vSphere Web Services SDK 文档

文件状态： [　]草稿 [√]正式发布 [　]正在修改	文件标识：	
	当前版本：	1.0
	作　　者：	尹哲
	完成日期：	

版本/状态	作者	参与者	起止日期	备注
1.0	尹哲		2012/12/13-2013/07/26	

目录

1、VMware vSphere and vSphere 管理 APIs：.....9
1.1 虚拟化和 VMware vSphere 组件.....9
1.2 vSphere 开发工具.....9
1.2.1 vSphere Web Services SDK.....9
1.2.2 CIM APIs.....10
1.2.3 vSphere SDK for Perl.....10
1.2.4 vSphere PowerCLI.....10
1.2.5 VIX API.....10
1.3 SDK 安装.....11
1.4 SDK 例子.....11
2、vSphere API 编程模型.....11
2.1 vSphere 客户端-服务端架构.....12
2.2 vShpere API 作为 Web Service.....13
2.2.1 WSDL 文件 and the Client-Side 代理接口.....13
2.2.2 网络访问 vShpere Web Service.....14
2.2.3 Language-Specific Classes and Methods.....14
2.3.4 影射 XML 数据类型到 Java and C# 数据类型.....15
2.3 访问 managed 对象.....15
2.4 访问 vSphere Server 数据.....16
2.4.1 检索服务器信息.....16
2.4.2 Working with Data Structures.....17
2.4.3 访问属性值.....17
2.4.4 未设置的可选属性.....20
2.4.5 属性名称和路径中的换码符.....21
3、客户端应用.....21
3.1 基本客户端应用功能.....21
3.2 一个 Java 例子应用概况.....22
3.2.1 Java 客户端例程.....23
3.3 Web Server Session 记号.....26
3.3.1 使用 JAX-WS 访问 HTTP Endpoint.....27
3.3.2 访问 vSphere Server.....29
3.3.3 关闭连接.....30
3.3.4 Using the Java Samples as Reference.....30
3.4 使用 Axis Bindings.....32
3.5 vSphere API 多版本.....33
3.6 Server 支持确定 API 版本.....34
4、数据中心清单.....34
4.1 清单概况.....35
4.1 清单层次和 and ServiceInstance.....35
4.1.1 层次中的文件夹.....36
4.2 ESX/ESXi 清单层次.....36
4.2 访问 Accessing Inventory Objects.....37

4.3 创建清单对象.....38
4.4 清单权利权限需求.....39
4.4.1 权限.....39
4.4.2 许可.....39
4.5 被管理的和独立的 ESX/ESXi 主机.....40
5、属性收集器.....40
5.1 PropertyCollector 介绍.....41
5.1.1 数据检索.....41
5.1.2 清单遍历和挑选对象.....41
5.2 属性收集相关的 vSphere 数据对象.....42
5.3 属性收集相关的 vSphere 方法.....42
5.4 PropertyCollector 例程(RetrievePropertiesEx).....43
5.5 清单遍历.....52
5.5.1 TraversalSpec 遍历.....52
5.5.2 SelectionSpec 遍历.....62
5.5.3 客户端数据同步(WaitForUpdatesEx).....70
5.5.4 服务器数据传输.....72
5.5.5 PropertyCollector 性能.....72
5.5.6 SearchIndex.....73
6、认证和授权.....73
6.1 认证和授权管理的相关对象.....73
6.2 ESX/ESXi 和 vCenter Server 的认证和授权.....74
6.2.1 ESX/ESXi 用户模式.....75
6.2.2 vCenter Server 用户模式.....75
6.2.3 vSphere 安全模式.....76
6.3.4 建立用户，组和许可.....78
6.3 从 UserDirectory 获取用户和组信息.....78
6.4 通过 HostLocalAccountManager 管理 ESX/ESXi 用户和组.....79
6.5 通过 AuthorizationManager 管理角色和许可.....80
6.5.1 使用角色来统一权限设置.....81
6.5.2 编辑例子角色来创建新的角色.....82
6.5.3 通过许可来授予权限 Granting Privileges Through Permissions.....83
6.6 通过 SessionManager 验证用户.....85
6.7 使用证书库来自动登陆.....86
6.7.1 证书库方法.....87
6.7.2 证书库备份文件.....87
6.7.3 证书库举例.....88
6.7.4 使用证书库指定角色和用户.....89
6.8 使用 LicenseManager 来管理许可证.....89
7、主机.....90
7.1 主机管理对象.....91
7.2 检索主机信息.....91
7.3 配置和重新配置主机.....92
7.4 管理主机生命周期.....93

7.4.1 重启和关机.....93
7.4.2 使用待机模式.....93
7.4.3 断开连接和重新连接主机.....94
7.4.4 查询和改变主机时间.....94
7.4.5 查询虚拟机内存的 Overhead.....95
8、存储.....95
8.1 存储管理对象.....95
8.2 存储介绍.....96
8.2.1 虚拟机如何访问存储.....97
8.2.2 数据存储.....98
8.3 选择使用存储 API.....99
8.4 配置磁盘分区.....100
8.5 多路径管理.....101
8.6 配置 iSCSI 存储.....102
8.7 创建和管理数据存储.....104
8.7.1 访问数据存储.....105
8.7.2 创建和编辑一个 VMFS 数据存储.....106
8.7.3 移除和更新数据存储.....108
8.7.4 通过 HostStorageSystem 管理 VMFS 数据存储.....108
8.8 管理 VMFS 卷拷贝(Resignaturing).....109
8.9 管理诊断分区.....110
8.9.1 检索诊断分区信息.....110
8.9.2 创建一个诊断分区.....111
8.10 代码示例.....111
9、网络.....111
9.1 网络管理对象和方法.....111
9.2 网络介绍.....112
9.3 虚拟网络标准交换机环境.....113
9.3.1 虚拟交换机.....113
9.3.2 端口组.....113
9.3.3 虚拟机网络接口.....114
9.3.4 VMkernel 网络接口.....114
9.3.5 物理网络适配器(pnic).....115
9.4 使用 vSS 建立网络.....115
9.4.1 检索网络配置信息.....115
9.4.2 添加一个虚拟交换机.....115
9.4.3 添加一个虚拟端口组.....116
9.4.4 添加一个 VMkernel 网络接口.....117
9.5 定义主机网络策略.....117
9.6 NIC Teaming.....118
9.7 建立 IPv6 网络.....119
9.8 添加网络服务.....120
9.8.1 添加一个 NTP 服务.....120
9.8.2 建立 IP 路由配置.....120

9.8.3 建立 SNMP....121
9.9 代码示例....121
10、虚拟机配置....122
10.1 虚拟机管理对象和方法....122
10.2 创建虚拟机和虚拟机模板....123
10.2.1 使用 VirtualMachineConfigSpec 来创建一个虚拟机....123
10.2.2 创建虚拟机模板....125
10.2.3 克隆一个虚拟机....125
10.2.4 模板转换成虚拟机....126
10.2.5 访问虚拟机信息....126
10.3 配置虚拟机....128
10.3.1 名称和位置....128
10.3.2 硬件版本....128
10.3.3 启动选项....129
10.3.4 操作系统....129
10.3.5 CPU 和内存信息....129
10.3.6 网络....131
10.3.7 光线通道 NPIV 设置....131
10.3.8 文件位置....132
10.4 为虚拟机添加设备....132
10.5 执行虚拟机电源操作....134
10.6 注册和注销虚拟机....135
10.7 自定义客户操作系统....136
10.8 安装 VMware Tools....137
10.9 虚拟机升级....137
11、虚拟机管理....137
11.1 虚拟机迁移....138
11.1.1 冷迁移....138
11.1.2 使用 VMotion 迁移....138
11.1.3 使用存储 VMotion....139
11.2 快照....139
11.2.1 创建一个快照....140
11.2.2 恢复到一个快照....141
11.2.3 删除一个快照....141
11.3 链接的虚拟机....141
11.3.1 链接的虚拟机和磁盘备份....141
11.3.2 创建一个链接的虚拟机....142
11.3.3 移除快照和删除链接虚拟机....144
11.3.4 在一个链接虚拟机群中迁移一个虚拟机....145
11.3.5 提升一个虚拟机磁盘....145
11.3.6 执行 Delta 磁盘的高级操作....146
12、虚拟应用....147
12.1 关于虚拟应用....148
12.1.1 管理概述....148

12.1.2 直接的和链接的子项.....148
12.1.3 OVF 包.....149
12.2 创建一个 VirtualApp.....150
12.3 管理 VirtualApp 子项.....150
12.4 导出一个虚拟应用.....151
12.4.1 获得导出租约.....151
12.4.2 下载磁盘.....151
12.4.3 产生 OVF 描述符.....152
12.4.4 完成租约.....152
12.5 导入一个 OVF 包.....152
12.6 虚拟应用生命周期.....153
12.6.1 上电和下电一个虚拟应用.....153
12.6.2 注销一个虚拟应用.....154
12.6.3 挂起一个虚拟应用.....154
12.6.4 销毁一个虚拟应用.....154
13、资源管理.....154
13.1 资源管理对象.....155
13.2 资源管理介绍.....155
13.3 资源分配.....156
13.3.1 资源池层次结构.....156
13.3.2 资源池管理指导原则.....157
13.3.3 群集概括.....157
13.4 创建和配置资源池.....158
13.4.1 了解扩展预留.....159
13.4.2 删除子资源池.....160
13.4.3 移动资源池或虚拟机到一个资源池中.....160
13.5 VMware DRS 和 VMware HA 群集介绍.....161
13.5.1 VMware DRS.....161
13.5.2 VMware HA.....162
13.6 创建和配置群集.....162
13.6.1 创建一个群集.....162
13.6.2 添加一个主机到一个群集中.....163
13.6.3 重新配置一个群集.....163
13.7 管理 DRS 群集.....164
13.8 管理 HA 群集.....164
13.8.1 首要和次要主机.....165
13.8.2 故障检测和主机网络隔离.....165
13.8.3 同时使用 VMware HA 和 DRS.....166
14、任务和计划任务.....166
14.1 创建任务.....166
14.1.1 Session 驻留.....167
14.1.2 取消任务.....167
14.1.3 使用 TaskInfo 决定任务状态.....167
14.1.4 监视 TaskInfo 属性.....168

14.2 访问和处理多任务.....170
14.2.1 使用 ViewManager 对象收集数据.....170
14.2.2 使用一个 TaskManager 接口来收集数据.....181
14.2.3 了解 ScheduledTaskManager 接口.....183
14.2.3.1.2 计划循环操作.....185
14.2.4 使用 TaskHistoryCollector.....188
15、事件和警告.....189
15.1 事件和警告管理对象.....189
15.2 了解事件.....190
15.2.1 使用 EventManager 管理事件.....190
15.2.2 事件数据对象.....191
15.2.3 设计事件信息内容格式.....192
15.2.4 创建自定义事件.....192
15.3 使用 EventHistoryCollector.....193
15.3.1 创建一个 EventHistoryCollector 过滤器.....194
15.3.2 管理 HistoryCollector.....194
15.4 使用警告.....194
15.4.1 获取警告列表.....195
15.4.2 创建警告.....195
15.5 使用 AlarmSpec 数据对象定义警告.....196
15.5.1 使用 AlarmExpression 指定警告触发条件.....197
15.5.2 指定警告动作.....198
15.6 删除或禁用一个警告.....199
16、vSphere 性能.....199
16.1 vSphere 性能数据收集.....200
16.2 PerformanceManager 对象和方法.....201
16.3 检索 vSphere 性能数据.....203
16.3.1 性能计数器例程(QueryPerf).....204
16.3.2 大规模的性能数据检索.....214
16.3.3 使用 QueryPerf 方法作为原始数据.....215
16.3.4 查询方法比较.....215
16.3.5 检索总计性性能数据.....216
16.4 性能计数器元数据.....216
16.5 性能时间间隔.....217
16.5.1 ESXi 服务器性能时间间隔.....218
16.5.2 vCenter 服务器性能时间间隔.....218
16.6 vSphere 性能和数据存储.....219
16.6.1 编辑历史时间间隔.....219
16.6.2 编辑性能计数器收集级别.....220
17、诊断和处理故障.....221
17.1 处理故障最佳实践.....221
17.2 配置文件和日志文件概述.....222
17.2.1 ESX/ESXi 日志文件.....223
17.2.2 虚拟机日志文件.....224

17.2.3 vCenter 服务器日志 225
17.3 编辑日志级别来获得详细的信息 225
17.3.1 在 ESX/ESXi 系统上设置日志级别 226
17.3.2 产生日志 227
17.3.3 在 vCenter 服务器系统上设置日志级别 227
17.4 使用 DianosticManager 228
17.5 使用 MOB 来查看 DiagnosticManager 229
17.6 产生诊断包 230
18、Managed 对象浏览 231
18.1 使用 MOB 来查看对象模型 231
18.1.1 访问 MOB 231
18.1.2 使用 MOB 导航 VMware 架构对象模型 232
18.2 使用 MOB 来调用方法 232
18.2.1 传入原始的数据类型给方法 233
18.2.2 传入原始数据队列给方法 233
18.2.3 传入复合结构给方法 233
19、HTTP 访问 vSphere 服务器文件 237
19.1 HTTP 访问介绍 237
19.2 HTTP 访问的 URL 语法 238
19.2.1 数据存储访问(/folder) 238
19.2.2 主机文件访问(/host) 239
19.2.3 更新包访问(/tmp) 240
19.2.4 HTTP 访问权限需求 240
20、权限参考 241
20.1 调用操作所需的权限 241

1、VMware vSphere and vSphere 管理 APIs：

VMware vSphere 支持健壮的、容错的虚拟化应用，网络和存储。vSphere 提供许多可选的组件和模块例如 VMware High Availability 和 VMware VMotion。VMware vSphere Web Services SDK 提控给 Web Services 开发者通过编程方式访问 vSphere 组件。

1.1 虚拟化和 VMware vSphere 组件

VMware 软件虚拟化了计算资源，包括 CPU，内存，存储和网络。虚拟化在计算资源，物理存储设备，网络和使用它们的应用程序中间提控了一个抽象层。

VMware vSphere 包括 ESX/ESXi，vCenter Server 和几个额外的服务产品。基本产品支持运行和管理虚拟机。配合额外的许可证，你可以使用高级的 vSphere distributed resource manaaement(DRS)和 high availability(HA)功能。

- ESX/ESXi 管理程序有能力支持多个虚机和其他虚拟组件，像存储和网络。
- vCenter Server 是一个有数据库支持的 Windows 服务，它为多个 ESX/ESXi 系统提供中央管理。
- vSphere Client 是一个图形化的管理工具来管理 vSphere。vSphere Client 运行在 Windows 系统上使用和 vSphere Web Services SDK 同样的接口来访问 vSphere 服务器。你可以运行任何 vSphere Client 支持使用的 Web Services SDK 的任务，实事上，SDK 提供的有些任务不能通过 vSphere Client 来执行。

1.2 vSphere 开发工具

VMware 支持通过 SDKs 和 scripting(脚本)工具来管理 vSphere.

1.2.1 vSphere Web Services SDK

vSphere Web Services SDK 是应用最广泛的管理 APIs。SDK 依靠 ESX/ESXi 和 vCenter Server 系统工作。作为 Web Services SDK，它采用的是中性语言。SDK 包括 Java、C#的存根（Stubs）和例子，还有一个包括 API 引用文档。

1.2.2 CIM APIs

VMware CIM APIs 为开发者提供了一套建立应用管理的 CIM(Common Information Model)接口。通过使用 VMware CIM APIs 开发者可以使用标准的 CIM-compliant 应用程序来管理 ESX/ESXi 主机。

CIM APIs 包括：

- CIM SMASH/Server 管理 API—System Management Architecture for Server Hardware(SMASH)可以自动和 DMTF(Distributed Management Task Force 分布式管理任务组)兼容。SMASH 允许 CIM 客户端监控服务器的健康状况。
- CIM Storage Management API—兼容 the Storage Network Industry Association(全球网络存储协会)的 Storage Managemetn Initiative Specification(SMI-S)标准,SMI-S 允许 CIM 客户端浏览在 ESX/ESXi 主机上的虚拟机，并可以访问存储资源。

1.2.3 vSphere SDK for Perl

vSphere SDK for Perl 通过简单的使用 Perl 脚本调用 vSphere API。管理员和开发人员可以通过使用 vSphere SDK for Perl 子程序中调用 vSphere API 实例。管理员可以使用 vSphere SDK for Perl 中提供的功能丰富的工具。

vSphere SDK for Perl 还包括了通过使用 CIMOM 的脚本来检索 ESX/ESXi 主机上的 CIM 数据的管理组建的 Web Services，一个服务提供标准 CIM 管理功能。vSphere SDK for Perl 还包括了管理 VMware Credential Store 和一个说明 credential store 使用的用例程序。

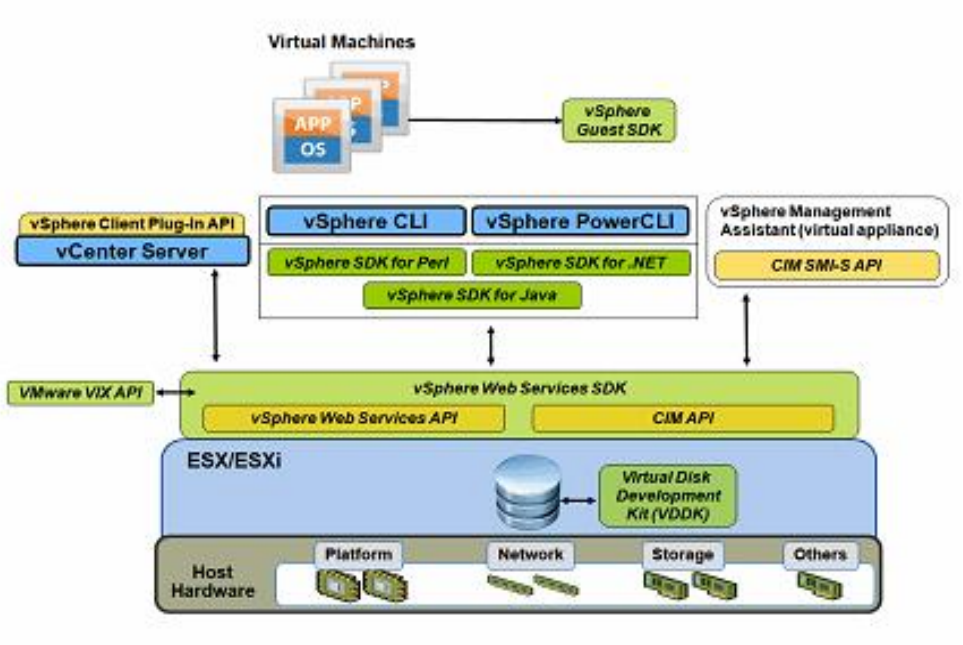
1.2.4 vSphere PowerCLI

VMware vSphere PowerCLI 为 vSphere API 提供了一个 Windows PowerShell 界面。vShpere PowerCLI 提供 PowereShell Cmdlets 来管理 vSphere 组件。另外，vSphere PowerCLI 包括了.Net 的开发包。

1.2.5 VIX API

VIX API 是一个提供脚本和编程来操作虚拟机的库.同时供脚本开发人员和应用开发人员使用.它很适合在公司内部建立自己的工具.它还可以软件供应商通过在自己的产品或制造管理虚机的产品中使用 VIX 来集成 VMware 产品。

图 1：图示说明不同的 vSphere APIs 和 CLIs 在虚拟基础架构中的位置。



1.3 SDK 安装

在你开始使用 vSphere Web Services SDK 编写程序之前，你必须首先下载软件并配置你的系统。本手册提供完全的 Java 和 C#开发讲解和详述了一个简单的安全的开发环境环境。

1.4 SDK 例子

SDK 包括一系列丰富的描述 SDK 特点的例子。提供两个系列的例子：

- Java 例子使用 SDK 中已经生成的 Java stubs（Java 桩）。
- C#例子使用 SDK 中已经生成的 Java stubs（Java 桩）。

两个例子系列中的代码包括一系列实用应用。

2、vSphere API 编程模型

vSphere API 已语言无关的 Web service 形式实现，客户端应用通过远程过程调用来访问 ESX/ESXi,vCenter Server systems 上的服务和组件。

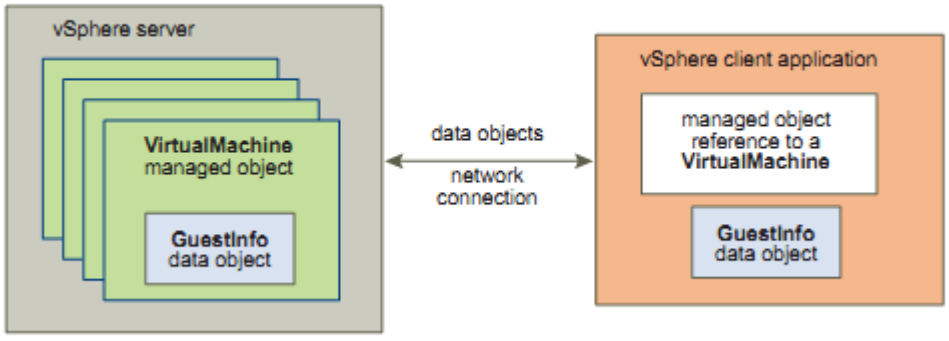
2.1 vSphere 客户端-服务端架构

VMware vSphere 客户端程序基于一个异步通讯的分布式架构模型，架构的基础基于 server-side managed object,client-side managed object references,和 data objects.

- **Managed object** 在 vSphere server(ESX/ESXi 或 vCenter Server system)上.代表着 vSphere 的服务和组件。服务包括 PropertyCollector,SearchIndex,PerformanceManager 和 ViewManager。组件包括 inventory objects 例如 VirtualMachine，Datastore 和 Folder。
- **Managed object references** 是客户端应用引用服务器端 managed objects.你的客户端应用程序通过使用 ManagedObjectReference objects 来向服务器端发送请求操作。在 objectd 的生命周期内 ManagedObjectReference 是唯一和持久有效地。当一个存在于清单中由于过期 sessions 或服务器重起而删除的 object 的引用会依然存在。假如你删除一个 object 例如一个虚机，然后又找回它，对它的引用将改变。
- **Data objects** 包含着关于 managed objects 的信息，你的客户端程序发送 data objects 和接收 data objects 与一个 vShpere server.不同的标准和功能 object 例如 VirtualMachineConfigSp 和 HostCapability.

图示：先是客户端应用和 vSphere 服务器。你的客户端有一个存在于服务器端的虚拟机的 managed object reference,一个虚拟机上的 GuestInfo data object 的拷贝。一个客户端必须包含一个 data object 的拷贝，因为，一来客户端请求类型，一个 vSphere server 可能会以一系列键-值对的形式发送一个和一个 managed object reference 保持联系的 data object 的 property data。

Figure 2-1. vSphere Server and Client



2.2 vShpere API 作为 Web Service

vSphere API 采用的是语言无关的 Web 服务，运行在 ESX/ESXi 和 vCenter 服务器上，vSphere API 采用 WS-I(web 服务互操作组织)的 Basic Profile 1.0 编译。WS-I Basic Profile 1.0 支持一下标准：

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1

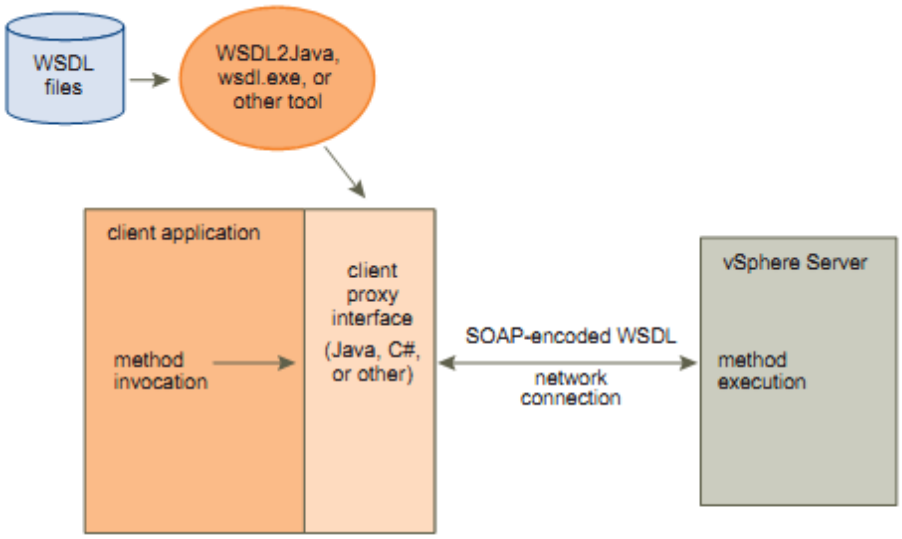
Web 服务技术支持像其他编程语言调用方法形式的操作。vSphere API Web 服务提供访问所有关于监控和管理 vSphere 组件的必要操作，例如计算机资源、虚拟机、网络、存储等等。

2.2.1 WSDL 文件 and the Client-Side 代理接口

vSphere Web Services SDK 提供一系列定义 vSphere Services API 的 WSDL(Web Services Description Language)，通过使用 Web-services 开发工具例如 JAX-WS wsimpot 或 Microsoft .Net wsdl.exe 来生成客户端代理代码(stubs).

客户端代理提供具体开发语言的 vSphere API，例如，Java 或 C#.代理使远程方法请求、组织 object data、其他分布特征、面向对象编程简易化。你的客户端应用调用代理接口方法。客户端代理使用 SOAP(Simple Object Access Protocol)来和 vSphere 服务器交换 WSDL 信息。

图例：表现了一个客户端应用使用客户端代理接口调用一个方法。代理接口基于 WSDL 定义。Client-Server 通过一个客户端代里接口通讯。



为了使用 VMware 客户端代里接口，你必须导入 vSphere API 客户端库文件到你的 C# 或 Java 应用中。

- C# using VimApi;
- Java import com.vmware.vim25.*;

2.2.2 网络访问 vShpere Web Service

你的客户端调用应用 vSphere API 通过 443 端口的 HTTPS(运行在加密的安全 Socker 层的 HTTP 连接)和 vSphere 服务器通讯.HTTPS 是默认协议.你可以设置服务器支持 HTTP. 一般在测试和开发环境中使用 HTTP,不建议用在生产环境上.

2.2.3 Language-Specific Classes and Methods

SOAP 工具根据相应的 WSDL 定义来产生特定语言的类和方法,工具同时产生一些不在 WSDL 文件中定义的对象和方法.

- Gerneratd objects. 额外的对象提供访问 vSphere Web Service 来建立 client-server 连接(VimServiceLocator,AppUtil)和声明为 vShpere API(VimPortTpye,VimService)定义的方法。
- Gernerated methods. 附加的方法是访问(getter)和设置(setter)属性的.例如 Java，方法命名股则是在属性名前添加 get 和 set 前缀，并且把属性名的第一个字母改为大写。

表 2-1 确认客户代理为 vSphere Web Services SDK WSDL 的定义

Table 2-1. Client Proxy Definitions

Element Access	Java	C#
Access to vSphere Web service (HTTPS/HTTP)	`VimServiceLocator` class	`AppUtil` class
Access to vSphere API methods	`VimPortType` class	`VimService` class
Access to vSphere API properties	`get`*`PropertyName`* and `set`*`PropertyName`* methods defined for data objects	`get` and `set` methods defined for properties
vSphere API data objects	Data objects in the vSphere API (see the *vSphere API Reference*) defined as objects in the proxy interface	

2.3.4 影射 XML 数据类型到 Java and C# 数据类型

在手册中，UML 类和对象图所使用的原始数据类型名例如 string 和 integer，没有 XML Schema 定义命名空间前缀(xsd:)。vShpere API Reference 包口完整的数据类型名，例如 xsd:string。数据类型对客户端应用编程使用的原始数据类型。

表 2-2 列出了一些常用的 XML 原始数据类型映射。

Table 2-2. Standard XML Schema Primitives to Java and .NET Data Type Mappings

XML Schema	Java	.NET Data Type
`xsd:base64binary`	`byte[]`	`Byte[]`
`xsd:boolean`	`boolean`	`Boolean`
`xsd:byte`	`byte`	`SByte`
`xsd:dateTime`	`java.util.Calendar`	`DateTime`
`xsd:decimal`	`java.math.BigDecimal`	`Decimal`
`xsd:double`	`double`	`Double`
`xsd:float`	`float`	`Single`
`xsd:int`	`int`	`Int32`
`xsd:string`	`java.lang.String`	`String`

2.3 访问 managed 对象

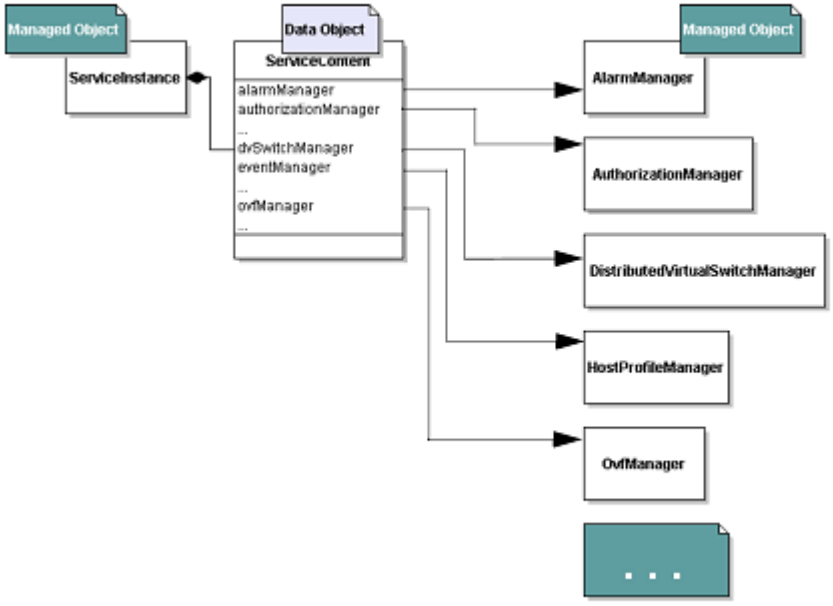
你的客户端应用通过 ServiceInstance managed 对象和和它相关联的 ServiceContent 数据对象来获得访问。ServiceContent 数据对象包括服务、管理实体和清单根目录的 managed 对象引用。

ServiceInstance managed 对象是 ESX/ESXi 和 vCenter Server 系统清单的根对象。服务器创建 ServiceInstance 和管理实例来在虚拟环境中提供服务。管理实例如 LicenseManager,PerpormanceManager,ViewManager.

