Citrix XenServer ® 6.2.0 ソフトウェア開発キットガイド

発行 2013/08/27
1.0 版

Citrix XenServer ® 6.2.0 ソフトウェア開発キットガイド

Version： 6.2.0

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

免責

商標

Citrix ®
XenServer ®
XenCenter ®

CITRIX

目次

CITRIX

第1章 はじめに

XenServerの開発者ガイドへようこそ。このドキュメントには、XenServerが提供するSDK（Software Development Kit：ソフトウェア開発キット）を理解し、使いこなすために必要な情報が記載されています。この情報から、アーキテクチャの背景知識、基本となるAPI、提供されているツール、迅速に開発を軌道に乗せる方法を理解できます。

CITRIX

第2章 導入

XenServerにはXML-RPCベースのAPIが用意されており、より多くの管理機能やツールを利用できます。XenServer APIはローカルのXenServerホストだけでなく、リモートシステムからもコールできます。.未加工のXML-RPCコールで直接XenServerの管理APIを使用するアプリケーションを作成することも可能ですが、各APIコールを主要プログラミング言語の第一級関数として使用可能にする言語バインディングを使用すると、アプリケーションをより簡単に開発できるようになります。XenServer SDKでは、C、C#、Java、Python、およびPowerShell用の言語バインディングとサンプルコードが提供されます。

2.1. システム要件と準備

SDKを使用するには、まずXenServerをインストールする必要があります。無償版のCitrix XenServerは、http://www.citrix.com/downloads/xenserver/からダウンロードできます。XenServerのインストールについて詳しくは、『インストールガイド』を参照してください。インストールが完了したら、**ホストIPアドレス**と**ホストのパスワード**を書き留めておきます。

2.2. ダウンロード

このSDKはZIPファイルとしてパッケージ化されており、http://www.citrix.com/downloads/xenserver/から無償でダウンロードできます。

2.3. 新機能

XenServer 6.2.0 SDKでは、C、C#、Java、PowerShell、およびPython用の言語バインディングによるAPIが提供されます。このリリースには、以下のようないくつかの変更および拡張が施されています。

- **C、C#、Java、およびPowerShell用のバインディング**
  XenServer 4.0のサポートは削除されています。
- **Powershellバインディング**
  このリリースに含まれているPowershell言語バインディングには、多くのバグ修正が施された従来のXenServer Powershell v1.0のスナップインと、新たに再設計されたXenServer Powershell v2.0スナップインの2つのバージョンがあります。詳しくは、『XenServerリリースノート』を参照してください。

2.4. 構成内容

SDKのZIPファイルの内容は、XenServer-SDKディレクトリに含まれています。このディレクトリに、以下の内容が抽出されます。一部のサブディレクトリには、個別のREADMEファイルが収録されています。提供されるサンプルコードは、言語バインディングにより異なる場合があります。このため、ほかのバインディング用のサンプルコードも参照することをお勧めします。

- **XenServer-SDK**
  - **libxenserver**
    C言語用のXenServer SDK。
    - **bin**
      libxenserverのバイナリ。
    - **src**
      libxenserverのソースコードとサンプル、およびこれらをビルドするためのメイクファイル。
  - **XenServer.NET**
    C#.NET言語用のXenServer SDK。
    - **bin**
      XenServer.NETのバイナリ。

- **samples**
  XenServer.NETのサンプル。
- **src**
  XenServer.NETのソースコード。

- **XenServerJava**
  Java言語用のXenServer SDK。
  - **bin**
    Javaのバイナリ。
  - **javadoc**
    Javaのドキュメント。
  - **samples**
    Javaのサンプル。
  - **src**
    Javaのソースコード、およびソースコードとサンプルをビルドするためのメイクファイル。
- **XenServerPSSnapIn**
  新たに再設計されたPowerShell言語用のXenServer SDK。このディレクトリには、Xenserver PowerShell Windowsインストーラ**XenServerPSSnapIn.msi**が格納されています。
  - **samples**
    PowerShellのサンプルスクリプト。
  - **src**
    Xenserver PowerShellのソースコード。
- **XenServerPSSnapIn_old**
  従来のPowerShell言語用のXenServer SDK。このディレクトリには、Xenserver PowerShell Windowsインストーラ**XenServerPSSnapIn.msi**が格納されています。
  - **samples**
    PowerShellのサンプルスクリプト。
  - **src**
    Xenserver PowerShellのソースコード。
- **XenServerPython**
  このディレクトリには、XenServer Pythonモジュール**XenAPI.py**が格納されています。ライブラリ**provision.py**は、サンプルコードで使用されます。
  - **samples**
    Pythonを使用したXenServer APIのサンプル。

CITRIX

第3章 XenServer APIの概要

この章では、XenServer API（単に「API」と呼びます）および関連付するオブジェクトモデルについて紹介します。APIの主な機能は次のとおりです。

- **XenServerホストの管理**
  APIを使用して、仮想マシン、ストレージ、ネットワーク、ホストの設定、およびプールを管理できます。パフォーマンスと状態のメトリクスもAPIでクエリできます。

- **永続オブジェクトモデル**
  副次的な影響を持つすべての操作（オブジェクトの作成、削除、パラメータの変更など）の結果は、XenServerシステムによって管理されるサーバー側のデータベースで保持されます。

- **イベントメカニズム**
  永続（サーバー側）オブジェクトが変更されたときにAPIを通じて通知されるように、クライアントを登録することができます。これによって、同時に実行するクライアントによるデータモデルの変更をアプリケーションで追跡できるようになります。

- **同期および非同期の呼び出し**
  すべてのAPIコールを同期的に呼び出す、つまり完了までブロックすることができます。また、実行に時間がかかる可能性のあるAPIコールを**非同期的**に呼び出すこともできます。非同期コールは、**タスク**オブジェクトへのリファレンスを即座に返します。このタスクオブジェクトの進行状況や状態情報について、APIからクエリすることができます。非同期に呼び出された操作が完了すると、タスクオブジェクトから結果（またはエラーコード）を取得できます。

- **リモート実行が可能でクロスプラットフォームなAPIコール**
  APIコールを発行するクライアントは、管理対象のホストに常駐している必要はありません。また、APIを実行するためにsshでホストに接続する必要もありません。APIコールは、XML-RPCプロトコルを使用してネットワーク経由で要求と応答を伝送します。

- **安全かつ認証されたアクセス**
  ホスト上で実行するXML-RPC APIサーバーは、セキュアソケット接続を受け付けます。このため、クライアントはHTTPSプロトコルでAPIを実行できます。さらに、すべてのAPIコールは、サーバーでユーザー名とパスワードを確認した上で確立されるログインセッションのコンテキストで実行します。これにより、XenServerへの安全かつ認証されたアクセスが提供されます。

3.1. APIの使用方法

APIへの手引きとして、まずXenServer上で新しい仮想マシンを作成するために必要なコールと、仮想マシンの開始/一時停止/再開/停止サイクルについて説明します。ここでは特定の言語のコードには言及しません。この段階では、「インストールおよび開始」のタスクを遂行するRPC呼び出しの順序について簡単に説明します。

3.1.1. 認証：セッションリファレンスの取得

まず初めに、**`Session.login_with_password(`*`<username>`*`, `*`<password>`*`, `*`<client_API_version>`*`)`**をコールします。APIはセッションベースのため、ユーザーはほかのコールを実行する前に、サーバーで認証されている必要があります。ユーザー名とパスワードが正しく認証された場合、このコールの結果が**セッションリファレンス**になります。これ以降のAPIコールでは、パラメータとしてセッションリファレンスを指定します。これにより、正しく認証されたAPIユーザーのみがXenServer上で操作を実行できるようになります。

3.1.2. 新しい仮想マシンのインストールで使用するテンプレートの一覧の取得

次に、ホスト上の「テンプレート」一覧をクエリします。テンプレートは特別な仮想マシンオブジェクトで、さまざまなゲストオペレーティングシステムに適したデフォルトパラメータが指定されています（**xe**

template-listコマンドを実行すると、XenServer上にインストールされているすべてのテンプレートを確認できます）。APIでテンプレート一覧を取得するには、**is_a_template**フィールドがtrueに設定されている仮想マシンオブジェクトをサーバーで検索します。これを行うには、**VM.get_all_records(session)**をコールします。ここでsessionパラメータには、**Session.login_with_password**コールで取得したセッションリファレンスを指定します。このコールによりサーバーがクエリされ、すべての仮想マシンオブジェクトリファレンスとそれらのフィールド値のスナップショットが返されます。

（この段階では、返されたオブジェクトリファレンスとフィールド値を特定のクライアント言語で操作することを想定していません。詳細については、言語固有のAPIバインディングの章で具体的に説明します。ここでは、APIコールによって返されたオブジェクトとフィールド値を読み取って操作するためのメカニズムが存在することを理解しておいてください）

すべての仮想マシンオブジェクトのフィールド値のスナップショットをクライアントアプリケーションのメモリ内に取得できたので、これをさらに反復処理し、**is_a_template**フィールドがtrueの仮想マシンオブジェクトを検索できます。次に、このクライアントアプリケーションでさらにテンプレートオブジェクトを反復処理して、**name_label**フィールドの値が「Debian Etch 4.0」（XenServerのデフォルトのLinuxテンプレートの1つ）であるリファレンスを取得します。

3.1.3. テンプレートからの仮想マシンのインストール

引き続き同じ例を使用して説明します。前の手順で選択した「Debian Etch 4.0」テンプレートを使って、新しい仮想マシンをインストールします。このインストールには、2つのAPIコールが必要です。

- 最初に、APIコール**VM.clone(session, t_ref, "my first VM")**を使用します。これにより、新しい仮想マシンオブジェクトを作成するために、**t_ref**で参照される仮想マシンオブジェクトが複製されます。このコールの戻り値は、作成された仮想マシンの仮想マシンリファレンスです。これを**new_vm_ref**とします。
- この段階で、**new_vm_ref**が参照しているオブジェクトは、複製元の**t_ref**が参照している仮想マシンオブジェクトと同様に、まだテンプレートの状態です。**new_vm_ref**を仮想マシンオブジェクトにするには、**VM.provision(session, new_vm_ref)**をコールします。このコールが戻ると、**new_vm_ref**オブジェクトの**is_a_template**フィールドがfalseに設定されます。これは、**new_vm_ref**が、標準的な（起動可能な）仮想マシンであることを示しています。

このコールの実行中にテンプレートのディスクイメージが作成されるため、プロビジョニング操作に数分かかることがあることに注意してください。このDebianテンプレートの例では、この段階で、新たに作成されたディスクにDebianルートファイルシステムも実際に設定されます。

3.1.4. 仮想マシンの起動/一時停止/再開/停止のサイクル

新たにインストールされた仮想マシンのオブジェクトリファレンスを取得できたので、仮想マシンのライフサイクル操作も簡単に実行できます。

- 仮想マシンを起動するには、**VM.start(session, new_vm_ref)**をコールします。
- 実行中の仮想マシンを一時停止するには、**VM.suspend(session, new_vm_ref)**をコールします。
- 一時停止した仮想マシンを再開するには、**VM.resume(session, new_vm_ref)**をコールします。
- 仮想マシンを完全にシャットダウンするには、**VM.shutdown(session, new_vm_ref)**をコールします。

3.1.5. ログアウト

アプリケーションでのXenServerホスト操作が終了したら、**Session.logout(session)**をコールすることをお勧めします。このコールにより、セッションリファレンスが無効になり、後続のAPIコールで使用できないようになります。また、セッションオブジェクトを格納していたサーバー側のメモリがこのコールにより解放されます。

非アクティブなセッションは最終的にタイムアウトになりますが、サーバーの同時セッション数はハードコードにより200までに制限されています。この制限に達した後で新たなログインが発生すると、最も古いセッション

オブジェクトが削除され、オブジェクトに関連付けられていたセッションリファレンスが無効になります。このため、サーバーに同時にアクセスするほかのセッションで問題が発生しないように、複数のアプリケーションで単一のセッションを共有し（単一セッションを複数の異なるクライアント・サーバー間**ネットワーク接続**で使用できます）、操作が終了したら明示的にログアウトすることをお勧めします。

