(task)
        update_progress_bar(progress)
        time.sleep(1)
session.xenapi.task.destroy(task)
```

注意：

请注意，运行良好的客户端在读取完结果或错误时，应记得删除通过异步操作创建的任务。如果任务数超过了内置阈值，则服务器将删除最先完成的任务。

4.1.6. 预订和侦听事件

服务器将在每次修改对象时生成事件（任务和指标类例外）。客户端可基于每个类订阅此事件流并接收更新，而不是采取频繁地轮询。事件包含三种类型：

- *add* - 创建某个对象时生成；
- *del* - 在即将销毁某个对象之前生成；
- *mod* - 对象的字段更改时生成。

事件还包含单调递增的 ID、对象类的名称和对象状态的快照，这类似于使用 get_record() 的结果。

客户端通过以下方式注册事件：通过类名称列表或特殊字符串“*”调用 event.register()。客户端通过执行 event.next() 来接收事件，该函数直到事件可用才会中断，并返回新事件。

注意：

由于服务器上所生成事件队列的长度有限，速度很慢的客户端可能无法以足够快的速度读取事件；如果发生这种情况，会返回 EVENTS_LOST 错误。为了处理这种情况，客户端应重新注册事件并检查在取消注册时是否未满足它们所等待的条件。

下面的 Python 代码段说明了如何输出由系统生成的每个事件的摘要（`Xenserver-SDK/XenServerPython/samples/watch-all-events.py` 中存在类似的代码）：