ServiceInstance 是首要的访问服务清单的入口。你的客户端应用开始于连接服务和创建对 ServiceInstance 的引用。在你连接到服务器后，你可以调用 ServiceInstance.RetrieveServiceContent 的方法来得到一个 ServiceContent 数据对象，ServiceContent 提供访问

图示 2-3 显示 ServiceInstance 和 ServiceContent 对象的对象模型.图表显示一些 ServiceContent managed 对象的引用和引用的目标对象.每一格 managed 对象引用代表一个明确的服务器上的 managed 对象包括它的类型和值(value 属性是隐式的字符串)

Figure 2-3. ManagedObjectReference Data Object



2.4 访问 vSphere Server 数据

为了获得虚拟架构的信息，你检索 managed 对象的属性。Managed 对象属性可以是一个简单的数据类型，例如 integer 或 string 数据，也可以是复合的数据类型例如包含一系列属性的数据对象。

2.4.1 检索服务器信息

通过一个 managed 对象的引用，你可以获得服务器列表对象和位于客户端的基于值得数据对象的信息。你的可以通过下列方法实现：

- 使用存取器(getter)方法，客户端代理接口为每一个数据对象属性提供存取器方法。你可以使用这些存取器来获得对象的值。

- 使用一个 PropertyCollector 来定位到一个已选择入口的服武器，获得指定的属性的值。
- 使用 SearchIndex managed 对象来获得你感兴趣的 managed 实体的 managed 对象引用。SearchIndex 能返回指定的 managed 实体的 manged 对象引用，如 ComputeResource,Datacenter,Folder,HostSystem,ResourcePool,VirtualMachine,通过给定的清单路径，IP 地址或 DNS 名称。

注意事项：你可以使用 API 方法来操作位于 vSphere 清单中的 managed 对象。一个更新 managed 对象属性的方法可能会更新其他 managed 对象的属性。服武器采用异步方式来更新清单。无法保证方法返回时清单已完成更新。可以使用 PropertyCollector 的 WaitForUpdatesEx 来获得属性的更改。

2.4.2 Working with Data Structures

属性中报还当前给定的服武器端对象的信息,属性值可以是下列类型:

- 简单数据类型,例如 string,Boolean 或 integer,举例，ManagedEntity managed 对象有一个字符串类型的 name 属性。
- 简单数据类型或数据对象的队列，例如，一个 HostSystem managed 对象包含一个指向寄宿于物理机上的虚拟机的 managed 对象引用(数据对象类型)的队列。在例如，SessionManager managed 对象有一个 sessionList 属性它是 UserSession 数据对象的队列。
- 预先定义值得枚举类型。值可以是简单数据类型或数据对象的集合。例如，一个虚拟机的 power 状态可以取下列三个值之一：poweredOn,poweredOff,suspended.
  属性的类型经常是个字符串，但是实际上属性期望其值为枚举类型中的一个值。例如当你设置 VirtualMachineConfigSpec.guestid 属性时你可以指定 VirtualMachineGuestOSIdentifier 中的一个元素作为一个字符串赋给它。
- 复合数据类型，例如，HostProfileConfigInfo 对象包含数据对象，一个数据对象的队列和一个字符串的队列。

2.4.3 访问属性值

使用复合数据结构和包含服务数据的队列：

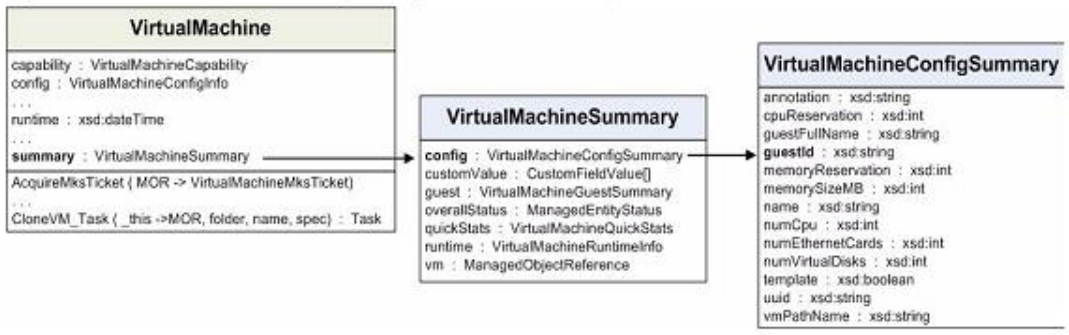
- 使用”.”标记符访问符合数据结构中的属性。
- 非强制转换属性值到队列类型。
- 使用键或索引值来访问队列中的值。

2.4.3.1 嵌入属性和符合数据结构中属性路径

vSphere 数据对象包括符合数据类型的属性，例如数据对象，嵌入的数据对象也可能包含的属性为数据对象。属性可以定位到几个级别。

举例，下图显示一个 VirtualMachine managed 对象的 UML 类图，拥有一个被定义成 xsd:dateTime 数据类型的运行时属性，还有一个 VirtualMachineSummary 数据对象的 summary 属性，VirtualMachineSummary 数据对象包括一个 config 属性，它是一个 VirtualMachineConfigSummary 数据对象。

Figure 2-4. VirtualMachine Managed Object and Nested Properties



2.4.3.2 xsd:anyType Arrays

vSphere API 使用非强制的 xsd:anyType 类型声明，一个 vSphere 客户端必须映射 xsd:anyType 的值到明确的数据类型上，一个 xsd:anyType 值代表一个单一的数据值或者一个队列。vSphere API 的 WSDL 定义了所有 vSphere 客户端能以队列形式发送接受的数据值地队列类型.对列类型使用前缀”ArrayOf”.例如 ArrayOfString 为字符类型值。

当客户端数据送到 vSphere Server 时，客户端必须采用明确的数据类型。例如，一个客户端为一个 ScheduledTask 定义一个 MethodAction。vSphere API 采用 xsd:anyType 类型定义 Action(MethodActionArgument.value 属性)参数。假如一个 Action 要获取一个队列参数，客户端必须设置一致的 MethodAction.argument[].value 为适当的 ArrayOf…类型。

当一个客户端从 vSphere 服务器接受到 xsd:anyType 类型数据，它必须转换数据成一个

明确的类型。例如，PropertyCollector 的方法 RetrievePropertyiesEx 返回一组 ObjectContent 数据对象。ObjectContent.propSet 属性是一系列 DynamicProperty 对象包含请求属性的值。每个 DynamicProperty 对象包含一个键-值对。属性值(DynamicProperty.val)是 xsd:anyType 类型的。它代表一个单一对象或一个队列对象。

当返回值是一个单一对象例如一个 Event,ManagedObjectReference 或 String，你可以直接转换为适合类型的变量，然而，当这个值是一个对象队列是，你不能直接转换 anyType 值到任何变量。

当 PropertyCollector 返回队列数据，当作 xsd:anyType 值。明确语义绑定包含队列对象定义例如 ArrayOfEvent，ArrayOfManagedObjectReference 和 ArrayOfString，和一致的”get” 方法。为了重 xsd:anyType 类型的属性中获得真实的队列，转换 DynamicProperty.val 成一致的队列类型和使用匹配 get 方法—例如，getEvent(),getManagedObjectReference()或 getString()。下列部分提供了例子来说明怎么转换返回值。代码使用 JAX-WS 生成 Java 绑定的 VMware vSphere Web Services SDK WSDL。每个代码片段使用下面逻辑：

- 使用 DynamicProperty.getVal()方法来得到 anyType 属性值。
- 为转化 anyType 值明确合适的队列类型。
- 使用一致的 get 方法来分配转换操作的结果到一个队列变量。

Event Array Example

```
/*
* Handing arrays of Evnet objects
* Cast the return value to ArrayOfEvent and use getEvent().
*/
List[] eventList = ((ArrayOfEvent) dynamicProp.getVal()).getEvent();
```

ManagedObjectReference Array Example

```
/*
* Handing array of ManagedObjectReference objects
* Cast the return value to ArrayOfManagedObjectReference and use
* getManagedObjectReference().
*/
List[] moralist = ((ArrayOfManagedObjectReference)
dynamicProp.getVal()).getManagedObjectReference();
```

String Array Example

```
/*
* Handing arrays of strings
* Cast the return value to ArrayOfString and use getString().
* getManagedObjectReference().
*/
```

2.4.3.3 Indexed Array and Key-Based Array Properties

VMware vSphere 数据结构包括队列属性，索引队列或基于关键字的队列。

- 索引队列可以通过索引值访问。索引队列常用于队列中数据位置固定的数据类型队列。例如，AuthorizationManager managed 对象的 roleList 属性是一个认证规则的队列。添加一个新规则到队列中不会改变队列中已存在元素的位置。
- 基于关键字队列常用于位置经常变动的队列。一个基于关键字的队列(类似于 Perl Hash 或 Python dictionary 的概念)使用唯一的，不变的值作为关键字来访问一个元素值。典型的，关键字是一个字符串，但是数字也可以被使用。例如，Enent 队列使用数字作为关键字。关联属性能指向基于关键字队列中的实体。例如，a.b.c[“xyz”]指向属性 c 的关键字为 xyz 的值。

  vSphere 管理对象模型使用基于关键字的队列监测管理对象引用。基于关键字的队列属性内容值可以通过关键字属性访问或通过一个 managed 对象引用来访问。这些域的值是唯一可以访问所有队列组件的。

2.4.4 未设置的可选属性

在 vSphere Web Services SDK 中的大多数数据对象存在可选得属性可以通过你的客户端或服务器进程或事件设置。假如你检索一个数据对象它有一个可选的属性没有被设置，服务器将不返回这个可选属性相对应的值。假如你调用存取器函数来的到这个属性值，返回值依赖于你所采用的编程语言。

例如，如果你采用 Java 或 C#，你的到这个没有设置的属性值是”null”.

图 2-5 显示 HostFirewallInfo 数据对象的部分属性.当你在 vSphere Web Services SDK API Reference 中查找属性时,你可以看到可选属性被标记为红色星号。

在这个例子中,defaultPolicy 属性总会被返回，但是 ruleset 属性假如不设置它将返回 null 值。

Figure 2-5. Data Object - HostFirewallInfo Properties

Properties

NAME	TYPE	DESCRIPTION
defaultPolicy	HostFirewallDefaultPolicy	Default firewall p
ruleset*	HostFirewallRuleset[]	List of configurec
Properties inherited from DynamicData		
dynamicProperty, dynamicType		

*Need not be set

2.4.5 属性名称和路径中的换码符

%被当作转义字符把特殊的字符嵌入到字符串中。例如%2f(或%2F)被解释围/字符。当一个字符创中包括%号时，使用%%。清单的开始路径从 root 文件夹开始(ServiceContent.rootFolder 属性)，靠/字符分割。

Table 2-4. Special Characters

Character	Description	Representation in URL
%	Percent sign	%25
/	Slash	%2F, %2f
\	Backslash	%5C, %5c
-	Dash	%2D, %2d
.	Dot	%2E, %2e
"	Double quotation mark	%2B, %2b

3、客户端应用

你通过 vSphere Web Services SDK 创建的任何客户端应用必须链接服务器和通过用户权限认证。当链接建立后，客户端应用通过 vSphere 服务来访问虚拟环境。

3.1 基本客户端应用功能

每个使用 vSphere API 的客户端应用都执行以下操作：

1、 创建到 vSphere Server Web Service 的链接。

2、 实例化一个指向 ServiceInstance 的本地代理对象。使用这个本地代理对象检索服务上的 ServiceContent 对象。ServiceContent 包含一个指向清单根目录和提供 vSphere services 的 managed 对象的引用。

3、 实例化一个可以访问 vSphere API 方法的本地代理对象。

4、 使用正确的证书(帐户、密码、and optionally the local)登陆到服务器。

5、 访问服务器端对象来检索数据和执行管理操作。

6、 关闭链接。

3.2 一个 Java 例子应用概况

本章提控一个完整的展示客户端基本功能的例子。这个客户端例子打印它连接得、可以检索信息得服务器得产品名称，服务类型，产品版本。

To build a simple vSphere client application

1、 Import the vSphere Web Service API 库：

Import com.vmware.vim25.*;

2、 Import 必须的 Java(JAX-WS 连接，bindings,SOAP)库:

Import java.util.*;

Import javax.net.ssl.HostnameVerifier;

Import javax.net.ssl.HttpsURLConnection;

Import javax.net.ssl.SSLSession;

Import javax.xml.ws.BindingProvider;

Import javax.xml.ws.soap.SOAPFaultException;

3、 创建 TestClient 类：

Public class TestClient {

4、 创建类的变量的声明和定义。使用 TrustManager 类来接受所有证书，参见”Accessing the HTTP Endpoint with JAX-WS”.

5、 使用 vSphere Web Services APIs 创建链接。

6、 从 vSphere 或 vCenter 服务器检索数据。在本例中，我们仅仅准备打印出产品名，服务类型和产品版本来证明客户端已连接和工作正确。

System.out.println(serviceContent.getAbout().getFullName());

System.out.println(“Server type is “ + serviceContent.getAbout().getApiType());

System.out.println(“API version is “ + serviceContent.getAbout().getVersion());

7、 使用 VimPort 对象来关闭连接。

3.2.1 Java 客户端例程

Java Test Client Application

```
Import com.vmware.vim25.*;
Import java.util.*;
Import javax.net.ssl.HostnameVerifier;
Import javax.net.ssl.HttpsURLConnection;
Import javax.net.ssl.SSLSession;
Import javax.xml.ws.BindingProvider;
Import javax.xml.ws.soap.SOAPFaultException;
Public class TestClient {
//Auhentication is handled by using a TrustManager and supplying a host verifier
//method.(The host name verifier(证明者) is declared in the main function
    Private static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
        Javax.net.ssl.X509TrustManager {
        Public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }
        Public Boolean isServerTrusted (java.security.cert.X509Certificated[] certs) {
            return true;
        }
        Public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }
        Public void checkServerTrusted(java.security.cert.X509Certificate[] certs,String
        authType) throws java.security.cert.CertificateException {
            return;
```

```
        }
        Public void checkClientTrusted(java.security.cert.X509Certificate[] certs,String
        authType) throws java.security.cert.CertificateException {
            return;
        }
    }
Public static void main(String[] args) {
    try {
        String servername = arg[0];
        String username = arg[1];
        String password = arg[2];
        String url = "https://" + servername + "/sdk/vimService";
        //Variables of the following types for access to the API methods
        //and to the vSphere inventory.
        //--ManagedObjectReference for the ServiceInstance on the Server
        //--VimService for access to the vSphere Web Service
        //--VimPortType for access to methods
        //--ServiceContent for access to managed object services
        ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
        VimService vimService;
        VimPortType vimPort;
        ServiceContent serviceContent;
        //Declare a host name verifier that will automatically enable
        //the connection.The host name verifier is invode during the SSL handshake(握手)
        HostnameVerifier hv = new HostnameVerifier() {
            public Boolean verify(String urlHostName, SSLSession session){
                return true;
            }
        };
        //Create the trust manager.
```

```
Javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
Javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
//Create ths SSL context.
Javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance(“SSL”);
//Create ths session context.
Javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
//Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
//Use the default socket factory to create the socket for the secure connection
Javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFacto
ry());
//Set the default host name verifier to enable the connection
HttpsURLConnection.setDefaultHostnameVerifier(hv);
//Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType(“ServiceInstance”);
SVC_INST_REF.setValue(“ServiceInstance”);
//Create a VimService object to obtain a VimPort binding provider
//The BindingProvider provides access to the protocol fields
//in request/response messages.Retrieve the request context
//which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
//Store the Server URL in the request context and specify true
//to maintain the connection between the client and server.
//The client API will include the Server’s HTTP cookie in its
//requests to maintain the session.if you do not set this to true, the Server will start
//a new session with each request.
```

```
            ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
            ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
            //Retrieve the ServiceContent object and login
            serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
            vimPort.login(serviceContent.getSessionManager(), username, password, null);
            //print out the product name,server type ,and product version
           System.out.println(serviceContent.getAbout().getFullName());
            System.out.println("Server type is " +    serviceContent.getAbout().getApiType());
            System.out.println("API version is " + serviceContent.getAbout().getVersion());
            //close the connection
            vimPort.logout(serviceContent.getSessionManager()):
        } catch( Exception e) {
            System.out.println("Conncetion Failed");
            e.printStackTrace():
        }
    }
```

使用下面命令编译代码：

C:> javac –classpath path-to-vim25.jar TestClient.java

使用下面命令执行编译好的类文件：

C:> java –classpaht path-to-vim25.jar TestClient web-service-url user-name user-password

3.3 Web Server Session 记号

和其他 Web 服务一样,vSphere Web 服务为每一个在 HTTP 头使用一个记号来确认 session 的客户端连接维持 session 的状态。vSphere 服务器在它的响应客户端连接请求时返回一个 session token 到客户端。随后在客户端和服务器端的通讯信息都自动包含这个标记。

每个在 SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\单独的例子中使用 JAX-WS TrustAllTrustCertificates 类，在例子 3-2 中讨论忽略证书，获得一个 session token，然后连接服务器。

CAUTION We DO NOT recommend that you trust all certificates in a production environment. Instead, you can look at the sample code to see how the JAX-WS libraries are used when making the connection, but set up an SSL policy that only allows connection to trusted certificates.

得到一个 cookie 并且放到 header 中的逻辑如下：

```
//cookie logic
List cookies = (List) headers.get("Set-cookie");
cookieValue = (String) cookies.get(0);
StringTokenizer tokenizer = new StringTokenizer(cookieValue,";");
cookieValue = tokenizer.nextToken();
String path = "$" + tokenizer.nextToken();
String cookie = "$Version=\"1\";" + cookieValue + ";" + path;
//set the cookie in the new request header
Map map = new HashMap();
map.put("Cookie",Collections.singleletonList(cookie));
((BindingProvider)
vimPort).getRequestContext().put(MessageContext.HTTP_REQUEST_HEADERS,map);
```

3.3.1 使用 JAX-WS 访问 HTTP Endpoint

通过 JAX-WS bindings 来访问 HTTP endpoint 的步骤包括 vSphere Web Service SDK Server URL,vSphere 服务器对象和变量。

- 步骤 1：在下面例子中我们使用一个 TrustManager 类来接受所有的证书。这对于产品环境不适个好主意。产品代码应该支持证书。

```
private static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
                                        javax.net.ssl.X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
    }
    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
    }
```

```
        public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
                return true;
        }
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
            String authType)    throw java.security.cert.CertificateException {
                return;
        }
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
            String authType)    throw java.security.cert.CertificateException {
                return;
        }
    }
```

- 步骤 2：在 main 方法中包括了服务器 URL 和证书作为参数

```
Public static void main(String[] args) {
    Try {
        String serverName = agrs[0];
        String userName = args[1];
        String password = args[2];
        String url = https:// + serverName + “sdk/vimService”;
```

- 步骤 3:声明下列类型变量来访问 vSphere 服务对象：
  - 为 ServcieInstance 声明 ManagerObjectRefernce
  - VimService 对象来访问 Web 服务
  - VimPortType 对象访问所有在 vSphere API 中定义的所有方法
  - ServiceContent 来访问服务器上的 managed 对象服务

下面 Java 代码段显示这些变量声明：

```
ManagedObjectReference SVC_INST_REF;
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;
```

- 步骤 4:声明一个 host name 可自动保证连接的验证者。Host name 验证者在 SSL 握

手期间被引用。

```
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostname, SSLSession session) {
        return true;
    }
};
```

- 步骤 5：示例 trust manager 对象

```
//Create the trust manager
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
```

- 步骤 6：创建 SSL 上下文

```
Javax.net.ssl.SSLContext    sc = javax.net.ssl.SSLContext.getInstance(“SSL);
```

- 步骤 7：创建 Session 上下文

```
javax.net.ssl.SSLSessionContext    sslsc = sc.getServerSessionContext();
```

- 步骤 8：初始化上下文；Session 上下文获得 trust manager

```
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
```

- 步骤 9：使用默认的 socket 工厂来创建安全连接的 socket

```
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

- 步骤 10：设置默认 host name 验证者来确保连接

```
HttpURLConnection.setDefaultHostnameVerifier(hv);
```

3.3.2 访问 vSphere Server

下列步骤使用 vSphere Web Services APIs 来创建连接：

- 步骤 1：为服务器上的 ServiceInstance 对象创建一个 managed 对象引用

```
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType(“ServiceInstance”);
SVC_INST_REF.setValue(“ServiceInstance”);
```

- 步骤 2: 创建一个 VimService 对象来获得一个 VimPort binding 提供者。BindingProvider 对象提供了访问在 request/response 消息中的协议字段。检索在 processing message request 中使用的 request 上下文。

  VimServiceLocator 和 VimPortType 对象提供访问 vSphere 服务器。getVimPort 方法返回一个提供访问 vSphere API 方法的 VimPortType 对象。

  vimService = new VimService();

  vimPort = vimService.getVimPort();

  Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
- 步骤 3:存储服务器 URL 在请求上下文中并且明确 true 来维持客户端和服务器端的连接。客户端 API 包括在请求中的 Server's HTTP cookie 来维持 session。假如你不设置这个为 true，服务器将开始一个新的 session 为每次请求。

  ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);

  ctxt.put(BindingProvider..SESSION_MAINTAIN_PROPERTY, true);
- 步骤 4：检索 ServiceInstance 内容(ServiceContent 数据对象)和在服务器上的日志

  serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);

  vimPort.login(serviceContent.getSessionManager(), username, password, null);

  isConnected = true;

3.3.3 关闭连接

使用 VimPort 对象来关闭连接，通常关闭你的服务器连接保证安全。

```
vimPort.logout(serviceContent.getSessionManager());
} catch (Exception e) {
        System.out.println("Connect Failed");
        e.printStackTrace();
}
```

3.3.4 Using the Java Samples as Reference

下面是在 SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\vm\VMPromoteDisks.java 程序片段，展示了另外服务器连接的实现。回顾在 vSphere Web Services SDK 单机 Java 例子,

使用相似的代码为你的客户端应用得到一个 session token.

Example 3-2:

```
private static String cookieValue = “”;
private static Map headers = new HashMap();
private static void trustAllHttpCertificates() throws Exception {
    javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
    javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
    trustAllCerts[0] = tm;
    javax.net.ssl.SSLContext    sc = javax.net.ssl.SSLContext.getInstance(“SSL”);
    javax.net.ssl.SSLSessionContext    sslsc = sc.getServerSessionContext();
    sslsc.setSessionTimeout(0);
    sc.init(null, trustAllCerts, null);
    javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
}
private static void connect() throws Exception {
    HostnameVerifier hv = HostnameVerifier() {
        public Boolean verify(String urlHostName, SSLSession session) {
            return true;
        }
    };
    trustAllHttpCertificates();
    HttpURLConnection.setDefaultHostnameVerifier(hv);

    SVC_INST_REF.setType(SVC_INST_NAME);
    SVC_INST_REF.setValue(SVC_INST_NAME);

    vimService = new VimService();
    vimPort = vimService.getVimPort();
    Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
    ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
```

```
        ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

        serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
        headers = (Map) ((BindingProvider) vimPort).getResponseContext().
            get(MessageContext.HTTP_RESPONSE_HEADERS);
        vimPort.login(serviceContent.getSessionManager(), username, password, null);
        isConnected = true;
        propCollectorRef = serviceContent.getPropertyCollector();
        rootRef = serviceContent.getRootFolder();
    }
```

3.4 使用 Axis Bindings

如果你还在使用 Axis bindings，你可以从 SOAP 消息上下文中提取 session token 并且复用它在同一个 session 中，如果需要，例如，你的客户端应用是多线程的，保存这个 token 可以在应用中的每一个线程中使用它。

如果连接被关闭，则 session 变为无效。

例子 3-3 展示了一个使用 Axis 客户端库的 Java 从当前 SOAP 信息上下文获得 session token。例子转化 token 到一个 String。使用 String 类型的 session token，你能保存它到客户端的本地文件中以便在随后复用。在你的连接步骤代码中，你可以检查一个已存在的 session 文件并且在需要的时候使用它。

```
org.apache.axis.client.Stub clientStub = (org.apache.axis.client.Stub) _service;
org.apache.axis.client.Call callObj = clientStub._getCall();
org.apache.axis.MessageContext msgContext = callObj.getMessageContext();
String sessionString = (String) msgContext.getProperty(
        org.apache.axis.transport.http.HTTPConstants.HEADER_COOKIE);
```

Session tokens 从 VirtualCenter Server 2.5 和 ESX/ESXi 3.5 返回，例如，可能像下列显示：

```
vCenter Server    vmware_soap_session = “63869259-F0FF-4DB5-9B3B-6493212AB9CD”
ESX/ESXi          vmware_soap_session = “52b1910ba-31ad-df35-95d1-210f86c55efb”
```

对于每一个连接，标识符存在于客户端和服务器段通讯的 HTTP header 中。Session tokens

可以在多个到Web service的连接中传递，默认，session在处于不活动状态下30分钟后失效。

```
_locator = new VimServiceLocator();
_locator.setMaintainSession(true);
_service = _locator.getVimPort(new URL(url));
```

3.5 vSphere API 多版本

当一个客户端应用连接到一个运行在 vSphere 服务器上(ESX/ESXi 或 vCenter Server 系统)的 Web service，服务器检测客户端的 API 版本来支持客户端的操作。

客户端应用使用 SOAP 信息来传输 API 版本信息到 vSphere 服务器。SOAP 信息包括一个 versionID 在 soapAction 属性中，详细资料由 SOAP 工具和客户端代理代码来透明处理。服务器根据客户端版本信息来调整其行为，显示客户端支持的 API 版本来支持客户端。

从 vSphere 4.0 开始，关于配支持的 API 版本信息保存在一个 XML 文件中，vimServiceVersions.xml,，存放在服务器上。

```
<?xml version=”1.0” encoding=”UTF-8” ?>
-<!—Copyright 2008-2010 VMware, Inc. All rights reserved. →
-<namespaces version=”1.0”>
    -<namespace>
        <name>urn:vim25</name>
        <version>5.0</version>
    -<priorVersions>
        <version>2.5u2</version>
        <version>2.5</version>
      </priorVersion>
    </namespace>
    -<namepace>
        <name>urn:vim2</name>
        <version>2.0</version>
      </namespace>
</namespace>
```

假如你正在开发的客户端应用必须支持多服务器版本在同一时刻(ESX/ESXi 5.0 和 ESX/ESXi 3.x),你必须获得服务器支持的 API 版本的信息并且在你的代码中提供逻辑选择使用或不使用的特性，建立在版本信息基础上。

3.6 Server 支持确定 API 版本

一个为同一个客户端代码定位 API 版本的方法是检索存放在服务器上的版本文件,假如你没有找到 vimServiceVersions.xml 文件，那你的服务器比 ESX/ESXi4.x,vCenter Server 4.x 还要早。

例子 3-5 VersionUtil.java 例子证明了这个通用方法。查看 VersionUtil.java 中的 getTragetNameSpaceAndVersion 方法。

例子 3-5 从获得的 API 版本信息中定义 URL。