3.1.6. インストールと起動の例：概要

ここまで、APIを使用して、XenServerテンプレートから仮想マシンをインストールし、その上で多数のライフサイクル操作を実行する方法について説明しました。これらの操作を実行するために必要なコール数が少ないことに注目してください。

- セッションを取得するためのコール：**`Session.login_with_password()`**。
- XenServer上の仮想マシン（およびテンプレート）オブジェクトをクエリするためのコール：**`VM.get_all_records()`**。このコールで返された情報を使用して、インストールするテンプレートを選択しました。
- 選択したテンプレートから仮想マシンをインストールするための2つのコール：**`VM.clone()`**、続いて**`VM.provision()`**。
- 作成された仮想マシンを起動するコール：**`VM.start()`**同様に、仮想マシンを一時停止、再開、およびシャットダウンのための各コール）。
- ログアウトのためのコール：**`Session.logout()`**。

以上の説明で重要なのは、API自体は複雑で包括的な機能を備えていますが、一般的なタスク（仮想マシンの作成やライフサイクル操作など）の実行は非常に単純で、少数の簡単なAPIコールしか必要としない点です。次の節は最初は少々手ごわいかもしれませんが、この点を念頭に置いて学習を進めてください。

3.2. オブジェクトモデルの概要

ここでは、APIのオブジェクトモデルの概要について説明します。ここで概説する各クラスのパラメータとメソッドについて詳しくは、『Citrix XenServer Management API』を参照してください。

最初に、APIを構成するいくつかのコアクラスについて概要を説明します（最初に見たときに、これらの定義が抽象的に思えたとしても心配しないでください。後続の節の説明を読み、次の章でコードサンプルを手順ごとに確認していくことで、これらの概念を具体的に理解できるようになるはずです）。

VM	VM（仮想マシン）オブジェクトは、XenServerホストまたはリソースプール上の特定の仮想マシンインスタンスを表します。メソッドには、**`start`**、**`suspend`**、および**`pool_migrate`**などがあります。パラメータには、*`power_state`*、*`memory_static_max`*、および*`name_label`*などがあります（前節の例で、VMクラスで標準的な仮想マシンとテンプレートの両方を操作できることを学習しました）。
Host	Host（ホスト）オブジェクトは、XenServerプール内の物理ホストを表します。メソッドには、**`reboot`**、および**`shutdown`**などがあります。パラメータには、*`software_version`*、*`hostname`*、および（IP）*`address`*などがあります。
VDI	VDIオブジェクトは、**仮想ディスクイメージ**を表します。仮想マシンに仮想ディスクイメージを接続すると、仮想マシン内にブロックデバイスが現れます。これにより、仮想ディスクイメージによりカプセル化されたビットを読み書きできるようになります。VDIクラスのメソッドには、**`resize`**、および**`clone`**などがあります。フィールドには、**`virtual_size`**、**`sharable`**などがあります（前節の例で、仮想マシンテンプレート上で**`VM.provision`**をコールしたときに、新たに作成されたディスクを表すいくつかのVDIオブジェクトが自動的に作成され、VMオブジェクトに接続されています）。

SR	SR（**ストレージリポジトリ**）オブジェクトは、仮想ディスクイメージのコレクションを集約し、仮想ディスクイメージのビットを格納している物理ストレージのプロパティをカプセル化します。パラメータには、*type*（XenServerでストレージリポジトリの仮想ディスクイメージの読み取り/書き込みに使用するストレージ固有のドライバを指定します）、*physical_utilisation*などがあります。メソッドには、**scan**（ストレージ固有のドライバを呼び出して、ストレージリポジトリに含まれる仮想ディスクイメージの一覧と、これらの仮想ディスクイメージのプロパティを取得します）、**create**（仮想ディスクイメージを格納できるように、物理ストレージのブロックを初期化します）などがあります。
ネットワーク	Network（ネットワーク）オブジェクトは、XenServerホストのインスタンスが動作する環境に存在するレイヤ2ネットワークを表します。XenServerではネットワークを直接管理しないため、これは単に物理ネットワークトポロジと仮想ネットワークトポロジをモデル化するだけの軽量なクラスです。特定のNetworkオブジェクトにVIFおよびPIF（下記参照）で**接続された**VMオブジェクトおよびHostオブジェクトは、互いにネットワークパケットを送信できます。

これらのクラスについて完全に理解している場合は、次の章のコードの説明に進んでも構いません。既に説明したクラスの一部のみを使用して、多数の役立つアプリケーションを作成することができます。理論的にクラスの説明を理解したい場合は、このまま読み進めてください。

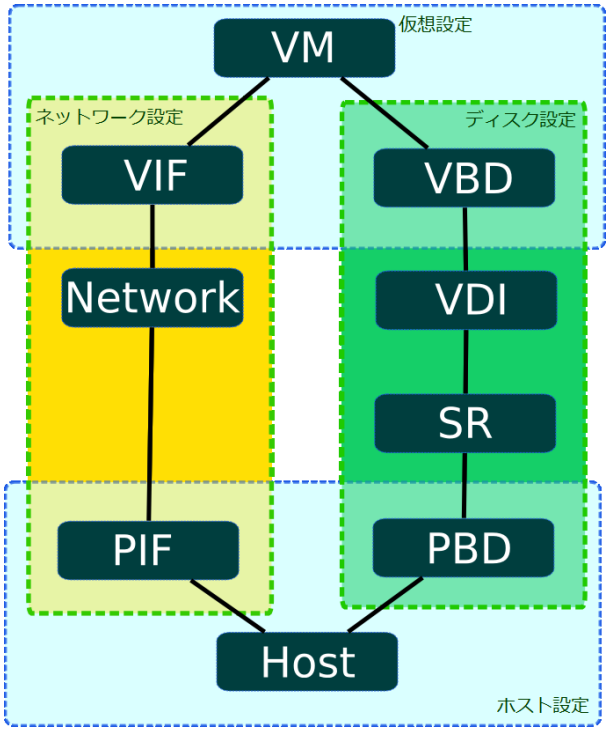
上記のクラスに加えて、**コネクタ**として動作するさらに4つのクラスがあります。これらのクラスは、VMオブジェクト、Hostオブジェクト、SRオブジェクト、およびNetworkオブジェクトの間の関係を指定します。これらのクラスの最初の2つが**VBD**（仮想ブロックデバイス）と**VIF**（仮想ネットワークインターフェイス）で、それぞれVMオブジェクトとVDIオブジェクト、およびVMオブジェクトとNetworkオブジェクトの接続形式を決定します。

VBD	VBD（**仮想ブロックデバイス**）オブジェクトは、仮想マシンと仮想ディスクイメージの間の接続を表します。仮想マシンが起動すると、そのVBDオブジェクトがクエリされ、接続するディスクイメージ（仮想ディスクイメージ）が決定されます。VBDクラスのメソッドには、**plug**（実行中の仮想マシンにディスクデバイスを**ホットプラグ**し、ディスクデバイスに含まれる指定された仮想ディスクイメージにアクセスできるようにします）、**unplug**（実行中の仮想マシンからディスクデバイスを**ホットアンプラグ**します）などがあります。フィールドには、**device**（指定された仮想ディスクイメージにアクセスできる、ゲスト内のデバイス名を決定します）などがあります。
VIF	VIF（**仮想ネットワークインターフェイス**）オブジェクトは、VMオブジェクトとNetworkオブジェクトの間の接続を表します。仮想マシンが起動すると、そのVIFオブジェクトがクエリされ、作成するネットワークデバイスが決定されます。VIFクラスのメソッドには、**plug**（実行中の仮想マシンにネットワークデバイスを**ホットプラグ**します）、**unplug**（実行中の仮想マシンからネットワークデバイスを**ホットアンプラグ**します）などがあります。

「コネクタクラス」の残りの2つは、HostオブジェクトとNetworkオブジェクト、およびHostオブジェクトとSRオブジェクトの接続形式を決定します。

PIF	PIF（**物理ネットワークインターフェイス**）オブジェクトは、Hostオブジェクトと Networkオブジェクトの間の接続を表します。ホストが（PIFを介して）Networkオブジェクトに接続している場合、指定されたホストからのパケットを送受信できるようになります。PIFクラスのフィールドには、**device**（PIFオブジェクトに対応するデバイス名。**eth0**など）、および**MAC**（PIFオブジェクトが表しているNICのMACアドレス）などがあります。PIFオブジェクトは、物理ネットワークインターフェイスとVLANの両方を表す点に注意してください（**VLAN**フィールドの値が正の整数であるインターフェイスがVLANです）。
PBD	PBD（**物理ブロックデバイス**）オブジェクトは、HostオブジェクトとSRオブジェクトの間の接続を表します。フィールドには、**currently-attached**（指定されたSRオブジェクトのストレージをホストで使用できるかどうか）、および**device_config**（指定されたホスト上で低レベルストレージデバイスを設定する方法を決定する、ストレージドライバ固有のパラメータを指定します。たとえば、NFSファイラで提供されるストレージリポジトリの場合、ファイラのホスト名と、ストレージリポジトリファイルを格納するファイラ上のパスを指定できます）などがあります。

図 3.1. 代表的なAPIクラス



仮想マシン、ホスト、ストレージ、およびネットワークを管理するためのAPIクラスの概要

図 3.1.「代表的なAPIクラス」は、仮想マシン、ホスト、ストレージ、およびネットワークの管理に関係するAPIクラスの概要を示しています。この図から、ストレージとネットワークの設定そして仮想マシンとホストの設定に、対称性があることがわかります。

3.3. 仮想ネットワークインターフェイスおよび仮想ブロックデバイスの操作

ここでは、より複雑なシナリオを例にして、仮想ストレージとネットワークデバイスに関連するさまざまなタスクをAPIを使って実行する方法について説明します。

3.3.1. ディスクの作成と仮想マシンへの接続

まず、新しい空のディスクイメージを作成し、実行中の仮想マシンにそのディスクイメージを接続します。既に実行中の仮想マシンがあり、そのAPIオブジェクトリファレンスがわかっている（たとえば、前出の節で説明した手順で仮想マシンを作成し、サーバーがそのリファレンスを返している）とします。また、XenServerで認証済みで、セッションリファレンスを取得済みであるとします。説明を簡潔にするために、この章の残りの部分ではセッションに関する説明を省略します。

3.3.1.1. 新しい空のディスクイメージの作成

まず初めに、物理ストレージ上でディスクイメージのインスタンスを作成します。これを行うには、**`VDI.create()`**をコールします。**`VDI.create`**コールでは、多くのパラメータを指定できます。たとえば、以下のパラメータがあります。

- **`name_label`**および**`name_description`**：人間が判読できる、ディスクの名前と説明です（ユーザーインターフェイスでの表示用など）。これらのフィールドは、特に必要がなければ空白のまま残してもかまいません。
- **`SR`**：仮想ディスクイメージのビットを格納する物理ストレージを表す、ストレージリポジトリのオブジェクトリファレンスです。
- **`read_only`**：このフィールドをtrueに設定すると、仮想ディスクイメージは読み取り専用形式で***のみ***仮想マシンに接続できます（**`read_only`**フィールドがtrueの仮想ディスクイメージを読み取り/書き込み形式で接続しようとすると、エラーが返されます）。

`VDI.create`コールを呼び出すと、XenServerで物理ストレージ上に空のディスクイメージが作成されます。さらに、関連付けられたVDIオブジェクト（物理ストレージ上のディスクイメージを参照するデータモデルインスタンス）が作成され、このVDIオブジェクトへのリファレンスが返されます。

物理ストレージ上のディスクイメージを表す方法は、作成した仮想ディスクイメージの格納先ストレージリポジトリの種類によって異なります。たとえば、ストレージリポジトリの種類がlvmの場合、新しいディスクイメージはLVMボリュームとして表されます。ストレージリポジトリの種類がnfsの場合、新しいディスクイメージはNFSファイラ上に作成されるスパースなVHDファイルとして表されます（APIで**`SR.get_type()`**コールを使用すると、ストレージリポジトリの種類をクエリできます）。