```
fmt = "%8s  %20s  %5s  %s"
session.xenapi.event.register(["*"])
while True:
    try:
        for event in session.xenapi.event.next():
            name = "(unknown)"
            if "snapshot" in event.keys():
                snapshot = event["snapshot"]
                if "name_label" in snapshot.keys():
                    name = snapshot["name_label"]
            print fmt % (event['id'], event['class'], event['operation'], name)
    except XenAPI.Failure, e:
        if e.details == [ "EVENTS_LOST" ]:
            print "Caught EVENTS_LOST; should reregister"
```

4.2. 语言绑定

4.2.1. C

SDK 在目录 `XenServer-SDK/libxenserver/src` 中包含 C 语言绑定源，以及用于将此绑定编译到库中的 Makefile。每个 API 对象都与一个头文件相关联，该文件中包含所有该对象的 API 函数的声明；例如，调用 VM 操作所需要的类型定义和函数都包含在 `xen_vm.h` 中。

C 绑定依赖项

支持的平台：	Linux
库：	语言绑定生成为由 C 程序链接的 `libxenserver.so`。
依赖项：	• XML 库（GNU Linux 上的 libxml2.so） • Curl 库 (libcurl2.so)

下面是 C 绑定中包含的几个简单示例：

- `test_vm_async_migrate`：演示如何使用异步 API 调用将正在运行的 VM 从从属主机迁移到池主服务器。
- `test_vm_ops`：此示例说明如何查询主机的功能、创建 VM 以及如何将全新的空磁盘映像连接到 VM，然后执行各种开关机周期（关闭后再打开）操作；
- `test_failures`：演示如何将错误字符串转换为 enum_xen_api_failure，反之亦然；
- `test_event_handling`：演示如何通过连接侦听事件。
- `test_enumerate`：演示如何枚举各种 API 对象。

4.2.2. C#

C# 绑定包含在 `XenServer-SDK/XenServer.NET` 目录中，并包含适用于在 Microsoft Visual Studio 下构建的项目文件。每个 API 对象都与一个 C# 文件相关联；例如，实现 VM 操作的函数包含在 `VM.cs` 文件中。

C# 绑定依赖项

支持的平台：	安装了 .NET 2.0 版的 Windows
库：	此语言绑定以 C# 程序所链接的动态链接库 `XenServer.dll` 的形式生成。
依赖项：	XenServer.dll 需要借助 `CookComputing.XMLRpcV2.dll` 与 xml-rpc 服务器通信。虽然其他版本也可以，但是我们对版本 2.1.0.6 进行了测试，因此建议您使用此版本。

目录 `XenServer-SDK/XneServer.NET/samples` 中的 C# 绑定包括以下三个示例（作为 `XenSdkSample.sln` 解决方案的单独项目）：

- `GetVariousRecords`：登录到 XenServer 主机，并显示有关主机、存储和虚拟机的信息；
- `GetVmRecords`：登录到 XenServer 主机，并列出所有 VM 记录；
- `VmPowerStates`：登录到 XenServer 主机，查找 VM，并使其进入各种电源状态。需要安装已经关闭电源的 VM。

4.2.3. Java

Java 绑定包含在 `XenServer-SDK/XenServerJava` 目录中，并包含适用于在 Microsoft Visual Studio 下构建的项目文件。每个 API 对象都与一个 Java 文件相关联；例如，实现 VM 操作的函数包含在 `VM.java` 文件中。

Java 绑定依赖项

支持的平台：	Linux 和 Windows

库：	此语言绑定以 Java 程序所链接的 Java 存档文件 `xenserver-6.2.0.jar` 的形式生成。
依赖项：	• `xenserver.jar` 需要借助 `xmlrpc-client-3.1.jar` 与 xml-rpc 服务器通信。 • 需要 `ws-commons-util-1.0.2.jar`，才能运行这些示例。

运行主文件 `XenServer-SDK/XenServerJava/samples/RunTests.java` 将运行一系列包含在同一目录中的示例：

- `AddNetwork`：添加未连接到任何 NIC 的新内部网络；
- `SessionReuse`：演示如何在多个连接之间共享会话对象；
- `AsyncVMCreate`：从内置模板中异步创建新 VM，启动和停止该 VM；
- `VdiAndSrOps`：执行各种 SR 和 VDI 测试，其中包括创建虚拟 SR；
- `CreateVM`：通过网络和 DVD 驱动器在默认 SR 上创建 VM；
- `DeprecatedMethod`：测试当调用已弃用的 API 方法时是否显示警告；
- `GetAllRecordsOfAllTypes`：检索所有对象类型的所有记录；
- `SharedStorage`：创建共享 NFS SR；
- `StartAllVMs`：连接至主机并尝试启动该主机上的每个 VM。

4.2.4. PowerShell

此版本中包含两个版本的 Powershell 语言绑定。

之前发布的 PowerShell 绑定包含在 `XenServer-SDK/XenPSSnapIn_old` 目录中。我们提供 Windows 安装程序 `XenServerPSSnapIn.msi` 和源代码，将 XenServer API 显示为 Windows PowerShell 1.0 cmdlet。