```
try{
    String wsdlUrlString = “”;
    String vimServiceXmlUrlString = “”;
    if((urlString.indexOf(https://) != -1) || (urlString.indexOf(http://) != -1)){
        wsdlUrlString = urlString.substring(0,urlString.indexOf(“/sdk”) + 4)
            + “/vimService?wsdl”;
        vimServiceXmlUrlString = urlString.substring(0,urlString.indexOf(“/sdk”) +4)
            + “/vimServiceVersions.xml”;
    } else {
        wsdlUrlString = https:// + urlString + “/sdk/vimService?wsdl”;
        vimServiceXmlUrlString = https:// + urlString + “/sdk/vimServiceVersions.xml”;
    }
}
```

4、数据中心清单

vSphere 清单是 vSphere 数据中心和数据中心中的对象的表现。了解数据中心中的对象关联关系可以帮助你访问清单各个层以便访问到你想操作的对象。

4.1 清单概况

vSphere 清单包括下列类型对象：

- 数据中心中的系统：主机、虚拟机和虚拟应用。
- 支持的组件：计算资源，数据存储，网络和虚拟设备。
- 可组织的组件：文件夹和数据中心。

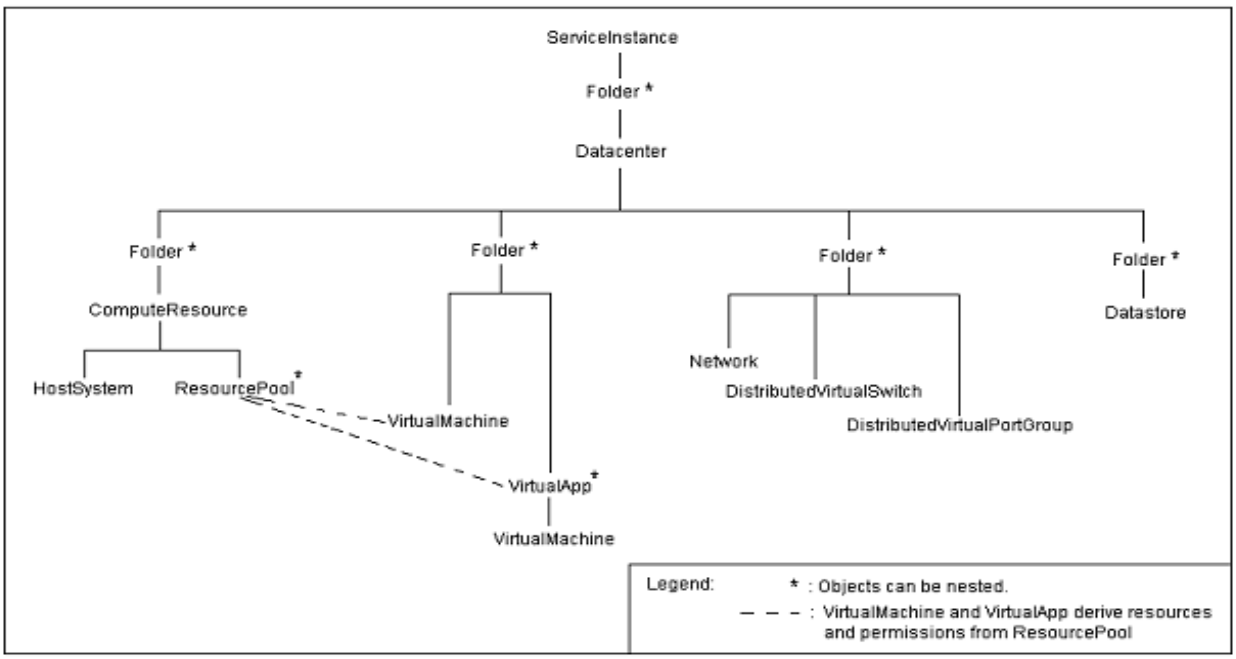
当你管理虚拟基础架构时,基于他们所在的清单来访问对象以及对象的属性和方法,了解清单结构对任何编程任务都起着决定性作用.你通常会以绑定一个 session 的 ServiceInstance 开始,ServiceInstance 是清单的根对象,从这里你可以访问清单的各个层.

4.1 清单层次和 and ServiceInstance

当你开启一个 seesion，vSphere 创建一个具有一个 root folder，一个 Datacenter 和四个具有不同清类型的单对象 folders 的 ServiceInstance。

当你访问一个 vCenter Server 系统是，层次展示如图 4-1，允许你访问的清单

Figure 4-1. vCenter Server Inventory Hierarchy



注意：假如你的 ESX/ESXi 主机被一个 vCenter Server 管理，你必须一直通过 vCenter Server 来访问你的主机。vCenter Server 保留所有同步、异步操作的轨迹，还有最新的状态和关于每一个 ESX/ESXi 主机的清单信息。因此，直接连接被 vCenter Server 管理的主机可

能得到不正确或不完整的数据。

当你的 ESX/ESXi 主机没有被 vCenter Server 管理时，你的应用可以直接连接每一个主机。

4.1.1 层次中的文件夹

假如你安装了 vCenter Server 系统，你可以在 root folder 下创建额外的 datacenters。对于每一个 DataCenter 对象，服务器自动创建下列 Folder 对象:

- 一个包括 VirtualMachine，template 和 VirtualApp 对象的 folder.
- 一个包括 ComputeResource 层次的 folder.
- 一个包括 Network,DistributedVirtualSwitch 和 DistributedVirtualPortgroup 对象的 folder.
- 一个包括 Datastore 对象的 folder.

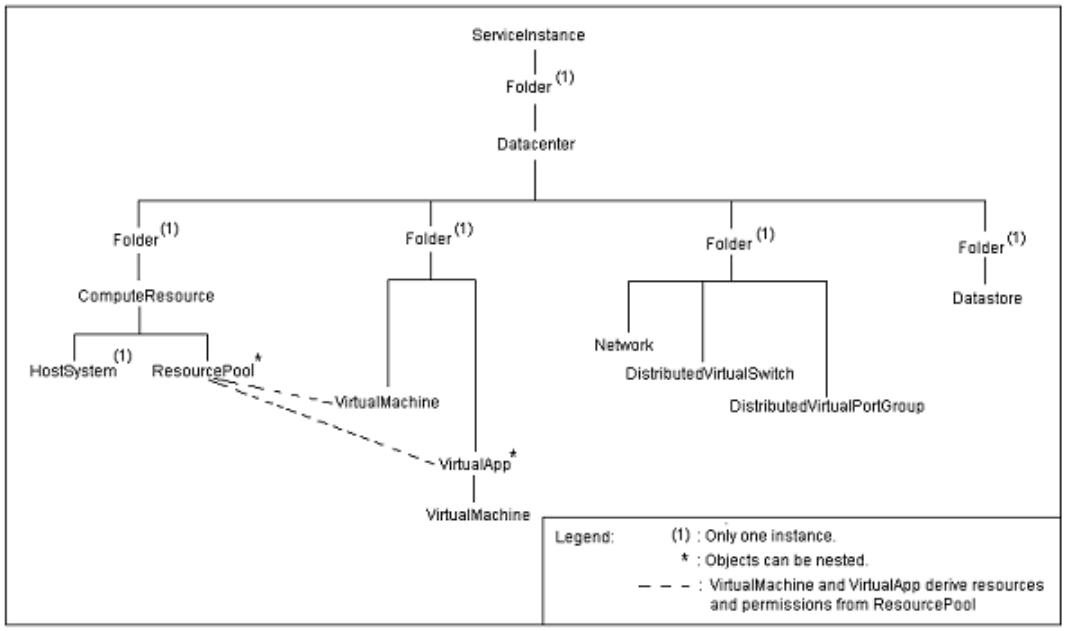
在一个大型部署中，嵌套结构允许你通过使用多个 folders 和 datacenters 来组织 datacenter 中的对象到一个简单的可管理的结构中。

为一个单独的 ESX/ESXi 系统，仅支持一个单独的 datacenter，Folder 管理实体不支持创建额外的 Folder 对象或 Datacenter 对象

4.2 ESX/ESXi 清单层次

当你直接访问 ESX/ESXi 主机，而不是通过 vCenter Server 来访问时，图 4-2 列出了可以访问清单的层次结构。

Figure 4-2. ESX/ESXi Inventory Hierarchy



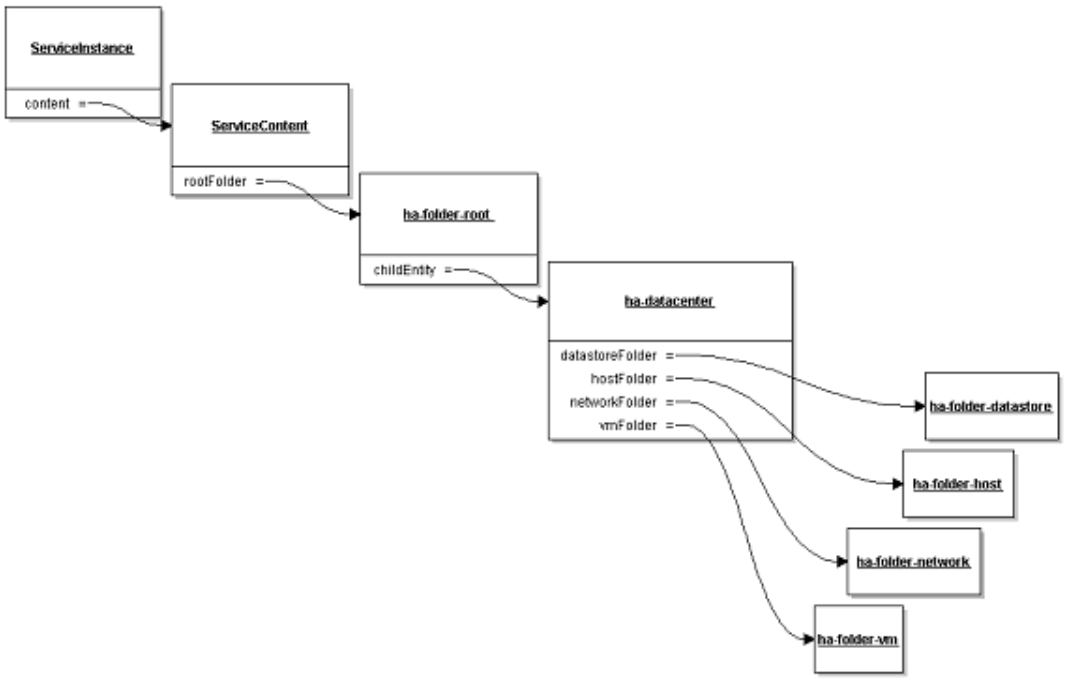
4.2 访问 Accessing Inventory Objects

为了从一个清单对象中检索信息，你通过 ServiceInstance 和清单 root 对象开始。你在属性集合中使用 TraversalSpec 来访问一个对象，通过属性来定位层次中的位置。

- 每一个 managed 对象都有一个 parent 属性来标识它在清单层次中的关系位置。
- Folder managed 对象都有一个 childEntity 属性来标识对象在一个 folder 实例中。

图 4-3 展示 childEntity 和 folder 属性来定义一个单独的 ESX/ESXi 系统清单中的默认对象。清单开始于 ServiceContent.rootFolder 属性。rootFolder 有一个针对一个 Datacenter managed 对象的 managed 对象的引用的 childEntity。

Figure 4-3. Instance Diagram of Root Folders in an Inventory



4.3 创建清单对象

Folder managed 实体提供创建下列 managed 实体的实例。

- Datacenter
- DistributedVirtualSwitch
- VirtualMachine
- Cluster
- Folder

当你创建了这些对象，他们自动创建在你所引用的创建方法文件夹里。

有时一些 managed 实体通过 Folder managed 实体上的方法来创建，有时直接实例化。例如，HostDatastoreSystem 拥有创建数据存储的方法如 CreateNasDatastore 和 CreateVmfsDatastore。

注意：当你创建清单对象时，你必须在主机能力允许的范围内，可以通过访问 HostSystem.capability 属性，它是一个 HostCapability 数据对象，例如，HostCapability 对象有明确的 maxSupportedVMs 属性。

4.4 清单权利权限需求

访问清单需要一个能连接服务器和获得一个有效的 session 的用户帐号。用户身份和调用这个 session 的当事人联系在一起。当一个客户端应用尝试访问一个清单中的对象时，服务器检查许 permission object or objects 并且和当事人的权限相比较。

例如，创建一个虚拟机需要结合 session 的规则有一下权限：

- VirtualMachine.Inventory.Create 在 folder 上创建虚拟机的权限
- Resource.AssignVMToPool 在虚拟机获得 CPU 和 Memory 资源的资源池上的权限

阅读 PerformanceManager managed 对象的 perfCounter 属性需要在 root folder 上的 System.View 权限。

注意：一些权限是针对 vCenter Server 上的对象或 ESX/ESXi 上的对象。例如，Alarm.Create 权限只是 vCenter Server 系统上结合 AlarmManager。

4.4.1 权限

一个权限是系统定义的需求结合 VMware vSphere managed 对象。权限是静态的和不可以为一个版本的产品所修改的。vSphere 组件的权限定义如下：

<group>[.<group>].privilege

For example:

Datacenter.Create

Host.Config.Connection

Host.Config.Snmp

4.4.2 许可

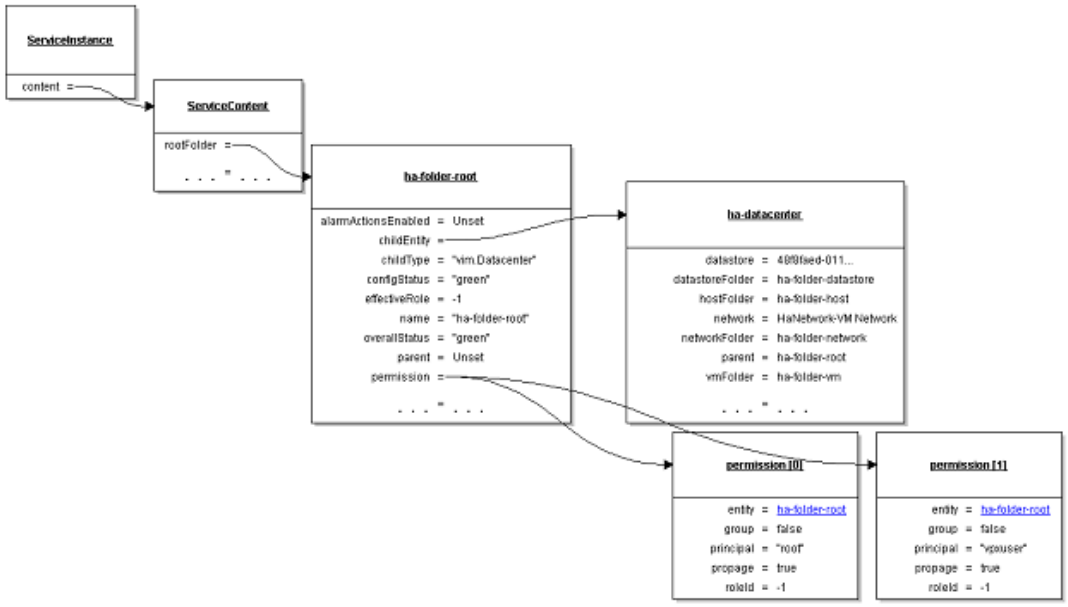
许可结合在特定的 managed 实体上的权力的规则。你使用许可到指定那个用户能访问那个 managed 实体。

一个子实体当它的父实体的 propagate 属性设置为 true 是它可以继承父实体的许可。一个在子实体上直接设置的认可将覆盖父实体的许可。为了授权认可到一个 Datatcenter 对象的所有子实体，分配许可到 Datacenter 对象和设置 Permission 对象的 propagate 属性为 true。

图 4-4 显示了 users root 和 vpxuser 都有在清单的 rootFolder 的许可。vpxuser 是当主机

添加到 vCenter Server 系统时由 vCenter Server 系统在主机上创建的帐号。vCenter Server 需要需要访问主机系统管理的清单对象，所以 vpxuser 帐号被授权到每个主机的 rootFolder。

Figure 4-4. Inventory and Permissions



4.5 被管理的和独立的 ESX/ESXi 主机

你可以已被管理或独立的方式运行 ESX/ESXi 主机

- 独立运行的 ESX/ESXi 主机是受限制的独立主机。一个独立的主机清单能支持多个虚拟机和多个资源池，但是它包含一个单一的默认 datacenter 和单一的 root folder。默认 datacenter 和 root folder 是在 vSphere 客户端上看不到的，他们存在于独立的主机清单中，他们在 MOB 中是可见的。
- 可管理的 ESX/ESXi 主机已经被添加到 vCenter Server 清单中了。可用的特征依赖于主机上有效的许可。例如，你可以定义两个或更多的主机为了 VMware DRS 资源管理或 VMware HA failover(出故障时自动备份)保护。

5、属性收集器

vSphere 服务器提供 PropertyCollector 服务来访问数据和监控改变。使用 PropertyColloctor 来获得 managed 对象的引用，managed 对象的属性值，监控和重新获得修改的属性值。

5.1 PropertyCollector 介绍

PropertyCollector 服务接口提供了一个监控和获取 managed 对象信息的道路，例如是否一个虚拟机已经开机或是否一个在集群中的主机掉线。

PropertyCollector 使用一个或多个过滤器来确定集合的范围,它有方法来检索数据。一个过滤器使用一组明确下列信息的数据对象：

- 集合操作中清单遍历的开始点。
- 清单遍历的路径。
- 对象和被采集的数据的属性。

一个 vSphere 服务器为每一个 session 创建一个默认的 PropertyCollector，并允许你创建多个，额外的 PropertyCollector 对象。创建附加的 PropertyCollector 对象，利用每个线程，来执行相互独立的集合操作。

5.1.1 数据检索

两种方式检索数据：

- 通过使用 RetrievePropertiesEx 和 ContinueRetrievePropertiesEx 方法来检索属性。这些方法执行单一的集合操作。
- 增加的属性检索，涉及到属性检测，使用 WatiForUpdatesEx 方法。对这个方法的初始调用检索一组基本的 managed 对象属性值。随后的调用检索最后一次检索后改变的属性。使用 WaitForUpdateEx 来监视清单或任何 managed 对象属性的改变。

5.1.2 清单遍历和挑选对象

PropertyCollector 过滤器的属性可以识别对象属性和对象在清单遍历中的路径。例如，你可以检索一个 VirtualMachine 对象的属性并且通过使用 VirtualMachine.network 属性明确一个检索路径来获得关联 Network 对象们的属性们。

你可以使用 vSphere view 对象(例如，ContainerView)在过滤器中来简化遍历描述。一个 View 维护一个清单对象的子集，因此假如有清单层次的改变，你不必重新创建 view。使用一个 View 来明确一组 PropertyCollector 可以为数据集合使用的对象。

5.2 属性收集相关的 vSphere 数据对象

表 5-1 提供了一个 PropertyCollector 数据对象概览。

Table 5-1. PropertyCollector Data Objects

Data Object	Description
`PropertyFilterSpec`	Provides access to object and property selection data. A `PropertyFilterSpec` must have at least one `ObjectSpec` and one `PropertySpec`.
`ObjectSpec`	Identifies the starting object for property collection. An `ObjectSpec` also identifies additional objects for collection.
`TraversalSpec`	Identifies the type of object for property collection. It also provides one or more paths for inventory traversal.
`SelectionSpec`	Acts as a placeholder reference to a `TraversalSpec`.
`PropertySpec`	Identifies properties for collection.
View objects	Identify a subset of the vSphere inventory objects.

5.3 属性收集相关的 vSphere 方法

PropertyCollector 支持下列方法来从服务器上获得对象和属性。

- 假如你的客户端应用没有和服务器状态保持同步，使用 RetrievePropertiesEx 方法。RetrievePropertiesEx 实例化一个过滤器，收集明确的对象和属性，以一个 ObjectContent 数据对象作为返回数据到你的客户端应用。服务器不添加这个过滤器到 PropertyCollector.filter 队列。服务器会在结果返回给你的客户端后销毁这个过滤器。
- 假如你的应用和服务器状态保持同步相应，使用 CreateFilter 和 WaitForUpdataesEx 方法。WatiForUpdatesEx 返回属性改变的描述，被过滤器编组的指定属性。

这两种方法，你创建一个 PropertyFilterSpec 数据对象来明确你要检索的服务器上的对象和属性。

Table5-2 显示 PropertyCollector 方法通过使用他们的上下文创建.

Table 5-2. PropertyCollector Methods

Method Context	Method	Description
Monitor properties using different filters	`CreatePropertyCollector`	Creates a new `PropertyCollector` object to monitor properties using different filters. The vSphere server handles requests for a `PropertyCollector` instance independently of any other instances of the `PropertyCollector` on the server.
	`DestroyPropertyCollector`	Destroys an instance of a `PropertyCollector` that was created by a call to `CreatePropertyCollector` from your client application.
Single collection operation	`RetrievePropertiesEx`	Retrieves property data for the specified managed objects.
	`ContinueRetrievePropertiesEx`	Retrieves additional property data for an operation started by `RetrievePropertiesEx`.
	`CancelRetrievePropertiesEx`	Cancels a `RetrievePropertiesEx` or `ContinueRetrievePropertiesEx` operation.
Incremental collection or monitoring operation	`WaitForUpdatesEx`	Retrieves changes to property data since the last `WaitForUpdatesEx` cycle. `WaitForUpdatesEx` blocks until it can satisfy the request or until the request times out. `WaitForUpdatesEx` supports chunked data transmission (see "Server Data Transmission" on page 69).
	`CancelWaitForUpdatesEx`	Cancels a `WaitForUpdatesEx` operation.
General	`CreateFilter`	Creates a new instance of a `PropertyFilter` managed object.

5.4 PropertyCollector 例程(RetrievePropertiesEx)

例子是用了一个 ContainerView 来有效的访问清单和一个包含一个 ObjectSpec，一个 TraversalSpec，一个 PorpertySpec 的 PropertyFilterSpec。程序执行下列任务：

- 接受命令行参数 vSphere server 名称(DNS 名字或 IP 地址),用户名,密码
- 连接到一个 vSphere server
- 使用一个 ContainterView 来创建清单子集;子集仅包含虚拟机.
- 为一个单独的检索操作使用 RetrievePropertyEx 方法
- 收集清单中所有虚拟机的名称并通过标准输出流打印出来.

To use the PropertyCollector for a single retrieval operation

1. 得到 ViewManager 和 PropertyCollector 的引用

   在例子中,sContent 是 ServiceContent 数据对象的变量，sContent 提供检索 vSphere 服务上的 managed object 引用提供方法。

   ManagedObjectReference viewMgrRef = sContent.getViewManager();

   ManagedObjectReference propColl = sContent.getPropertyCollector();

2. 为虚拟机创建一个 container view

   methods 是 VimPortType 对象变量.VimPortType 定义 Java 方法符合 vSphere API 方

法 .createContainerView 参数包括 (清单根目录， sContent.getRootFolder) 和 type(“VirtualMachine”)明确 ViewManager 在根文件夹开始选择虚拟机，true 值确定当超出跟目录时是否递归检测子文件路径下的虚拟机来添加到 view。Container view 提供到清单中所有虚拟机的引用。

```
List<String> vmList = new ArrayList<String>();
vmList.add(“VirtualMachine”);
ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef
    ,sContent.getRootFolder(),
    vmList,
    true);
```

3. 创建一个对象详述来定义清单导航的起始点

   ObjectSpec.obj 属性定义了起始对象(the container view).例子中只收集虚拟机数据，所以 skip 属性设置为 true 来忽略 container view 自身。

```
ObjectSpec oSpen = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);
```

4. 创建一个遍历描述来明确集合路径

   TraveralSpec 属性的 type 和 path 定义遍历信息。TraversalSpec.type 明确对象类型。TraversalSpec.path 明确在 type 对象中的一个属性。PropetyCollector 使用 path 对象来选择附加的对象。

   例子中使用了一个单一的 TraversalSpec 来在 container view 中的可用虚拟机列表中穿行。下面的代码片断指定了用于 ContainerView 对象的 TraversalSpec.type 属性和指定用于 ContainerView view 属性的 TraversalSpec.path 属性。Skip 属性设置为 false，所以 PropertyCollector 将收集 path 对象的数据(在 container view 中的虚拟机).

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName(“traverseEntities”);
tSpec.setPath(“view”);
tSpec.setSkip(false);
tSpec.setType(“ContainerView”);
```

5. 添加 TraversalSpec 到 ObjectSpec.selectSet 队列。

oSpec.getSelectSet().add(tSpec);

6. 确定要检索的属性

   例子程序创建了一个 PropetySpec 数据对象来明确要收集的属性。Type 属性被设置成 VirtualMachine 来匹配在 container view 中的对象。pathSet 属性明确一个或多个在 type 对象中的属性。

   本例中明确了 VirtualMachine.name 属性。

   PropetySpec pSpec = new PropertySpec();

   pSpec.setType(“VirtualMachine”);

   pSpec.getPathSet().add(“name”);

7. 添加对象和属性声明到属性过滤器说明中

   一个 PropetyFilterSpec 必须拥有至少一个 ObjectSpec 和一个 PropertySpec

   PropertyFilterSpec fSpec = new PropertyFilterSpec();

   fSpec.getObjectSet().add(oSpec);

   fSpec.getPropSet().add(pSpec);

8. 创建一个过滤器列表添加到其中

   List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();

   fSpecList.add(fSpec);

9. 检索数据

   通过 RetrivePropertiesEx 方法来调用一个单独的属性集合操作。例程传递一个定位好的 PropetyFilterSpec 和一个空的 options 结构给方法。默认的 RetrieveOptions.maxObjects 明确没有最大返回对象数限制。ProrpertyCollector 可以强制设置最大值。假如收集的对象大于最大值，PropetyCollector 返回一个标记在 RetrieveResult 数据对象中，这个标记通过 ContinueRetrievePropertyiesEx API 方法来检索剩下的属性。

   RetrieveOptions ro = new RetrieveOptions();

   RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);

10. 打印虚拟机名称

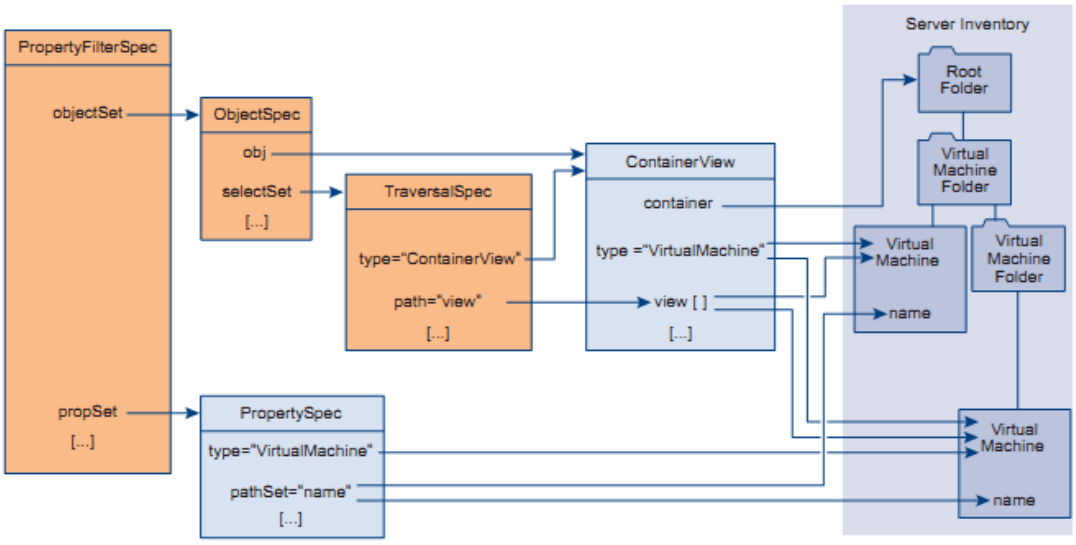
    下面的代码片断遍历在 RetrieveResult 对象中返回的 ObjectContent 对象。每一个对象(ObjectContent),在内循环中打印 name-value 对。

    If(props != null)

```
{
    for (ObjectContent oc : props.getObjects())
    {
        String vmName = null;
        String path = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if( dps != null)
        {
            for(DynamicProperty dp : dps)
            {
                vmName = (String) dp.getVal();
                path = dp.getName();
                System.out.println(path + " = " + vmName);
            }
        }
    }
}
```

图 5-1 显示了在例程中使用的对象。图表现了清单元素直接或间接定义的属性。不同的对象没有显示其全部的属性

Figure 5-1. Property Filter Specification



Example 5-1 Simple PropetyCollector Example(Java)

```
import com.vmware.vim25.*;
import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

//PropetyCollector example
//command line input: server name,user name,password
public class PCollector{
    private static void collectProperties(VimPortType method,
        ServiceContent sContent) throws Exception {
        //Get references to the ViewManager and PropertyCollector
        ManagedObjectReference viewMgrRef = sContent.getViewManager();
        ManagedObjectReference propColl = sContent.getPropertyCollector();
        //use a container view for virtual machines to define the traversal
        //- invoke the VimPortType method createContainerView(corresponds
        // to the ViewManager method) – pass the ViewManager MOR and
        // the other parameters required for the method invocation
        //-createContainerView takes a string[] for the type parameter;
        // declare an arrayList and add the type string to it
        List<String> vmList = new ArrayList<String>();
        vmList.add("VirtualMachine");
        ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
            sContent.getRootFolder(),
            vmList,
            true);
        // create an object spec to define the beginning of the traversal;
        // container view is the root object for this traversal
```

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);
// create a traversal spec to select all objects in the view
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName(“traversalEntities”);
tSpec.setPath(“view”);
tSpec.setSkip(false);
tSpec.setType(“ContainerView);

// add the traversal spec to the object spec;
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the list
oSpec.getSelectSet().add(tSpec);
// specify the property for retrieval (virtual machine name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType(“VirtualMachine”);
pSpec.getPathSet().add(“name”);
// create a PropertyFilterSpec and add the object and property specs to it;
// use the getter method to reference the mapped XML representation of the
// lists and add the specs directly to the list
PropertyFilterSpec fSpec = new ProperyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
// create a list for filters and add the spec to is
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
```

```
        RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);
        // go through the returned list and print out the data
        if (props != null)
        {
            for (ObjectContent oc : props.getObjects()) {
                String vmName = null;
                String path = null;
                List<DynamicProperty> dps = oc.getPropSet();
                if (dps != null) {
                    for (DynamicProperty dp : dps) {
                        vmName = (String) dp.getVal();
                        path = dp.getName();
                        System.out.println(path + " = " + vmName);
                    }
                }
            }
        }
    }
    // Authentication is handled by using a TrustManager and supplying
    // a host name verifier method (The host name verifier is declared in the main function)
    // For the purposes of this example,this TrustManager implementation will accept all
    // certificates. This is only appropriate for a development environment. Production code
    // should implement certificate support.
    private static class TrustAllTrustManager implement javax.net.ssl.TrustManager,
        javax.net.ssl.X509TrustManager {
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }
        public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
```

```
        }
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                String authType)
                throws java.security.cert.CertificateException {
                return;
        }
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                String authType)
                throws java.security.cert.CertificateException {
                return;
        }
}
public static void main(String[] args) throws Exception {
        // arglist variables
        String serverName = args[0];
        String userName = args[1];
        String password = args[2];
        String url = "https://" + serverName + "/sdk/vimService";
        // Variables of the following types for access to the API methods
        // and to the vSphere inventory.
        // -- ManagedObjectReference for the ServiceInstance on the Server
        // -- VimService for access to the vSphere Web service
        // -- VimPortType for access to methods
        // -- ServiceContent for access to managed object services
        ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
        VimService vimService;
        VimPortType vimPort;
        ServiceContent serviceContent;
        // Declare a host name verifier that will automatically enable
        // the connection. The host name verifier is invoked during
```

```
// the SSL handshake.
HostnameVerifier hv = new HostnameVerfier() {
    public boolean verify(String urlHostname, SSLSession session) {
        return true;
    }
};
// Create the trust manager
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
// Create the ssion context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(
    sc.getSocketFactory());
// Set the default host name verifier to enable the connection
HttpsURLConnection.setDefaultHostnameVerifier(hv);
// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message request.
vimService = new VimService();
```