> **注：**
>
> ストレージリポジトリの種類によっては、設定したブロックサイズで分割できるように*`virtual-size`*の値が切り上げられる可能性があります。

3.3.1.2. ディスクイメージの仮想マシンへの接続

この時点で、実行中の仮想マシンと、作成したばかりの新しい仮想ディスクイメージが存在しています。これらはどちらもXenServerホスト上に存在する独立したオブジェクトで、まだ関連付けられていません。そこで、次にこれらの仮想ディスクイメージと仮想マシンを関連付けるリンクを作成します。

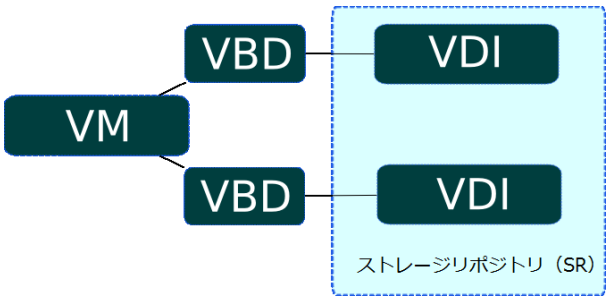
接続は、VBD（**仮想ブロックデバイス**）オブジェクトと呼ばれる新しい「コネクタ」オブジェクトを作成することで確立されます。VBDオブジェクトを作成するには、**`VBD.create()`**コールを呼び出します。**`VBD.create()`**コールでは、多くのパラメータを指定できます。たとえば、以下のパラメータがあります。

- **`VM`**：仮想ディスクイメージを接続する仮想マシンのオブジェクトリファレンスです。
- **`VDI`**：接続する仮想ディスクイメージのオブジェクトリファレンスです。
- **`mode`**：仮想ディスクイメージを読み取り専用形式または読み取り/書き込み形式のどちらで接続するかを指定します。
- **`userdevice`**：ゲスト内のブロックデバイスを指定します。仮想マシン内で実行するアプリケーションは、ブロックデバイスを通じて仮想ディスクイメージのビットを読み書きできます。

- **type**：仮想ディスクイメージを仮想マシン内で標準的なディスクとして表示するか、CDとして表示するかを指定します（このフィールドは、Linux仮想マシンよりWindows仮想マシンでより重要な意味を持ちますが、この章では詳細について割愛します）。

VBD.createを呼び出すと、XenServer上にVBDオブジェクトが作成され、そのオブジェクトリファレンスが返されます。ただし、このコール自体が実行中の仮想マシンに副次的な影響を与えることはありません（実行中の仮想マシン内にブロックデバイスは作成されません）。VBDオブジェクトが存在していてもゲスト内のブロックデバイスがアクティブでないということは、VBDオブジェクトの**currently_attached**フィールドがfalseに設定されているということです。

図 3.2. 2つのVDIオブジェクトが関連付けられたVMオブジェクト



2つのVDIオブジェクトが関連付けられたVMオブジェクト

図 3.2.「2つのVDIオブジェクトが関連付けられたVMオブジェクト」は、VM、VBD、VDI、およびSRの各オブジェクトの間の関係を示しています。この例では、VMオブジェクトに2つのVDIオブジェクトが接続されています。また、VMオブジェクトとそのVDIオブジェクトの間の接続を形成する2つのVBDオブジェクトがあり、VDIオブジェクトは同じSRオブジェクト内に存在しています。

3.3.1.3. 仮想ブロックデバイスのホットプラグ

この段階で仮想マシンを再起動すると、VBDオブジェクトに対応するブロックデバイスが現れます。XenServerは、仮想マシンの起動時にすべての仮想ブロックデバイスをクエリし、対応する各仮想ディスクイメージをアクティブに接続します。

仮想マシンの再起動は非常に良い手段ですが、ここでは**実行中**の仮想マシンに新しい空のディスクを接続しようとしていることを思い出してください。これは、新しいVBDオブジェクトの**plug**メソッドを呼び出すことで達成できます。**plug**コールが正常終了すると、VBDオブジェクトが関連付けられているブロックデバイスが実行中の仮想マシン内部に現れます。これは実行中の仮想マシンから見ると、新しいディスクデバイスが**ホットプラグ**されたとゲストオペレーティングシステムに認識させることになります。このことをAPI側から見ると、VBDオブジェクトの**currently_attached**フィールドがtrueに設定されるということになります。

当然のことながら、VBDオブジェクトの**plug**メソッドには、対になる**unplug**メソッドがあります。VBDオブジェクトの**unplug**メソッドを呼び出すと、実行中の仮想マシンからブロックデバイスが**ホットアンプラグ**され、それに応じて、VBDオブジェクトの**currently_attached**フィールドがfalseに設定されます。

3.3.2. ネットワークデバイスの作成と仮想マシンへの接続

仮想マシンの仮想ネットワークインターフェイスの設定に関連するAPIコールは、多くの面で仮想ディスクデバイスの設定に関連するコールに似ています。そのため、ここでは、APIオブジェクトモデルを使用してネットワークインターフェイスを作成する方法について詳細に説明することはせず、代わりに、仮想**ネットワークデバイス**の設定と仮想**ストレージデバイス**の設定の対称性について簡単に概説します。

ネットワークにおいて、VBDクラスに相当するのがVIFクラスです。APIにおいて、VBDオブジェクトが仮想マシン内のブロックデバイスを表現するものであるのと同様に、VIFオブジェクト（**仮想ネットワークインターフェイス**）は仮想マシン内のネットワークデバイスを表現するものです。VBDオブジェクトがVMオ

ブジェクトとVDIオブジェクトを関連付けるのに対し、VIFオブジェクトはVMオブジェクトとNetworkオブジェクトを関連付けます。VBDオブジェクトと同様に、VIFオブジェクトにも、VIFオブジェクトに関連付けられている（ゲスト内の）ネットワークデバイスが現在アクティブなのか非アクティブなのかを決定する、**currently_attached**フィールドがあります。仮想マシンの起動時にその仮想マシンのVIFオブジェクトがクエリされ、各VIFオブジェクトに対応するネットワークデバイスが仮想マシン内で作成される点もVBDオブジェクトに似ています。同様に、VIFオブジェクトにも、実行中の仮想マシンでネットワークデバイスをホットプラグまたはホットアンプラグするための**plug**メソッドと**unplug**メソッドがあります。

3.3.3. ネットワークとストレージのためのホストの設定

ここまでで、仮想マシン内でブロックデバイスとネットワークデバイスの設定を管理するために、それぞれVBDクラスとVIFクラスを使用することについて学習しました。ホストのストレージとネットワークを管理するには、これらに類似する2つのクラスを使用します。PBD（**物理ブロックデバイス：Physical Block Device**）とPIF（**物理ネットワークインターフェイス：Physical [network] InterFace**）です。

3.3.3.1. ホストのストレージ設定：PBD

まず、PBDクラスから説明します。**PBD_create()**コールでは、以下のようなパラメータを指定できます。

パラメータ	説明
host	物理ブロックデバイスを使用できる物理コンピュータ
SR	物理ブロックデバイスが接続するストレージリポジトリ
device_config	ホストのストレージリポジトリ用バックエンドドライバに提供される文字列マップで、物理ストレージデバイスの設定に必要な低レベルパラメータを含みます。このデバイス上でストレージリポジトリが認識されます。*device_config*フィールドの内容は、物理ブロックデバイスが接続するストレージリポジトリの種類によって異なります（**xe sm-list**コマンドを実行すると、ストレージリポジトリの種類の一覧を確認できます。この一覧の**configuration**フィールドによって、各種のストレージリポジトリで指定できる*device_config*パラメータの値が決定されます）。

たとえば、種類がnfs（仮想ディスクイメージがVHDファイルとして格納されるNFSファイラ上のディレクトリ）のSRオブジェクト**s**があり、Hostオブジェクト**h**から**s**にアクセスできるようにするとします。この場合、**PBD.create()**を呼び出して、ホスト**h**、ストレージリポジトリ**s**、および**device_config**パラメータの値（以下のマップ）を指定します。

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

このパラメータは、ストレージリポジトリ**s**がホスト**h**上でアクセス可能であり、さらに、ストレージリポジトリ**s**にアクセスするために、NFSサーバー`my_nfs_server.xensource.com`上のディレクトリ`/scratch/mysrs/sr1`をマウントする必要があることをXenServerホストに通知します。

VBDオブジェクトと同様に、PBDオブジェクトにもアクティブなのか非アクティブなのかを決定する**currently_attached**フィールドがあります。**PBD.plug**メソッドおよび**PBD.unplug**メソッドを呼び出すことで、ストレージリポジトリを特定のホストに接続したり切断したりすることができます。

3.3.3.2. ホストのネットワーク設定：PIF

ホストのネットワーク設定には、PIFオブジェクトを使用します。PIFオブジェクトがNetworkオブジェクト**n**をHostオブジェクト**h**に接続する場合、**n**に対応するネットワークは、PIFオブジェクトのフィールドで指定され

る物理ネットワークインターフェイス（または、VLANタグを加えた物理ネットワークインターフェイス）にブリッジされます。

たとえば、Hostオブジェクト**h**をNetworkオブジェクト**n**に接続するPIFオブジェクトが存在し、PIFオブジェクトの**device**フィールドが`eth0`に設定されているとします。これは、ネットワーク**n**上のすべてのパケットが、ホストのネットワークデバイス`eth0`の物理NICにブリッジされることを意味します。

3.4. 仮想マシンのエクスポートとインポート

仮想マシンをファイルにエクスポートし、それをXenServerホストにインポートすることができます。エクスポートプロトコルはシンプルなHTTP（S） GETであり、仮想マシンがプールのメンバ上にある場合はマスタ上で実行する必要があります。標準HTTP基本認証で認証するか、セッションリファレンスを取得済みの場合はそのセッションを使用できます。エクスポートする仮想マシンを指定するには、UUIDまたはリファレンスを使用します。エクスポートを追跡するには、タスクを作成し、そのリファレンスを使用できます。仮想マシンのディスクにプールメンバからのみアクセスできる場合は、要求がリダイレクトされる可能性があります。

次の引数がコマンドラインで渡されます。

引数	説明
session_id	認証に使用するセッションのリファレンス。HTTP基本認証を使用しない場合にのみ必要です。
task_id	操作を追跡するタスクオブジェクトのリファレンス。オプションであり、エクスポートを追跡するタスクオブジェクトを作成した場合にのみ必要です。
ref	仮想マシンのリファレンス。UUIDを使用しない場合にのみ必要です。
uuid	仮想マシンのUUID。リファレンスを使用しない場合にのみ必要です。

次は、LinuxコマンドラインツールcURLを使用する例です。

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

このコマンドにより、指定した仮想マシンがファイル`exportfile`にエクスポートされます。

メタデータのみをエクスポートするには、URIとして`http://server/export_metadata`を使用します。

インポートプロトコルも同様で、HTTP（S） PUTが使用されます。`session_id`および`task_id`引数はエクスポートの場合と同じです。`ref`および`uuid`は使用しません。仮想マシンンのリファレンスとUUIDは新たに生成されます。次の追加パラメータがあります。

引数	説明
restore	**true**の場合、元の仮想マシンの置き換え処理としてインポートが扱われます。VIFオブジェクト上のMACアドレスがエクスポート時と変わらないため、元の仮想マシンが実行中の場合は競合が発生します。
force	**true**の場合、チェックサムエラーが無視されます（デフォルトでは、チェックサムエラーが検出された場合に仮想マシンが破棄されます）。
sr_id	仮想マシンのインポート先のSRオブジェクトのリファレンス。デフォルトでは、**Pool.default_SR**にインポートされます。

もう一度、cURLを使用する例を次に示します。

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

このコマンドにより、サーバー上のデフォルトのストレージリポジトリに仮想マシンがインポートされます。

> **注：**
>
> sr_uuidを指定せず、かつデフォルトのストレージリポジトリが未設定の場合、「DEFAULT_SR_NOT_FOUND」というエラーメッセージが返されます。