旧的 PowerShell 绑定依赖项

支持的平台：	安装了 .NET Framework 3.5 和 Powershell v1.0 的 Windows
库：	`XenServerPSSnapIn.dll`
依赖项：	需要借助 `CookComputing.XMLRpcV2.dll` 与 xml-rpc 服务器通信。虽然其他版本也可以，但是我们对版本 2.1.0.6 进行了测试，因此建议您使用此版本。

此示例脚本包含在旧 Powershell 绑定的 `XenServer-SDK/XenPSSnapIn_old/samples` 目录中：

- `AutomatedTestCore.ps1`：登录到 XenServer 主机，创建存储库和 VM，然后执行各种开关机周期（关闭后再打开）操作。

经过重新设计的全新 PowerShell 绑定包含在 `XenServer-SDK/XenPSSnapIn` 目录中。我们提供 Windows 安装程序 `XenServerPSSnapIn.msi` 和源代码，将 XenServer API 显示为 Windows PowerShell 2.0 cmdlet。

PowerShell 绑定依赖项

支持的平台：	安装了 .NET Framework 3.5 和 Powershell v2.0 的 Windows
库：	`XenServerPSSnapIn.dll`

依赖项：	需要借助 `CookComputing.XMLRpcV2.dll` 与 xml-rpc 服务器通信。虽然其他版本也可以，但是我们对版本 2.1.0.6 进行了测试，因此建议您使用此版本。

此示例脚本包含在 Powershell 绑定的 `XenServer-SDK/XenPSSnapIn/samples` 目录中：

- `AutomatedTestCore.ps1`：登录到 XenServer 主机，创建存储库和 VM，然后执行各种开关机周期（关闭后再打开）操作。

4.2.5. Python

Python 绑定包含在单个文件 `XenServer-SDK/XenServerPython/XenAPI.py` 中。

Python 绑定依赖项

支持的平台：	Linux
库：	XenAPI.py
依赖项：	无

SDK 包含 7 个 python 示例：

- `fixpbds.py` - 重新配置用于访问共享存储的设置；
- `install.py` - 安装 Debian VM、将其连接到网络、启动并等待它报告其 IP 地址；
- `license.py` - 将全新许可证上载到 XenServer 主机；
- `permute.py` - 选择一组 VM 并使用 XenMotion 在主机之间同时移动它们；
- `powercycle.py` - 选择一组 VM 并启动；
- `shell.py` - 用于测试的简单交互 shell；
- `vm_start_async.py` - 说明如何异步调用操作；
- `watch-all-events.py` - 注册所有事件，并在这些事件发生时显示详细信息。

4.2.6. 命令行接口 (CLI)

除了使用原始 XML-RPC 或提供的语言绑定之一，第三方软件开发人员还可以使用 XE 命令行接口 xe 与 XenServer 主机集成。默认情况下，xe CLI 安装在 XenServer 主机上；独立的远程 CLI 也可用于 Linux。在 Windows 上，`xe.exe` CLI 可执行文件随 XenCenter 一起安装。

CLI 依赖项

支持的平台：	Linux 和 Windows
库：	无
二进制：	xe（Windows 上的 xe.exe）
依赖项：	无

CLI 允许从脚本或其他程序直接调用几乎每个 API 调用，以静默方式完成所需的会话管理。《XenServer 管理员指南》中详细介绍了 XE CLI 语法和功能。有关其他资源和示例，请访问 Citrix 知识中心。

注意：

当从 XenServer 主机控制台运行 CLI 时，可以通过 Tab 键自动补齐命令名称和参数。

4.3. 完整的应用程序示例

本节介绍了两个使用 API 的真实程序的完整示例。

4.3.1. 使用 XenMotion 同时迁移 VM

此 python 示例（包含在 `XenServer-SDK/XenServerPython/samples/permute.py` 中）说明如何使用 XenMotion 在资源池中的主机之间同时移动 VM。此示例使用异步 API 调用并显示如何等待完成一组任务。

此程序开头部分是一些标准样本并导入 API 绑定模块

```
import sys, time
import XenAPI
```

接下来此程序解析包含服务器 URL、用户名、密码和大量迭代的命令行参数。用户名和密码用于建立传递到函数 `main` 的会话，该函数以循环形式进行多次调用。请注意使用 `try: finally:` 确保程序在结束时从其会话注销。

```
if __name__ == "__main__":
    if len(sys.argv) <> 5:
        print "Usage:"
        print sys.argv[0], " <url> <username> <password> <iterations>"
        sys.exit(1)
    url = sys.argv[1]
    username = sys.argv[2]
    password = sys.argv[3]
    iterations = int(sys.argv[4])
    # First acquire a valid session by logging in:
    session = XenAPI.Session(url)
    session.xenapi.login_with_password(username, password)
    try:
        for i in range(iterations):
            main(session, i)
    finally:
        session.xenapi.session.logout()
```

`main` 函数检查系统中每个正在运行的 VM，筛选出控制域（控制域是系统的一部分，无法由用户控制）。该函数将构建正在运行的 VM 及其当前主机的列表。