```
            vimPort = vimService.getVimPort();
            Map<String, Object> ctxt = ((BingdingProvider) vimPort).getRequestContext();
            // Store the Server URL in the request context and specify true
            // to maintain the connection between the client and server
            // The client API will include the Server's HTTP cookie in its
            // requests to maintain the session. if you do not set this to true,
            // the Server will start a new session with each request.
            ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
            ctxt.put(BindingProvider.SESSION.MAINTAIN_PROPETY, true);
            // Retrieve the ServiceContent object and login
            serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
            vimPort.login(serviceContent.getSessionManager(), username,
                  password, null);
            // retrieve data
            collectProperties( vimPort, serviceContent);
            // close the connection
            vimPort.logout(serviceContent.getSessionManager()):
        }
    }
```

5.5 清单遍历

例子 5-1 使用了一个 ContainerView 来明确收集操作开始对象.这是最简单的方法来建立一个过滤器,是用一个单一的 view 引用来让 PropertyCollector 访问一系列对象。为了从清单中选择对象，一个过滤器包括 TraversalSpec 和可能用到的 SelectionSpec 对象。使用这些对象来制定对象选择基于在一个 view 里的引用，并且当超出了这些对象时(或超出了对象在 ObjectSpec.obj 中的定义)延伸清单遍历。

5.5.1 TraversalSpec 遍历

使用一个 TraversalSpec 对象来确定一个 managed 对象类型和一个在它的类型中的遍历

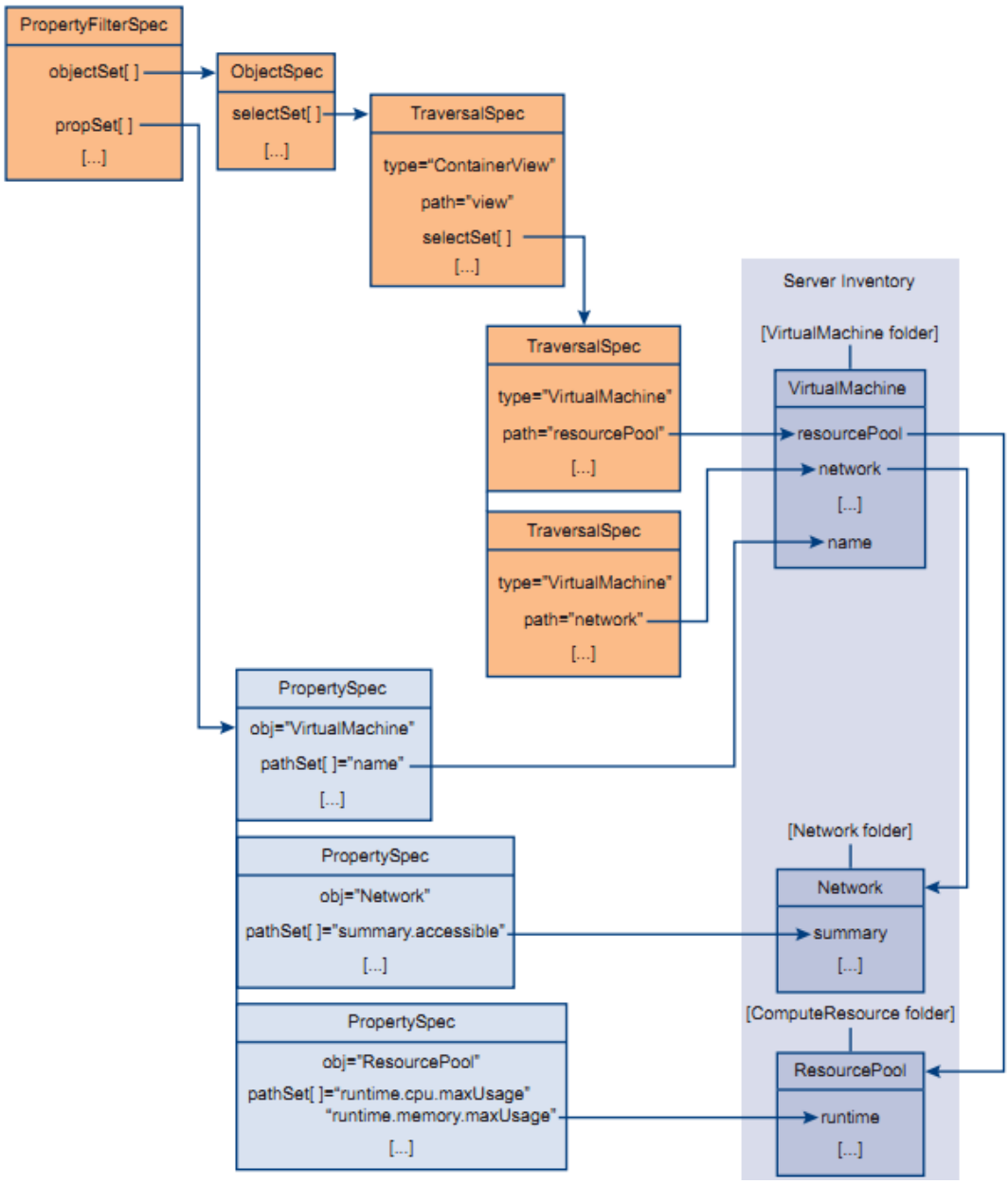
属性。TraversalSpec 包括下列属性：

- type – 确定一个清单对象类型
- path – 明确这个类型对象中的一个 managed 对象引用属性.这个属性提供了从这个对象延伸出来的遍历路径.
- selectSet – 明确了选择对象集合的选项列表为了附加对象遍历路径。PropetyCollector 在 selectSet 队列中应用 TraversalSpec 到遍历结果中(TraversalSpec.path 的目标)。selectSet 队列也可以包含 SelectionSpec 对象；SelectionSpec 是一个 TraversalSpec 的引用。
- skip – 指示是否为路径对象收集属性。

在清单遍历过程中，PropertyCollector 应用 PropertySpec 对象 or 对象集合(PropertyFilterSpec.propSet)到对象上。清单遍历开始于 ObjectSpec.obj 定义的对象和按照 TraversalSpec 的路径集合继续执行。假如 PropertySpec.type 和当前对象的类型匹配，并且 skip 属性为 false，PropertyCollector 发送 PropertySpec.pathSet 属性集合到你的客户端。

图示 5-2 陈述了定义遍历虚拟机对象的 PropertyFilterSpec。这个过滤器使用了一个 ContainerView 当作起始点。TraversalSpec 为 ContainerView 明确了 view 属性来访问视图的虚拟机。图示显示了 TraversalSpec 对象集合为和 Network 和 ResourcePool 结合的虚拟机扩展了导航。PropertyCollector 应用这些 TraversalSpec 对象集合到每一个在视图列表中的虚拟机。图示也展示了 PropertySpec 对象集合收集虚拟机，网络和资源池对象们的数据。

Figure 5-2. Inventory Navigation



例子 5-2 展示了机遇例子 5-1 的 java 代码片断

To define invertory traversal

1. 为虚拟机创建一个 ContainerView
2. 创建一个 ObjectSpec 使用 container view 为收集起始点。
3. 创建一个 TraversalSpec 应用到 ContainerView 来选择虚拟机对象。
4. 创建附加的 TraversalSpec 对象集合来选择附加对象集合。

   ContainerView 的 SelectSet 列表拥有两个 TraversalSpec 对象集合。它们都明确了一个虚拟机对象的上下文。一个对象使用网络属性来扩展遍历 Network managed 对象，另一个使用 resourcePool 属性来扩展遍历 ResourcePool managed 对象。

5. 创建 PropertySpec 对象集合来检索虚拟机，网络和资源池属性集合。

   为了检索嵌入在数据对象中的属性，PropertySpec.PathSet 属性使用”.”标记来明确属性路径。

Example 5-2 Inventory Traversal

```
import com.vmware.vim25.*
import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.soap.SOAPFaultException;
// PropertyCollector example
// command line input: server name, user name, password
public class PCollector_traversal {
    private static void collectProperties(VimPortType methods,
        ServiceContent sContent) throws Exception {
        // Get reference to the ViewManager and PropertyCollector
        ManagedObjectReference viewMgrRef = sContent.getViewManager();
        ManagedObjectReference propColl = sContent.getPropertyCollector();
        // use a container view for virtual machine to define the traversal
        // - invoke the VimPortType method createContainerView (corresponds
        // to the ViewManager method) – pass the ViewManager MOR and
        // the other papameters required for the method invocation
        // (use a List<String> for the type parameter’s string[])
        List<String> vmList = new ArrayList<String>();
        vmList.add(“VirtualMachine”);
        ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
            sContent.getRootFolder(), vmList, true);
        // create an object spec to define the beginning of the traversal;
        // container view is the root object for this traversal
```

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObject(cViewRef);
oSpec.setSkip(true);
// create a traversal spec to select all objects in the view
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName(“traversalEntities”);
tSpec.setPath(“view”);
tSpec.setSkip(false);
tSpec.setType(“ContainerView”);
// add the traversal spec to the object spec
// the accessor method (getSelectSet) returns a reference
// to the mothod XML representation of the list; using this
// reference to add the spec will update the selectSet list
oSpec.getSelectSpec().add(tSpec);
// extend from virtual machine to network
TraversalSpec tSpecVmN = new TraversalSpc();
tSpecVmN.setType(“VirtualMachine”);
tSpecVmN.setPath(“network”);
tSpecVmN.setSkip(false);
// extend from virtual machine to resourcepool
TraversalSpec tSpecVmRp = new TraversalSpec();
tSpecVmRp.setType(“VirtualMachine”);
tSpecVmRp.setPath(“VirtualMachine”);
tSpecVmRp.setSkip(false);
// add the network and resource pool traversal specs
// to the virtual machine traversal;
// the accessor method (getSelectSet) returns a reference to the
// mapped XML representation for the list; using this
// reference to add spec will update the selectSet list
tSpec.getSelectSet().add(tSpecVmN);
```

```
tSpec.getSelectSet().add(tSpecVmRp);
// specify the properties for retrieval (virtual machine name, network summary
// accessible, rp runtime props);
// the accessor method (getPathSet) returns a reference to the mapped XML
// representation of the list; using this reference to add the property
// name will update the pathSet list
PropertySpec pSpec = new PropertySpec();
pSpec.setType(“VirutalMachine”);
pSpec.getPathSet().add(“name”);

PropertySpec pSpecNs = new PropertySpec();
pSpecNs.setType(“Network”);
pSpecNs.getPathSet().add(“summary.accessible”);

PropertySpec pSpecRPr = new PropertySpec();
pSpecRPr.setType(“ResourcePool”);
pSpecRPr.getPathSet().add(“runtime.cpu.maxUsage”);
pSpecRPr.getPathSet().add(“runtime.memory.maxUsage”);
pSpecRPr.getPathSet().add(“runtime.overallStatus”);
// create a PropertyFilterSpec and add the object and
// proprety specs to it; use the getter methods to reference
// the mapped XML representation of the lists and add the specs
// dirtectly to the objectSet and propSet lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
fSpec.getPropSet().add(pSpecNs);
fSpec.getPropSet().add(pSpecRPr);
// create a list for the filters and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
```

```
fSpecList.add(fSpec);
// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,
      fSpecList, ro);
// go through the returned list and print out the data
if (props != null) {
      for (ObjectContent oc : props.getObjects()) {
            String value = null;
            String path = null;
            List<DynamicProperty> dps = oc.getPropSet();
            if (dps != null) {
                  for (DynamicProperty dp : dps) {
                        path = dp.getName();
                        if( path.equals(“name”))) {
                              value = (String) dp.getVal();
                        }
                        else if (path.equals(“summary.accessible”)) {
                              // summary.accessible is a boolean
                              value = String.valueOf( dp.getVal());
                        }
                        else if (path.equals(“runtime.cpu.maxUsage”)) {
                              // runtime.cpu.maxUsage is an xsd:long
                              value = String.valueOf( dp.getVal());
                        }
                        else if (path.equals(“runtime.overalStatus”)) {
                              // runtime.overalStatus is a ManagedEntityStatus enum
                              value = .String.valueOf( dp.getVal());
                        }
                        System.out.println(path + “ = “ + value);
```

```
                        }
                    }
                }
            }
        }
        // Authentication is handled by using a TrustManager and supplying
        // a host name verifier method (The host name verifier is declared
        // in the main function)
        // For the purposes of this example this TrustManager implementation
        // will accept all certificates. This is only appropriate for
        // a development environment. Production code should implement certificate support
        private    static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
            javax.net.ssl.X509TrustManager {
            public java.security.cert.X509Certificate[] getAcceptedIssuers() {
                return null;
            }
            public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
                return true;
            }
            public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
                return true;
            }
            public void CheckServerTrusted(java.security.cert.X509Certificate[] certs,
                String authType) throws java.security.cert.CertificateException {
                return;
            }
            public void CheckClientTrusted(java.security.cert.X509Certificate[] certs,
                String authType) throws java.security.cert.CertificateException {
                return;
            }
```

```
}
public static void main(String [] args) throws Exception {
    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String url = "https://" + serverName + "/sdk/vimService";
    // Variables of the following types for access to the API methods
    // and to the vSphere inventory.
    // -- ManagedObjectReference for the ServiceInstance on the Server
    // -- VimService for access to the vSphere Web service
    // -- VimPortType for access to methods
    // -- ServiceContent for access to managed object services
    ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
    VimService vimService;
    VimPortType vimPort;
    ServiceContent serviceContent;
    // Declare a host name verifier that will automatically enable
    // the connection. The host name verifier is invoked during
    // the SSL handshake.
    HostnameVerifier hv = new HostnameVerfier() {
        public boolean verify(String urlHostname, SSLSession session) {
            return true;
        }
    };
    // Create the trust manager
    javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
    javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
    trustAllCerts[0] = tm;
    // Create the SSL context
```

```
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance(“SSL”);
// Create the ssion context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(
      sc.getSocketFactory());
// Set the default host name verifier to enable the connection
HttpsURLConnection.setDefaultHostnameVerifier(hv);
// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType(“ServiceInstance”);
SVC_INST_REF.setValue(“ServiceInstance”);
// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message request.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BingdingProvider) vimPort).getRequestContext();
// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server
// The client API will include the Server’s HTTP cookie in its
// requests to maintain the session. if you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION.MAINTAIN_PROPETY, true);
// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
```

```
            vimPort.login(serviceContent.getSessionManager(), username,
                  password, null);
            // retrieve data
            collectProperties( vimPort, serviceContent);
            // close the connection
            vimPort.logout(serviceContent.getSessionManager()):
        }
    }
```

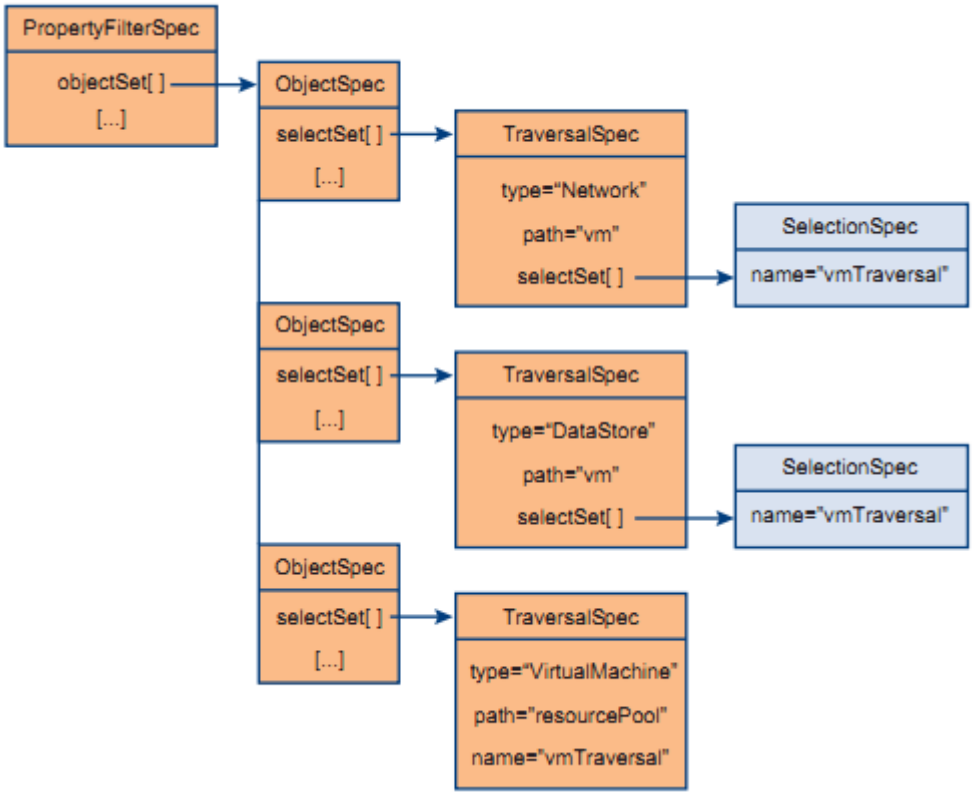
5.5.2 SelectionSpec 遍历

在 ObjectSpec 和 TraversalSpec 对象中的 selectSet 队列可以包括 TraversalSpec 对象和 SelectionSpec 对象.SelectionSpec 是 TraversalSpec 对象的基础类。SelectionSpec 定义了 name 属性。你可以在一个 selectSet 队列中使用一个 SelectionSpec 对象作为一个指定的 TraversalSpec 对象的引用。通过使用 SelectionSpec 引用，你可以重用一个 TraversalSpec 和定义递归遍历。

5.5.2.1 简单引用 SelectionSpec

使用 SelectionSpec 引用来避免 TraversalSpec 的重复声明。在 SelectionSpec 引用中明确的 TraversalSpec 必须是包含在同一个 PropertyFilterSpec 中。图 5-3 显示了 SelectionSpec 对一个虚拟机 TraversalSpec 引用的使用。SelectionSpec 引用和 Network 和 Datastore 遍历相关联。

Figure 5-3. SelectionSpec Reference



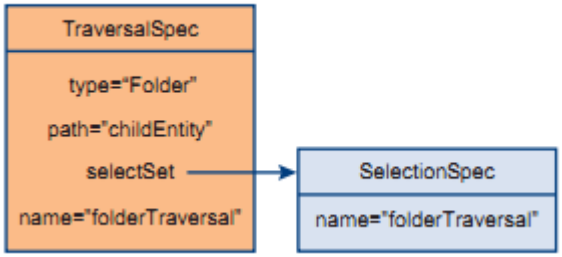
假如ObjectSpec.selectSet队列包含一个SelectionSpec,TraversalSpec引用必须明确一样的对象类型。TraversalSpec.type 必须和在 ObjectSpec.obj 中明确的对象类型匹配。PropertyCollector应用TraversalSpec到对象并且使用TraversalSpec.path属性来扩展它的遍历。

5.5.2.2 Recursive Traversal

使用一个SelectionSpec来应用一个TraversalSpec到自身遍历的结果中。使用递归过滤器结构，创建一个明确了 TraversalSpec 名称的 SelectionSpec 并且添加它到指定的TraversalSpec.selectSet中。递归结构扩展了清单遍历超过了TraversalSpec对象直接代表的路径。

你可以在任何可以嵌套的清单对象上使用递归遍历。例如，在一个 vCenter Server 上，文件夹可以位于任意的深度。为了描述一个通过连续的文件夹的遍历路径，你可以添加一个SelectionSpec到文件夹的TraversalSpec。SelectionSpec必须引用这个TraversalSpec。图5-4表示了一个TraversalSpec和它关联的SelectionSpec来嵌套文件夹遍历。

Figure 5-4. Recursive TraversalSpec and SelectionSpec



To define recursive invertory traversal

1. 使用 SearchIndex managed 对象来检索位于顶层的虚拟机文件夹的 managed 对象引用。

   这个文件夹被当作清单遍历的开始点。
2. 创建一个 ObjectSpec 对象来引用顶层的虚拟机文件夹。
3. 创建一个 SelectionSpec 对象来通过名字来引用文件夹的 TraversalSpec。
4. 为文件夹对象创建一个带有名称的 TraversalSpec。

   TraversalSpec.path 属性为遍历任何子对象明确了 Folder.childEntity 属性。
5. 添加 SelectionSpec 到 TraversalSpec 中来创建递归过滤器。
6. 添加 TraversalSpec 到 ObjectSpce 中。
7. 为文件夹名称创建一个 PropertySpec。
8. 添加对象和属性声明到 PropertyFilterSpec。
9. 调用 RetrievePropertiesEx 方法。

Example5-3 Nested Folder Traversal

```
import com.vmware.vim25.*;
import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;
// PropetyCollector example
// command line input: server name, user name, password
public class nestedTraversal {
    private static void collectorProperties(VimPortType methods,
```

```
ServiceContent sContent) throws Exception {
    // Get reference to the PropertyCollector
    ManagedObjectReference propColl = sContent.getPropertyCollector();
    // get the top-level vm folder mor
    ManagedObjectReference sIndex = sContent.getSearchIndex();
    ManagedObjectReference rootVmFolder =
        methods.findByInventoryPath(sIndex,”datacenterl/vm”);
    // create an object spec to define the beginning of the traversal;
    // root vm folder is the root object for this traversal
    ObjectSpec oSpec = new ObjectSpec();
    oSpec.setObj(rootVmFolder);
    oSpec.setSkip(true);
    // folder traversal reference
    SelectionSpec sSpecF = new SelectionSpec();
    sSpecF.setName(“traverseFolder”);
    // create a folder traversal spec to select childEntity
    TraversalSpec tSpecF = new TraversalSpec();
    tSpecF.setType(“Folder”);
    tSpecF.setPath(“childEntity”);
    tSpecF.setSkip(false);
    tSpecF.setName(“traverseFolder”);
    // use the SelectionSpec as a reflexive(反射) spec for the folder traversal;
    // the accessor method (getSelectSet) returns a reference to the
    // mapped XML representation of the list; using this reference
    // to add the spec will update the list
    tSpecF.getSelectSet().add(sSpecF);
    // add folder traversal to object spec
    oSpec.getSelectSet().add(tSpecF);
    // specify the property for retrieval (folder name)
    PorpertySpec pSpec = new PropertySpec();
```

```
pSpec,setType(“Folder”);
pSpec.getPathSet().add(“name”);
// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter method to reference
// the mapped XML representation of the lists and add the specs
// directly to the lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
// create a list for the filter and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,fSpecList,ro);
// go through the returned list and print out the data
if ( props != null) {
      for (ObjectContent oc : progs.getObjects()) {
            String folderName = null;
            String path = null;
            List<DynamicProperty> dps = oc.getPropSet();
            if (dps != null) {
                  for(DynamicProperty dp : dps) {
                        folderName = (String) dp.getVal();
                        path = dp.getName();
                        System.out.println(path + “ = “ + folderName);
                  }
            }
      }
}
```

```
        }
    // Authentication is handled by using a TrustManager and supplying
    // a host name verifier method (The host name verifier is declared
    // in the main function)
    // For the purposes of this example this TrustManager implementation
    // will accept all certificates. This is only appropriate for
    // a development environment. Production code should implement certificate support
    private    static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
        javax.net.ssl.X509TrustManager {
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }
        public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }
        public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }
        public void CheckServerTrusted(java.security.cert.X509Certificate[] certs,
            String authType) throws java.security.cert.CertificateException {
            return;
        }
        public void CheckClientTrusted(java.security.cert.X509Certificate[] certs,
            String authType) throws java.security.cert.CertificateException {
            return;
        }
    }
    public static void main(String [] args) throws Exception {
        // arglist variables
        String serverName = args[0];
```

```
String userName = args[1];
String password = args[2];
String url = "https://" + serverName + "/sdk/vimService";
// Variables of the following types for access to the API methods
// and to the vSphere inventory.
// -- ManagedObjectReference for the ServiceInstance on the Server
// -- VimService for access to the vSphere Web service
// -- VimPortType for access to methods
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;
// Declare a host name verifier that will automatically enable
// the connection. The host name verifier is invoked during
// the SSL handshake.
HostnameVerifier hv = new HostnameVerfier() {
    public boolean verify(String urlHostname, SSLSession session) {
        return true;
    }
};
// Create the trust manager
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
// Create the ssion context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
```

```
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(
    sc.getSocketFactory());
// Set the default host name verifier to enable the connection
HttpsURLConnection.setDefaultHostnameVerifier(hv);
// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message request.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BingdingProvider) vimPort).getRequestContext();
// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server
// The client API will include the Server's HTTP cookie in its
// requests to maintain the session. if you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION.MAINTAIN_PROPETY, true);
// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(), username,
    password, null);
// retrieve data
collectProperties( vimPort, serviceContent);
```

```
            // close the connection
            vimPort.logout(serviceContent.getSessionManager()):
        }
    }
```

5.5.3 客户端数据同步(WaitForUpdatesEx)

为了在客户端保持服务器对象的状态(通过监视清单属性),使用 CreateFilter 和 WaitForUpdatesEx 方法.WaitForUpdatesEx 方法支持增量检索模式.

注意:你使用的增量检索过滤器会存在整个 seesion 期间直到你销毁它。

5.5.3.1 Property Filters

PropertyCollector 可以有一个或多个相关联的 PropertyFilter 对象。PropertyFilter 有一个或多个相关联的 PropertyFilterSpec 对象。PropertyFilterSpec 在被方法 RetrievePropetiesEx 使用时有一个有限的生命期；当结果返回给客户端后服务器销毁这个过滤器。为一个增长的属性收集操作序列，WaitForUpdatesEx 方法依赖于可用于多次调用的 ProperyFilterSpec 对象。

为创建持久的属性过滤器声明，请使用 CreateFilter 方法。当你调用 CreateFilter，传递一个 PropertyFilterSpec 对象给方法。该方法添加一个新的过滤器到和方法请求相关的 PropertyCollector 中并且返回一个到新过滤器的引用。创建好过滤器后，你可以添加附加的 PropertyFilterSpec 对象。你不能和在其它 session 中的 PropertyCollertor 共享一个过滤器。

5.5.3.2 WatiForUpdatesEx

WaitForUpdatesEx 方法在特定的等待时间内支持轮询机制为属性收集。

当你调用 WaitForUpdatesEx 时要明确下列参数：

- 一个 PropetyCollector 实例的 Managed 对象引用
- version value 是一个连续的值。第一次调用 WaitForUpdatesEx 是，确定一个空字符串(“”)来检索特定属性的完整的结果结合。随后的调用必须使用前一次调用返回的版本值。假如你不包括版本值，服务器会返回所有。
- options 明确了在单一一个相应中传输的数据总量(WaitOptions.maxObjectUpdates 属性)和 PropertyCollector 应该等待更新的时间(WaitOptions.maxWaitSeconds 属性)。

WaitOptions.maxWaitSeconds 属性值决定 PropertyCollector 是采用立即检索还是轮询机制。当你使用等待时间为 0 时调用 WaitUpdatesEx，它立即检查更新并返回。当你使用大于 0 的等待时间时，方法会等待到设定的时间结束或改变发生。WaitForUpdatesEx 阻塞你的进程知道更新发生或超时。超时时间受 maxWaitSeconds 的值影响，是收集属性值更新的时间总数，并且是 PropertyCollector 策略。

假如属性收集操作超时,并且请求的属性没有更新,PropertyCollector 返回 null 值来响应 WaitForUpdatesEx.

- maxWaitSeconds 是一个可选属性，假如你没有设置一个值，PropertyCollector 会一直等待更新，因此，假如未设置 maxWaitSeconds，在检索到所有的数据后 waitForUpdatesEx 方法将阻塞线程，等待到和 vSphere 服务器的 TCP 连接超时。你的代码可以通过下列方法中的一种来处理这个情况：在单独的线程中调用 waitForUpdatesEx；查找特定的更新然后停止调用；改变 TCP 连接超时时间 (BindingProviderProperties.CONNECT_TIMEOUT)。
- maxWaitSeconds 设置为 0 代表立即调用并且响应。PropetyCollertor 检测所有被和这个 PropertyCollector 实例关联的过滤器指明的属性更新。PropertyCollector 返回任何结果或 null(没有更新)。
- maxWaitSeconds 大于 0 时循环等待。PropertyCollector 在设定的 maxWaitSeconds 时间内没有更新将返回 null。

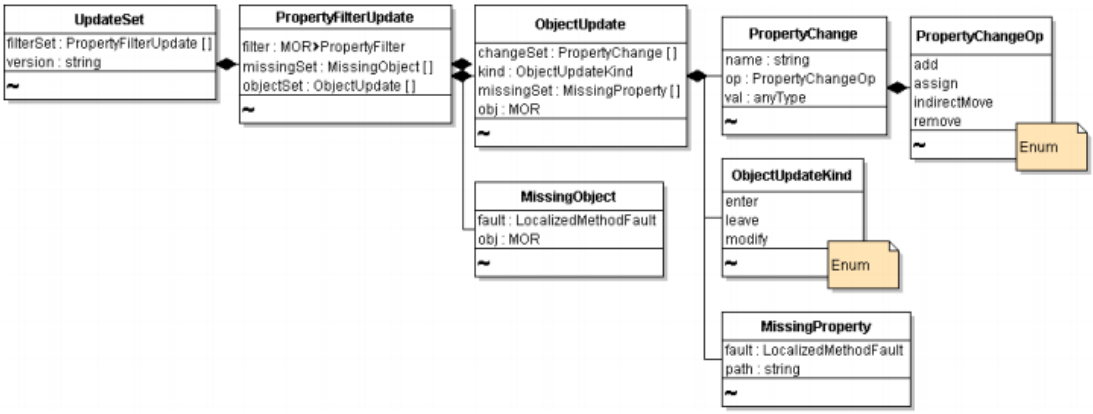
表 5-3 显示了有利的和无利的两个操作。

Table 5-3. WaitForUpdatesEx Operations Compared

Operation	Advantages	Disadvantages
`MaxWaitSeconds=0`	Returns only properties that have changed since the version specified. Returns changed data only, providing better network utilization than `RetrieveProperties`.	Returns an empty set even when nothing has changed on the server. Depending on your client application, this might be inefficient.
`MaxWaitSeconds>0`	Blocks thread until an update occurs. Efficient use of network resources. The only operation that you can cancel.	Blocks processing thread until updates occur. However, this call can be cancelled so you can monitor the time the operation is taking and cancel if necessary.