もう1つ例を示します。

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

このコマンドにより、サーバー上の指定されたストレージリポジトリに仮想マシンがインポートされます。

メタデータのみをインポートするには、URIとして`http://server/import_metadata`を使用します。

3.4.1. Xen仮想アプライアンス（XVA）仮想マシンのインポート形式

XenServerでは、人間が判読できる従来の仮想マシン入力形式であるXVAがサポートされます。ここでは、XVAの構文と構造について説明します。

XVAは、XMLメタデータとディスクイメージのセットを含んでいる単一のディレクトリで構成されます。XVA形式の仮想マシンは、直接実行するためのものではありません。XVAパッケージ内のデータは圧縮されており、永続ストレージにアーカイブするか、パッケージを展開して実行できる仮想マシンサーバー（XenServerホストなど）に転送することを目的としています。

XVAはハイパーバイザーの種類に依存しないパッケージ形式なので、ほかのプラットフォームでXVA仮想マシンのインスタンスを作成する、簡単なツールを作成できます。XVAでは特定の実行時形式を指定しません。たとえば、ファイルイメージ、LVMボリューム、QCoWイメージ、VMDKイメージ、またはVHDイメージとしてディスクのインスタンスを作成できます。XVA仮想マシンは何度でもインスタンス化でき、そのたびに異なる実行時形式にすることができます。

XVAでは次のことを行いません。

- シリアル化またはトランスポートの特定の形式を指定する。
- 仮想マシン（またはテンプレート）のインストール時のカスタマイズ用メカニズムを提供する。
- インストール後に仮想マシンをアップグレードする方法に対応する。
- アプライアンスとして動作する複数の仮想マシンが通信する方法を定義する。

これらの問題は、関連するOpen Virtual Applianceの仕様で対処されます。

XVAは単一のディレクトリであり、このディレクトリには必ず`ova.xml`と呼ばれるファイルが含まれています。このファイルには、XVAに含まれる仮想マシンが記述されます。詳しくは、第3.2節を参照してください。ディスクはサブディレクトリに格納され、ova.xmlから参照されます。ディスクデータの形式については、第3.3節を参照してください。

この章の残りのページでは、次の用語を使用します。

- HVM：ハードウェアの仮想化サポートを活用して、オペレーティングシステムのカーネルを変更しないで実行するモード。
- PV：ハードウェアの仮想化サポートが不要な、特別に変更された「準仮想化」カーネルをハイパーバイザー上で明示的に実行するモード。

ova.xmlファイルには次の要素が含まれます。

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

version属性の値は、XVAの仕様のバージョンを示します。この場合は0.1です。<appliance>要素には<vm>要素が1つだけ存在します（Open Virtual Applianceの仕様では、複数の<vm>要素を指定できます）。

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

各<vm>要素により単一の仮想マシンが定義されます。name属性は将来的な内部使用を目的としており、ova.xmlファイル内で固有である必要があります。name属性の値として、任意の有効なUTF-8文字列を設定できます。各<vm>要素には次の必須要素があります。

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

ユーザーインターフェイスに表示される仮想マシンの短い名前です。

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

ユーザーインターフェイスに表示される仮想マシンの説明です。<label>要素および<shortdesc>要素では、文字列の先頭と末尾の空白が無視されます。

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

<config>要素には、仮想マシンのメモリサイズをバイト単位で表すmem_set属性と、CPU数を表すvcpus属性があります。

各<vm>要素にはブロックデバイスを表す<vbd>要素が含まれます。この要素の有無や数は任意で、次の形式で記述されます。

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

次の属性があります。

device　仮想マシンに認識される物理デバイスの名前です。Linuxゲストにはsd[a-z]、Windowsゲストにはhd[a-d]を使用します。

function　値がrootの場合は、このディスクを使用してゲストを起動します（Linuxのルート、つまり「/」のファイルシステムが存在するという意味ではないことに注意してください）。rootが設定されたデバイスは1つのみである必要があります。仮想マシンの起動については、第3.4節を参照してください。ほかの文字列は無視されます。

mode　デバイスが読み取り/書き込み可能である場合の値はw、読み取り専用の場合の値はroです。

vdi　ブロックデバイスが接続するディスクイメージの名前です（<vdi>要素で表わされます）。

各<vm>要素に、<hacks is_hvm="false" kernel_boot_cmdline="root=/dev/sda1 ro"/>のような<hacks>要素が含まれる場合があります。この<hacks>要素はXenServerで生成されるXVAファイルに存在しますが、将来削除される予定です。is_hvm属性には、仮想マシンをHVMモードで起動する必要があるかどうかを示すtrueまたはfalseが指定されます。kernel_boot_cmdline属性には、pygrubでゲストを起動するときの追加のカーネルコマンドライン引数が含まれます。

<appliance>要素には、<vm>要素のほか、<vdi>要素が含まれます。この要素の有無や数は任意で、次の形式で記述されます。

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

各<vdi>要素により、単一のディスクイメージが定義されます。次の属性があります。

- name：仮想ディスクイメージの名前で、<vbd>要素のvdi属性により参照されます。任意の有効なUTF-8文字列を設定できます。
- size：必要なイメージのバイト単位のサイズです。

- source：イメージデータの場所を記述するURIです。現在設定できるのは「file://」で始まるURIのみで、ova.xmlを格納するディレクトリからの相対パスである必要があります。
- ディスクデータの形式です（第3.3節を参照してください）。

type属性の値が「dir-gzipped-chunks」の<vdi>要素により、単一ディスクイメージエンコーディングが定義されます。各イメージは、次のような一連のファイルを含むディレクトリによって表わされます。

```
-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
```

chunk-XXXXXXXXX.gzという名前の各ファイルは、1e9バイト（1GB。1GiBではない）のローブロックデータを含んだGZIPファイルです。一部のファイルシステムでのファイルサイズの上限を超えないように、この小さなサイズが選択されています。ファイルを展開して結合すると、元のイメージが復元されます。

XenServerでは2つのメカニズムで仮想マシンを起動できます。1つはpygrubで抽出された準仮想化カーネルを使用する方法で、もう1つはHVMモードを使用する方法です。現在の実装では、<hacks>要素のis_hvm属性で指定されたメカニズムが使用されます。

この節の残りのページでは、XVAとしてパッケージ化されたシンプルなDebian仮想マシンを例にして説明します。この仮想マシンには2つのディスクがあります。1つは5,120MiBのルートファイルシステムで、pygrubでゲストを起動するために使用されます。もう1つは512MiBで、スワップに使用されます。メモリは512MiBで、仮想CPUを1つ使用します。

このDebian仮想マシンの最上位ディレクトリは、次のように表示されます。

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

この最上位ディレクトリ内には、次のように、各ディスク用の2つのサブディレクトリと、単一のova.xmlファイルがあります。

```
$ 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
```

各ディスクのサブディレクトリには、GZIPで圧縮された1GBのローディスクブロックであるいくつかのファイルがあります。

```
$ 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
```

このDebian仮想マシンのova.xmlには、次のような内容が記述されます。

```
<?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に関する注意事項

3.5.1. datetime

このAPIにおけるdatetimeの制御方法は、XML-RPC仕様に従いません。APIでは、datatime文字列の後に「Z」が付加されます。これにより、時刻がUTC形式であることを示します。

3.6. 参考資料

この章では、APIとそのオブジェクトモデルについての概要を学習しました。この章の目的は、APIの詳細を説明することではなく、次の章のコードサンプルを読解し、さらに『Citrix XenServer Management API』の詳細な情報を理解するための背景知識を提供することです。

より詳細な情報は、さまざまな場所から入手できます。

- 『XenServer管理者ガイド』には、**xe** CLIの概要が記載されています。多くの**xe**コマンドではAPIがそのまま使用されるため、**xe**を使用することは、この章で説明したAPIオブジェクトモデルを理解する手がかりになります。
- 次の章のコードサンプルでは、さまざまなクライアント言語でのAPIコーディングの具体例について説明します。
- 『Citrix XenServer Management API』では、プログラミング上のAPIの動作とXML/RPCメッセージの形式についてより詳しく説明します。
- XenServerホストのdom0自体に、APIを使用するスクリプトが含まれています。たとえば、/opt/xensource/libexec/shutdownは、仮想マシンを完全にシャットダウンするPythonプログラムです。このスクリプトは、ホスト自体をシャットダウンするときに呼び出されます。

CITRIX

第4章 APIの使用

この章では、実際のプログラムでXenServer管理APIを使用してXenServerホストおよび仮想マシンを管理する方法について説明します。まず、典型的なクライアントアプリケーションについて順を追って説明し、APIを使用して一般的なタスクを実行する方法を具体的に示します。サンプルコードではPythonの構文が用いられていますが、ほかのプログラミング言語でも同様の方法を使用できます。言語バインディング自体については後述します。章の最後では、2つの完全な例について解説します。

4.1. 典型的なアプリケーションの構造

ここでは、XenServer管理APIを使用する典型的なアプリケーションの構造について説明します。大半のクライアントアプリケーションは、XenServerホストに接続し（ユーザー名とパスワードを使用するなどして）認証を受ける処理から始まります。認証に成功すると、サーバーで「セッション」オブジェクトが作成され、クライアントへリファレンスが返されます。このリファレンスは引数として、それ以降のすべてのAPIコールに渡されます。いったん認証されると、クライアントでほかのオブジェクト（XenServerホストや仮想マシンなど）へのリファレンスを検索して、それらに対する操作を呼び出すことができます。操作は、同期的または非同期的に呼び出すことができます。非同期操作の場合は、特殊なタスクオブジェクトが操作の状態と進行状況を表します。これらのアプリケーション要素については、次の節で詳しく説明します。

4.1.1. 低レベルトランスポートの選択

APIコールは、2つのトランスポートで実行できます。

- IPネットワークを介したポート443（HTTPS）上のSSL暗号化TCPトランスポート
- ローカルUNIXドメインソケット`/var/xapi/xapi`を介したプレーンテキストトランスポート

SSL暗号化TCPトランスポートはホスト外部からのすべてのトラフィックで使用され、UNIXドメインソケットはXenServerホスト上で実行されているサービスで使用されます。SSL暗号化TCPトランスポートでは、すべてのAPIコールをプールマスタに送信する必要があります。送信できないと、*`HOST_IS_SLAVE`*エラーが返されます。このエラーには、パラメータとしてマスタのIPアドレスが含まれます。

高可用性が有効なプールなどでは、プールマスタとして動作するホストが変更される場合があります。したがって、クライアントでは次のステップを実装して、プールマスタの変更を検出し、必要に応じて新しいマスタに接続するようにしてください。

プールマスタの変更に対処するには

1. ホストサーバー一覧の更新をサブスクライブし、プール内のホストの現在の一覧を保持します。
2. プールマスタへの接続に失敗した場合は、応答があるまで一覧内のすべてのホストへの接続を試行します。
3. 最初に応答するホストから`HOST_IS_SLAVE`エラーが返されます。このメッセージには、（そのホストが新しいマスタでない場合）新しいプールマスタの識別情報が含まれます。
4. 新しいプールマスタに接続します。

> **注：**
>
> 例外として、UNIXドメインソケットを介して送信されるすべてのメッセージは、正しいホストに透過的に転送されます。

4.1.2. 認証とセッションの処理

多くのAPIコールは、第1パラメータとしてセッションリファレンスを取ります。有効なリファレンスを指定しないと、*`SESSION_INVALID`*エラーが返されます。セッションリファレンスは、**`login_with_password`**関数にユーザー名とパスワードを指定して取得します。

注：

例外として、ローカルUNIXドメインソケットを介してコールが実行された場合、ユーザー名とパスワードは無視され、常にコールは成功します。

各セッションにはタイムスタンプのlast activeフィールドがあり、これはAPIコールごとに更新されます。XenServerには、アクティブなセッション数の上限を200とする制限が組み込まれています。この制限を越えると、last activeフィールドの値が最も古いセッションが削除されます。さらに、last activeフィールドの値が24時間より古いセッションもすべて削除されます。したがって、次の重要事項を守る必要があります。

- セッションリークを防ぐために、必ずアクティブなセッションからログアウトする。
- *SESSION_INVALID*エラーが返された場合にサーバーに再度ログインするようにする。

次のサンプルコードでは、UNIXドメインソケットを介した接続が確立され、セッションが作成されます。