```
def main(session, iteration):
    # Find a non-template VM object
    all = session.xenapi.VM.get_all()
    vms = []
    hosts = []
    for vm in all:
        record = session.xenapi.VM.get_record(vm)
        if not(record["is_a_template"]) and \
           not(record["is_control_domain"]) and \
           record["power_state"] == "Running":
            vms.append(vm)
            hosts.append(record["resident_on"])
    print "%d: Found %d suitable running VMs" % (iteration, len(vms))
```

接下来循环主机列表：

```
# use a rotation as a permutation
    hosts = [hosts[-1]] + hosts[:(len(hosts)-1)]
```

然后，每个 VM 通过 XenMotion 移动到该循环下的新主机中（即，在列表中第 2 个位置的主机上运行的 VM 将移至列表中第 1 个位置的主机上，依此类推）。要并行执行各移动，应使用 VM.pool_migrate 的异步版本并构建任务引用列表。请记住传递到 VM.pool_migrate 的 *live* 标志；该标记会使 VM 在运行状态下移动。

```
tasks = []
    for i in range(0, len(vms)):
        vm = vms[i]
        host = hosts[i]
        task = session.xenapi.Async.VM.pool_migrate(vm, host, { "live": "true" })
        tasks.append(task)
```

然后，轮询任务列表以查看是否已完成：

```
finished = False
    records = {}
    while not(finished):
        finished = True
        for task in tasks:
            record = session.xenapi.task.get_record(task)
            records[task] = record
            if record["status"] == "pending":
                finished = False
        time.sleep(1)
```

当所有任务都退出挂起状态（即已成功完成、已失败或取消）后，将再次轮询这些任务以查看是否已全部成功完成：

```
allok = True
    for task in tasks:
        record = records[task]
        if record["status"] <> "success":
            allok = False
```

如果任一任务失败，则会打印详细信息、引发异常并保留任务对象以供将来检查。如果所有任务都成功完成，则销毁任务对象，且函数将返回。

```
if not(allok):
        print "One of the tasks didn't succeed at", \
            time.strftime("%F:%HT%M:%SZ", time.gmtime())
        idx = 0
        for task in tasks:
            record = records[task]
            vm_name = session.xenapi.VM.get_name_label(vms[idx])
            host_name = session.xenapi.host.get_name_label(hosts[idx])
            print "%s : %12s %s -> %s [ status: %s; result = %s; error = %s ]" % \
                  (record["uuid"], record["name_label"], vm_name, host_name,      \
                   record["status"], record["result"], repr(record["error_info"]))
            idx = idx + 1
        raise "Task failed"
    else:
        for task in tasks:
            session.xenapi.task.destroy(task)
```

4.3.2. 使用 XE CLI 克隆 VM

此示例是一个 bash 脚本，使用 XE CLI 克隆 VM，请注意，如果处于打开状态，应先将其关闭。

此示例开头部分是一些样本，这些样本首先检查是否设置了环境变量 *XE*：如果设置了该变量，则将其视为指向 CLI 的完整路径；否则认为 XE CLI 在当前路径中。接下来，该脚本会提示用户输入服务器名称、用户名和密码：

```
# Allow the path to the 'xe' binary to be overridden by the XE environment variable
if [ -z "${XE}" ]; then
  XE=xe
fi

if [ ! -e "${HOME}/.xe" ]; then
  read -p "Server name: " SERVER
  read -p "Username: " USERNAME
  read -p "Password: " PASSWORD
  XE="${XE} -s ${SERVER} -u ${USERNAME} -pw ${PASSWORD}"
fi
```

接下来该脚本会检查其命令行参数。只需要一个参数：要克隆的 VM 的 UUID：

```
# Check if there's a VM by the uuid specified
${XE} vm-list params=uuid | grep -q " ${vmuuid}$"
if [ $? -ne 0 ]; then
        echo "error: no vm uuid \"${vmuuid}\" found"
        exit 2
fi
```

然后，该脚本将检查 VM 的电源状态，如果 VM 正在运行，则将尝试干净关闭。使用事件系统以等待 VM 进入“Halted”状态。

注意：

XE CLI 支持命令行参数 *--minimal*，使用该参数（适于从脚本使用）时，打印输出中将没有多余空格或格式。如果返回多个值，则使用逗号分隔。

```
# Check the power state of the vm
name=$(${XE} vm-list uuid=${vmuuid} params=name-label --minimal)
state=$(${XE} vm-list uuid=${vmuuid} params=power-state --minimal)
wasrunning=0

# If the VM state is running, we shutdown the vm first
if [ "${state}" = "running" ]; then
        ${XE} vm-shutdown uuid=${vmuuid}
        ${XE} event-wait class=vm power-state=halted uuid=${vmuuid}
        wasrunning=1
fi
```

此时将克隆该 VM，新 VM 的 *name_label* 设置为 *cloned_vm*。

```
# Clone the VM
newuuid=$(${XE} vm-clone uuid=${vmuuid} new-name-label=cloned_vm)
```

最后，如果原始 VM 原来正在运行而后来被关闭，则它将和新 VM 一起启动。