WatiForUpdatesEx 方法返回一个 UpdateSet 数据对象，这个复合数据结构如图 5-5

Figure 5-5. UpdateSet Data Object Returned by WaitForUpdates Operations



5.5.4 服务器数据传输

属性收集可能包含大量的检索数据，随着收集请求而定属性数目。vSphere 服务器在传送收集的数据到客户端时支持分段数据传送，或块传送。假如收集的数据超出了块大小，服务器在一个响应中返回一个块的数据，并且提示额外的数据可以被检索。关于块大小的信息，请参考 RetrieveOptions.maxObjects 和 WaitOptions.maxObjectUpdates 属性的描述。

- WaitForUpdatesEx 方法返回一个 UpdateSet 数据对象。UpdateSet.truncated 属性明确是否必须 WaitForUpdatesEx 再次检索额外的数据。假如 truncated 为 true，WaitForUpdatesEx 方法返回一个 version 字符串来表明块数据。当客户端程序收到一个存在额外的数据指示时，再次调用 WaitForUpdatesEx 来检索下一个块数据时必须把返回的 UpdateSet.version 字符串传递给 WaitForUpdatesEx。
- RetrievePropertiesEx 方法返回一个 RetrieveResult 数据对象。RetrieveResult.token 属性明确了是否需要调用 ContinueRetrievePropertiesEx 方法来检索额外的数据。假如 token 属性有一个值，代表分块的数据。当客户端应用接收到一个明确存在额外数据的指示时，返回的 token 必须传给接下来调用的 ContinueRetrievePropertiesEx 方法来检索下一个块数据。

Version 字符串和 tokens 是顺序的。客户端程序必须按照值得顺序轨迹。假如在收集操作中发生了个错误，重新执行操作时要使用刚才放生错误前的 version 字符串或 token。

5.5.5 PropertyCollector 性能

有下列因素影响在任何给定的 session 中的 ProeprtyCollector 性能

- 对象的个数
- 属性的个数
- 属性数据密度(复合的，嵌套的数据)
- 在服务器上的对象和属性变化的频率
- 遍历的深度(属性遍历个数)

另外，一个 vSphere 服务受 PropertyCollector 实例数和支持跨越在服务器上的所有 sessions 的过滤器实例数量影响。

对于 VirtualCenterServer2.5 来说，一个包含 2,000 个虚拟机的清单能同时支持 25 个客户端应用，可接受大概 12 个适度的复合过滤器监控在所有 2,000 台虚拟机上。假如过滤器包含复合的遍历对象，可预见性能将会降低，除非你减少穿过服务器的过滤器数量。

为了降低 PropertyCollector 系统开销和客户端网络流量，同 PropertyCollector 一起使用 view 对象。

5.5.6 SearchIndex

SearchIndex managed 对象提供了一系列方法来检索对于 vSphere 清单中 managed 对象的引用。你可以通过 managed 对象清单路径，IP 地址，数据存储路径，DNS 名称和许多其它标示属性来查找。例如，你知道一个虚拟机的 IP 地址，你可以通过 SearchIndex.FindByIp 方法获得它的 managed 对象引用。你可以使用 SearchIndex 来获得服务器对象的引用，然后使用这个引用作为属性收集的开始对象。

6、认证和授权

VMware vSphere 执行机制确保只有有效的用户可以访问虚拟架构组件。在 API 中的每一个属性和方法都和权限相关联，只有具有符合的权限才能访问实体。

6.1 认证和授权管理的相关对象

VMware vSphere 提供下列接口来为认证用户保护虚拟架构组件接口不受为授权操作的破坏。

- HostLocalAccountManager 通常被用来创建和管理在 ESX/ESXi 系统上的用户帐户

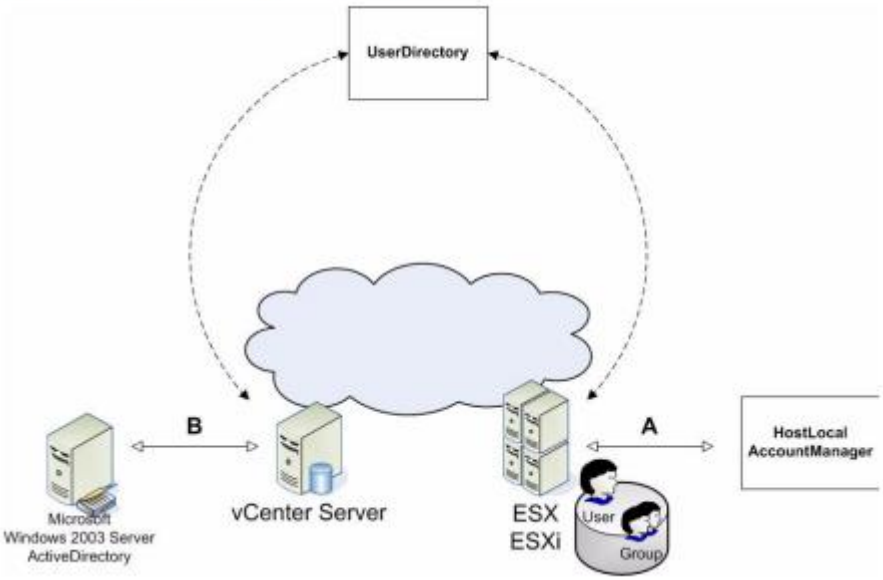
和组。已认证的用户可以在服务器上察看对象或调用操作依赖于它们的帐户权限。

- AuthorizationManager 保护 vSphere 组件防止非授权的访问。访问组件基于角色的：Users 和 groups 被赋予包括在 vSphere 对象上浏览和执行操作权限的规则。AuthorizationManager 可以创建规则，编辑规则，设置在实体上的许可，管理 managed 对象和许可之间的关系。
- UserDirectory 提供了查表机制，返回用户帐户信息到 AuthorizationManager 或另一个请求者，例如一个客户端应用。
- SessionManager 提供目标服务器系统上的认证架构接口。
  - 对于 vCenter 服务器系统，SessionManager 集成了 Microsoft SSPI(Security Service Provider Interface)，可以支持使用 Microsoft Windows 本地账户或活动域的单点登陆
  - 对于 ESX/ESXi 系统，SessionManager 支持认证定义在主机系统上的用户，例如由 vSphere Client 或由程序通过 HostLocalAccountManager API 创建的帐户。
- 即使一个用户被授权可以在一个 vSphere 对象上执行操作，当主机的许可证或特性不被允许的情况下操作会失败。你可以使用 LicenseManager 和 LicenseAssignment 来管理许可证。

6.2 ESX/ESXi 和 vCenter Server 的认证和授权

当客户端应用连接服务器时,例如 vSphere Client 或 vSphere Web Services SDK 应用，服务器端有几种交互机制来认证一个人类用户。因为 ESX/ESXi 使用基于 Linux 的认证，而 vCenter 服务器是一个 Windows 服务，俩个系统是用不同的方法来操作用户帐户。图 6-1 显示了两个不同的与 VMware vSphere 服务器相关的用户管理机制。

Figure 6-1. Managed Objects for Handling User Accounts



这些服务一起工作来保证仅有认证过的用户才能连接 ESX/ESXi 或 vCenter 服务系统，它们仅能访问的那些对象—folders，virtual machines，datacenters，virtual services，等等—具有必需的权限和被授权可以使用或浏览。

而外的，vSphere Web Services SDK 支持通过证书库自动登陆。

6.2.1 ESX/ESXi 用户模式

当用户使用他们的帐户和凭据进入客户端应用时，服务器检查正确的用户帐户库和确认真实的用户帐户和相关连的凭据。当然，凭据由密码组成，但是 vSphere 也支持证书，例如 X.509 证书。然后认证的用户可以访问授权使用的对象。如果用户身份存在于目标系统或在一个支持的目录服务上则认证成功。

ESX/ESXi 补充了标准的 Linux 架构，包括 Linux pluggable authentication module(PAM)机制为用户帐户创建和管理。VMware 认证守护(vmware-authd)作为 PAM 的一个模块运行。你可以通过 HostLocalAccountManager 来创建和管理在 ESX/ESXi 系统上的用户帐户。

6.2.2 vCenter Server 用户模式

vCenter 服务器是基于 Windows 服务的，适用本地的 Windows 功能和 Windows 用户模型来身份确认和认证。vCenter Server Web 服务关联 vCenter 服务器安装机器上的 Windows 用户帐户。vCenter 服务器的管理员帐户必须是本地 Windows Administrators 组的成员。VMware 建议创建一个专用的 Windows 用户帐号来安装和管理 vCenter 服务器系统。

其他 vCenter 服务器来连接 Web 服务的用户也必须拥有一个 Windows 帐户在本地 Administrators 组。

注意：即使一个用户在 ESX/ESXi 主机和 vCenter 服务器上拥有一样的名字，这两个用户拥有不同的帐户。

使用 Microsoft Active Directory 的组织可以使用包含在 Windows 2003Server 域控制器或 Active Directory 服务中用户身份验证来访问它们的虚拟化的基础设施。Microsoft Active Directory 验证支持所有的运行在基于 Windows 的 vSphere Web Services SDK 应用。

VMware vSphere Web Services SDK 客户端应用可以通过 SSPI 来认证到本地机器并且建立一个远程连接。

6.2.3 vSphere 安全模式

ESX/ESXi 和 vCenter Server 间的认证和授权细节不一致，自身的模型在两个系统上是一致的。它依赖于权限、角色和许可。

6.2.3.1 权限

权限是系统定义访问一个 VMware vSphere 对象的必要条件。由 VMware.Privilege 定义的权限是静态的，不能为某一个版本的产品而改变。每个 managed 对象有一个或多个主要的权限，当事人(用户、组成员)必须引用一个操作或查看一个属性。例如，managed 实体像 Folder 和 VirtualMachine 需要当事人在实体上具有 System.Read 权限来浏览它的属性值。

vSphere API Reference 包括关于调用操作和浏览属性的权限要求在 Required Privileges 标签中，文档页为每一个 vSphere 组件的 managed object.Privileges 定义如下：

<group>[.<group>].privilege

For example:

Datacenter.Create

Host.Config.Connection

Host.Config.Snmp

一个权限可以明确到 vCenter Server 或 ESX/ESXi 系统上。例如，Alarm.Create 权限可以定义在 vCenter Server.Setting 通过 AlarmManager 服务接口完成，需要一个运行的 vCenter Server 系统。

权限要求应用到系统对象不论 客户端应用如何尝试访问服务器内容(vSphere Client，CLI，SDK)。例如，你能使用下列 URL 来访问虚拟机数据存储文件：

https://<hostname>/folder[/<path>]/?dcPath=<datacenter_path>[&dsName=<datastore_name>]

URL 可以访问清单中的一个 Datastore 对象。你必须拥有访问各个层次中的对象权限，与 URL 中的元素相一致。

Table 6-1. Privileges Required for Datastore Objects Apply Regardless of Access Mechanism

Object Associated with File	URL Element	Required Privileges
Root folder	/folder	System.View
Datacenter	?dcPath	Datastore.Browse
		Datastore.FileManagement
Datastore	&dsName	Datastore.Browse
		Datastore.FileManagement
Host	/host	Host.Config.AdvancedConfig
	/tmp/	Host.Config.SystemManagement

6.2.3.2 角色

一个角色是预先定义好的一系列权限。用户通过角色被赋予到对象的权限。当你给一个用户或组分配认可时，是在配置清单中的对象和角色组合。一个单独的用户对于清单中不同的对象拥有不同的角色。

例如，假如你有两个资源池在你的清单中，资源池 A 和资源池 B，你可以配置一个特殊的用户角色 Virtual Machine User 在资源池 A 和 ReadOnly 角色在资源池 B。这些指定允许用户在资源池 A 上打开虚拟机。在资源池 B 上，用户可以浏览虚拟机状态，但不能打开虚拟机。

6.2.3.3 许可

在 vSphere 中，一个许可有一个用户或组和为一个清单对象分配的角色组成，例如一个虚拟机或 ESX/ESXi 主机。许可授权用户执行指定角色的对象活动。

例如，为一个 ESX/ESXi 主机设置内存，一个用户必须被赋予包括 Host.Configuration.Memory 权限。通过为不同的对象配置不同的角色到用户或组，你可以控制用户在你的 vSphere 环境中的操作。

6.3.4 建立用户，组和许可

建立用户，组和许可包口下列任务：

1. 得到权限需求和系统关联权限和例子角色信息。
   a) 找到 vSphere 对象操作所需的权限，察看 API Reference。
   b) 找到系统角色和例子角色可以执行的操作。
2. 假如必要，创建附加的角色。
3. 检索关于已存在的用户和组的信息。
4. 使用许可关联角色和用户、组。

在运行时，使用 SessionManager 来登录服务器。SessionManager.Login 方法需要用户 ID 和密码来确定期间可以访问服务器内容。

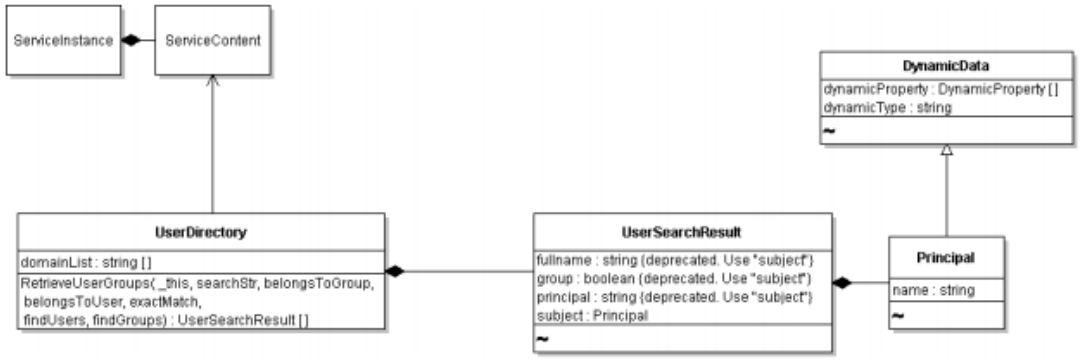
6.3 从 UserDirectory 获取用户和组信息

UserDirectory managed 对象允许客户端应用获取关于 VMware vSphere 服务器上用户和组的信息。属性和结果变化依赖于服务器是一个 vCenter Server 还是一个 ESX/ESXi 系统。

- vCenter Server system。域控制器，活动目录，本地 Windows 帐户库。
- ESX/ESXi 主机。Linux 主机上的密码文件在/etc/passwd。

例如，vCenter Server 用户帐户可以被存在于 UserDirectory 的 domainList 属性中的 Windows Acive Directory 服务器上或域控制器上管理。对于 ESX/ESXi 系统，domainList 属性为空。

Figure 6-2. UserDirectory Managed Object



UserDirectory 允许你使用 RetrieveUserGroups 方法来获得关于用户和组的信息。方法可以得到主机上的用户帐户列表，能基于明确准则来过滤结果来查找特定的用户或组。你可以通过用户名，组名来精确匹配或一部分字段来模糊匹配。

对于 ESX/ESXi 系统，查找返回在 passwd 文件中的全部用户和组。假如这个文件包含 Network Information System(NIS)或 NIS+用户和组，RetrieveUserGroups 同样返回这些帐户。

对于 vCenter Server，查找受到特定的 Windows 域限制。假如域为删减版，查找执行在本地用户和组上。

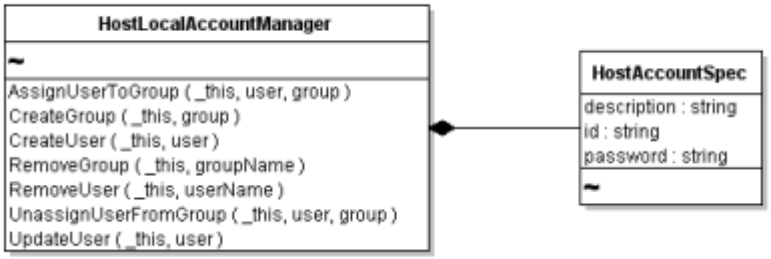
注意：不要配置 ESX/ESXi 系统使用 NIS 或 NIS+，除非它接受通过 UserDirectory.RetrieveUserGroups API 获得可用的包括 NIS(或 NIS+)用户和组的信息。

6.4 通过 HostLocalAccountManager 管理 ESX/ESXi 用户和组

HostLocalAccountManager managed 对象支持用户和组的管理任务。HostLocalManager 仅仅在 ESX/ESXi 上有效。

注意：vCenter Server 系统使用 Microsoft Windows user 管理功能。

Figure 6-3. HostLocalAccountManager Managed Object



HostLocalManager 允许你通过下列方法来管理用户和组：

- **User Management**.CreateUser,RemoveUser,UnassignUserFormGroup,AssignUserToGroup, UpdateUser
- **Group Management**.CreateGroup,RemoveGroup

对于这些方法，你必须定义 HostAccountSpec 数据对象满足目标系统上的需求。需求包括密码长度要求，使用 dictionary words 的限制，等等。

To create a user account on an ESX/ESXi system

1. 得到目标系统上 HostLocalAccountManager 的 managed 对象引用。
2. 创建一个 HostAccountSpec 数据对象来定义用户帐户的属性，包括描述和密码。

   定义帐户名称和密码符合你的 ESX/ESXi 系统的用户帐户命名规则和密码要求，例如最小长度，字符集和其他要求。
3. 调用 HostLocalAccountManager.CreateUserAccount 方法，传递 managed 对象引用(step 1)和 HostAccountSpec 数据对象(step 2)。

To assign a user to a group on an ESX/ESXi system

1. 创建一个 HostAccountSpec 数据对象定义好组的属性(描述，id)。不需要为组设置密码。
2. 调用 HostLocalAccountManager.CreateGroup 方法，传递 HostAccountSpec 数据对象。现在 user 和组都存在了，你可以添加用户到组中。
3. 调用 HostLocalAccountManager.AssignUserToGroup 方法，传递 managed 对象引用到 HostLocalAccountManager，userid，groupid 到调用中。

在 ESX/ESXi 系统上创建了用户和组后，你可以通过 AuthorizationManager 方法来授权这些用户来访问虚拟组件。

6.5 通过 AuthorizationManager 管理角色和许可

AuthorizationManager 是为处理分配给你通过 HostLocalAccountManager 定义的用户和组上的许可和角色提供的服务接口。AuthorizationManager 方法允许你创建、编辑、管理角色和许可，并能获得定义在系统内的角色和许可的相关信息。假如一个预先定义的角色不是你所需要的，定义一个新的饱含最小的你所需要的权限。

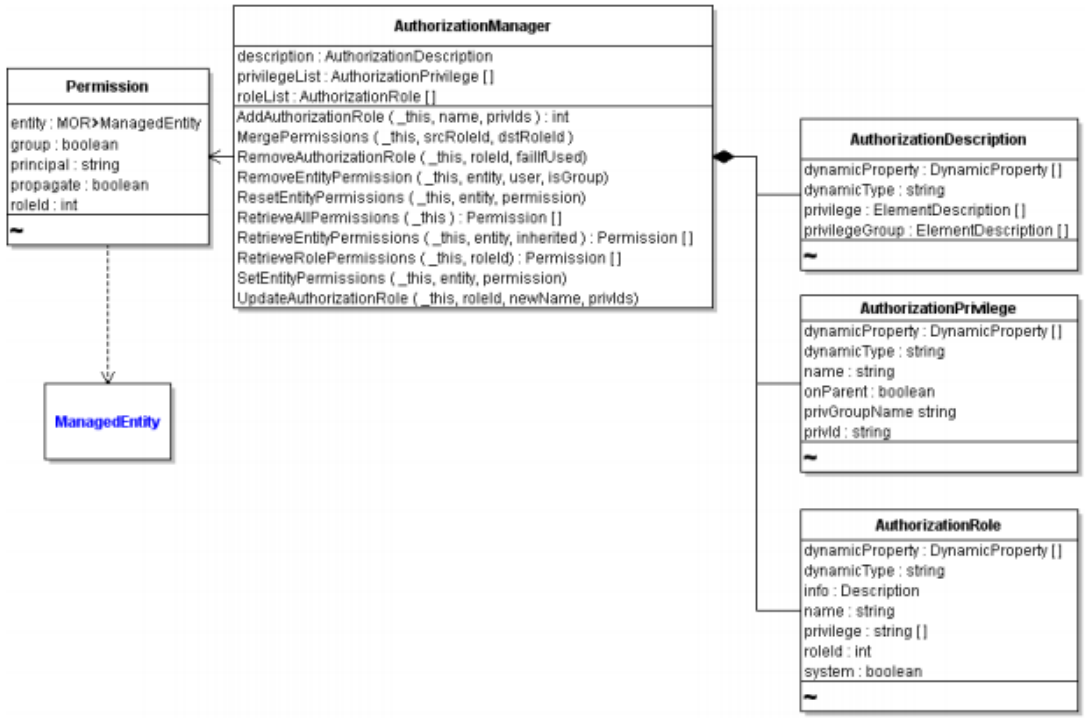
AuthorizationManager 允许访问和限制访问特殊的服务对象依赖于和这个对象相关联的许可。

AuthorizationManager 包括管理角色和许可的方法有：

- **角色管理.**AddAuthorizationRole,RemoveAuthorizationRole,UpdateAuthorizationRole。
- **许可管理.**MergePermissions， RemoveEntityPermission， ResetEntityPermissions，RetrieveAllPermissions ， RetrieveEntityPermissions ， RetrieveRolePermissions ，SetEntityPermissions。

图 6-4 显示了 AuthorizationManager 的这些方法在一个 UML 图标中，还有和它相关的数据对象。

Figure 6-4. AuthorizationManager Managed Object



AuthorizationManager 属性允许访问的信息。举例：

- privilegeList 属性返回一个定义在系统上的所有权限的列表，作为 AuthorizationPrivilege 数据对象的队列。由 VMware 定义的权限，对象和属性饱含在系统里。这些权限是固定的不可以由客户端应用修改的。
- roleList 属性返回一个所有当前正确定义的角色，包括系统定义的队列，作为 AuthorizationRole 数据对象队列。

6.5.1 使用角色来统一权限设置

一个角色是一个命名的饱含一个或更多权限的集合。一个角色通常为了一群具有共同的责任的人而定义的。例如，administrators。每一个角色可以有零个或多个权限。ESX/ESXi 定义系统角色和用户角色。

- **System roles**.不能编辑或删除。
- **User roles.** 应用到不同的用户团体或限制 add-on-tools 的访问。vCenter Server 和 ESX/ESXi 系统里饱含了几个预先定义的用户角色。你可以使用这些预先定义的用户角色当作起始点来创建新的角色。

表 6-2 描述了角色更多细节亮种类型和列出了当前可用的作为例子的角色。

Table 6-2. System and Sample Roles

Type	Role name	Role ID	Description
System Roles	Administrator	-1	Superuser access. Encompasses the set of all defined privileges. See Table D-3, "Privileges Granted to the Administrator Role," on page 224 for an example list from a vCenter Server system. This role cannot be deleted. By default, the Administrator role is granted to the user or group that owns the root node.
	Anonymous	-4	Cannot be granted. Default access role associated with any user account that has logged in.
	No Access	-5	No access. Explicitly denies access to the user or group with this role. Assigning this role to a user account prevents the user from seeing any objects. Use the `No Access` role to mask subobjects under a higher-level object that has propagated permissions defined.
	Read-Only	-2	Read-only access. Encompasses the set of all nonmutable privileges. (`System.Anonymous`, `System.Read`, and `System.View`). Equivalent to a user role with no permissions. Users with this role can read data or properties and call query methods, but cannot make changes to the system.
	View	-3	Visibility access consisting of `System.Anonymous` and `System.View` privileges. Cannot be granted.
Sample Roles	Virtual Machine Administrator	1	Set of privileges necessary to manage virtual machines and hosts within the system.
	Datacenter Administrator	2	Set of privileges necessary to manage resources, but not interact with virtual machines.
	Virtual Machine Provider	3	Set of privileges necessary to provision resources.
	Virtual Machine Power User	4	Set of privileges for a virtual machine user that can also make configuration changes and create new virtual machines.
	Virtual Machine User	5	Set of privileges necessary to use virtual machines only. Cannot reconfigure virtual machines.
	ResourcePool Administrator	6	Available on vCenter Server systems only.
	VMware Consolidated Backup Utility	7	Available on vCenter Server systems only. Set of privileges necessary to run the Consolidated Backup Utility.

6.5.2 编辑例子角色来创建新的角色

列在表 6-2 里系统角色不能被修改或删除。然而，你可以创建新的角色，或编辑例子角色。

使用 API 来创建新的角色

1. 由在 ServiceInstance.content 里的 ServiceContent 对象开始，取得到服务器上 AuthorizationManager 的一个 managed 对象引用。
2. 调用 AddAuthorizationRole 方法。参数是一个 AuthorizationManager 的引用，一个角色的名称(字符串)，应该分配给角色的权限队列(字数串队列)。

   AddAuthorizationRole 返回一个整形(xsd:int)数值代表着 roleId，系统分配给新定义角色的。
3. 在后面的代码中，使用这个 roleId 分配这个角色给明确的用户或组。

6.5.3 通过许可来授予权限 Granting Privileges Through Permissions

当你使用一个 AuthorizationManager 对象来分配或编辑许可时，你使用一个 Permission 数据对象。Permission 通过一个当事人和一系列权限相关联。一个 Permission 确定了：

- 这个许可应用的用户或组(准则)。
- 角色包含应该赋权给用户或组的权限。
- 到许可应用的实体上的 managed 对象引用。

每一个 managed 实体至少有一个和它相关联的 Permission 对象。一个 managed 实体可能有多个 Permission 分配给它。有效的授予不同的权限到不同的用户或组。许可可以被 managed 实体明确定义或通过继承。

6.5.3.1 Obtaining information About Permissions

具有管理员角色的用户可以获得关于许可的信息在不同的细节级别。

- 对于一个许可对象队列，调用 AuthorizationManager.RetrieveAllPermissions 方法。
- 对于明确的清单对象，例如 managed 实体，folders，datacenter，virtual services，调用 AuthorizationManager.RetrieveEntityPermissions 方法。
- 对于定义在系统里的角色，调用 AuthorizationManager.RetrieveRolePermissions 方法。

6.5.3.2 Setting,Changing,or Deleting Permissions

Permission 数据对象关联于当事人(用户，组)在某个对象上执行操作所需要的权限。当事人通过他们的角色来拥有权限。在对象上设置或更新许可，使用 AuthorizationManager.SetEntityPermissions 方法。

在实体上设置许可：

1. 通过关联与 ServiceInstance 的 ServiceContent 对象获得一个关于服务器的 AuthorizationManager 引用。
2. 创建一个 Permission 数据对象来定义用户(或组)的名称，角色，许可应该使用在得实体,和是否许可应用在实体的字对象上。

   举例，下列代码片断在清单中的 root 目录上创建一个许可赋给一个管理员用户角色到 root 目录和它的子目录。

```
Permission per = new Permission();
per.setGroup(false);
per.setPrincipal("new_user_name");
per.setRoleId(-1);
per.setPropagate(true);
per.setEntity(rootFolder);
```

许可不能对复合实体直接设置。对于复合实体，设置许可在父实体上和设置 propagate 标志为 true 来应用许可到子实体上。

是用一套新的许可替换已有的许可，使用 AuthorizationManager.ResetEntityPermissions 方法。

6.5.3.3 Impat of Group Membership on Permissions

用户可以是多个组的成员。系统按照下列方式处理属于多个组的成员：

- 许可被应用到清单对象从包含对象到每一个它的实体。
- 假如一个用户没有明确的用户级别许可，组级别许可直接赋予给用户。
- 多组成员拥有在所属组在同一对象许可的合集。
- 用户级别的许可一般优先于组级别许可。

6.5.3.4 Applying Permission to a Managed Entity

例子 6-1 显示了创建用户帐户和应用一个许可到一个实体来基于角色赋予用户帐户访问权限。在例子中分配角色 ID 为 4，定义为“Virtual Machine Power User”。例子是用 AuthrizationManager 来把许授予用户并且关联许可和清单中的 managed 实体，例子中实体为 rootFolder。

Example 6-1. Creating a User Account

```
…
ManagedObjectReference _authManRef = _sic.getAuthorizationManager();
public class CreateUser {
    private static AppUtil appUtil = null;
    private void createUser() throws Exception {
```

```
        ManagedObjectReference hostLocalAccountManager =
                appUtil.getConnection().getServiceContent().getAccountManager();
        ManagedObjectReference hostAuthorizationManager =
                appUtil.getConnection().getServiceContent().getAuthorizationManager();
        // Create a user
        HostAccountSpec hostAccountSpec = new HostAccountSpec();
        hostAccountSpec.setId(userName);
        hostAccountSpec.setPassword(password);
        hostAccountSpec.setDescription(“ my delegated admin auto-agent software”);
        appUtil.getConnection().getService().createUser(hostLocalAccountManager,
            hostAccountSpec);
        ManagedObjectReference rootFolder = appUtil.getConnection().getServiceContent().
            getRootFolder();
        Permission permission = new Permission();
        Permission.setPrincipal(userName);
        // Assign the Virtual Machine Power User role
        permission.setRoleId(4);
        permission.setPropagate(true);
        permission.setEntity(rootFolder);
        appUtil.getConnection().getService().setEntityPermissions(hostAuthorizationManager,
            rootFolder, new Permission [] {permission});
    }
}
```

6.6 通过 SessionManager 验证用户

SessionManager managed 对象控制用户访问服务器。SessionManger 包括登陆服务器，获得一个 session，退出等方法。SessionManager 定义了许多对象的生命周期和可见度。Session-specific 对象在创建他的 session 外是不可见的。

注意：每一个用户 session 使用系统资源来在服务器端创建锁。同时存在很多 sessions

可能导致服务器性能下降。默认，vCenter 服务器在 30 分钟后结束一个 session。

当一个用户帐户成功认证后，SessionManager 返回一个 UserSession 数据对象到客户端应用。在 session 存活期间 session 和用户帐户相关联。客户端应用可以保存 session 到本地一个安全文件中，在以后或重新连接服务器时使用。你也可以配置一个 ESX/ESXi 或 vCenter 服务器来支持本地 sessions，支持用户使用主机上的凭证基于那些权限来登陆。

SessionManager 提供下列能力：

- **Log in and log out(登陆和退出).**陆到 ESX/ESXi 或 vCenter 服务器系统，获得一个 session，退出是基本操作。当一个 session 结束时，所有的 session-specific 对象都将被销毁。
- **Impersonation(扮演).**一个用户 session 采用另一个用户 session 的认证级别。扮演通常在 Web 中基于作为一个和其他后端服务器或进程交互的中心账户的中间层应用函数的脚本。当访问代表客户端资源时 Windows 服务扮演一个客户端。SessionManager 支持扮演通过它的 ImpersonnateUser 方法。
- **Delegation(委 托).** 一 个 代 表 本 地 用 户 运 行 的 客 户 端 应 用 可 以 调 用 SessionManager.AcquireLocalTicket 方法来获得一个一次性的用户名和密码来登陆。委托对于运行在本地控制台上的基于主机的功能是很有用的。

假如和 session 关联的用户帐户没有执行一个动作的必要许可，AuthroizationManager 返回一个 NoPermission 错误到客户端应用。

6.7 使用证书库来自动登陆

为了帮助无人监控应用自动登录,vSphere Web Services SDK 包括了客户端证书存储开发库和工具提供一个更安全的方式来实现自动登录过程。开发库消除了系统管理员要保存密码在本地脚本中的需求。

注意：这些开发库建立在 vSphere Web Services SDK 上。

证书存储拥有下列组件：

- 一个持久的文件(证书存储备份文件)来存储认证证书。当前，仅有密码被支持。持久文件映射一个 ESX/ESXi 主机上的远程用户到这个服务器上的一个用户的密码上。
- C#，Java 和 Perl 开发库可以通过编程来管理证书存储。

- Java 和 Microsoft PowerShell-基于命令行的功能也可以来管理证书存储。

表 6-3 里列出了额外的开发库，vSphere Web Servcies SDK 包括 CredentialStoreAdmin 工具来创建，测试，管理证书存储。你可以使用工具来测试证书库的内容，例如，产生用户帐户和密码。

假如你使用证书存储客户端开发库，显示在表 6-3 中，你必须在运行你的应用得客户端机器上建立证书存储。

Table 6-3. Credential Store Client Libraries

Package com.vmware.security.credstore (Java)	Namespace VMware.Security.CredentialStore(C#)
CredentialStore.java	CredentialStoreFactory.cs
CredentialStoreFactory.java	CredentialStore.cs

6.7.1 证书库方法

Table 6-4. Credential Store Client Methods

Java	C#	Description
`addPassword(hostname, username, password)`	`AddPassword(hostname, username, password)`	Stores the password for the specified host and user. Overwrites any existing password for that user in the credential store. Creates the default credential store backing file in the default location (if it does not exist).
`removePassword(hostname, username)`	`RemovePassword(hostname, username)`	Deletes the password for the specified user from the credential store.
`clearPasswords()`	`ClearPasswords()`	Deletes all passwords from the credential store.
`getPassword(hostname, username)`	`GetPassword(hostname, username)`	Returns the password for the specified host and user from the credential store.
`getHosts()`	`GetHosts()`	Returns the set of hosts contained in the credential store.
`getUsernames(hostname)`	`GetUsernames(hostname)`	Returns the collection of all user names that have passwords stored for the specified `hostname`.
`close()`	`Close()`	Closes the credential store, preventing further method invocations. Releases associated resources.

6.7.2 证书库备份文件

证书存储备份文件是一个 XML 文件被保存在客户机本地以便运行时访问。除非明确的指明，一般备份放到下列位置：

- Linux.$HOME/.vmware/credstore/vicredentials.xml
- Windows Vista.

C:\Users\[user_name]\AppData\Rooming\VMware\credstroe\vicredentials.xml

- Windows XP and Windows 2000.