```
import XenAPI

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

4.1.3. オブジェクトリファレンスの検索

アプリケーションが認証されたら、次に、オブジェクトの状態をクエリしたりオブジェクトに対する操作を呼び出したりするために、オブジェクトへのリファレンスを取得します。すべてのオブジェクトに、次のような「暗黙」のメッセージセットがあります。

- **get_by_name_label**：特定のラベルを持つ特定のクラスのすべてのオブジェクトの一覧を返します。
- **get_by_uuid**：UUIDによって特定される単一のオブジェクトを返します。
- **get_all**：特定のクラスのすべてのオブジェクトへのリファレンスセットを返します。
- **get_all_records**：特定のクラスの各オブジェクトのレコードへのリファレンスマップを返します。

たとえば、すべてのホストを一覧するには、次のように指定します。

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

「my first VM」という名前のすべての仮想マシンを検索するには、次のように指定します。

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

注：

オブジェクトの*name_label*フィールドの値は固有であるとは限らないため、**get_by_name_label** APIコールは、単一のリファレンスではなくリファレンスセットを返します。

上記のオブジェクト検索メソッドに加え、ほとんどのオブジェクトでも、ほかのオブジェクトへのリファレンスがフィールドに含まれています。たとえば、次のコールによって、特定のホスト上で実行中の仮想マシンのセットを検索できます。

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

4.1.4. オブジェクトに対する同期操作の呼び出し

オブジェクトリファレンスを取得したら、それらに対する操作を呼び出すことができます。たとえば、仮想マシンを開始するには、次のように指定します。

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

すべてのAPIコールはデフォルトでは同期的に呼び出され、操作が完了または失敗するまで戻りません。たとえば、**VM.start**は、仮想マシンが起動を開始するまで戻りません。

> **注：**
>
> **VM.start**コールが戻るときには、仮想マシンが起動を開始しています。いつ起動が終了したかを判別するには、ゲスト内エージェントが**VM_guest_metrics**オブジェクトで内部統計情報を報告するまで待機します。

4.1.5. タスクを使用した非同期操作の管理

時間のかかる操作（**VM.clone**や**VM.copy**など）の管理を簡素化するため、関数を同期（デフォルト）と非同期の2通りの形式で使用できます。各非同期関数は、進行中の操作に関する情報を含む、以下のリファレンスを返します。

- 保留状態かどうか。
- 成功したか、失敗したか。
- 進行状況（0から1の範囲で表します）。
- 操作によって返される結果やエラーコード。

アプリケーションで**VM.clone**操作の進行状況を追跡し、進行状況バーを表示するには、次のようなコードを使用します。

```
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：**
>
> いわゆる「行儀のよい」クライアントアプリケーションを作成するには、結果またはエラーの読み取りが完了したら、非同期操作により作成されたタスクを必ず削除する必要があります。組み込みの上限値をタスク数が超えた場合、最も古い完了済みのタスクがサーバーにより削除されます。

4.1.6. イベントのサブスクライブおよびリッスン

タスクおよび測定値クラスを除き、オブジェクトが変更されるたびに、サーバーでイベントが生成されます。クライアントは、頻繁にポーリングする代わりにクラス単位でこのイベントストリームをサブスクライブし、更新を受信できます。イベントには3つの種類があります。

- *add*：オブジェクトが作成されたときに生成されます。
- *del*：オブジェクトが破棄される直前に生成されます。
- *mod*：オブジェクトのフィールドが変更されたときに生成されます。

イベントには、単調増加ID、オブジェクトクラスの名前、および**get_record()**の結果に相当するオブジェクト状態のスナップショットも含まれます。

クライアントは、クラス名一覧または特殊文字の「*」を指定して**event.register()**をコールすることで、イベントを登録します。また、イベントが入手可能になるまで待機して新しいイベントを返す**event.next()**を実行することで、イベントを受信します。

注：

サーバー上の生成済みイベントのキューの長さは限定されているため、非常に低速なクライアントでは十分な速さでイベントを読み取れない場合があります。その場合は、EVENTS_LOSTエラーが返されます。クライアントには、この種の状況に対処する用意が必要です。そのためには、イベントを再登録する処理と、待機している条件が未登録の間に真になっていないことを確認する処理を追加します。

次のサンプルPythonコードでは、システムによって生成された各イベントの概要が出力されます（同様のコードがXenserver-SDK/XenServerPython/samples/watch-all-events.pyに含まれています）。

```
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. 言語バインディング

4.2.1. C

SDKのXenServer-SDK/libxenserver/srcディレクトリに、C言語バインディングのソースと、バインディングをライブラリにコンパイルするメイクファイルが収録されています。各APIオブジェクトは、そのオブジェクトのすべてのAPI関数の宣言が含まれているヘッダーファイルに関連付けられています。たとえば、仮想マシンの操作を呼び出すために必要な型定義と関数は、すべてxen_vm.hに含まれています。

Cバインディングの依存関係

サポートされるプラットフォーム：	Linux
ライブラリ：	言語バインディングは、Cプログラムによってリンクされるlibxenserver.soとして生成されています。
依存関係：	• XMLライブラリ（GNU Linux上のlibxml2.so） • Curlライブラリ（libcurl2.so）

Cバインディングの以下の単純なサンプルコードが収録されています。

- test_vm_async_migrate：非同期APIコールを使用して、実行中の仮想マシンをメンバホストからプールマスタに移行します。
- test_vm_ops：ホスト機能をクエリして、仮想マシンを作成し、新しい空のディスクイメージを仮想マシンに接続した後、さまざまな電源操作を実行します。
- test_failures：エラー文字列をenum_xen_api_failureに変換したり、その逆を行ったりします。
- test_event_handling：接続上のイベントをリスンします。
- test_enumerate：さまざまなAPIオブジェクトを列挙します。

4.2.2. C#

C#バインディングはXenServer-SDK/XenServer.NETディレクトリに収録されています。Microsoft Visual Studioでのビルドに適したプロジェクトファイルも提供されています。各APIオブジェクトは、1つのC#ファイルに関連付けられています。たとえば仮想マシンの操作を実装する関数は、VM.csファイルに含まれています。

C#バインディングの依存関係

サポートされるプラットフォーム：	.NET Framework Version 2.0をインストールしたWindows
ライブラリ：	言語バインディングは、C#プログラムによってリンクされるダイナミックリンクライブラリであるXenServer.dllとして生成されています。
依存関係：	XenServer.dllとXML-RPCサーバーの間の通信を可能にするために、CookComputing.XMLRpcV2.dllが必要です。Version 2.1.0.6でテストされています。ほかのバージョンでも動作する場合もありますが、このバージョンを使用することをお勧めします。

C#バインディングの3つの単純なサンプルコードが、XenSdkSample.slnソリューションの個別のプロジェクトとしてXenServer-SDK/XneServer.NET/samplesディレクトリに収録されています。

- GetVariousRecords：XenServerホストにログインし、ホスト、ストレージ、および仮想マシンに関する情報を表示します。
- GetVmRecords：XenServerホストにログインし、すべての仮想マシンレコードのリストを出力します。
- VmPowerStates：XenServerホストにログインし、仮想マシンを検索してさまざまな電源状態に切り替えます。シャットダウン状態の仮想マシンが必要です。

4.2.3. Java

JavaバインディングはXenServer-SDK/XenServerJavaディレクトリに収録されています。Microsoft Visual Studioでのビルドに適したプロジェクトファイルも提供されています。各APIオブジェクトは、1つのJavaファイルに関連付けられています。たとえば仮想マシンの操作を実装する関数は、VM.javaファイルに含まれています。

Javaバインディングの依存関係

サポートされるプラットフォーム：	LinuxおよびWindows
ライブラリ：	言語バインディングは、Javaプログラムによってリンクされるjavaアーカイブファイルであるxenserver-6.2.0.jarとして生成されています。
依存関係：	• xenserver.jarとXML-RPCサーバーの間の通信を可能にするために、xmlrpc-client-3.1.jarが必要です。 • サンプルを実行するために、ws-commons-util-1.0.2.jarが必要です。

メインのXenServer-SDK/XenServerJava/samples/RunTests.javaファイルを実行すると、同じディレクトリ内の以下の一連のサンプルが実行されます。

- AddNetwork：NICに接続されていない新規内部ネットワークを追加します。
- SessionReuse：1つのセッションオブジェクトを複数の接続で共有します。

- `AsyncVMCreate`：組み込みのテンプレートから新規仮想マシンを非同期的に作成して、その仮想マシンを起動して停止します。
- `VdiAndSrOps`：ストレージリポジトリやVDIに関するさまざまなテストを実行します（ダミーストレージリポジトリの作成など）。
- `CreateVM`：1つのネットワークとDVDドライブを持つ仮想マシンをデフォルトストレージリポジトリ上に作成します。
- `DeprecatedMethod`：廃止予定のAPIメソッドがコールされたときに警告が表示されるかどうかをテストします。
- `GetAllRecordsOfAllTypes`：すべての種類のオブジェクトのすべてのレコードを取得します。
- `SharedStorage`：共有NFSストレージリポジトリを作成します。
- `StartAllVMs`：ホストに接続してそのホスト上の各仮想マシンを起動します。

4.2.4. PowerShell

このリリースには、2つのバージョンのPowershell言語バインディングが含まれています。

従来のPowerShellバインディングは、`XenServer-SDK/XenPSSnapIn_old`ディレクトリに収録されています。Windowsインストーラ`XenServerPSSnapIn.msi`と、XenServer APIをWindows PowerShell 1.0コマンドレットで使用するソースコードが提供されています。

従来のPowerShellバインディングの依存関係

サポートされるプラットフォーム：	.NET Framework 3.5とPowershell 1.0をインストールしたWindows
ライブラリ：	`XenServerPSSnapIn.dll`
依存関係：	XML-RPCサーバーとの通信を可能にするために、`CookComputing.XMLRpcV2.dll`が必要です。Version 2.1.0.6でテストされています。ほかのバージョンでも動作する場合もありますが、このバージョンを使用することをお勧めします。

従来のPowershellバインディングのサンプルスクリプトが、`XenServer-SDK/XenPSSnapIn_old/samples`ディレクトリに収録されています。

- `AutomatedTestCore.ps1`：XenServerホストにログインして、ストレージリポジトリおよび仮想マシンを作成し、さまざまな電源操作を実行します。

新たに再設計されたPowerShellバインディングは、`XenServer-SDK/XenPSSnapIn`ディレクトリに収録されています。Windowsインストーラ`XenServerPSSnapIn.msi`と、XenServer APIをWindows PowerShell 2.0コマンドレットで使用するソースコードが提供されています。

PowerShellバインディングの依存関係

サポートされるプラットフォーム：	.NET Framework 3.5とPowershell 2.0をインストールしたWindows
ライブラリ：	`XenServerPSSnapIn.dll`
依存関係：	XML-RPCサーバーとの通信を可能にするために、`CookComputing.XMLRpcV2.dll`が必要です。Version 2.1.0.6でテストされています。ほかのバージョンでも動作する場合もありますが、このバージョンを使用することをお勧めします。

Powershellバインディングのサンプルスクリプトが、`XenServer-SDK/XenPSSnapIn/samples`ディレクトリに収録されています。

- `AutomatedTestCore.ps1`：XenServerホストにログインして、ストレージリポジトリおよび仮想マシンを作成し、さまざまな電源操作を実行します。