```
# If the VM state was running before cloning, we start it again
# along with the new VM.
if [ "$wasrunning" -eq 1 ]; then
        ${XE} vm-start uuid=${vmuuid}
        ${XE} vm-start uuid=${newuuid}
fi
```

CITRIX

第 5 章 使用 HTTP 与 XenServer 交互

XenServer 在每台主机上提供一个 HTTP 接口，这些接口可用于执行各种操作。本章将介绍相关的可用机制。

5.1. VM 导入和导出

由于 VM 的导入和导出操作需要花费一些时间才能完成，所以提供了用于导入和导出操作的异步 HTTP 接口。要使用 XenServer API 执行导出操作，可构建一个 HTTP GET 调用以提供有效会话 ID、任务 ID 和 VM UUID，如以下伪代码所示：

```
task = Task.create()
result = HTTP.get(
  server, 80, "/export?session_id=<session_id>&task_id=<task_id>&ref=<vm_uuid>");
```

对于导入操作，可以使用 HTTP PUT 调用，如以下伪代码所示：

```
task = Task.create()
result = HTTP.put(
  server, 80, "/import?session_id=<session_id>&task_id=<task_id>&ref=<vm_uuid>");
```

5.2. 获取 XenServer 性能统计信息

XenServer 记录与您的 XenServer 安装的各个方面的性能有关的统计信息。这些指标将永久存储以供长期访问和进行历史趋势分析。当存储对 VM 可用时，统计信息将在 VM 关闭时写入磁盘。统计信息存储在为各个 VM（包含控制域）和服务器保留的 RRD（轮询数据库）中。RRD 位于运行 VM 的服务器上，如果 VM 未处于运行状态，则位于池主服务器上。RRD 也会每天备份。

> 警告：
>
> 在 XenServer API 的早期版本中，可以使用 VM_metrics、VM_guest_metrics、host_metrics 方法及相关方法获取即时性能指标。不推荐使用此类方法，而是支持使用本章介绍的 HTTP 处理程序从 VM 和服务器上的 RRD 下载统计信息。请注意，默认情况下旧的指标不返回任何内容。要还原为 XenServer 的早期版本中的定期统计轮询，请将您主机上的 *other-config:rrd_update_interval=<interval>* 参数设置为下列值之一，然后重新启动您的主机：
>
> never　这是默认值，表示不执行定期轮询。
>
> 1　每 5 秒执行一次轮询。
>
> 2　每 1 分钟执行一次轮询。
>
> 默认情况下，旧版指标 API 不会返回任何值，因此要运行使用旧版监视协议的监视客户端，必须启用此键。

统计信息以不同的粒度存储，最多保留一年时间。平均值和最近值以下面的时间间隔存储：

- 在过去 10 分钟内每隔 5 秒
- 在过去 2 个小时内每隔 1 分钟
- 在过去 1 周内每隔 1 小时
- 在过去 1 年内每隔 1 天

RRD 作为未经压缩的 XML 保存到磁盘。当 RRD 存储全年统计信息时，每个 RRD 写入磁盘的大小范围为 200 KiB 到大约 1.2 MiB。

> 警告：

如果统计信息无法写入磁盘（例如，当磁盘已满时），统计信息将丢失，并将使用 RRD 的上次保存版本。

可通过 HTTP 以 XML 格式下载统计信息，例如使用 wget。有关 XML 格式的信息，请参阅 http://oss.oetiker.ch/rrdtool/doc/rrddump.en.html 和 http://oss.oetiker.ch/rrdtool/doc/rrdxport.en.html。HTTP 身份验证可以采取用户名和密码的形式或会话令牌的形式。参数附加到 URL，在问号 (?) 之后，并以 & 分隔。

要获取主机上的所有 VM 统计信息的更新，URL 应为如下形式：

```
http://<username>:<password>@<host>/rrd_updates?start=<secondssinceepoch>
```

此请求将为正在查询的特定主机上的每个 VM 返回 rrdtool `xport` 样式 XML 格式的数据。为区分导出中的列与哪个 VM 相关联，`legend` 字段将以 VM 的 UUID 作为前缀。

要同时获取主机更新，请使用查询参数 *`host=true`*：

```
http://<username>:<password>@<host>/rrd_updates?start=<secondssinceepoch>&host=true
```

时间段越短则步长越小，这意味着，请求查询统计信息的时间段越短，获得的统计信息越详细。

其他 rrd_updates 参数

cf=*`<ave|min|max>`*	数据合并模式
interval=*`<interval>`*	要报告的值之间的间隔

注意：

默认情况下，只有 `ave` 统计信息可用。要获取 VM 的 `min` 和 `max` 统计信息，请运行以下命令：

```
xe pool-param-set uuid=<pool_uuid> other-config:create_min_max_in_new_VM_RRDs
```

获取主机的所有统计信息：