  C:\Documents and Setting\[user_name]\Application Data\VMware\creadstore\vicredentials.xml

证书存储基于每个用户保留在本地—每一个用户有他或她自己的证书存储备份文件。

注意：证书存储备份文件使用文件系统级别的许可来确保密码保持机密性。使用适当的文件许可来保护证书存储备份文件。

例子 6-2 显示了读取和写入 XML 元素到文件

Creadential Store File Format

```
<?xml version=”1.0” encoding=”UTF-8”?>
    <viCrendentials>
        <version>1.0</version>
        <passwordEntry>
            <host>mi6.vmware.com</host>
            <username>agent007</username>
            <password>IhWSlsaIhtsw2FbIh0w2F2</password>
        </passwordEntry>
        <passwordEntry>
            ….
        </passwordEntry>
    </viCrendentials>
```

6.7.3 证书库举例

CreateUser 和 SimpleAgent 例子程序展示了如何使用证书存储客户库：

- CreateUser 例子基于随机数生成方案为服务器创建一个用户账户和密码。例子用这个信心填写本地的证书存储备份文件。假如备份文件不存在，它将在默认位置创建它。

  当你运行 CreateUser，明确一个 ESX/ESXi 系统的名称，管理用户名和密码。一个在服务器上创建的用户账户名称和密码。明确的忽略证书--除非你的系统有安全的

连接到目标。不要使用—忽略证书在生产环境。

java com.vmware.samples.simpleagent.CreateUser –server <servername> --url https://<servername>/sdk --username <adminuser> --password <pwd> --ignorecert ignorecert

注意：CreateUser 例子程序是为了展示目的不应该在生产代码中使用。例子打破了系统管理员角色的最少权限被赋权给用户账户的原则。永远不要在一个生产环境中使用。

- SimpleAgent 例子程序展示了如何使用证书存储库函数来在运行时提取用户账户和密码来以非交互方式使用户生效。

  java com.vmware.samples.simpleagent.SimpleAgent <servername>

6.7.4 使用证书库指定角色和用户

VMware 推荐应用最少的权限到任何代理软件或在生产环境中使用证书库自动登录的应用程序。根据用户要做的任务分配给用户账户在系统上最小的权限。

像下面明确角色和用户：

1. 为每一个基于 SDK 的应用，使用一个明确的拥有合适权限的角色，新创建或重新定义。

   例如，假如你开发一个类似代理应用自动启动 VMware Consolidated Backup 功能，你可以使用“VMware Consolidated Backup Utility”角色(roleID7)。

   假如在你的应用中没有找到合适的预先定义的用户角色，创建一个应用所需要权限的角色。
2. 为代理或应用程序创建一个用户账户。
3. 应用在第一步创建的角色到在第二步创建的用户账户上。
4. 存储用户账户和密码到证书存储中，使用 CredentialStoreAdministration tool。

永远不要授权管理员权限到一个关联自动脚本或软件代理用户账户，尤其是使用证书存储的。

6.8 使用 LicenseManager 来管理许可证

当你要在 vSphere 环境中执行任务时，你必须拥有许可证才可以做。许可应用到

ESX/ESXi 主机，vCenter 服务器，明确特征的例如 VMware HA 或 VMware vMontion。

vSphere Datacenter Administration Guide 解释了怎样通过使用 vSphere 客户端管理 ESX/ESXi 和 vCenter 服务许可证，并且给出了关于许可证密钥的背景信息，和相关的文章。

你也可以使用 LicenseManager 和 LicenseAssignmentManager managed 对象来管理许可证。你可以使用 LicenseManager 来明确的管理 vSphere4.0 以前在 ESX/ESXi 系统上发布的可用的许可证池。你可以使用 LicenseAssignManager，通过有效的 LicenseManager.LicenseAssignmentManager 属性获得，来管理分配给 vCenter 服务器清单实体的许可证。你能检索信息，添加许可证，和移除许可证。

检索信息

- 检索 LicenseManager.evaluation 和 LicenseManager.licenses 属性来获得关于评估许可证和完全许可证的信息
- 调用 LicenseManager.DecodeLicense 来解码许可证信息。调用返回一个 LicenseManagerLicenseInfo 数据对象，封装了许可证的信息。
- 调用 LicenseAssignmentManager.QueryAssignedLicenses 来获得关于已分配许可证的信息。

添加许可证

- 调用 LicenseManager.AddLicense，传递一个许可证密钥，来添加一个许可证到有效的许可证清单中。
- 调用 LicenseAssignmentManager.UpdateAssignedLicense，传递一个许可证密钥，来为一个实体更新许可证，例如，一个主机系统。

移除许可证

- 调用 LicenseAssignmentManager.RemoveAssignedLicense 来移除一个实体上的所有许可证，传递需要移除许可证的实体。你随后可以把这些许可证分配给其他的实体。
- 调用 LicenseManager.RemoveLicense，传递一个许可证密钥，来从有效的许可清单中移除一个许可证。

7、主机

在你的 vSphere 环境中有许多操作调用配置运行在虚拟化层上的 ESX/ESXi 主机。你可

以配置存储和网络，并且这些设置直接影响虚拟机。你同时必须管理主机的其他方面。

7.1 主机管理对象

vSphere Web Services SDK 包括几个主机管理对象。

核心的管理对象是 HostSystem。HostSystem 的每个属性是一个封装了一些关于主机信息的数据对象。例如，capability 属性是一个 HostCapability 对象，runtime 属性是一个 HostRuntimeInfo 对象。

HostSytem 方法允许你在主机上执行某些任务。然而，许多任务不能通过 HostSystem 方法来执行，但是可以通过关联到 HostSystem 的 managed 对象里的方法来执行。例如，你要管理主机时间可以使用 HostDateTimeSystem 和要管理内核模块可以使用 HostKernelModuleSystem。

7.2 检索主机信息

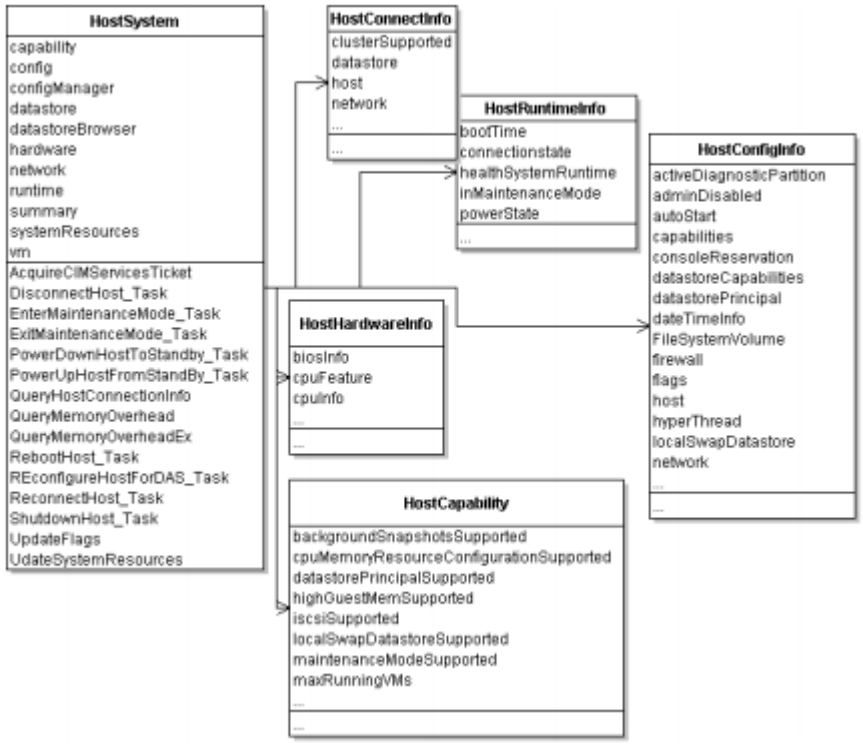
你通过访问为 HostSystem 定义的数据对象来检索关于主机的信息。

- HostSystem.capability 是一个 HostCapability 对象。HostCapability 属性说明了主机支持的特性。例如，maintenanceModeSupported 或 recursiveResourcePoolsSupported。
- HostSystem.runtimeInfo 是一个 HostSystemRuntimeInfo 对象包含几个详细描述当前主机状态信息的对象。例如，你可以通过 HealthSystemRuntime 对象获得健康状态或通过 HostPowerState 对象来获得电源状态。
- HostSystem.hardware 是一个 HostHardwareInfo 对象，允许你来检索主机的硬件配置信息包括 CPU 和 NUMA 信息和内存大小。
- HostSystem.config 是一个 HostConfigInfo 对象。这个数据对象封装了一套主机配置信息用来显示和配置一个主机。你只能在连接主机后访问 managed 主机上的 HosConfigInfo 对象。

HostSystem 有几个附加的属性允许你直接访问虚拟机，数据存储，和系统关联的网络。

QueryHostConnectionInfo，QueryMemoryOverhead，QueryMemoryOverheadEx 方法都可用来信息检索。

Figure 7-1. HostSystem and Information Properties



7.3 配置和重新配置主机

当你配置或重新配置一个 ESX/ESXi 主机时，你通常不直接使用 HostSystem 中的方法，而是通过对系统一部分配置有效的 managed 对象来设置。例如，HostNetworkSystem 允许你来配置网络，HostAuthorizationManager 管理用户，组和主机上的许可。

一些方法定义在本地 HostSystem 上。

- **CIM 管理**- AcpuireCimServicesTicket.额外关于使用 vSphere CIM 的信息，请查看 VMware CIM APIs 文档。
- **主机生命周期** – RebootHost_Task，ShutdownHost_Task，PowerDownHostToStandBy_Task，PowerUpHostFromStandBy_Task，DisconnectHost_Task，ReconnectHost_Task。
- **维护模式** – EnterMaintenanceMode_Task，ExitMaintenanceMode_Task。
- **更新**– UpdateFlags，UpdateIpmi，UpdateSystemResources。

7.4 管理主机生命周期

一个主机的生命周期依赖是否主机是独立主机或有 vCenter Server 系统管理的。

7.4.1 重启和关机

你可以重起和关闭被管理和独立的主机。ShutdownHost_Task 方法不是在所有主机上都支持。检查主机 shutdownSupported 的能力。

你可以使用一个 force 参数调用两个方法，明确是否来重起主机不论虚拟机是在运行当中或其他操作正在主机上运行。假如你设置参数为 false，当主机在维护模式时主机只能被重起。

- ShutdownHost_Task – 关闭一个主机。假如是直接连接主机，客户端不会得到任务成功的返回指示，但是会暂时失去到主机的联接。假如方法不成功，一个错误将被返回。
- RebootHost_Task – 重新启动一个主机。假如命令成功，然后主机重新启动。直接连接主机的客户端 不会接受到任务成功的指示，但是会暂时性的失去到主机的连接。假如方法不成功，一个错误将被返回。

7.4.2 使用待机模式

待机是在主机不支持运行或虚拟机加电时的一个电源状态。VMware 电源管理模块可以撤离和设置主机进入待机模式来节省电量。主机可以通过使用 PowerUpHostFromStandBy_Task 来远程加电。

下列方法支持待机模式。两个方法可以取消。

- PowerDownHostToStandBy_Task – 使主机进入待机模式，处于待机模式的主机可以通过远程上电。这个命令仅在主机的 standBySupported 为 true 时才被支持。

  当任务在运行时，没有虚拟机可以被加电没有配置任务可以在主机上执行。

  调用这个方法不能直接发起任何操作来撤离或下电已经开机的虚拟机。然而，假如 VMware DRS 开启，vCenter Server 迁移一个关机的虚拟机或建议迁移到一个不同的主机，建立在自动级别。假如主机是一个群集的一部分这个任务由一个具有方法 evacuatePowerOffVms 参数设置为 true 的 vCenter Server 目标发起。除非所有的关

机状态的虚拟机都在其他的主机上注册，否则任务不会成功。

- PowerUpHostFromStandBy_Task – 主机脱离待机模式。假如命令成功，主机唤醒并且开始发送心跳。方法可以被自动调用该能力由 VMware DRS 添加到集群中，假如主机没有在维护模式下。

7.4.3 断开连接和重新连接主机

你可以添加一个主机到 vCenter Server 系统使其成为一个 managed 主机。你可以随后断开和重新连接主机，例如，刷新代理。

你可以使用下列方法，由通过 vCenter Server 系统来访问主机方式来支持的。

- QueryHostConnectionInfo – 返回一个 HostConnectionInfo 对象，和通过 Datacenter.QueryConnectionInfo 返回的是同一个对象。对象中的信息可以被一个连接向导使用，类似于 vSphere 客户端中使用的向导。
- DisconnectHost_Task – 从一个主机断开连接并且通知 vCenter Server 系统来停止发送心跳到这个主机。
- ReconnectHost_Task – 重新连接主机到 vCenter Server。这个进程重新安装代理和重新配置主机，假如它已经和服务器脱离同步。重新连接进程检查正确的许可证和这个主机上的 CPUs 的数量，确定正确的代理套数被安装，确定网络和数据存储被发现和注册在 vCenter Server 系统上。

客户端应用程序可以改变 IP 地址和主机的端口当执行一个重新连接操作时。当客户端想保留已经存在的原数据时这个功能很有用的，例如统计，警告，权限，甚至主机正在更改 IP 地址。

7.4.4 查询和改变主机时间

HostDateTimeSystem 支持日期和时间关联设置在一个主机上并且支持 NTP 设置。

HostDateTimeSystem.dateTimeInfo 属性允许你来检索和设置日期和时间信息。

HostDateTimeInfo 数据对象的属性包含两个数据对象用来日期时间的管理。

- HostNTPConfig 包含一个可供主机使用的 NTP 服务器列表。
- HostDateTimeSystemTimeZone 明确了时区包括 GMT 时差，时区的标识符和名称。

你还可以通过调用一个 HostDateTimeSystem 方法查询主机时间信息

- QueryAvailableTimeZones – 检索主机上可用的时区列表。方法是用公共的域 tz 时区数据库。方法返回一个 HostDateTimeSystemTimeZone 对象队列。
- QueryDateTime – 返回当前主机的日期和时间。

你可以编辑主机的日期时间信息通过调用下列 HostDateTimeSystem 中的一个方法

- RefreshDateTimeSystem – 刷新日期和时间关联设置到任何可能发生的改变。
- UpdateDateTime – 更新主机上的日期和时间通过把日期和时间传递到方法中。使用警告。网络延迟或操作延迟导致时间扭曲。
- UpdateDateTimeConfig – 更新主机上日期和时间配置。你可以调用这个方法通过一个 HostDateTimeConfig 参数，允许你来确定 NTP 配置和时区。

7.4.5 查询虚拟机内存的 Overhead

每一个加电的虚拟机需要一定数量的内存供它自己使用。另外，主机必须拥有一些内存开支为每个可用的虚拟机。为了找出内存开销，可以调用 HostSystem.QueryMemoryOverheadEx 方法。这个方法使用一个 virtualMachineConfigInfo 数据对象作为一个参数，确定加电的具有哪些特性的虚拟机必须的开销数量。

该方法返回内存需求总数，字节表示。

8、存储

一个虚拟机使用一个虚拟磁盘来存储它的操作系统，程序文件和其他数据。一个虚拟是一个大的物理文件，或一组文件，他们可以像其他文件一样被拷贝，移动，存档和备份。为了存储和操作虚拟磁盘文件，主机需要专用的存储空间。ESX/ESXi 支持多种方式的存储。被 vCenter Server 管理的主机可以共享存储。

8.1 存储管理对象

你可以通过访问 HostSystem managed 对象来支持存储管理。

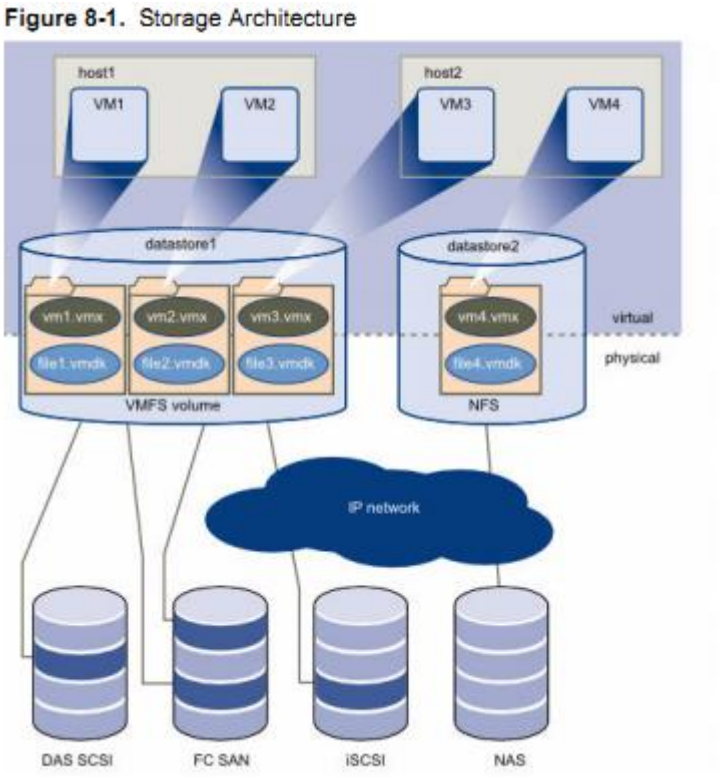
- HostStorageSystem – HostSystem.storageSystem 属性是一个到 ESX/ESXi 系统的 HostStorageSystem 的 managed 对象引用。HostStorageSystem 是一个低级别的接口通常使用来配置物理存储。

- HostDatastoreSystem – HostSystem.datastoreSystem 属性是一个 HostDatastoreSystem managed 对象的 managed 对象引用。HostDatastoreSystem 的方法允许你创建，配置，扩展和移除数据存储。HostStorageSystem 支持访问和配置物理存储，HostDatastoreSystem 支持访问和配置逻辑存储通过主机可以使用虚拟机上的卷(Datastore managed 对象)。
- HostDatastoreBrowser – 提供到一个或多个数据存储的访问。数据存储上的细项是文件包括了配置文件，虚拟机磁盘文件和其他个和虚拟机关联的数据文件。
- Datastore – Datastore managed 实体提供挂载数据存储，浏览数据存储和获得和数据存储相关联的虚拟机的信息地方法。
- HostDiagnosticPartition – 支持创建和查询你的 ESX/ESXi 主机的诊断分区。

8.2 存储介绍

VMware vSphere 存储架构包含隐藏和管理复杂的和不同的物理存储子系统的虚拟化层，如图 8-1。

Figure 8-1. Storage Architecture



8.2.1 虚拟机如何访问存储

虚拟机为他们的操作系统，应用软件和其他数据文件使用虚拟磁盘，一个虚拟磁盘是以一个 VMDK 文件形式存储在数据存储上。虚拟磁盘对于虚拟机的操作系统隐藏了物理存储层。不论你的主机使用的什么型号的存储设备，虚拟磁盘对于虚拟机总是以一个本地的 SCSI 设备出现。因此，你可以在虚拟机中运行非特定存储设备的操作系统，例如 SAN。

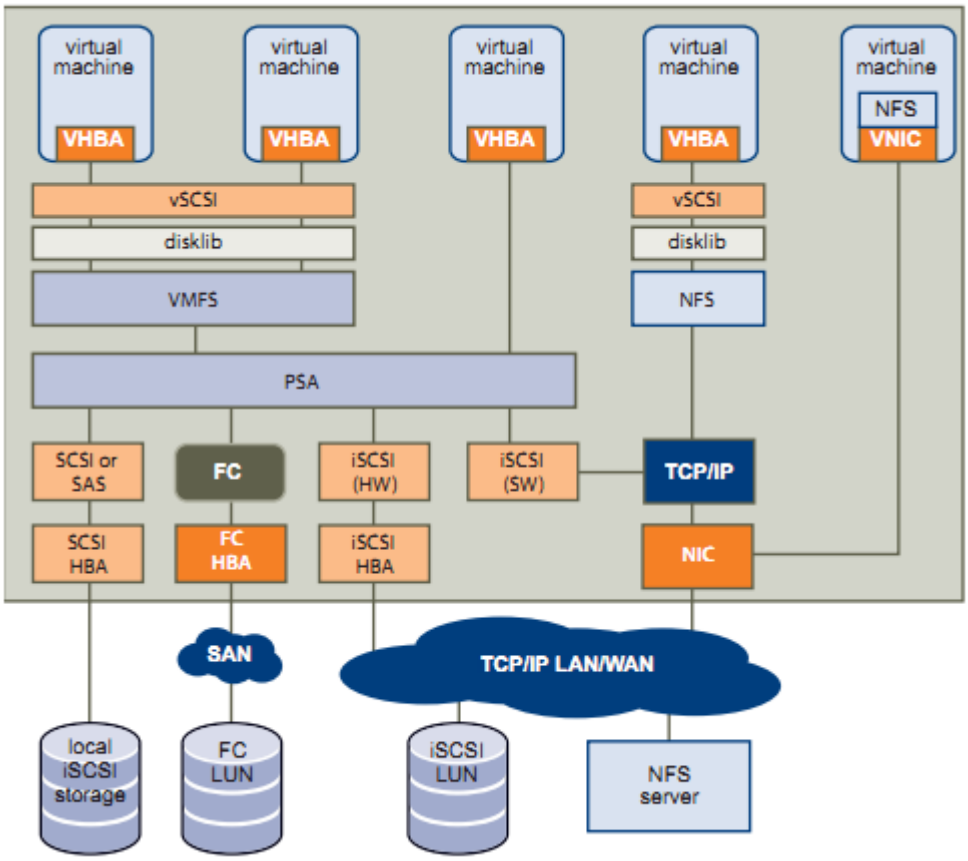
当一个虚拟机和存储在数据存储上的虚拟磁盘文件通讯时，它发出 SCSI 命令。因为数据存储可能存在于不同类型的物理存储上，这些命令被打包成其他的形式，依赖于 ESX/ESXi 主机和物理存储设备连接所采用的协议。

为了运行在每一个虚拟机上的应用程序和客户操作系统，存储子系统已一个虚拟 SCSI 控制器形式来连接到一个或多个虚拟 SCSI 磁盘，如图 8-1 上半部分表示的。这些控制器对于虚拟机都是 SCSI 类型的控制器，包括扩展了 VirtualSCSIController 的对象如下：

- ParaVirtualSCSIController
- VirtualBusLogicController
- VirtualLsiLogicController
- VirtualLsiLogicSASController

一个虚拟机如何正确的访问存储依赖于主机的安装。图 8-2 给出了不同可能性的概况。

Figure 8-2. Storage API Architecture



8.2.2 数据存储

一个数据存储是一个可管理的存储实例，通常被用来存储虚拟机文件包括日志文件，脚本，配置文件，虚拟机磁盘文件等等。vSphere 支持两种类型的数据存储，VMFS 和 NAS。

- 假如你想使用一个 NAS 卷，使用 CreateNasDatastore 挂载它挂载和使用 RemoveDataStore 来卸载。这俩个命令明确是在 host 上使用的，你必须在你想要挂载或卸载数据存储的每一个主机上调用 create 和 remove 方法。
- 为了创建一个 VMFS 数据存储，调用 CreateVmfsDatastore，传入任何存在的磁盘。调用返回的结果，磁盘被格式化为 VMFS 类型并且数据存储自动被挂载到当你执行了重新扫描后可以看到该磁盘的所有主机上。当你调用 RemoveDatastore 在一个 VMFS 数据存储上，数据储存被销毁。执行重新扫描后，数据存储对所有 ESX/ESXi 系统不再可用。相比之下 NAS 数据存储，你不必为每一个主机调用创建和移出数据储存方法。

一个 ESX/ESXi 主机自动发现相关联的逻辑单元上(LUNs)的 VMFS 卷在启动或重新扫描主机总线适配器时。当你创建了一个 VMFS 数据存储时，数据存储标签是基于 VMFS 卷

标基础上的。假如和已经存在的数据存储冲突，标签将被添加一个后缀来保持唯一。VMFS 卷标保持不变。

销毁一个 VMFS 数据存储将移除组成 VMFS 卷的分区。

数据存储能跨越多个物理存储设备。一个单一个 VMFS 卷可以包括从一个在物理主机上的本地 SCSI 磁盘阵列，一个 Fibre Channel SAN disk farm 或 iSCSI SAN disk farm 的一个或多个 LUNs。ESX/ESXi 系统检测被添加到任何物理存储子系统上的新的 LUNS。当用户查询可用的设备列表时，新的被发现的设备也在其中。你可以在物理主机或存储子系统不下电的情况下扩展已经存在的 VMFS 卷的存储能力。

假如任何在VMFS卷中的LUNs失败或不可用，仅仅数据在其LUN上的虚拟机受影响。一个例外是具有第一个跨区卷(multi-extent volum)的长度的 LUN。所有其它的虚拟磁盘驻留在其它 LUNs 上的虚拟机正常使用。

8.3 选择使用存储 API

HostStorageSystem APIs 是底层级别的足够执行 VMFS 维护操作。他们需要分区细节的知识和 VMFS extent 布局。他们不强制 VMFS 最佳实践如分区对齐和最佳的 VMFS 快大小，他们允许你在同一个 LUN 上的不同数据存储中混合设置 extent 并且在大多数情况下通过添加 extent 来扩展是更好的方式。

HostDatastoreSystem APIs 主要被使用来管理 VMFS 卷。他们不需对存储系统彻底了解，并且执行最佳实践。

图 8-3 给出了不同 APIs 的概况；表 8-1 显示了通过 APIs 执行的任务。

Figure 8-3. Storage APIs

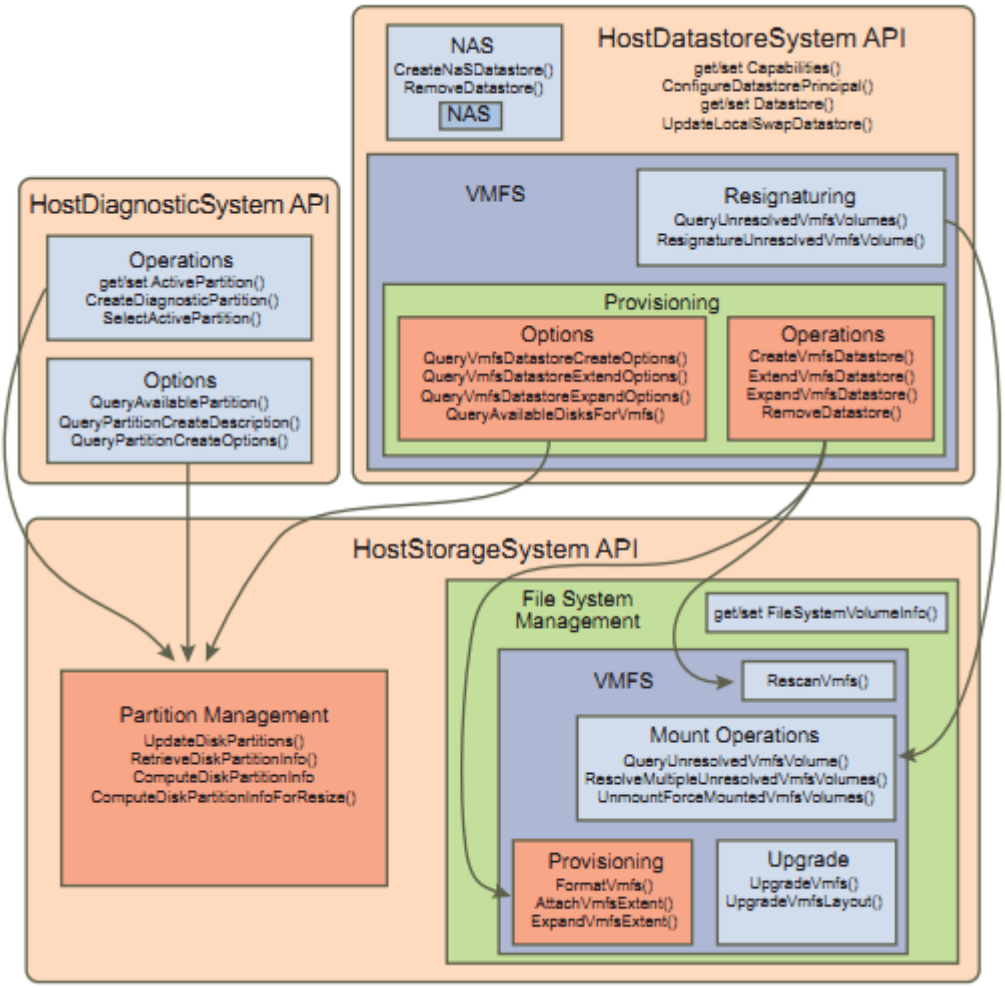


Table 8-1. Storage API Overview

Managed Object	Task	See
`HostStorageSystem`	Low-level operations associated with individual hosts, such as resizing or updating disk partitions.	"Configuring Disk Partitions" on page 93
`HostStorageSystem`	Multipath management.	"Multipath Management" on page 94
`HostStorageSystem`	iSCSI Storage setup and configuration.	"Configuring iSCSI Storage" on page 94
`HostDatastoreSystem`	Creating and managing VMFS datastores and remote datastores.	"Creating and Managing Datastores" on page 96
`HostDatastoreSystem` `HostStorageSystem`	Managing VMFS volume copies (resignature or force mount).	"Managing VMFS Volume Copies (Resignaturing)" on page 99
`HostDiagnosticSystem`	Creating an managing diagnostic partitions.	"Managing Diagnostic Partitions" on page 100

8.4 配置磁盘分区

HostStorageSystem 管理底层级别的存储组件包括 HBAs，SCSI LUNs，文件系统卷等等。

你可以使用 API 在创建，扩大或扩展一个 VMFS 文件系统前建立分区。

- ComputeDiskPartitionInfo – 计算磁盘分区信息基于明确的磁盘布局。为一个使用由 HostDiskPartitionLayout 对象制定布局的磁盘，服务器会计算一个新的 HostDiskPartitionInfo 对象。在 HostDiskPartitionLayout 对象里，你为分区提出详细的块范围列表，和可选择的总数和块大小。当更新一个磁盘分区时你可以时候在 HostDiskPartitionSpec 里面的信息。
- ComputeDiskPartitionInfoForResize – 计算磁盘分区信息来支持对于一个已经给给定的分区重新设置大小。重新设置大小的磁盘分区信息以一个 HostDiskPartitionInfo 对象返回。在重新设置磁盘分区大小时你可以使用在 HostDiskPartitionSpec 里面的信息。
- RetrieveDiskPartitonInfo – 允许你得到确定磁盘的设备路径明的队列和返回一个 HostPartitionInfo 对象对列为每一个磁盘。
- UpdateDiskPartitions – 通过提供一个分区详细规格(HostDiskPatitionSpec)和设备名称来改变一个磁盘上的分区。

在你已经为主机更新了磁盘分区后，你必须执行一个重新扫描通过使用下列之一的方法。完成重新扫描可能会花很长时间。

- RefreshStorageSystem – 刷新存储信息和设置接受改变，但是不要明确的执行命令来发现新设备。
- RescanAllHba – 重新扫描所有主机总线适配器为新的存储设备。这个方法可能会花费很长时间。
- RescanHba – 重新扫描一个指定的主机总线适配器为新设备。

HostStorageSystem 方法也可以用在建立 iSCSI 存储。

8.5 多路径管理

ESXi 配置手册和 ESX 配置手册包括了一个广泛的为失效备份和负载均衡使用多路径的讨论。你可以使用 vSphere 客户端，esxcli 命令，或下列命令来管理多路径。使用 HostStorageSystem.multipahtStateInfo 属性来访问描述给定的主机上的一个多路径运行时状态的 HostMultipathStateInfo 数据对象。

- EnableMultipathPath – 使一个设备的失效路径起作用。使用

HostMultipathStateInfoPath 或 HostMultipathInfoPath 的路径名。

- QueryPathSelectionPolicyOptions – 得到一套 path-selection-policy 选项。这些选项决定了设备可以使用的路径由多重路径管理。一个 HostMultipathInfo 数据对象明确了被多路径管理的设备。
- QueryStorageArrayTypePolicyOptions – 获得一套 storage-array-type 策略选项。这些选项决定了由多路径管理一个设备的 storage-array-type 策略可用。一个 HostMultipathInfo 数据对象明确了由多路径管理的设备。
- SetMultipathLunPolicy – 为一个 LUN 更新路径选择。通过 HostMultipathInfoLogicalUnit 对象的 LUN UUID 来确定 LUN。
- DisableMultipathPath – 使一个有效的设备路径无效。使用从 HostMultipathStateInfoPath 或 HostMultipathInfoPath 得到的路径名称。

8.6 配置 iSCSI 存储

vSphere 支持软件 iSCSI，依赖于硬件 iSCSI，和独立的硬件 iSCSI。

下列 HostStorageSystem 方法可以用于 iSCSI 存储管理。

- 添加一个动态或静态的目标
  - AddInternetScsiSendTarget – 添加发送目标实体到主机总线适配器发现列表如果 DiscoveryProperties.sendTargetsDiscoveryEnabled 标志为 true。
  - AddInternetScsiStaticTargets – 添加静态目标实体到主机总线适配器发现列表。DiscoveryProperty.staticTargetDiscoveryEnable 标志必须为 true。
- 配置目标
  - UpdateInternetScsiAdvanceOptions – 更新 iSCSI 主机总线适配器或发现地址的高级选项和与其关联的目标。
  - UpdateInternetScsiAlias – 更新一个 iSCSI 主机总线适配器的别名。
  - UpdateInternetScsiAuthenticationProperties – 为一个或多和一个 iSCSI 主机总线适配器关联的个目标或发现地址更新认证属性。
  - UpdateInternetScsiDigestProperties – 为 iSCSI 主机总线适配器或发现地址跟新 digest 属性和与其关联的目标。
  - UpdateInternetScsiDiscoveryProperties – 为一个 iSCSI 主机总线适配器更新

discovery 属性。

- UpdateInternetScsiIPProperties – 为一个 iSCSI 主机总线适配器更新 IP 属性。
- UpdateInternetScsiName – 为一个 iSCSI 主机总线适配器更新名字。
- UpdateSoftwareInternetScsiEnabled – 使在 VMkernel 里的软件 iSCSI 生效或无效。

- 移除一个动态或静态目标
  - RemoveInternetScsiSendTargets – 从主机总线适配器发现列表中移除发送的目标实体。DiscoveryProperty.sendTargetsDiscoveryEnabled 必须设置为 true。假如任何一个提供作为参数的目标不在已存在的列表里面，其他目标被移除并且抛出异常。
  - RemoveInternetScsiStaticTargers –从主机总线适配器发现列表中移除静态目标。DiscoveryProperty.staticTargetsDiscoveryEnabled 必须设置为 true。假如任何一个提供作为参数的目标不在已存在的列表里面，其他目标被移除并且抛出异常。

iSCSI 启动器和目标有唯一的，永久的 iSCSI 名称和地址。一个 iSCSI 名称正确的定位一个 iSCSI 启动器或目标，无论物理位置在哪里。名称必须是 EUI 或 IQN 格式，由存储供应商硬件明确提供。

在一个系统上建立 iSCSI 之前，你必须创建一个专用的 VMkernel 网络接口。

To enable the VMkernel to support software iSCSI

1. 获得一个到主机系统的 HostStorageSystem 的 managed 对象引用。
2. 调用 UpdateSoftwareInternetScsiEnabled 方法，传递 HostStorageSystem 的引用和 true 值。

To configure iSCSI intiators

1. 访问在主机系统上可用的 HBAs 列表。

   你可以通过使用 HostSystem 作为开始点来创建一个属性收集器。参看第五章，“Property Collector”。通过 HostSystem.config 属性，你可以通过指定的属性路径获得主机总线适配器列表(队列)：

   config.storageDevice.hostBusAdapter

   属性路径返回一个主机适配器的队列，例如：

   hostBusAdapter[“key-vim.host.BlockHba-vmhba32”]

   hostBusAdapter[“key-vim.host.BlockHba-vmhba33”]

hostBusAdapter[“key-vim.host.BlockHba-vmhba34”]

hostBusAdapter[“key-vim.host.BlockHba-vmhba35”]

hostBusAdapter[“key-vim.host.BlockHba-vmhba1”]

…..