4.2.5. Python

Pythonバインディングは、単一のファイルの`XenServer-SDK/XenServerPython/XenAPI.py`に含まれています。

Pythonバインディングの依存関係

サポートされるプラットフォーム：	Linux
ライブラリ：	XenAPI.py
依存関係：	なし

SDKには、Pythonの8つのサンプルコードが含まれています。

- `fixpbds.py`：共通ストレージにアクセスするための設定値を再設定します。
- `install.py`：Debian仮想マシンをインストールし、それをネットワークに接続して起動し、仮想マシンによりIPアドレスが報告されるまで待機します。
- `license.py`：新しいライセンスをXenServerホストにアップロードします。
- `permute.py`：仮想マシンのセットを選択し、XenMotionを使用してホスト間でそれらを同時に移行します。
- `powercycle.py`：仮想マシンのセットを選択し、それらの電源を入れ直します。
- `shell.py`：テスト用の単純なインタラクティブシェルです。
- `vm_start_async.py`：操作を非同期的に呼び出す方法を示します。
- `watch-all-events.py`：すべてのイベントを登録し、イベントが発生したときに詳細を出力します。

4.2.6. コマンドラインインターフェイス（CLI）

サードパーティのソフトウェア開発者は、未加工のXML-RPCや提供されている言語バインディングに加えて**xe**コマンドラインインターフェイスを使用して、ソフトウェアをXenServerに統合できます。xe CLIは、XenServerホスト上にデフォルトでインストールされます。また、Linuxではスタンドアロンのリモート CLIも使用できます。Windowsの場合は、`xe.exe` CLI実行ファイルがXenCenterと一緒にインストールされます。

CLIの依存関係

サポートされるプラットフォーム：	LinuxおよびWindows
ライブラリ：	なし
バイナリ：	xe（Windowsの場合はxe.exe）
依存関係：	なし

CLIを使用すると、スクリプトやそのほかのプログラムからほとんどすべてのAPIコールを直接呼び出し、必要なセッション管理を自動的に実行できます。xe CLIの構文と機能について詳しくは、『XenServer管理者ガイド』を参照してください。このほかの情報やサンプルコードについては、Citrix Knowledge Centerを参照してください。

注：

XenServerホストのコンソールからCLIを実行する場合は、コマンド名と引数の両方でTabキーによる自動補完機能を使用できます。

4.3. 完全なアプリケーションの例

ここでは、APIを使用する実際のプログラムの完全な例を2つ挙げて説明します。

4.3.1. XenMotionによる複数の仮想マシンの同時移行

このPythonのサンプル（`XenServer-SDK/XenServerPython/samples/permute.py`）は、XenMotionを使用して、複数の仮想マシンをリソースプール内のサーバーの間で同時に移行します。このサンプルでは、非同期APIコールを使用し、タスクセットが完了するまで待機します。

プログラムの冒頭には標準ボイラープレートテキストがあり、次にAPIバインディングモジュールをインポートします。

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

次に、サーバーのURL、ユーザー名、パスワード、および反復処理の回数を含むコマンドライン引数を解析します。ユーザー名とパスワードはセッションを確立するときに使用されます。このセッションは`main`関数に渡され、さらにループ内で複数回呼び出されます。`try:`と`finally:`が使用されていることに注目してください。これにより、プログラムの終了時に必ずセッションからログアウトします。

```
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`関数は、システム内で実行中の各仮想マシンを調べ、**コントロールドメイン**（システムの一部でユーザーは制御できません）をフィルタ（除外）します。そして、実行中の仮想マシンとそれらの現在のホストの一覧を作成します。

```
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))
```

次にホスト一覧をローテーションします。

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

その後、このローテーションに従い、XenMotionを使って各仮想マシンを新しいホストに移行します（一覧の2番目のホストで実行中の仮想マシンを1番目のホストに移行するなど）。各移行処理を並行して実行するために、非同期版の**VM.pool_migrate**を使用して、タスクリファレンスの一覧を作成します。**VM.pool_migrate**に渡される*live*フラグに注意してください。これにより、仮想マシンを停止せずに移行することができます。

```
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)
```

その後、完了したことを確認するために、タスク一覧をポーリングします。

```
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)
```

すべてのタスクが保留（**pending**）状態でなくなったら（正常終了、失敗、またはキャンセル状態になったら）、すべてのタスクが成功したかどうかを確認するために、もう一度ポーリングします。

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

タスクのいずれかが失敗した場合は詳細を出力し、例外を生成して、さらに調査するためにそのタスクオブジェクトを残します。すべてのタスクが成功した場合はタスクオブジェクトを破棄し、関数が戻ります。

```
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. xe CLIによる仮想マシンの複製

この**bash**スクリプトのサンプルは、xe CLIを使用して仮想マシンをシャットダウンしてから複製します。

プログラムの冒頭のボイラープレートでは、環境変数*XE*が設定されているかどうかをチェックします。この環境変数が設定されている場合はその変数がCLIのフルパスを参照し、設定されていない場合は現在のパス上にxe CLIがあるとして動作します。次に、サーバー名、ユーザー名、およびパスワードの入力をユーザーに促します。

```
# 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
```

次に、コマンドライン引数をチェックします。複製する仮想マシンのUUIDのみが必要です。

```
# 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
```

その後、仮想マシンの電源状態を確認し、実行中の場合は完全なシャットダウンを試みます。イベントシステムを使用して、仮想マシンが停止状態（halted）になるまで待機します。

> **注：**
>
> xe CLIでは、コマンドライン引数*--minimal*がサポートされます。この引数を指定すると、出力内容から余分な空白や書式が削除され、スクリプトからの使用に適した形式が使用されます。複数の値が返される場合は、コンマで区切られます。

```
# 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
```