```
http://<username:password@host>/host_rrd
```

获取 VM 的所有统计信息：

```
http://<username:password@host>/vm_rrd?uuid=<vm_uuid>
```

CITRIX

第 6 章 XenServer API 扩展

XenAPI 是一个用于管理虚拟机的生命周期的常规综合性接口，具有很大的灵活性，XenAPI 提供程序可以实现特定功能，如置备存储或处理控制台。XenServer 具有多个扩展，我们可以在自己的 XenCenter 界面中使用这些扩展提供的有用功能。本章介绍这些机制的工作方法。

通常采用为各种对象指定 *other-config* 映射键的方式来提供 XenAPI 扩展。此参数用来表示 XenServer 的特定版本支持某一功能，但该功能不是长期功能。我们一直在评估 API 的功能以进行提升，这需要对界面的特性有很好的了解。我们始终欢迎您就其中一些扩展的使用情况提供开发人员反馈，以帮助我们做出这些决策。

6.1. VM 控制台转发

大多数 XenAPI 图形界面都希望能够访问 VM 控制台，以便将这些控制台如同物理机一样呈现给用户。存在多种可用控制台类型，具体取决于来宾系统的类型或是否正在访问物理主机控制台：

控制台访问

操作系统	文本	图形	优化的图形
Windows	否	VNC，使用 API 调用	RDP，直接来自来宾系统
Linux	是，通过 VNC 调用和 API 调用	否	VNC，直接来自来宾系统
物理主机	是，通过 VNC 调用和 API 调用	否	否

硬件辅助 VM（例如 Windows）通过 VNC 直接提供图形控制台。不存在基于文本的控制台，使用图形控制台不需要来宾系统网络。建立来宾系统网络后，更高效的方法是，设置“远程桌面访问”并使用 RDP 客户端直接连接（必须在 XenAPI 外部执行此操作）。

半虚拟化 VM（例如 Linux 来宾系统）可直接提供本机文本控制台。XenServer 提供了一个名为 vncterm 的实用程序，用来将此基于文本的控制台转换为图形 VNC 表示形式。此控制台无需来宾系统网络即可使用。与前面的 Windows 版本类似，Linux 版本通常在来宾系统内部配置 VNC，然后通过来宾系统网络接口直接与其连接。

物理主机控制台仅可用作 `vt100` 控制台，可通过在控制域中使用 vncterm 来通过 XenAPI 将其作为 VNC 控制台显示。

VNC 基于 RFB（远程 Framebuffer）协议，这是在 RFB 协议中指定的。第三方开发人员应提供他们自己的 VNC 查看器，许多免费提供的实施可用来实现此目的。查看器支持的最低版本是 RFB 3.3。

6.1.1. 使用 API 检索 VNC 控制台

使用传递到主机代理的特殊 URL 可以检索 VNC 控制台。API 调用序列如下所示：

1. 客户端到主节点/443：XML-RPC: Session.login_with_password()。
2. 主节点/443 到客户端：返回用于后续调用的会话引用。
3. 客户端到主节点/443：XML-RPC: VM.get_by_name_label()。
4. 主节点/443 到客户端：返回特定 VM（或“控制域”— 如果您希望检索物理主机控制台）的引用。
5. 客户端到主节点/443：XML-RPC: VM.get_consoles()。
6. 主节点/443 到客户端：返回与 VM 相关联的控制台对象的列表。
7. 客户端到主节点/443：XML-RPC: VM.get_location()。

8. 返回一个 URI，描述所请求控制台的位置。这些 URI 的形式为：https://192.168.0.1/console?ref=OpaqueRef:c038533a-af99-a0ff-9095-c1159f2dc6a0。
9. 客户端到 192.168.0.1：HTTP CONNECT "/console?ref=(...)"

最终的 HTTP CONNECT 与标准略有不符，因为 HTTP/1.1 RFC 指定它应该只是主机和端口，而不是 URL。完成 HTTP 连接后，该连接随后可以直接用作 VNC 服务器，无需其他 HTTP 协议操作。

此方案需要从客户端直接访问控制域的 IP，如果存在阻止此类连接的网络地址转换 (NAT) 设备，则无法正确运行。可以使用 CLI 从客户端检索控制台 URI，然后执行连接检查。

使用 CLI 检索控制台 URI

1. 通过运行以下命令检索 VM UUID：

   ```
   xe vm-list params=uuid --minimal name-label=name
   ```

2. 通过以下命令检索控制台信息：

   ```
   xe console-list vm-uuid=uuid
   uuid ( RO): 714f388b-31ed-67cb-617b-0276e35155ef
   vm-uuid ( RO): 8acb7723-a5f0-5fc5-cd53-9f1e3a7d3069
   vm-name-label ( RO): etch
   protocol ( RO): RFB
   location ( RO): https://192.168.0.1/console?ref=(...)
   ```

使用类似 ping 的命令行实用程序可测试与 *location* 字段中提供的 IP 地址的连接。

6.1.2. 禁用 Linux VM 的 VNC 转发

创建和销毁 Linux VM 时，主机代理会自动管理将文本控制台转换为 VNC 的 vncterm 进程。希望直接访问文本控制台的高级用户可以禁用该 VM 的 VNC 转发。之后，仅可以从控制域直接访问文本控制台，XenCenter 等图形界面将无法呈现该 VM 的控制台。

使用 CLI 禁用 Linux VNC 控制台

1. 启动来宾系统之前，在 VM 记录上设置下列参数：

   ```
   xe vm-param-set uuid=uuid other-config:disable_pv_vnc=1
   ```

2. 启动 VM。
3. 通过以下命令使用 CLI 检索 VM 的基础域 ID：

   ```
   xe vm-list params=dom-id uuid=<uuid> --minimal
   ```

4. 在主机控制台上，直接连接到文本控制台：

   ```
   /usr/lib/xen/bin/xenconsole <domain_id>
   ```

此配置是一个高级过程，我们建议您不要将文本控制台直接用于大量的 I/O 操作，而是通过 SSH 或其他一些基于网络的连接机制连接到来宾系统。

6.2. 半虚拟化 Linux 安装

由于必须引导 Xen 感知内核，而不是使用硬件辅助技术简单地安装来宾系统，半虚拟化 Linux 来宾系统的安装较为复杂。而这具有在仿真开销不足时使安装速度接近于本机安装速度的好处。XenServer 支持安装多个不同的 Linux 版本，并尽可能精简此过程。

最后，在控制域中将出现一个名为 eliloader 的特殊引导加载器，它启动时会读取 VM 记录中的各种 *other-config* 键，并执行特定于版本的安装行为。

- *install-repository* - 必须指定。存储库的路径；“http”、“https”、“ftp”或“nfs”。应指定为目标安装程序将使用的形式，但不应包含前缀，例如 method=。

- *install-vnc* - 默认值：false。在安装期间，在可以使用的位置使用 VNC。