2. 通过这个队列，选择你想要配置的主机适配器(HostHostBusAdapter 入口)并且得到它的 key 属性，主机适配器的设备名字是一个字符创。
3. 通过检索 HostHostBusAdapter 对象属性来确定适配器的能力。
4. 配置启动器。
   - 对于独立的硬件启动器，配置 IP 地址。
   - 对于软件启动器，在 VMkernel 中启动软件启动器。
5. 通过调用 HostStorageSystem.UpdateInternetScisiName 配置 iSCSI 名称和运行 HostStorageSystem.UpdateInternetScsisAlias 配置别名。
6. 通过调用 HostStorageSystem.UpdateInternetScisiHbaDiscoveryProperties 来配置目标发现。

   这个方法取得一个你可以配置的 HostInternetScisiHbaDiscoveryProperties 数据对象。
7. (可选)通过调用 HostStorageSystem.UpdateInternetScisiAuthenticationProperties 来设置认证信息。

   你传递到方法里的 HostInternetScsiHbaAuthenticationProperties 对象包括了配置 CHAP 属性和共有的 CHAP。
8. 配置目标访问。
9. 重新扫描 HBAs。

   重新扫描使 HBAs 发现新的存储设备。你也可以使用 HostStorageSystem.RescanHba 来重新扫描一个单一的 HBA，明确的 HBA ID 作为参数，或使用 HostStorageSystem.RescanAllHba 来重新扫描所有 HBAs。

8.7 创建和管理数据存储

每一个数据存储是一个逻辑上的容器，类似于一个逻辑卷上的一个文件系统，主机存放虚拟磁盘文件和其他虚拟机文件的地方。数据存储隐藏了具体的物理存储设备并且提供统一的存储虚拟机文件模式。

HostDatastoreSystem managed 对象提供了创建和管理数据存储方法。所有的 HostDatastoreSystem 方法需要一个 HostDatastoreSystem 的 managed 对象引用，当它被创建后返回一个到 Datastore 对象的引用。

HostDatastoreSystem 允许你来创建和扩展，查询，移除或更新数据存储。HostDatastoreSystem 也允许你通过调用 ConfigureDatastorePrincipal 来为一个主机配置一个数据存储负责人。所有的虚拟机关联文件的 I/O 在这个用户下执行。

VMFS provisioning 任务经常以下列形式运行：

1. 调用 QueryAvailableDisksForVmfs 来为包含 VMFS 数据存储得到合适的磁盘的子集。QueryAvailableDisksForVmfs 得到一个可以被使用来容纳 VMFS 数据存储扩展的磁盘列表。你可以通过提供一个数据存储名称来从列表中得到指定的 VMFS 数据存储。这个操作不返回当前被 VFMS 数据存储使用的磁盘，也不返回管理的 LUNs 和被 RDMs 引用的磁盘。RDM 磁盘不可以被 VMFS 数据存储使用。
2. 通过调用下列之一方法来获得关于 provisioning 选项信息，传入选择好的磁盘 ：
   - QueryVmfsDatastoreCreateOptions – 取得关于在磁盘上创建一个新的 VMFS 数据存储的选项信息。方法返回一个 VmfsDatastoreOption 数据对象队列。
   - QueryVmfsDatastoreExpandOptions – 取得关于扩展一个已经存在的 VMFS 数据存储长度的选项信息。
   - QueryVmfsDatastoreExtendOptions – 取得关于为一个磁盘扩展一个已经存在的 VMFS 数据存储的选项信息。
3. 假如需要，通过调用 HostStorageSystem.ComputeDiskPartitionInfo 来改变布局，然后调用 HostStorageSystem.UpdateDiskPartition 来重新设置分区大小。
4. 调用 CreateVmfsDatastore，ExtendVmfsDatastore，或 ExpandVmfsDatastore 来完成 VMFS provisioning 操作。

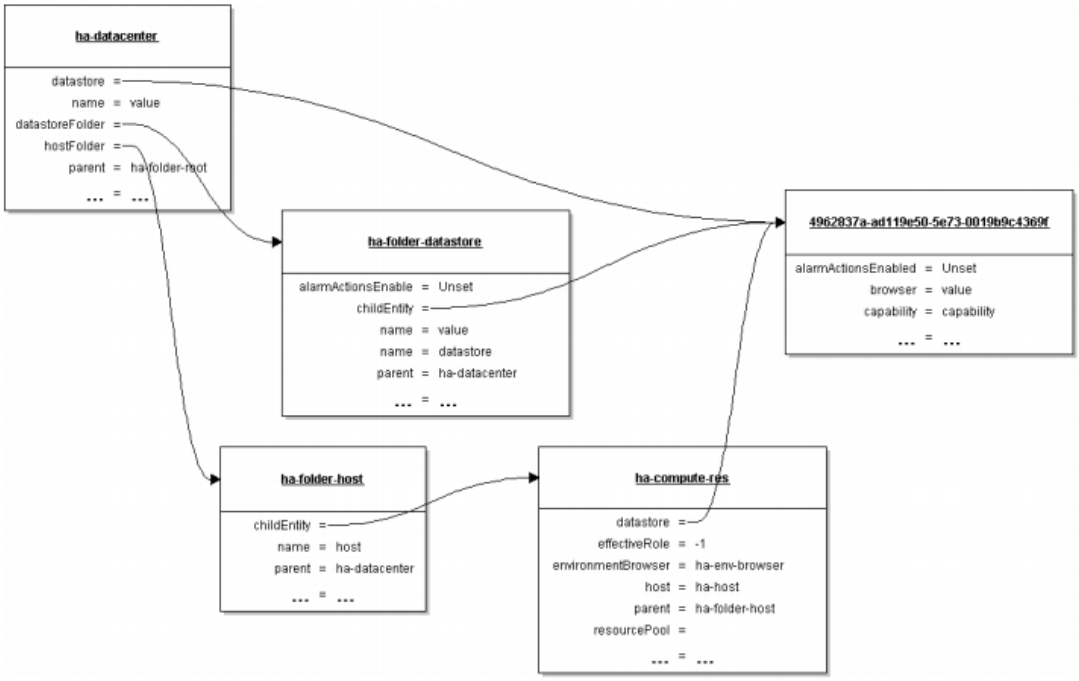
8.7.1 访问数据存储

图 8-4 说明了你如何访问和指定数据存储。

- 每一个数据中心 managed 对象有 datastore 属性包含了一个数据存储队列。
- 每一个数据中心 managed 对象有一个指向在这个数据中心中包含数据存储的文件夹(或是文件夹层级结构)引用的 datastoreFolder 属性。

- 每一个数据中心 managed 对象有一个指向包含了计算资源的文件夹(或是文件夹层级结构)引用的 hostFolder 属性，计算资源包括主机和集群。对于这个数据中心，每个 HostSystem 或 ComputeResource 有一个包含 Datastore managed 对象队列的 datastore 属性。

Figure 8-4. Datastore Managed Object



8.7.2 创建和编辑一个 VMFS 数据存储

一个数据存储是一个可管理的存储实体，通常被用来存放虚拟机文件的仓库包括日志文件，脚本，配置文件，虚拟磁盘等等。

VMFS 是一个专有的文件系统由 VMware 为了虚拟机而设计的。VMFS 适合存储小数量的大数据文件例如虚拟磁盘。这些文件大多数被一个单独的主机使用。VMFS 不同于其他各式的文件系统例如 FAT16/FAT32 等等，它可以通过连接同一个 SAN LUN 来被多个主机访问。

你可以在任何主机可以访问的基于 SCSI 存储设备上建立一个 VMFS。VMFS 卷创建，延伸，扩展需要第一个分区操作和 VMFS 卷操作。

建立磁盘分区

建立磁盘分区包括以下任务：

1. 调用 HostStorageSystem.RetrieveDiskPartitionInfo 来检索关于已经存在分区信息。
2. 调用 HostStorageSystem.ComputeDiskPartition，传入希望的磁盘布局。服务器相应请求来为一个指定的磁盘计算一个新分区信息对象并且返回一个你需要在

HostDiskPartitionSpec 使用的 HostDiskPartitionInfo 对象，HostDiskPartitionSpec 传递给 UpdateDiskPartitions。

3. 调用 HostSystemStorage.UpdateDiskPartitions 传入 HostDiskPartitionSpec 来更新分区。

Creating the VMFS Datastore

创建 VMFS 数据存储包括以下任务：

1. 为你的存储需求配置和安装任何第三方适配器和调用 HostStoreageSystem.RescanAllHba 来重新扫描适配器。
2. 调用 HostDatastoreSystem.QueryAvailableDisksForVmfs 来获得包含 VMFS 数据存储的磁盘信息。

   这个方法过滤当前被存在的 VMFS 使用的磁盘，除非该磁盘正在被扩展(extend)。同时它也过滤了管理 LUNs 和被 RDMs 引用的磁盘。这些磁盘 LUNs 不合适被 VMFS 使用。方法返回一个 HostScisiDisk 对象队列。
3. 调用 HostStorageSystem.QueryVmfsDatastoreCreateOptions 来获得关于创建一个新的 VMFS 数据存储的选项信息。调用返回一个允许你访问合适数据存储的 UUIDs 的 VmfsDatastoreCreateOption 数据对象队列。
4. (可选)假如没有合适你的 VMFS 卷的分区，你可能不得不创建他们。使用 HostStorageSystem 里面的 ComputeDiskPartitionInfo 和 UpdateDiskPartitions 方法。
5. 创建数据存储。
   - 调用 HostDatastoreSystem.CreateVmfsDatastore 来创建一个 VMFS 数据存储。这个方法取得一个 VmfsDatastoreCreateSpec 数据对象包括一个分区，一个 HostVmfsSpec 和一个可选的 extent(长度)。HostVmfsSpec 允许你指定块大小，extent，主版本和 VMFS 卷标名。
   - 调用 HostDatastoreSystem.CreateNasDatastore 来创建一个基于数据存储的网络连接存储。

你以后可以通过下列方法来扩展和延伸 VMFS 数据存储。

- 首先调用 QueryVmfsDatastoreExpandOptions 然后调用 ExpandVmfsDatastore 来扩展一个已存在的 VMFS 数据存储使用 VmfsDatastoreExpandSpec 数据对象提供的规格(包含 extent 名称和分区信息)。假如需要，ExpandVmfsDatastore 可以扩展数据存储到数据存储预置的大小。
- 首先调用 QueryVmfsDatastoreExtendOptions 然后调用 ExtendVmfsDatastore 来延伸

一个已经存在的 VMFS 数据存储使用 VmfsDatastoreExtendSpec 数据对象提供的规格。

8.7.3 移除和更新数据存储

- RemoveDatastore – 从一个主机上删除一个数据存储
- UpdateLocalSwapDatastore – 为这个主机选择 LocalSwapDatastore。这个设置的任何改变都会影响这个主机上的挂起的虚拟机上电或重新运新，或当这个主机上事迁移到这个主机。在这个主机上上电的虚拟机不受影响。

8.7.4 通过 HostStorageSystem 管理 VMFS 数据存储

大多数时候，对于创建和管理 VMFS 数据存储 Datastore 的方法适合的。然而，在下列的有些情况下 HostStorageSystem 命令更合适：

- AttachVmfsExtent – 通过把一个磁盘分区当作一个 extent 来延伸一个 VMFS。
- ExpandVmfsExtent – 通过磁盘分区规格按照说明扩展一个 VMFS extent。
- FormatVmfs – 基于你传入的 HostVmfsSpec 的磁盘分区信息来格式化一个新的 VMFS。返回一个 HostVmfsVolume 表示一个新的 VMFS 文件系统。HostVmfsVolume 包括块大小，磁盘的 VMFS extents 的分区名称列表，其他的如 UUID 信息。
  这个命令时底层的 API，你可以用来显示的给磁盘分区。在多数情况下，Datastore VMFS 命令是更合适的。
- RescanVmfs – 重新扫描一个 VMFS 实例。
- UpdateVmfs – 更新 VMFS 到当前 VMFS 版本。

更新和升级

- HostStorageSystem.UpdateScsiLunDisplayName – 更新和一个 SCSI LUN 关联的易变的显示名称。SCSI LUN 将被为 LUN UUID 识别的。
- HostStorageSystem.UpgradeVmLayout – 遍历所有注册的虚拟机。为每一个虚拟机，升级布局和记录一个事件。方法调用后，VirtualMachineFileLayout 数据对象里的信息将是正确的。

8.8 管理 VMFS 卷拷贝(Resignaturing)

默认的，ESX/ESXi 主机挂载所有 VMFS 数据存储。在一个 LUN 上的分区中创建的每一个 VMFS 数据有一个唯一的 UUID，它存储在文件系统的超级块中。额外的，源 LUN 的 LUN ID 是唯一的且存储在 VMFS 元数据中。

当一个 LUN 被复制或制作一个拷贝，LUN 拷贝结果和远 LUN 是完全相同的，字节对字节拷贝。因此，假如源 LUN 包括一个具有 UUID X 的 VMFS 数据存储，LUN 拷贝好像包含一个完全一样的 VMFS 数据存储，或一个 VMFS 数据存储拷贝，具有完全一样的 UUID X。ESX/ESXi 可以决定是否是否一个 LUN 包含 VMFS 数据存储拷贝，并且考虑未解决的拷贝不自动挂载它。

在 LUN 拷贝上产生数据是可行的，假如你确定源 LUN 没有在使用你可以强行挂载拷贝，或注册拷贝。当你执行数据存储 resignaturing，请注意以下要点：

- 数据存储 resignaturing 是不可逆的，因为它覆写了源的 VMFS UUID。
- 包含 VMFS 数据存储的 LUN 拷贝重新标记后将不能作为一个 LUN 拷贝处理，而是作为一个和这个拷贝源无关的独立的数据存储出现。
- 一个在有效期的数据存储仅仅当它所有的 extents 在线时可以被重新标记。
- Resignaturing 进程崩溃和容错。假如进程中断，你可以随后重新执行它。
- 你可以挂载和其他数据存储 UUIDs 没有冲突的新的 VMFS 数据存储，例如在 LUN 快照层结构中的一个祖先或孩子。

最简单的方法来重新标记未解决的卷，可以使用 HostDatastoreSystem.ResignatureUnresolvedVmfsVolume_Task 方法。这个方法分配一个新的 DiskUuid 到一个 VMFS 卷，但是保持它的内容完好的。方法跨主机来支持安全卷共享并且它合适大多数情况。

你可以替代底层的 HostStorageSystem 方法来查找，强行挂载，或卸载未解决得卷：

- HostStorageSystem.QueryUnresovedVolume – 取得没有束缚的 VMFS 卷列表。为了跨主机来共享一个卷，一个 VMFS 卷受限于它底下的块设备存储。当一个底层的块拷贝被执行来拷贝或移动 VMFS 卷，复制的卷不受限制。
- HostStorageSystem.ResoveMultipleUnresolvedVmfsVolumes – 重新标志或强制挂载非受限的 VMFS 卷。这个方法用一个 HostUnresolvedVmfsResolutionSpec 数据对象作为输入。HostUnresolvedVmfsResolutionSpec.resolutionSpec 属性是

一个 HostUnresolvedVmfsResolutionSpec 数据对象队列包括一个 HostUnresolvedVmfsResolutionSpecVmfsUuidResolution 枚举类型。这个枚举类型取值为 forceMount 或 resignature。

- UnmountForceMountedVmfsVolume – 卸载一个强制挂载的 VMFS 卷。当一个底层块拷贝被执行来复制或移动 VMFS 卷时，复制的卷不受限制。为了这个 VMFS 卷可用，一个解决操作被应用。作为解决操作的部分，你可一决定保持原始的 VMFS UUID。一旦解决操作应用，VMFS 卷将挂载到它可用的主机上。方法允许你卸载 VMFS 卷当没有注册的虚拟机使用的时候。

8.9 管理诊断分区

你的主机必须拥有一个诊断分区(dump partition 转储分区)来存储调试的 core dump 和 VMware 技术支持使用。

建议为每一个主机分配一个 100MB 的诊断分区。假如超过一个 ESX/ESXi 主机使用同一个 LUN 作为诊断分区，这个 LUN 必须被化区这样所有的 EXS/ESXi 主机可以访问它。每个主机需要 100MB 的空间，因此 LUN 的大小决定多少服务器可以共享它。每一个 ESX/ESXi 主机被影射到一个诊断槽位。你可以使用 FibreChannel 或硬件 iSCSI 来建立一个 SAN LUN。SAN LUN 不支持通过一个软件的 iSCSI 启动器访问。

注意：假如俩个共享一个诊断分区的主机失败并且保存 core dump(磁芯信息转储)到同一个槽位，core dump 可能丢失。为了收集 core dump 数据，主机失败后马上重新启动一个主机并且提取日志文件。假如其他的主机在你收集前一个失败的主机诊断数据前失败，第二个主机不保存 core dump。

8.9.1 检索诊断分区信息

HostDiagnosticSystem managed 对象允许你通过下列途径来检索信息。

- 从 HostDiagnosticSystem.activePartition 属性中检索 HostDiagnosticPartition 对象来检查活动的分区的属性。
- 调用 HostDiagnosticPartition.QueryAvailablePartition 方法来检索得到一个可用的诊断分区列表，结果按照适合程度排序。
- 调用 HostDiagnosticPartition.QueryPartitionCreateOptions 方法来检索得到一个拥有

足够空间来包含一个指定类型的诊断分区的磁盘列表，结果按照适合程度排序。

8.9.2 创建一个诊断分区

创建一个诊断分区需要你通过一种查询方法找到一个合适的分区。然后你检索一个创建规格，然后执行真实的创建动作。

创建一个诊断分区

1. 通过调用 HostDiagnosticPartition.QueryAvailablePartition 或 HostDiagnosticPartition.QueryPartitionCreateOptions 来找到一个合适的分区。
2. 调用 HostDiagnosticPartition.CreateDiagnosticPartition，传入一个 HostDiagnosticPartitionCreateSpec，包括关于诊断类型，Id，存储类型等等。

成功时，方法创建一个分区并，如果明确了 active 参数为 true 将使它成为活动分区。失败时，诊断分区可能存在，但是不是活动的甚至分区不支持使其活动。

8.10 代码示例

Table 8-2. Sample Applications that Demonstrate Storage and Datastore Operations

Java (SDK\vsphere-ws\java\JAXWS\samples\com\vmware\)	C# (SDK\vsphere-ws\dotnet\cs\samples\)
scsilun\SCSILunName.java	SCSILunName\SCSILunName.cs
--	SCSILunName\SCSILunName.csproj
--	SCSILunName\SCSILunName2008.csproj
--	SCSILunName\SCSILunName2010.csproj
httpfileaccess\GetVMFiles.java	GetVirtualDiskFiles\GetVirtualDiskFiles.cs

9、网络

在你添加存储和虚拟机到 ESX/ESXi 系统前，你应该有了完整的网络设置。

9.1 网络管理对象和方法

vSphere Web Services SDK 包括下列对象和方法来管理网络配置：

- HostNetworkSystem – Managed 对象代表了主机的网路配置。对象的属性指向你可

以为网络管理的网络数据对象，包括 HostDnsConfig 和 HostIpRouteConifg。

HostNetworkSystem 属性允许你访问 HostNetCapabilities 和 HostNetworkInfo 数据对象，访问和编辑 HostNetworkConfig 数据对象。

HostNetworkSystem 拥有检索和改变网络配置的方法。

- HostNetworkConfig – 允许你为主机详细规划网络配置。你可以通过运行 HostNetworkSystem.UpdateNetworkConfig 方法来应用这个网络配置。
- Network – 代表一个可以由主机或虚拟机访问的网络。这可以是一个物理网络或一个逻辑网络，例如一个 VLAN。

  当你添加一个主机到一个 vCenter Server 系统中时，或当你添加一个虚拟机到一个一个 ESX/ESXi 主机时，一个网络自动被添加。
- HostSystem.QueryHostConnectionInfo 和 Datacenter.QueryConnectionInof 俩个都返回一个 HostConnectInfo 数据对象，描述当前网络配置。
- HostSNMPSystem – 支持 SNMP(Simple Network Management Protocol)建立。

9.2 网络介绍

为你的 ESX/ESXi 主机建立网络可能包含几个部分：

- 建立一个或多个虚拟交换机。虚拟交换机提供在同一主机或不同主机上虚拟机之间的连接。虚拟交换机也支持 VMkernel 网络访问来 VMotion，iSCSI 和 NFS。在 ESX 上，虚拟交换机支持服务控制台网络。你在每台主机上独立建立虚拟交换机。

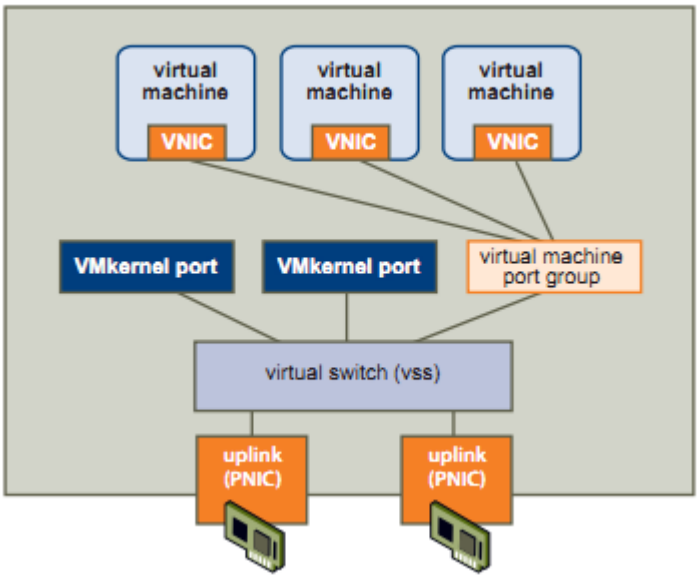
  从 vSphere4.0 以后，你可以使用一个网络标准交换机(vSS)或一个网络分布式交换机(vDS)。文档主要讨论 vSS。
- 添加虚拟机端口组。一个虚拟机总是通过一个端口组来访问网络。
- 为虚拟机指定适配器。被指定的适配器作为一个虚拟设备，作为虚拟机安装时的一部分配置。
- 添加 VMkernel 网络接口，例如，为了支持 iSCSI 存储或 VMotion。
- 配置一个无力适配器(pnic)，实际的连接从主机到网络。你可以配置 pnic 通过 HostNetworkSystem.pnic 属性，是一个 PhysicalNic 数据对象。你可以通过 VirtualSwitch.pnic 属性来指定一组 pnics 与虚拟交换机关联，VirtualSwith.pnic 取得一个物理网络适配器的队列。

- 主机的网络设置(IP 路由，DNS，SNMP)。

9.3 虚拟网络标准交换机环境

一个虚拟网络标准交换机(vSS)可以在俩个虚拟机间内部通讯也可以和使虚拟机连接外部网络。图 9-1 显示了一个 vSS 环境的包含的元素。

Figure 9-1. vSS Environment



9.3.1 虚拟交换机

vSS 的网络中虚拟交换机它自己是中心。vSS 可以在同一个主机(私有网络)上的两台虚拟机间发送网络流量或发送数据到一个外部网络(公共网络)。公共网络使用以太网网络适配器和物理主机关联(上行适配器)。

默认的 vSwitch 的逻辑端口数是 120，但是可以修改成更多的端口数。每一个端口能连接到一个一个虚拟机的一个网络适配器，或这个物理机上的一个上行适配器。

当两个或多个虚拟机被连接到同一个 vSS 上，它们直接的网络通讯被路由为本地。假如一个上行适配器和这个 vSS 连接，每一个虚拟机可以访问这个适配器所连接的外部网络。

9.3.2 端口组

端口组在一个公共的配置下聚合了多个端口。

每个端口组由一个网络标签标识，在当前主机上是唯一的。网络标签网络标签可以虚拟

机方便的配置访问主机。在数据中心中的所有端口组被物理的连接到同一个网络(每个都可以接受到其他人的广播)都被分配了同一个标签。相反地，假如两个端口组不能相互接受到彼此的广播，他们有着不同的标签。

你可以使用一个 VLAN ID 来限制端口组流量到一个物理网络内的逻辑以太网段。一个端口组要是可以到达位于其他 VLANs 的端口组，VLAN ID 必须被设置为 4095。假如你使用 VLAN IDS，你必须改变端口组标签和 VLAN IDs 所以标签属性代表连通性。

9.3.3 虚拟机网络接口

当你创建一个虚拟机时，包括了一个 VirtualMachineConfigSpec，相应的包括了一个 VirtualDeviceConfigSpec。、VirtualDeviceConfigSpec 的 device 属性是一个 VirtualDevice 数据对象。一个可用的虚拟设备是一个 VirtualEthernetCartd。你可以使用 VirtualEthernetCard 中的一个子类型来明确这个虚拟卡的使用和制定 MAC 地址和是否 wake-on-LAN 是否可以在这个虚拟卡上适用。适配器的数量是有限制的。

9.3.4 VMkernel 网络接口

VMkernel 提供的网络服务(iSCSI，NFS，VMotion)适用一个 TCP/IP 栈在 VMkernel 里。这个 TCP/IP 栈完全区别于在服务控制台里使用的 TCP/IP 栈。每一个这些 TCP/IP 栈访问的各个网络被连接到一个或多个虚拟交换机上的一个或多个端口组上。

VMware VMkernel TCP/IP 网络栈通过下列方式来处理 iSCSI，NFS，VMotion。

- iSCSI 当作一个虚拟机的数据存储。
- 对于直接挂载.ISO 文件的 iSCSI，对于虚拟机表现为 CD-ROMs。
- NFS 当作一个虚拟机数据存储。
- 对于直接挂载.ISO 文件的 NFS，对于虚拟机表现为 CD-ROMs。
- 通过 VMotion 迁移。

假如你有两个或多个物理 NICs 对于 iSCSI，你可以通过使用端口绑定来为软件 iSCSI 创建多路径。

一个新安装的 ESX/ESXi 系统不包括 VMkernel 网络接口。当你希望通过 VMotion 来迁移一个虚拟机时，你的 VMkernel 网络栈必须提前创建。当你想使用一个使用 TCP/IP 网络通讯的存储时，例如 iSCSI，你必须为这个存储设备提供一个独立的 VMkernel 网络接口。

你必须创建任何你需要的 VMkernel 端口。

9.3.5 物理网络适配器(pnic)

pnic 指向物理网络适配器期间能被主要的操作系统认到。当适用 vSphere Web Services SDK 时，你能直接控制这个适配器。当适用 vSphere 客户端 UI 时，你代替的控制上行适配器。在 vSS 环境中，每一个 pnic 由一个关联的上行适配器。在一个 vDS 环境中，你使用一个 DVS uplink 代替一个上行适配器。

9.4 使用 vSS 建立网络

你能使用 HostNetworkSystem managed 对象来访问和控制 ESX/ESXi 系统网络元素。

9.4.1 检索网络配置信息

你可以按照下列步骤来检索关于网络配置信息：

- HostNetworkConfig 对象属性，你可以 HostNetworkSystem.networkConfig 访问，允许你检索配置信息。这个信息是广泛的包括物理适配器，虚拟交换机，虚拟网络接口等等。