次に、仮想マシンを複製し、新しい仮想マシンの*name_label*を*cloned_vm*に設定します。

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

最後に、複製元の仮想マシンが実行中であった場合は、その仮想マシンと新しい仮想マシンの両方を起動します。

```
# 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 HTTPによるXenServerとの対話

XenServerでは各ホストにHTTPインターフェイスを介してアクセスし、さまざまな操作を実行できます。この章では、利用できるメカニズムについて説明します。

5.1. 仮想マシンのインポートとエクスポート

仮想マシンのインポートとエクスポートには時間がかかるので、インポートとエクスポートの操作に対して非同期HTTPインターフェイスが提供されています。XenServer APIを使用してエクスポートするには、次のコードサンプルのようにHTTP GETコールを作成して、有効なセッションID、タスクID、および仮想マシンのUUIDを提供します。

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

インポートするには、次のコードサンプルのようにHTTP PUTコールを使用します。

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

5.2. XenServerパフォーマンス統計値の取得

XenServerでは、さまざまな側面のパフォーマンスについて統計値が記録されます。メトリクスは永続的に格納されるため、履歴データに基づく傾向分析や長期間のアクセスが可能です。仮想マシンでストレージを使用できる場合、これらの統計値は仮想マシンのシャットダウン時にディスクに書き込まれます。統計値はラウンドロビンデータベースに格納されます。このデータベースは（コントロールドメインを含む）各仮想マシンおよびサーバーごとに保持されます。ラウンドロビンデータベースは、仮想マシンが実行中の場合はそのホストサーバーに、仮想マシンが実行中でない場合はプールマスタに格納されます。ラウンドロビンデータベースは毎日バックアップされます。

警告：

以前のバージョンのXenServer APIで
は、**VM_metrics**、**VM_guest_metrics**、**host_metrics**の各メソッドおよび関連するメソッドにより瞬間的なパフォーマンスメトリクスを取得できました。これらのメソッドは、この章で説明するHTTPハンドラを使用するために廃止されました。HTTPハンドラを使用して、仮想マシンおよびサーバー上のラウンドロビンデータベースから統計値をダウンロードします。デフォルトでは、従来の測定値の戻り値が0であることに注意してください。旧バージョンのXenServerの定期的な統計ポーリングの機能を使用するには、*other-config:rrd_update_interval*=<interval>パラメータに次のいずれかの値を設定して、ホストを再起動します。

never　これがデフォルト設定で、定期的なポーリングを実行しません。

1　5秒間隔でポーリングを実行します。

2　1分間隔でポーリングを実行します。

以前のメトリクスAPIはデフォルトで値を返さないため、従来の監視プロトコルを使用する監視クライアントを実行するには、このキーを有効にする必要があります。

統計値は最長1年間、値が取得された時期に応じて異なる粒度で保持されます。平均および最新の値は次の間隔で保存されます。

- 直近の10分間は5秒間隔
- 過去2時間は1分間隔

- 過去1週間は1時間間隔
- 過去1年間は1日間隔

ラウンドロビンデータベースは、未圧縮のXMLファイルとしてディスクに保存されます。各ラウンドロビンデータベースのサイズは、200KiBから、1年分の統計値を格納した時点で約1.2MiBになります。

> **警告：**
>
> ディスク領域の不足などで統計値を書き込めない場合、統計値は失われ、最後に保存されたバージョンのラウンドロビンデータベースが使用されます。

統計値は、HTTPを使用してXML形式でダウンロードできます（**wget**を使用するなど）。XML形式については、http://oss.oetiker.ch/rrdtool/doc/rrddump.en.htmlおよびhttp://oss.oetiker.ch/rrdtool/doc/rrdxport.en.htmlを参照してください。HTTP認証はユーザー名とパスワードでも、セッショントークンでも実行できます。パラメータはURLの後に疑問符（?）を付けて追加し、アンパサンド（&）で区切ります。

ホスト上のすべての仮想マシン統計値の更新を取得するには、次のURLを使用します。

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

この要求により、rrdtool `xport`コマンドの出力に使用されるXML形式で、クエリ対象のサーバー上の各仮想マシンについてデータが返されます。エクスポートデータのどの列がどの仮想マシンのデータを示すのかを区別できるように、`legend`フィールドの値の前に仮想マシンのUUIDが付加されます。

ホストの更新も取得する場合は、クエリパラメータ*`host=true`*を使用します。

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

期間が短くなるにつれて粒度が細かくなります。つまり、要求する統計値の期間が短いほど、詳細な値を取得することになります。

rrd_updatesの追加パラメータ

cf=<ave|min|max>　　データ統合モードです。

interval=<interval>　　報告する値の間隔です。

> **注：**
>
> デフォルトでは、`ave`統計値だけが使用可能です。仮想マシンの`min`と`max`の統計値を取得するには、次のコマンドを実行します。
>
> ```
> xe pool-param-set uuid=<pool_uuid> other-config:create_min_max_in_new_VM_RRDs
> ```

ホストのすべての統計値を取得するには次のコマンドを実行します。

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

仮想マシンのすべての統計値を取得するには次のコマンドを実行します。

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

CITRIX

第6章 XenServer API拡張

XenAPIは仮想マシンのライフサイクル管理のための汎用的かつ総合的なインターフェイスで、XenAPIプロバイダで特定の機能（ストレージプロビジョニングやコンソール処理など）を柔軟に実装できます。XenServerに組み込まれている拡張により、XenCenterのインターフェイスで使用する便利な機能が提供されます。これらのメカニズムのしくみについて、この章で説明します。

通常、XenAPIの拡張を使用するには、さまざまなオブジェクトに対して*other-config*マップキーを指定します。このパラメータを使用するということは、XenServerの特定のリリースでその機能がサポートされていることを意味しますが、将来的に長期間サポートされるということでは**ありません**。API化する機能については常時検討していますが、そのためにはインターフェイスの本質が十分に周知されている必要があります。これらの拡張の使用方法について、フィードバックをお寄せください。今後の製品向上のための貴重な資料として、参考にさせていただきます。

6.1. 仮想マシンコンソールの転送

ユーザーに対して仮想マシンコンソールを物理マシンのように表示するために、多くのXenAPIのグラフィックインターフェイスで仮想マシンコンソールへのアクセスが必要になります。ゲストの種類、またアクセス先が物理ホストコンソールであるかどうかに応じて、いくつかの種類のコンソールを使用できます。

コンソールのアクセス

オペレーティングシステム	テキスト	グラフィック	最適化グラフィック
Windows	なし	VNC（APIコールを使用）	RDP（ゲストから直接）
Linux	あり（VNCおよびAPIコール経由）	なし	VNC（ゲストから直接）
物理ホスト	あり（VNCおよびAPIコール経由）	なし	なし

Windowsなどのハードウェア支援型仮想マシンでは、VNCでグラフィックコンソールを直接提供します。テキストベースのコンソールはなく、またグラフィックコンソールを使用するためのゲストネットワーク設定は不要です。ゲストネットワークが確立されている場合は、リモートデスクトップアクセスを設定し、RDPクライアントを使用して直接接続する方が効率的です（これはXenAPIの外部で行う必要があります）。

Linuxなどの準仮想化仮想マシンでは、ネイティブテキストコンソールを直接提供します。XenServerには**vncterm**と呼ばれるユーティリティが組み込まれており、このテキストベースのコンソールをグラフィックVNCに変換できます。このコンソールは、ゲストネットワークがなくても機能します。前述のWindowsの場合と同様に、Linuxディストリビューションでよく使用されるのは、ゲストでVNCを設定してゲストのネットワークインターフェイスを介して直接接続する方法です。

物理ホストのコンソールは`vt100`コンソールとしてのみ使用できます。このコンソールには、コントロールドメインで**vncterm**を実行することによって、VNCコンソールとしてXenAPIを介してアクセスできます。

RFB（Remote Framebuffer）はVNCで使用されるプロトコルで、『The RFB Protocol』で規定されています。サードパーティの開発者は、自由に使用できる多くの実装を利用して独自のVNCビューアを開発できます。ビューアがサポートする必要のある最低バージョンはRFB 3.3です。

6.1.1. APIによるVNCコンソールの取得

VNCコンソールは、ホストエージェントに渡される特殊なURLを介して取得します。APIコールは次の順序で呼び出します。

1. クライアントからマスタ/443：XML-RPC：**`Session.login_with_password()`**。
2. マスタ/443からクライアント：後続のコールで使用されるセッションリファレンスを返します。
3. クライアントからマスタ/443：XML-RPC：**`VM.get_by_name_label()`**。
4. マスタ/443からクライアント：特定の仮想マシンのリファレンスを返します（物理ホストコンソールを取得する場合はコントロールドメインのリファレンス）。
5. クライアントからマスタ/443：XML-RPC：**`VM.get_consoles()`**。
6. マスタ/443からクライアント：仮想マシンに関連付けられているコンソールオブジェクトの一覧を返します。
7. クライアントからマスタ/443：XML-RPC：**`VM.get_location()`**。
8. 要求されたコンソールの位置を記述するURIを返します。URIの形式は、https://192.168.0.1/console?ref=OpaqueRef:c038533a-af99-a0ff-9095-c1159f2dc6a0のようになります。
9. クライアントから192.168.0.1：HTTP CONNECT "/console?ref=(...)"

最後のHTTP CONNECTはやや標準から外れています。HTTP/1.1 RFCでは、HTTP CONNECTはURLではなくホストとポートのみを取るべきとされているためです。HTTP接続が完了したら、その接続をVNCサーバーとして直接使用することができます。追加のHTTPプロトコル操作は不要です。

このスキーマでは、クライアントからコントロールドメインのIPアドレスに直接アクセスする必要があり、NAT（Network Address Translation：ネットワークアドレス変換）デバイスによりアクセスが拒否される場合には正しく機能しません。CLIを使用してクライアントからコンソールURIを取得し、接続状態を調べることができます。

CLIでコンソールURIを取得するには

1. 次のコマンドを実行して、仮想マシンのUUIDを取得します。

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

2. 次のコマンドを実行して、コンソール情報を取得します。

   ```
   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=(...)
   ```

pingなどのコマンドラインユーティリティを使用して、*location*フィールドのIPアドレスへの接続状態をテストします。

6.1.2. Linux仮想マシンのVNC転送の無効化

Linux仮想マシンを作成したり破棄したりするとき、テキストコンソールをVNCに変換する**vncterm**プロセスがホストエージェントによって自動的に管理されます。テキストコンソールに直接アクセスしたい上級ユーザーは、その仮想マシンのVNC転送を無効にすることができます。その後、テキストコンソールにはコントロールドメインからのみ直接アクセスできるようになり、XenCenterなどのグラフィックインターフェイスではその仮想マシンのコンソールを表示できなくなります。

CLIを使用してLinux VNCコンソールを無効にするには

1. ゲストを起動する前に、仮想マシンレコードに次のパラメータを設定します。

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

2. 仮想マシンを起動します。

3. 次のCLIコマンドを実行して、仮想マシンのドメインIDを取得します。

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

4. ホストコンソール上で次のコマンドを実行して、テキストコンソールに直接接続します。

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

この設定は上級者用です。負荷の高い入出力操作にテキストコンソールを使用することは推奨されません。その代わり、SSHなどのネットワークベースの接続方法を使用して、ゲストに接続してください。

6.2. 準仮想化Linuxのインストール

Xen対応のカーネルを起動する必要があるため、準仮想化Linuxゲストのインストールは複雑で、ハードウェア支援機能を使用するゲストのインストールとは異なります。ただし、エミュレーションのオーバヘッドがないため、ネイティブに近いインストール速度が得られるという利点があります。XenServerではさまざまなLinuxディストリビューションのインストールがサポートされるため、ここでは標準的な手順として説明します。

準仮想化Linuxをインストールするために、**eliloader**と呼ばれる特殊なブートローダーがコントロールドメインに組み込まれています。このブートローダーによって仮想マシンレコードのさまざまな*other-config*キーが起動時に読み取られ、ディストリビューション固有のインストール処理が実行されます。

- *install-repository*：必須キーで、リポジトリのパスです。値はhttp、https、ftp、またはnfsです。ターゲットインストーラと同じ要領で指定しますが、method=などのプレフィックスは含めません。
- *install-vnc*：デフォルトではfalseです。インストール中、使用可能であればVNCを使用します。
- *install-vncpasswd*：デフォルトでは未指定です。使用するVNCパスワードです。ターゲットディストリビューションのコマンドラインで入力が可能な場合に指定します。
- *install-round*：デフォルトでは1です。現在のブートローダー処理が何回目かを示します。ユーザーはこの値を変更できません。

6.2.1. Red Hat Enterprise Linux 4.1/4.4

eliloaderを使用して起動を2回行います。1回目は、インストーラ`initrd`と`/opt/xensource/packages/files/guest-installer`からのカーネルを返します。2回目は、仮想マシンから追加のアップデートディスクを削除し、ブートローダーを**pygrub**に切り替えた後、通常の起動を行います。

Red Hat社ではこれらのディストリビューションにXenカーネルを提供していないため、この方法でXenServerのカスタムカーネルを代わりに使用する必要があります。

6.2.2. Red Hat Enterprise Linux 4.5/5.0

Red Hat Enterprise Linux 4.4の場合と同様ですが、カーネルとRAMディスクをユーザー指定のネットワークリポジトリから直接ダウンロードし、ブートローダーを直ちに**pygrub**に切り替える点が異なります。pygrubは直ちに実行されず、次回起動時に解析のみ行われる点に注意してください。

Red Hatベンダのネットワークリポジトリからアップストリームカーネルを直接インストールすることができます。XenServerに付属のISOイメージ`xs-tools.iso`では、さまざまなXen関連の問題を修正する、アップデートされたカーネルも提供されています。

6.2.3. SUSE Linux Enterprise 10 Service Pack 1

このインストールでは2回の起動が必要です。1回目の起動では、ネットワークリポジトリからカーネルとRAMディスクをダウンロードして起動します。2回目の起動では、ディスクを調べてインストールされたカーネルとRAMディスクを見つけ、*PV-bootloader-args*を設定してゲストファイルシステムにこれらのパスを反映させ

ます。このプロセスは、**pygrub**の代わりにSUSEで使用される**domUloader**をエミュレートします。最後に、ブートローダーを**pygrub**に設定し、通常の起動を開始します。

SUSE Linux Enterprise Server 10のインストール方法では、カーネルとRAMディスクのパスがゲストの`menu.lst`ではなく仮想マシンレコードに保存されます。YaSTパッケージマネージャは有効な`menu.lst`を書き込まないため、この方法でインストールする必要があります。

6.2.4. CentOS 4.5/5.0

eliloaderによって認識されるいくつかのMD5チェックサムが異なることを除き、CentOSのインストール方法は上記のRed Hatの場合と同様です。

6.3. Xenstoreエントリの仮想マシンへの追加

仮想マシンの種類に応じて特別な操作を行うゲストエージェントを、仮想マシンにインストールしたい場合があります。これを行うには、仮想マシンの作成時に設定される、`vm-data`と呼ばれる特別なXenstoreネームスペースを使用します。このネームスペースを設定するには、仮想マシンレコードの*`xenstore-data`*マップを使用します。

仮想マシンのXenstoreノード*foo*を設定するには

1. 次のコマンドを実行して、仮想マシンレコードの*`xenstore-data`*パラメータを設定します。

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

2. 仮想マシンを起動します。
3. Linuxベースの仮想マシンの場合、XenServer Toolsをインストールしてから**`xenstore-read`**を使用してXenstoreにノードが存在することを確認します。

   注：

   `vm-data`で始まるプレフィックスのみが許可され、このネームスペースにないものはすべて仮想マシンの起動時に無視されます。

6.4. セキュリティの強化

悪意のあるゲストからの攻撃に備えて、XenServer 6.2.0以降のコントロールドメインには、さまざまなセキュリティ強化が施されています。この変更により開発済みのアプリケーションの動作に問題が生じることはありませんが、ほかのディストリビューションとは異なる動作として、ここで説明します。