- *install-vncpasswd* - 默认值：empty。要使用的 VNC 密码（如果可以使用目标版本的命令行提供一个密码）。
- *install-round* - 默认值：1。当前引导加载器循环。用户无法编辑（如下所示）。

6.2.1. Red Hat Enterprise Linux 4.1/4.4

eliloader 用于进行两轮引导。在第一轮引导中，它通过 `/opt/xensource/packages/files/guest-installer` 返回安装程序 `initrd` 和内核。然后，在第二轮引导中，它从 VM 中删除附加更新磁盘，接着从当前引导加载器切换到 pygrub，然后开始正常引导。

此序列是必需的，因为 Red Hat 不为这些版本提供 Xen 内核，因而会改用这些版本的 XenServer 自定义内核。

6.2.2. Red Hat Enterprise Linux 4.5/5.0

与 RHEL4.4 安装完全类似，不同之处只是直接从用户指定的网络存储库下载内核和 Ramdisk，以及将引导加载器立即切换到 pygrub。请注意，pygrub 不会立即执行，而仅会在下一次引导时解析。

通过网络检索，用户可直接从其网络存储库安装上游 Red Hat 供应商内核。`xs-tools.iso` 内置 ISO 映像上还提供了更新的 XenServer 内核，修复了各种与 Xen 相关的缺陷。

6.2.3. SUSE Enterprise Linux 10 SP1

要求两轮引导过程。第一轮引导过程从网络存储库下载内核和 Ramdisk，然后引导它们。接着，第二轮引导过程将检查磁盘以查找安装的内核和 Ramdisk，并设置 *PV-bootloader-args* 以反映来宾文件系统内的那些路径。此进程模拟 domUloader，SUSE 将其用作 pygrub 的替代方法。最后，此引导加载器将设置为 pygrub 并加以执行，以开始正常引导。

SLES 10 安装方法表示内核和 Ramdisk 的路径存储在 VM 记录中而不是来宾系统 `menu.lst` 中，但这是它运行的唯一方式，因为 YAST 软件包管理器未写入有效的 `menu.lst`。

6.2.4. CentOS 4.5 / 5.0

CentOS 安装机制与前面提到的 Red Hat 安装类似，只是 eliloader 识别的一些 MD5 校验不同。

6.3. 向 VM 添加 Xenstore 条目

开发人员可能希望在 VM 中安装来宾系统代理，以根据 VM 类型执行特殊的操作。为将此信息告知来宾系统，可使用名为 `vm-data` 的特殊 Xenstore 命名空间，此命名空间在创建 VM 时填充，它通过 VM 记录中的 *xenstore-data* 映射填充。

在 VM 中填充 Xenstore 节点 *foo*

1. 设置 VM 记录中的 *xenstore-data* 参数：

   ```
   xe vm-param-set uuid=<vm_uuid> xenstore-data:vm-data/foo=bar
   ```

2. 启动 VM。
3. 如果是基于 Linux 的 VM，请安装 XenServer Tools，然后使用 xenstore-read 验证该节点在 Xenstore 中是否存在。

   注意：

   仅允许使用以 `vm-data` 开头的前缀，启动 VM 时将默认忽略任何不在此命名空间中的内容。

6.4. 安全功能增强

XenServer 6.2.0（及更高版本）中的控制域具有多种安全功能增强，以加强抵御恶意来宾系统攻击的能力。开发人员不应将任何正常功能的丢失视为这些更改导致的结果，不过下面将这些更改作为其他版本行为的变体进行了归档。

- 插槽接口 xenstored，可使用 `libxenstore` 进行访问。这些接口由 `xs_restrict()` 限制。
- 设备 `/dev/xen/evtchn`，可通过调用 `xs_evtchn_open()`（在 `libxenctrl` 中）进行访问。可以使用 `xs_evtchn_restrict()` 限制句柄。
- 设备 `/proc/xen/privcmd`，可通过 `xs_interface_open()`（在 `libxenctrl` 中）进行访问。可以使用 `xc_interface_restrict()` 限制句柄。一些特权命令很难加以限制（例如，生成任意层的超级调用），只是在受限句柄上禁用这些命令。
- 受限句柄以后无法被授予更多特权，因此必须关闭并重新打开接口。仅当此进程随后无法打开更多句柄时，才能保证安全。

现在可以将控制域特权用户空间接口限制为仅对特定域发挥作用。下面是受此更改影响的三个接口：

- qemu 设备仿真进程和 vncterm 终端仿真进程作为非 root 用户 ID 运行，并被限制为在空目录中运行。它们使用上述限制 API 以在可能的情况下放弃权限。
- 对 xenstore 的访问受速率限制，以防止恶意来宾系统造成控制域上发生拒绝服务的情况。这可以作为具有受限填充速率的令牌桶实现，其中大多数操作使用一个令牌，打开一个事务需要 20 个令牌。这些限制设置得很高，即使是在负载操作下运行大量并发来宾系统，也永远达不到这些限制。
- VNC 来宾系统控制台仅绑定到 `localhost` 接口，以使它们不会对外部公开，即使已通过用户干预禁用了控制域数据包过滤器也不会公开。

6.5. 网络接口的高级设置

虚拟网络接口和物理网络接口具有一些高级设置，可使用 *`other-config`* 映射参数配置这些设置。这包含一组自定义 ethtool 设置及一些其他设置。

6.5.1. ethtool 设置

开发人员可能希望为物理网络接口和虚拟网络接口配置自定义 ethtool 设置。这可以通过 *`other-config`* 映射参数使用 *`ethtool-<option>`* 键来完成。

键	说明	有效设置
ethtool-rx	指定是否启用 RX 校验和检查	如果为 `on` 或 `true`，则启用该设置，如果为 `off` 或 `false`，则禁用该设置
ethtool-tx	指定是否启用 TX 校验和检查	如果为 `on` 或 `true`，则启用该设置，如果为 `off` 或 `false`，则禁用该设置
ethtool-sg	指定是否启用分散/集中	如果为 `on` 或 `true`，则启用该设置，如果为 `off` 或 `false`，则禁用该设置
ethtool-tso	指定是否启用 tcp 分段卸载	如果为 `on` 或 `true`，则启用该设置，如果为 `off` 或 `false`，则禁用该设置
ethtool-ufo	指定是否启用 UDP 段卸载	如果为 `on` 或 `true`，则启用该设置，如果为 `off` 或 `false`，则禁用该设置

键	说明	有效设置
ethtool-gso	指定是否启用通用分段卸载	如果为 on 或 true，则启用该设置，如果为 off 或 false，则禁用该设置
ethtool-autoneg	指定是否启用自动协商	如果为 on 或 true，则启用该设置，如果为 off 或 false，则禁用该设置
ethtool-speed	设置设备速度，单位采用 MB/秒	10、100 或 1000
ethtool-duplex	设置全双工或半双工模式	half 或 full

例如，使用 xe CLI 对虚拟 NIC 启用 TX 校验和检查：

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-tx="on"
```

或：

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-tx="true"
```

使用 xe CLI 将物理 NIC 上的双工设置设为半双工：

```
xe vif-param-set uuid=<VIF UUID> other-config:ethtool-duplex="half"
```

6.5.2. 其他设置

还可以通过将 *promiscuous* 键设置为 on，在 VIF 或 PIF 上设置混杂模式。例如，使用 xe CLI 对物理 NIC 启用混杂模式：

```
xe pif-param-set uuid=<PIF UUID> other-config:promiscuous="on"
```

或：

```
xe pif-param-set uuid=<PIF UUID> other-config:promiscuous="true"
```

VIF 对象和 PIF 对象都具有 *MTU* 参数，该参数是只读的，可提供接口的最大传输单位的当前设置。可以通过 *other-config* 映射参数使用 *mtu* 键覆盖物理或虚拟 NIC 的默认最大传输单位。例如，使用 xe CLI 将虚拟 NIC 上的 MTU 重置为使用 Jumbo 帧：

```
xe vif-param-set uuid=<VIF UUID> other-config:mtu=9000
```

请注意，更改基础接口的 MTU 是一个高级实验功能，如果单个资源池中的 NIC 上存在不同的 MTU，可能会导致意外的副面影响。

6.6. SR 名称的国际化

在安装时创建的 SR 现在包含一个 *other_config* 键，指示 SR 名称国际化的方式。

other_config["i18n-key"] 可能为下列键之一：

- *local-hotplug-cd*
- *local-hotplug-disk*
- *local-storage*
- *xenserver-tools*

此外，*other_config["i18n-original-value-<field name>"]* 在创建 SR 后会提供该字段的值。如果 XenCenter 发现一个记录，其 *SR.name_label* 等于 *other_config["i18n-original-*

value-name_label"]（即此记录自从在 XenServer 安装期间创建后未更改），则将应用国际化。换言之，XenCenter 将忽略该字段的当前内容，而使用与用户自己的语言相对应的值。

如果您按自己的需要更改 *SR.name_label*，它将不再与 *other_config["i18n-original-value-name_label"]* 相同。因此，XenCenter 将不应用国际化，而是保留您指定的名称。

6.7. 从 XenCenter 隐藏对象

通过将键 `HideFromXenCenter=true` 添加到对象的 *other_config* 参数，可以从 XenCenter 隐藏网络、PIF 和 VM。此功能面向了解自己正在执行什么操作的 ISV，而不是供日常用户的常规使用。例如，您可能想隐藏某些 VM，因为它们在您的环境中是不应由常规用户直接使用的克隆 VM。

在 XenCenter 中，可以使用查看菜单使隐藏的网络、PIF 和 VM 可见。

CITRIX

第 7 章 XenCenter API 扩展

除了记录的 API 之外，下面的部分详细介绍了我们所做的假设和 API 扩展。扩展在词典中编码为特殊键值对，如 `VM.other_config`。

7.1. 池

键	语义
pool.name_label	空 name_label 表明池应在树视图中隐藏。
pool.rolling_upgrade_in_progress	如果池处于滚动升级期间，则显示该池。

7.2. 主机

键	语义
host.other_config["iscsi_iqn"]	主机的 iSCSI IQN。
host.license_params["expiry"]	主机许可证的到期日期，采用 UTC ISO 8601 时间。
host.license_params["sku_type"]	主机许可证类型，例如 Server 或 Enterprise。
host.license_params["restrict_pooling"]	如果池功能受主机的限制，则返回 `true`。
host.license_params["restrict_connection"]	可以从 XenCenter 进行连接的连接数目受限制。
host.license_params["restrict_qos"]	如果已在主机上启用“Quality of Service”（服务质量）设置，则返回 `true`。
host.license_params["restrict_vlan"]	如果在主机上创建虚拟网络受限制，则返回 `true`。
host.license_params["restrict_pool_attached_storage"]	如果在此主机上创建共享存储受限制，则返回 `true`。
host.software_version["product_version"]	返回主机的产品版本。
host.software_version["build_number"]	返回主机的内部版本号。
host.software_version["xapi"]	返回主机的 API 修订号。
host.software_version["package-linux"]	如果已安装 Linux Pack，则返回“installed”。
host.software_version["oem_build_number"]	如果主机是 OEM 版本，则返回其修订号。
host.logging["syslog_destination"]	为 XenServer 系统记录程序获取或设置目标（空值表示本地日志记录）。
host.logging["multipathing"]	如果已在此主机上启用存储多路径，则为“true”。

键	语义
host.logging["boot_time"]	提供主机引导时间的浮点 Unix 时间。
host.logging["agent_start_time"]	提供控制域管理后台程序启动时间的浮点 Unix 时间。

7.3. VM

键	语义
VM.other_config["default_template"]	此模板是 Citrix 安装的一个模板。它用于选择性地在树视图中隐藏对象，为它们使用不同的图标，并禁止删除。
VM.other_config["xensource_internal"]	此模板是特殊模板，如 P2V 服务器模板。这些内容被 UI 完全隐藏。
VM.other_config["install_distro"] == "rhlike"	此模板适用于 RHEL 4.5、RHEL 5 或 CentOS 等效键。它用于在安装期间提示安装存储库，其中包括支持在 Miami 上从 ISO/CD 安装，以及修改 NFS URL 以适用于这些安装程序。
VM.other_config["install_distro"] in { "rhel41" | "rhel44" }	此模板适用于 RHEL 4.1、RHEL 4.4 或 CentOS 等效键。它用于在安装期间提示安装存储库，以及修改 NFS URL 以适用于这些安装程序。没有适用于这些模板的 ISO 支持。
VM.other_config["install_distro"] == "sleslike"	此模板适用于 SLES 10 和 SLES 9。它用于在安装期间提示安装存储库，这一点与 EL5 相同，但它不修改 NFS URL。XenServer 6.2.0 中提供对 SLES 10 的 ISO 支持。使用 *install-methods* 区分该平台上的 SLES 9 和 SLES 10。
VM.other_config["install-repository"] == "cdrom"	请求从 VM 的附加 CD 驱动器中的存储库（而不是 URL）安装。
VM.other_config["auto_poweron"]	获取或设置引导服务器时是否启动 VM，“true”或“false”。
VM.other_config["ignore_excessive_vcpus"]	获取或设置当 VM 所包含的 VCPU 多于其主机所包含的物理 CPU 时是否忽略 XenCenter 的警告，设置为 `true` 表示忽略。
VM.other_config["HideFromXenCenter"]	获取或设置 XenCenter 是否在树视图中显示 VM，设置为“true”表示隐藏。
VM.other_config["import_task"]	获取创建此 VM 的导入任务。
VM.HVM_boot_params["order"]	仅获取或设置 HVM VM 上的 VM 引导顺序，例如，“CDN”将按照以下顺序引导：首先引导磁盘，再引导 CD 驱动器，然后引导网络。

键	语义
VM.VCPU_params["weight"]	为 VM 的 VCPU 获取或设置 IONice 值，范围为 1 到 65536，65536 是最高值。
VM.pool_migrate(..., options['live'])	`true` 表示实时迁移。XenCenter 始终使用此键。
VM.other_config["install-methods"]	此模板中显示安装方法的以逗号分隔的列表。可以包 括“cdrom”、“nfs”、“http”或“ftp”。
VM.other_config["last_shutdown_time"]	上次关闭或重新启动此 VM 的时间，格式为 UTC ISO8601 日期时间。
VM.other_config["p2v_source_machine"]	通过 P2V 过程导入此 VM 所用的源计算机。
VM.other_config["p2v_import_date"]	通过 P2V 过程导入 VM 的日期。格式为 UTC ISO8601 日期时间。

7.4. SR

键	语义
SR.other_config["auto-scan"]	将自动扫描 SR 以查看更改。在 XenCenter 创建的所有 SR 上进行设置。
SR.sm_config["type"]	设置类型，例如将物理 CD 驱动器 SR 的类型设置为 `cd`。

7.5. VDI

键	语义
VDI.type	`user`（而不是 `system`）用于表示“如果将此磁盘连接到 VM，允许/不允许通过 GUI 删除 VDI”。目的是防止您损坏 VM（应将其卸载）。`suspend` 和 `crashdump` 分别记录挂起和内核转储。当前尚未使用 `ephemeral`。
VDI.managed	所有未管理的 VDI 都完全隐藏在 UI 中。它们是 VHD 链中的分支点或未使用的 LUN-per-VDI 磁盘。
VDI.sm_config["vmhint"]	此 VDI 支持的 VM 的 UUID。这是在通过用户界面创建 VDI 时设置的，目的是提高某些存储后端的性能。

7.6. VBD

键	语义
VBD.other_config["is_owner"]	如果已设置该键，则可能在卸载 VM 时删除此磁盘。

键	语义
VBD.other_config["class"]	设置为与 ionice 的“尽最大努力”设置相对应的整数。

7.7. 网络

键	语义
network.other_config["automatic"]	如果此键包含 `false` 以外的任何值，新 VM 向导将默认创建连接到此网络的 VIF。
network.other_config["import_task"]	获取创建此网络的导入任务。

7.8. VM_guest_metrics

键	语义
PV_drivers_version["major"]	获取 VM 的半虚拟化驱动程序版本的主要版本号。
PV_drivers_version["minor"]	获取 VM 的半虚拟化驱动程序版本的次要版本号。
PV_drivers_version["micro"]	获取 VM 的半虚拟化驱动程序版本的 Micro（内部版本号）。

7.9. 任务

键	语义
task.other_config["object_creation"] == "complete"	对于与 VM 导入相关的任务，将在创建所有对象（VM 和网络）后设置此标志。在导入 VM 向导中，这对于我们随后重新映射所有需要它的网络很有用。