- `libxenstore`を使用してアクセスする、ソケットインターフェイスの**`xenstored`**。インターフェイスは`xs_restrict()`によって制限されます。
- `libxenctrl`の`xs_evtchn_open()`を呼び出してアクセスする、デバイスの`/dev/xen/evtchn`。ハンドルは`xs_evtchn_restrict()`を使用して制限できます。
- `libxenctrl`の `xs_interface_open()`を呼び出してアクセスする、デバイスの`/proc/xen/privcmd`。ハンドルは`xc_interface_restrict()`を使用して制限できます。一部の特権コマンド（任意のハイパーコールを呼び出す機能など）はその性質上制限することが難しいため、制限付きハンドルでは使用を禁じられています。
- 制限付きハンドルに後からより高度な権限を付与することはできないため、インターフェイスをいったん終了してから、再開する必要があります。それ以降プロセスでさらにハンドルが開かれる場合は、セキュリティを維持できません。

コントロールドメインの特権ユーザースペースのインターフェイスは、特定のドメインのみを対象に動作するように制限できるようになりました。この変更によって影響を受けるインターフェイスは3つです。

- デバイスエミュレーションの**qemu**プロセスと端末エミュレーションの**vncterm**プロセスは、ルート以外のユーザーIDで実行され、空のディレクトリに制限されます。可能な場合は、上記の制限APIを使用して権限を放棄します。
- 悪意のあるゲストによるコントロールドメインへのサービス拒否攻撃を防ぐため、Xenstoreへのアクセスには制限が設定されています。この制限は、トークンバケットという方式で実装されています。ほとんどの操作に1トークンが、トランザクションのオープンには20トークンが必要です。多数のゲストで同時に負荷の大きな操作を実行するような場合でも上限に達することがないよう、十分に高い制限値が設定されています。
- VNCゲストコンソールは`localhost`インターフェイスにのみバインドされるので、ユーザーがコントロールドメインのパケットフィルタを無効にしても、外部からVNCゲストコンソールにアクセスできるようにはなりません。

6.5. ネットワークインターフェイスの詳細設定

仮想ネットワークインターフェイスと物理ネットワークインターフェイスにはいくつかの詳細設定があり、*`other-config`*マップパラメータを使用して設定できます。この詳細設定には、一連のカスタムethtool設定と、そのほかの設定があります。

6.5.1. ethtool設定

仮想ネットワークインターフェイスと物理ネットワークインターフェイス用にカスタムのethtool設定が必要な場合があります。この設定は、*`other-config`*マップパラメータで*`ethtool-`* <replaceable> option </replaceable> キーを使用して行います。

キー	説明	有効な設定
ethtool-rx	RXチェックサム機能を有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-tx	TXチェックサム機能を有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-sg	Scatter/Gatherを有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-tso	TCPセグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-ufo	UDPフラグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-gso	汎用セグメンテーションオフロードを有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-autoneg	自動ネゴシエーションを有効にするかどうかを指定します。	有効にする場合は`on`または`true`、無効にする場合は`off`または`false`です。
ethtool-speed	デバイス速度をMB/秒単位で設定します。	10、100、または1000

キー	説明	有効な設定
ethtool-duplex	全二重モードまたは半二重モードを設定します。	半二重モードにする場合はhalf、全二重モードにする場合はfullです。

たとえば、xe CLIを使用して仮想NICのTXチェックサムを有効にするには、次のコマンドを実行します。

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

または

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

xe CLIを使用して物理NICを半二重モードに設定するには、次のコマンドを実行します。

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

6.5.2. そのほかの設定

仮想ネットワークインターフェイスまたは物理ネットワークインターフェイスを無作為検出モードに設定することもできます。これを行うには、*promiscuous*キーにonを指定します。たとえば、xe CLIを使用して物理NICの無作為検出モードを有効にするには、次のコマンドを実行します。

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

または

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

VIFオブジェクトとPIFオブジェクトには*MTU*パラメータがあります。このパラメータは読み取り専用で、インターフェイスの最大送信単位の現在の設定を示します。*other-config*マップパラメータで*mtu*キーを使用すると、物理NICまたは仮想NICのデフォルトの最大送信単位を上書きすることができます。たとえば、xe CLIを使用してジャンボフレームを使用するように仮想NICのMTUをリセットするには、次のコマンドを実行します。

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

使用されるインターフェイスのMTUを変更することは高度で実験的な機能であることと、単一のリソースプール内の複数のNIC間でMTUが異なると予期しない副作用を起こす可能性があることに注意してください。

6.6. ストレージリポジトリ名の国際化対応

インストール時に作成されるストレージリポジトリに、名前を国際化する方法を示す*other_config*キーが付加されました。

*other_config["i18n-key"]*は、次のいずれかの値を取ります。

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

さらに、*other_config["i18n-original-value-*<replaceable>field name</replaceable>"]は、ストレージリポジトリの作成にそのフィールドに設定されていた値を提供します。XenCenterで*SR.name_label*が*other_config["i18n-original-value-name_label"]*と等しいレコードが検出された場合（XenServerのインストール時に作成されてからレコードが変更されていない場合）、国際化が適用されます。つまり、XenCenterでそのフィールドの現在の値が無視され、ユーザー言語に適した値が使用されます。

独自の目的のために*SR.name_label*を変更すると、*other_config["i18n-original-value-name_label"]*と同じではなくなります。したがって、XenCenterで国際化が適用されず、その代わりにユーザーが指定する名前が保持されます。

6.7. XenCenterでのオブジェクトの非表示化

ネットワーク、物理ネットワークインターフェイス、および仮想マシンは、`HideFromXenCenter=true`キーをそのオブジェクトの*other_config*パラメータに追加することで、XenCenterで非表示にすることができます。この機能はXenServerに習熟した独立系ソフトウェアベンダのために用意されているもので、日常的なユーザーのためのものではありません。たとえば、環境内の一般ユーザーが直接使用すべきでない、特定の複製仮想マシンを隠す必要がある場合などに使用します。

XenCenterでは、**［表示］**メニューを使用して、隠しオブジェクト（非表示のネットワーク、物理ネットワークインターフェイス、および仮想マシン）を表示することができます。

CITRIX

第7章 XenCenter API拡張

この章では、文書化されたAPIとは別に開発されたAPIについて、その前提条件と合わせて詳しく説明します。拡張は、`VM.other_config`のようなディクショナリに、特定のキー/値ペアとしてコード化されています。

7.1. pool

キー	説明
pool.name_label	name_labelに値がない場合は、プールをツリー表示から隠す必要があることを示します。
pool.rolling_upgrade_in_progress	プールがローリングアップグレード中である場合に存在します。

7.2. host

キー	説明
host.other_config["iscsi_iqn"]	ホストのiSCSI IQNです。
host.license_params["expiry"]	ホストのライセンスの有効期限です。ISO 8601のUTC形式です。
host.license_params["sku_type"]	ホストのライセンスの種類です。Server、Enterpriseなどです。
host.license_params["restrict_pooling"]	プール化がホストにより制限されている場合は`true`を返します。
host.license_params["restrict_connection"]	XenCenterから確立できる接続数を制限することを示します。
host.license_params["restrict_qos"]	ホストでQoS（サービス品質）設定が有効である場合は`true`を返します。
host.license_params["restrict_vlan"]	ホストで仮想ネットワークの作成が制限されている場合は`true`を返します。
host.license_params["restrict_pool_attached _storage"]	ホストで共有ストレージの作成が制限されている場合は`true`を返します。
host.software_version["product_version"]	ホストの製品バージョンを返します。
host.software_version["build_number"]	ホストのビルド番号を返します。
host.software_version["xapi"]	ホストのAPIリビジョン番号を返します。
host.software_version["package-linux"]	Linux Packがインストール済みである場合は`installed`を返します。
host.software_version["oem_build_number"]	ホストがOEMバージョンである場合はそのリビジョン番号を返します。

キー	説明
host.logging["syslog_destination"]	XenServerのシステムログの書き込み先を取得または設定します。ローカルの場合は`null`です。
host.logging["multipathing"]	ホストでストレージのマルチパスが有効な場合は`true`です。
host.logging["boot_time"]	浮動小数点UNIX時間でのホストの起動時刻です。
host.logging["agent_start_time"]	浮動小数点UNIX時間でのコントロールドメイン管理デーモンの開始時刻です。

7.3. VM

キー	説明
VM.other_config["default_template"]	テンプレートがCitrixによりインストールされたものであることを示します。このAPIは、デフォルトテンプレートについてツリー表示から選択的に隠したり、異なるアイコンを使用したり、削除を禁止したりするときに使用します。
VM.other_config["xensource_internal"]	テンプレートがP2Vサーバーテンプレートのような特殊なものであることを示します。このようなテンプレートはユーザーインターフェイスに表示されません。
VM.other_config["install_distro"] == "rhlike"	テンプレートがRed Hat Enterprise Linux 4.5、Red Hat Enterprise Linux 5、またはCentOSの同等のバージョンのものであることを示します。このテンプレートは、インストール時にインストールリポジトリの入力を要求するため（XenServer 4.1以降でISO/CDからのインストールもサポートします）、そしてインストーラに適合するようにNFS URLを変更するために使用します。
VM.other_config["install_distro"] in { "rhel41" | "rhel44" }	テンプレートがRed Hat Enterprise Linux 4.1、Red Hat Enterprise Linux 4.4、またはCentOSの同等のバージョンのものであることを示します。このテンプレートは、インストール時にインストールリポジトリの入力を要求するため、そしてインストーラに適合するようにNFS URLを変更するために使用します。これらのテンプレートではISOはサポートされません。

キー	説明
VM.other_config["install_distro"] == "sleslike"	テンプレートがSUSE Linux Enterprise Server 10およびSUSE Linux Enterprise Server 9のものであることを示します。このテンプレートは、EL5の場合と同様に、インストール時にインストールリポジトリの入力を要求するために使用します。ただし、NFS URLは変更されません。XenServer 6.2.0では、SUSE Linux Enterprise Server 10のインストールでISOがサポートされます。このプラットフォームでは、*install-methods*を使用してSUSE Linux Enterprise Server 9とSUSE Linux Enterprise Server 10を区別します。
VM.other_config["install-repository"] == "cdrom"	URLではなく、仮想マシンに接続されているCDドライブ内のリポジトリからのインストールを要求します。
VM.other_config["auto_poweron"]	サーバーの起動時に仮想マシンを起動するかどうかを取得または設定します。`true`または`false`です。
VM.other_config["ignore_excessive_vcpus"]	仮想マシンにそのホストが搭載する物理CPUより多くVCPUが設定されている場合に、XenCenterの警告を無視するかどうかを取得または設定します。無視する場合は`true`です。
VM.other_config["HideFromXenCenter"]	XenCenterのツリー表示に仮想マシンを表示するかどうかを取得または設定します。隠す場合は`true`です。
VM.other_config["import_task"]	仮想マシンを作成したインポートタスクを取得します。
VM.HVM_boot_params["order"]	HVMモードの仮想マシンでのみ、仮想マシンの起動順序を取得または設定します。たとえば、`DCN`は起動順序がディスク、CDドライブ、ネットワークであることを示します。
VM.VCPU_params["weight"]	仮想マシンのVCPUのionice値を取得または設定します。値の範囲は1から65536で、65536が上限です。
VM.pool_migrate(..., options['live'])	`true`はライブマイグレーションを示します。XenCenterでは常にこの設定が使用されます。

キー	説明
VM.other_config["install-methods"]	テンプレートで使用できるインストール方法の、コンマ区切りの一覧です。cdrom、nfs、http、またはftpが含まれます。
VM.other_config["last_shutdown_time"]	仮想マシンの最終シャットダウンまたは再起動の日時です。ISO 8601のUTC形式です。
VM.other_config["p2v_source_machine"]	仮想マシンがP2Vプロセスでインポートされた場合は、インポート元のマシンです。
VM.other_config["p2v_import_date"]	P2Vプロセスでインポートされた場合は、仮想マシンのインポート日時です。ISO 8601のUTC形式です。

7.4. SR

キー	説明
SR.other_config["auto-scan"]	ストレージリポジトリの変更が自動的にスキャンされます。XenCenterで作成されたストレージリポジトリはすべてスキャンされます。
SR.sm_config["type"]	物理CDドライブであるストレージリポジトリについて、cdを種類として設定します。

7.5. VDI

キー	説明
VDI.type	userの場合は、ディスクが仮想マシンに接続されている場合にグラフィックユーザーインターフェイスで仮想ディスクイメージを削除できます。systemの場合は削除できません。後者を使用する目的は、仮想マシンの破損を防ぐことです。仮想ディスクイメージを削除する代わりに、仮想マシンをアンインストールする必要があります。suspendおよびcrashdumpでは、それぞれ一時停止ダンプとコアダンプが記録されます。ephemeralは現在使用されていません。
VDI.managed	管理されていないすべての仮想ディスクイメージが、ユーザーインターフェイスで非表示になります。これらは、VHDチェーンの分岐点または未使用のLUN-per-VDIディスクです。

キー	説明
VDI.sm_config["vmhint"]	仮想ディスクイメージでサポートされる仮想マシンのUUIDです。特定のストレージバックエンドのパフォーマンスを向上するために、ユーザーインターフェイスを介して仮想ディスクイメージが作成されるときにこの値が設定されます。

7.6. VBD

キー	説明
VBD.other_config["is_owner"]	設定されている場合は、仮想マシンのアンインストール時にディスクを削除できます。
VBD.other_config["class"]	**ionice**のBest Effort設定に対応する整数を設定します。

7.7. network

キー	説明
network.other_config["automatic"]	`false`以外の値が設定されている場合、**［新規VM］**ウィザードにより、デフォルトでこのネットワークに接続する仮想ネットワークインターフェイスが作成されます。
network.other_config["import_task"]	ネットワークを作成したインポートタスクを取得します。

7.8. VM_guest_metrics

キー	説明
PV_drivers_version["major"]	仮想マシンの準仮想化ドライバのメジャーバージョンを取得します。
PV_drivers_version["minor"]	仮想マシンの準仮想化ドライバのマイナーバージョンを取得します。
PV_drivers_version["micro"]	仮想マシンの準仮想化ドライバのマイクロバージョン（ビルド番号）を取得します。

7.9. task

キー	説明
task.other_config["object_creation"] == "complete"	仮想マシンのインポートに関連するタスクについて、すべてのオブジェクト（VMおよびNetwork）が作成されるとこのフラグが設定されます。その後で、仮想マシンの**［インポート］**ウィザードですべての必要なネットワークを再マップすると便利です。