  你也可以使用 HostNetworkConfig 来改变配置。
- HostNetworkInfo 对象属性，你可以通过 HostNetworkSystem.networkInfo 来访问，允许你检索实时运行信息。

9.4.2 添加一个虚拟交换机

调用 HostNetworkSystem.AddVirtualSwitch 方法来添加一个或多个虚拟交换机。传入虚拟交换机的名称和一个 HostVirutalSwitchSpec 数据对象作为参数。

在 HostVitualSwitchSpec 中你可以去顶 MTU，端口数，网络策略和桥详细规格。桥指定了这个虚拟交换机如何连接到物理适配器。当前支持的连接桥提供网络适配器(NIC)组功能通过使用一个物理设备列表并且，可选的，一个信号监测来测试和物理适配器的连接性。

当你创建了虚拟交换机后，你可以使它来连接到一个 pnic 来连接到外部，和一个 VMkernerl 端口或一个端口组。

添加一个虚拟交换机

1. 获得关于当前网络配置的信息。

   你可以使用一个属性收集器来检索 HostNetworkSystem managed 对象和它的几个属性，例如 networkInfo。

2. 定义一个 HostVirtualSwitchSpec 明确虚拟交换机的属性。你可以指定端口数(56 到 4088 在 ESXi 系统上)和 HostNetworkPolicy。

3. 调用 HostNetworkSystem.AddVirtualSwitch 来添加一个虚拟交换机。指定一个唯一的名称和 HostVirtualSwitchSpec 定义了交换机的属性。

下列是从 AddVirtualSwitch.java 摘取的片段：

Example 9-1. Add a Virtual Switch

```
vswitchId = vSwitch42;
…
ManagedObjectReference nwSystem = configMgr.getNetworkSystem();
HostVirtualSwitchSpec spec = new HostVirtualSwitchSpec();
spec.setNumPorts(8);
service.addVirtualSwitch(nwSystem, vswitchId, spec);
System.out.println(“ : Successful createing : “ + vswitchId);
```

9.4.3 添加一个虚拟端口组

端口组允许你区分通过一个虚拟交换机的不同类型的流量。你也可以把端口组当作通讯或安全策略配置的分界线。对于 ESXi 系统，默认的端口组是管理网络和 VM 网络。对于 ESX 系统，默认端口组是服务控制台和 VM 网络。

当你创建一个端口组，你可以为它指定一个 VLAN ID。VLANs 是 ESX/ESXi 网络的重要组成部分，因为它可以对流量来分组。例如，你可以创建一个分离的网络区域为 VMotion，管理和开发。使用 VLANS，你仅需为每一个网络区域有一个分离的上行适配器和一个单独的连接到这个适配器的虚拟交换机。可以最大的减少你需要的交换机的数量。

添加一个虚拟端口组

1. 定义一个 HostPortgroupSpce，为每一端口组，你可以指定网络策略，VLAN ID，端口组属于所属的虚拟交换机。

2. 调用 HostNetworkSystem.AddPortGroup，传入 PortGroupSpec。

9.4.4 添加一个 VMkernel 网络接口

VMkernel 网络接口为 VMkernel TCP/IP 栈提供了网络访问。你必须为你的 ESX/ESXi 系统创建新的 VMkernel 端口假如你计划使用 VMotion，VMware FT，或 iSCSI 和 NAS 存储。一个 VMkernel 端口由一个虚拟交换端口和一个 VMkernel 接口组成。

To add a VMkernel Network Interface

1. 创建一个 HostVirtualNicSpec 数据对象，在这个对象里，你可以在一个 HostIpConfig 数据对象里确定 IP 配置。对于 vSS，确定端口组属性。对于 vDS，去顶 distributedVirtualPort 属性。
2. 调用 HostNetworkSystem.AddVirtualNic，传入 HostVirtualNicSpec。
3. 然后为 iSCSI 或 NAS 使用 VMkernel 网络接口，或调用 HostVmotionSystem.SelectVnic 方法来为 VMotion 使用这个 VMkernel NIC。

Example 9-2，Adding a VMkernel Network Interface

```
private HostVirtualNicSpec createVNicSpecification() {
    HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
    HostIpConfig ipConfig = new HostIpConfig();
    ipConfig.setDhcp(false);
    ipAddr = cb.get_option(“ipaddress”);
    ipConfig.setIpAddress(ipAddr);
    ipConfig.setSubnetMask(“255.255.255.0”);
    vNicSpec.setIp(ipConfig);
    return vNicSpec;
}
HostVirtualNicSpec vNicSpec = createVNicSpecification();
service.addVirtualNic(nwSystem, portGroup, vNicSpec);
```

9.5 定义主机网络策略

当你配置主机网络时，你可以为网络定义明确的策略。HostNetworkPolicy 数据对象类型描述为虚拟交换机和端口组描述了网络策略。假如没有为端口组显式的明确定义，端口组继承和它结合的虚拟交换机的策略。

这些策略通过下列数据对象作为 HostNetworkPolicy 属性来定义。

- HostNicTeamingPolicy – 定义连接到物理网络。包括了失败标准，活动和备用的 NICs，故障转移和负载均衡信息。
- HostNetworkSecurityPolicy – 为网络定义安全策略。
- HostNetworkTrafficShapingPolicy – 确定参数为三个流量特性：平均带宽，带宽峰值和最大突发量。

你也可以通过分配一个整形数值给 HostPortgroupSpec.vlandid 属性来明确 VLAN 策略。VMkernel 会当数据包通过虚拟交换机时添加标签或去掉标签。

9.6 NIC Teaming

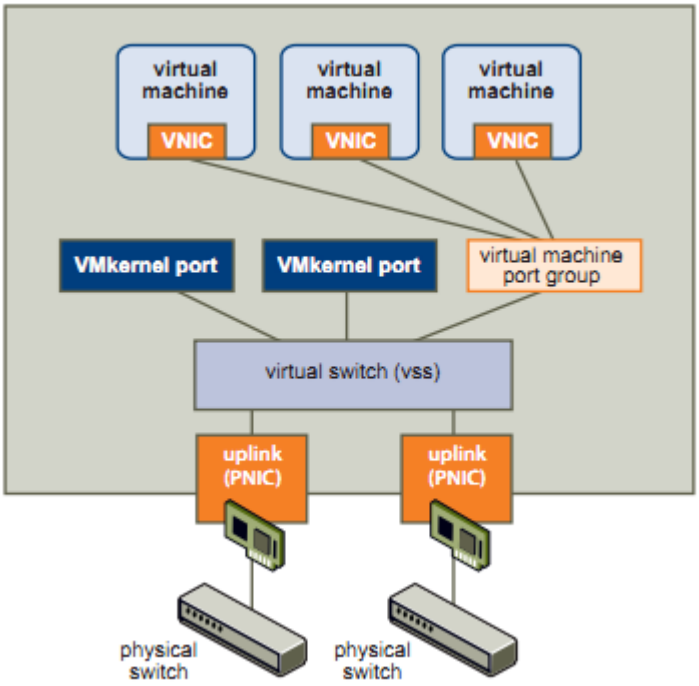
虚拟机连通过一个虚拟交换机连接到公共网络，相应的，连接到物理网络接口(pnic)。当物理适配器或适配器网络连接失败时，关联的虚拟交换机和所有的端口组和虚拟机的连接能力将丢失。

为了解决这个问题，你可以在你的环境中为每一个虚拟交换机连接到两个上行的适配器上。每一个上行适配器连接到两个不同的物理交换机上。然后这个组可以在物理网络和虚拟网络中的一些或所有成员提供共享网络负载，也可以在一个硬件失败或一个网络中断的事件中提供被动的故障转移。

你可以通过 HostNetworkPolicy 来建立 NIC 组。到 HostNicTeamingPolicy 的路径是：HostConfigSpec.network.vswitch[].spec.policy.nicTeaming

假如你为一个虚拟交换机设置了 NIC 组，HostVirtualSwitchSpec.bridge 属性必须设置为 HostVirtualSwitchBondBridge。

Figure 9-2. NIC Teaming



9.7 建立 IPv6 网络

vSphere 支持 Internet 协议版本 4 (IPv4)和 Internet 协议版本 6(IPv6)环境。在 IPv6 中，你能使用 vSphere 特性例如具有 IPv6 的 NFS。

一个在 vSphere 中的 IPv6 规格的配置包含了提供 IPv6 地址，可以是静态地址或为所有关联 vSphere 网络接口使用 DHCP。IPv6 地址可以通过路由器通告使用无状态自动配置。

你可以通过改变 HostIpConfig.ipV6Config 属性来为一个主机设置 IPv6 网络，ipV6Config 属性是一个 HostIpConfigIPV6AddressConfiguration 数据对象。HostIpConfigIPV6AddressConfiguration 允许你确定是否开启自动配置功能，是否为 IPv6 开启 DHCP 功能，和一队 IPv6 地址(HostIpConfigIpV6Address 数据对象)。

HostIpConfigIpV6Address 允许你确定关于 IPv6 地址的所有方面信息包括地址状态，地址(除非 DHCP 开启)，生命周期，操作，起点，前缀长度。下列代码片断展示了 VMkernel NIC 通过路由器通告和 DHCP 自动获得 IPv6 地址。当调用程序时用户在命令行上提供 IP 地址。例子通过 cb.get_option 检索地址。

Example 9-3. IPv6 Setup

```
private HostVirtualNicSpec createVNicSpecification() {
```

```
        HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
        HostIpConfig ipConfig = new HostIpConfig();
        //setting the vnic to get an automatic ipv6 address from router advertisements
        //and through dhcp
        ipV6Config = new HostIpConfigIpV6AddressConfiguration();
        ipV6Config.setAutoConfigurationEnabled(true);
        ipV6Config.setDhcpV6Enabled(true);
        ipConfig.setIpV6Config(ipV6Config);
        vNicSpec.setIp(ipConfig);
        return vNicSpec;
    }
```

9.8 添加网络服务

你可以使用 HostConfigManager 的属性和方法来为你的 EXS/ESXi 系统建立网络服务。

9.8.1 添加一个 NTP 服务

HostConfigManager.dateTimeSystem 属性包含一个 HostDateTimeSystem 数据对象。这个对象允许你进行 NTP，日期和时间关联的配置。

- 查询和更新日期、时间信息通过定义在 HostDateTimeSystem 中的方法。
- 编辑 HostDateTime.dateTimeInfo 属性，它包含了一个 HostDateTimeInfo 对象，来建立 NTP。NTP 信息被存储载 HostDateTimeInfo.ntpConfig 属性中，它是一个 HostNtpConfig 对象。HostNtpConfig 对象的 server 属性包含一个时间服务器列表，由 IP 地址或域名全称确定。

注意：你可以开启和停止 NTP 守护进程和通过使用 HostServiceSystem 对象来检索它的信息。

9.8.2 建立 IP 路由配置

你可以使用 HostNetworkSystem.UpdateIPRouteConfig 方法来为一个 ESX/ESXi 系统确定 IP 路由配置。方法用一个 HostIPRouteConfig 数据对象作为参数。在这个对象里，你可以明

确默认网关地址和 IPv6 网关地址。数据对象也允许你明确在 ESX 上的服务控制台网关设备。

9.8.3 建立 SNMP

简单网络管理协议(SNMP)允许管理程序到监控和控制联网的设备。vCenter 服务器和 ESX/ESXi 系统包括不同的 SNMP 代理：

- vCenter 服务器上的 SNMP 代理可以发送 traps 当 vCenter 服务器系统启动或一个 vCenter 服务器上警告被触发时。vCenter 服务器 SNMP 代理功能仅作为一个 trap 发射器并不支持其他 SNMP 操作例如 GET。
- ESX/ESXi4.0 和以后版本中的 SNMP 代理嵌入在 ESX/ESXi 主机守护中(hostd)可以发送 traps 和接收轮询请求例如 GET 请求。

ESX 在 ESX/ESXi4.0 以前的发布版本中包括一个 Net-SNMP-based 代理。你可以继续在由你的硬件提供商或第三方管理应用提供的具有 MIBs 功能的 ESX4.x 中使用这个 Net-SNMP-based 代理。然而，为使用 VMware MIB 文件你不许使用嵌入的 SNMP 代理。为了在同一时间使用基于代理的 NET-SNMP 和嵌入式的 SNMP 代理，要在一个非默认端口建立一个代理监听。默认的，两个代理使用同一个端口。

- HostSnmpSystem.ReconfigureSnmpAgent 允许通过一个 HostSnmpConfigSpec 来明确代理属性。这个数据对象允许你在 HostSnmpDestination 对象中指定 SNMP 端口，read only communities，和 trap 代理。HostSnmpDestination 对象允许你确定 community，一个主机和通知监听端口。
- HostSnmpSystem.SendTestNotification 允许你测试你的配置。

一个在 HostSnmpSystem.Limits 属性里的 HostSnmpSystemAgentLimits 数据对象指定了代理的限制。

9.9 代码示例

表 9-1 列出了饱含在 vSphere SDK 中的例程。

Table 9-1. Networking Sample Applications

Java (SDK\vsphere-ws\java\JAXWS\samples\com\vmware\host)	C# (SDK\vsphere-ws\dotnet\cs\samples\)
AddVirtualNic.java	AddVirtualNic\AddVirtualNic.cs
	AddVirtualNic\AddVirtualNic.csproj
	AddVirtualNic\AddVirtualNic2008.csproj
	AddVirtualNic\AddVirtualNic2010.csproj
AddVirtualSwitch.java	AddVirtualSwitch\AddVirtualSwitch.cs
	AddVirtualSwitch\AddVirtualSwitch.csproj
	AddVirtualSwitch\AddVirtualSwitch2008.csproj
	AddVirtualSwitch\AddVirtualSwitch2010.csproj
AddVirtualSwitchPortGroup.java	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.cs
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.csproj
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2008.csproj
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2010.csproj
RemoveVirtualNic.java	RemoveVirtualNic\RemoveVirtualNic.cs
	RemoveVirtualNic\RemoveVirtualNic.csproj
	RemoveVirtualNic\RemoveVirtualNic2008.csproj
	RemoveVirtualNic\RemoveVirtualNic2010.csproj
RemoveVirtualSwitch.java	RemoveVirtualSwitch\RemoveVirtualSwitch.cs
	RemoveVirtualSwitch\RemoveVirtualSwitch.csproj
	RemoveVirtualSwitch\RemoveVirtualSwitch2008.csproj
	RemoveVirtualSwitch\RemoveVirtualSwitch2010.csproj
RemoveVirtualSwitchPortGroup.java	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.cs
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.csproj
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2008.csproj
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2010.csproj

10、虚拟机配置

一个虚拟机是一个软件的计算机，像一个物理的计算机，运行着一个操作系统和应用程序。虚拟机兼容所有标准 x86 计算机。每一个虚拟机封装了一个完整的计算环境和独立于底层硬件运行。

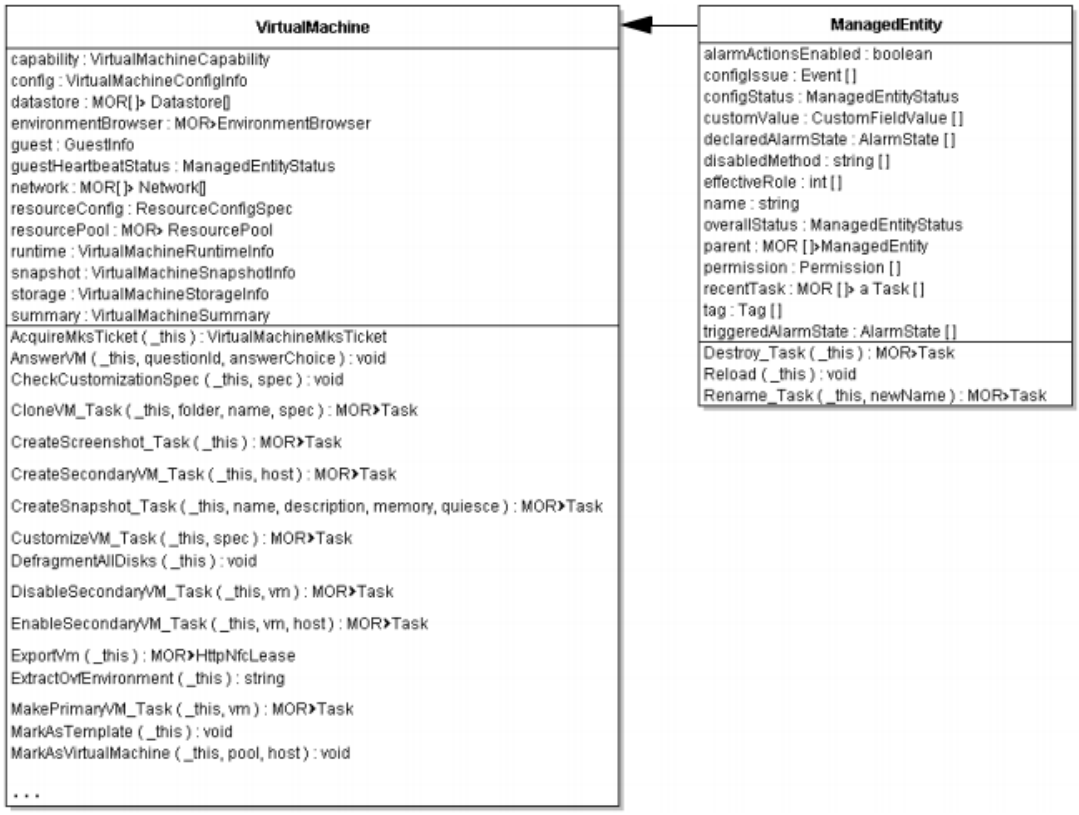
10.1 虚拟机管理对象和方法

虚拟机是你的 vSphere 环境中的中心元素。通过调用 Folder.CreateVM_Task 来创建一个虚拟机，然后使用 VirtualMachine managed 对象的属性和方法来配置这个虚拟机。大多数属性指向数据对象在方法中使用。图例 10-1 显示了部分方法和属性。

客户端程序通常访问和操作下列和虚拟机关联的对象：

- VitualMachine – Managed 对象应用于大多数虚拟机操作。包括创建模板，克隆，快照，执行电源操作，客户机系统管理和安装 VMware Tools。
- VirtualMachineConfigInfo – 数据对象允许你检索一个虚拟机的配置信息。
- VirtualMachineCloneSpec – 数据对象允许你为一个克隆操作制定虚拟机属性。作为 VirtualMachine.CloneVM_Task 的参数。

Figure 10-1. VirtualMachine Managed Object with Some Properties and Methods



10.2 创建虚拟机和虚拟机模板

为了创建一个虚拟机，你使用 Folder.CreateVM_Task 方法。这个方法使用一个 VirtualMachineConfigSpec 数据对象作为参数。VirtualMachineConfigSpec 允许你明确你创建的虚拟机的属性信息。

假如你需要几个一样的虚拟机，你可以转换一个已经存在的虚拟机为一个模板并从这个模板创建多个拷贝(克隆)。你也可以通过克隆一个已经存在的虚拟机来直接创建多个虚拟机。

10.2.1 使用 VirtualMachineConfigSpec 来创建一个虚拟机

使用明确的虚拟机属于通过 Folder.CreateVM_Task 方法来创建一个虚拟机。你必须指定

一个主机或一个资源池(或两者都指定)。虚拟机使用主机或资源池里的 CPU 和内存。

10.2.1.1 Calling the CreateVM_Task Method

调用 Folder.CreateVM_Task 方法是用下列参数来创建一个虚拟机：

- _this – 你想放置虚拟机的文件夹。
- config – VirtualMachineConfigSpec 数据对象明确了 CPU，内存，网络等等信息。
- pool – 资源池，虚拟机获得从中获得资源。
- host – HostSystem managed 对象代表了虚拟机运行于地目标主机。假如你在一个独立的主机上调用这个方法，省略这个参数。假如目标主机是 VMware DRS 群集里的一部分，这个参数可选；假如没有指定主机，系统自动选择一个。

注意：所有的对象必须位于同一个数据中心。

10.2.1.2 Specifying Virutal Machine Attributes with VirtualMachineConfigSpec

虚拟机的自定义通过 VirtualMachineConfigSpec 的属性，它作为参数传给 Folder.CreateVM_Task。例如，你可以为虚拟机确定名称，启动选项，CPUs 个数，内存。所有 VirtualMachineConfigSpec 的属性多支持任意的增加的改变。

下列 VMCreate 例子代码段程序展示了如何定义一个 VirtualMachineConfigSpec。

Example 10-1. 定义一个 VirtualMachineConfigSpec 数据对象

```
VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
    …
    vmConfigSpec.setName(“MyVM”);
    vmConfigSpec.setMemoryMB(new Long(Interger.parseInt 500));
    vmConfigSpec.setNumCPUs(Interger.parseInt 4);
    vmConfigSpec.setGuestId(cb.get_option(“guestid”));
    …
```

VMware SDK SDK/samples/Axis/java/com/vmware/apputils/vim/VMUtils.java 定义了更多更广泛的虚拟机包括一个 Floppy，CD-ROM，disk，和虚拟网卡。

当你创建一个虚拟机时，虚拟机文件被添加到虚拟机所在的存储上。

10.2.1.3 Additional Configuration Information

VirtualMachineConfigInfo 和 VirtualMachineConfigSpec 对象提供 extraConfig 属性来添加附加的配置信息。extraConfig 属性是一个 key/value 队列，来定义配置选项。服务器位虚拟机在.vmx 文件中存储 extraConfig 选项信息。当 vSphpere API 版本的发展，一个 extraConfig 选项可能会变成一个标准的配置属性来作为已经定义的清单数据模型的一部分。在这种情况下，你必须为访问使用标准数据模型属性；不能使用 extraConfig 属性。

10.2.2 创建虚拟机模板

模板允许你创建多个虚拟机具有同样特性，例如 CPU 和内存资源,虚拟硬件类型。
一个虚拟机模板是一个不能上电的也不能位于一个资源池的虚拟机。

你可以通过调用 VirtualMachine.MarkAsTemplate 来转换任何一个关闭了电源的虚拟机成为模板。转换完成后，源虚拟机将不再存在。你可以使用这个模板来创建同一配置的多个克隆。

10.2.3 克隆一个虚拟机

一个克隆是一个虚拟的拷贝。一个虚拟机和一个克隆主要区别在于 VirtualMachine.config.template 属性设置为 true。

你可以通过下列方式之一来创建一个克隆：

- 假如你不再需要一个虚拟机的实例，但是你想使用这个虚拟的配置作为一个模板，使用 VirtualMachine.MarkAsTemplate 方法。这个方法设置 config.template 属性为 true，并且禁用这个虚拟机。
- 假如你想使用一个已经存在的虚拟机作为模板，但是又想保留虚拟机，调用 VirtualMachine.CloneVM_Task 方法来创建这个虚拟的一个副本。

假如你使用 VirtualMachine.CloneVM_Task 方法，你可以自定义这个克隆的属性通过在 VirtualMachineCloneSpec 数据对象中明确他们，当作参数传递给该方法。

下列是 VMClone.java 例子中的代码片断展示了如何自定义一个克隆和指定一个新的位置。

Example 10-2. Cloning a Virtual Machine

VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();

VirtualMachineRelocateSpec relocSpec = new VirtualMachineRelocateSpec();

cloneSpec.setLocation(relocSpec);

cloneSpec.setPowerOn(false);

cloneSpec.setTemplate(false);

String clonedName = cloneName;

ManagedObjectReference cloneTask = service.cloneVM_Task(vmRef, vmFolderRef, cloneName ,cloneSpec);

VirtualMachine.CloneVM_Task 方法接受源虚拟机，目标文件夹，名称，和 VirtualMachineCloneSpec 作为参数。

VirtualMachineCloneSpec 数据对象包括位置，电源状态，和是否克隆为模板。位置，相应的，是一个 VirtualMachineRelocateSpec 数据对象指定目标位置(数据存储，磁盘，主机或资源池)和在磁盘上的任何变化。

10.2.4 模板转换成虚拟机

你可以改变一个模板回到一个可操作的虚拟机。

- 转换模板到一个虚拟机，在这个模板上调用 MarkAsVirutalMachine 方法。你必须为这个虚拟机指定一个资源池和，可选的，一个主机。主机和资源池必须在同一个 ComputeResource 下。当操作完成，模板不再存在。
- 为了保留模板，通过在这个模板上调用 CloneVM_Task 方法来克隆这个模板。在 VirtualMachineCloneSpec(spec 参数)中，设置 template 属性为 false。

10.2.5 访问虚拟机信息

当你已经创建了一个虚拟机，你可以通过 VirtualMachineConfigInfo 属性来检索关于虚拟机的信息。

10.2.5.1 检查默认文件

当你已经创建了一个虚拟机，几个文件被生成和放置在由

VirtualMachineConfigSpec.files 属性指定的目录中。

Table 10-1. Virtual Machine Files

File	Usage	File Description	File Format
.vmx	*.vmname*.vmx	Virtual machine configuration file.	ASCII
.vmxf	*vmname*.vmxf	Additional virtual machine configuration files, available, for example, with teamed virtual machines.	ASCII
.vmdk	*vmname*.vmdk	Virtual disk file.	ASCII
.flat.vmdk	*vmname*.flat.vmdk	Preallocated virtual disk in binary format.	Binary
.vswp	*vmname*.vswp	Swap file.	
.nvram	*vmname*.nvram or nvram	Non-volatile RAM. Stores virtual machine BIOS information.	
.vmss	*vmname*.vmss	Virtual machine suspend file.	
.log	vmware.log	Virtual machine log file.	ASCII
#.log	vmware-#.log	Old virtual machine log files. # is a number starting with 1.	ASCII

假如你正在使用快照，下列额外的文件可能被用到。

Table 10-2. Virtual Machine Snapshot Files

File Extension	Usage	File Description
.vmsd	*vmname*.vmsd	Virtual machine snapshot file.
.vmsn	*vmname*.vmsn	Virtual machine snapshot data file.
**.delta.vmdk		Snapshot difference file. A number preceding the extension increases with more snapshots.
**.vmdk		Metadata about a snapshot.
-Snapshot#.vmsn		Snapshot of virtual machine memory. Snapshot size is equal to the size of you virtual machine's maximum memory.

10.2.5.2 检查默认设备

当你创建一个虚拟机时，你同时创建了一系列的默认设备，基于和你的 SDK 关联的硬件版本。你可以使用 EnvironmentBrower.QueryConfigOption 方法查看这些设备。例如，默认创建的 IDE 控制器。大多数默认设备的属性是不能更改的。

然而，你可以添加下列可选设备到设备集：VirtualSerialPort，VirtualParallelPort，VirtualFloppy，VirtualCdrom，VirtualUSB，VirtualEthernetCard，VirtualDisk，和 VirtualSCSIPassThrough。

注意：不要试着用 VirtualMachineConfigSpec.devceChange 方法来改变默认设备的属性，因为 deviceChange 方法不适用于默认设备属性。

10.3 配置虚拟机

你可以在创建(Folder.CreateVM_Task)或克隆(VirtualMachine.CloneVM_Task)时配置一个虚拟机。你也可以使用 VirtualMachine.ReconfigVM_Task 重新配置虚拟机。然而，不要使用 VirtualMachine.ReconfigVM_Task 来创建或添加一个磁盘。

10.3.1 名称和位置

你可以使用 VirtualMachineConfigSpec.name 属性来为虚拟机设置显示名称。名称参数中的任何%(百分号)字符将被忽略，除非它前面使用转义字符。客户端可以忽略其他在名称参数中的字符。

使用一个注释字段来提供虚拟机的描述信息。为了删除一个已存在的描述信息，设置注释为一个空字符串。

虚拟机的位置在创建过程中隐式的确定了，因为你调用了 Folder.CreateVM_Task 方法并且指定了资源池和可选得虚拟机应该属于的目标主机。

10.3.2 硬件版本

一个虚拟机的硬件版本表明底层虚拟硬件描述了一个虚拟机支持的特性。例如 BIOS，虚拟槽位数，最大 CPUs 数，最大内存配置，其他硬件特性。

对于一个新创建的虚拟机，默认硬件版本是虚拟机所在主机上最新可用的版本。为了增强兼容性，你可以使用老版本来创建一个虚拟机。在虚拟机创建过程中指定 VirtualMachineConfigSpec.version 属性来做到。对于已经存在的虚拟机，调用 VirtualMachine.UpgradeVM_Task 方法。

虚拟机的硬件版本可以比 ESX/ESXi 主机支持的最高版本低，它在下列条件下运行：

- 迁移一个在 ESX/ESXi3.x 或以前主机上创建的虚拟机到 ESX/ESXi4.x 主机。
- 在 ESX4.x 主机上创建一个虚拟机使用一个在 ESX/ESXi3.x 或以前版本主机上创建的虚拟磁盘。
- 添加一个在 ESX/ESXi3.x 或以前版本主机上创建的虚拟磁盘到在 ESX4.x 主机上创建的虚拟机上。

硬件版本比 4 低的虚拟机可以运行在 ESX/ESXi4.x 主机上，但是会降低性能和兼容性。

尤其是，当具有硬件版本比 4 低得虚拟机存在于一个 ESX/ESXi4.x 主机上时你不能添加或移除它的虚拟设备。为了更好的利用这些虚拟机，更新虚拟硬件。

10.3.3 启动选项

你可以通过设置 VirtualMachineConfigSpec.bootOptions 属性来控制一个虚拟机的启动行为。属性中的 VirtualMachineBootOpt 数据对象允许你设定下列属性：

- bootDelay – 在开始启动序列前延迟，毫秒级。
- bootRetryDelay – 在一个启动重新开始前延迟，毫秒级。这个属性仅在 bootRetryEnabled 属性为 true 是考虑。
- bootRetryEnabled – 假如设置为 true，虚拟机启动失败后延迟 bootRetryDelay 时间段后重新尝试启动。
- enterBIOSSetup – 假如设置为 true，虚拟机在下次启动时进入 BIOS 设置。虚拟机重新设置这个标志为 false，以后的启动进程将正常。

10.3.4 操作系统

你指定的客户操作系统影响支持的设备和可用的虚拟机 CPUs 数。在下列两个属性中你指定客户操作系统。

- guestosid – 指定一个在 VirtualMachineGuestOsIdentifier 中的常量，字数串行式。
- alternateGuestName – 客户操作系统全名称。假如 guestosid 是 VirtualMachineGuestOsIdentifier 里面的一个以 other*开始的值时使用这个属性。

10.3.5 CPU 和内存信息

VirtualMachineConfigSpec 数据对象允许你来配置 CPU 和内存设置。

10.3.5.1 CPU 和内存资源分配

为了分配资源，使用 VirtualMachineConfigSpec 的 cpuAllocation 和 memoryAllcoation 属性。这两个属性包含了 ResourceAllocationInfo 对象，它拥有下列属性：

- reservation - 保证留给虚拟机可用资源的总数。假如资源的利用少于预留，其他运行的虚拟机可以使用这些资源。

- limit – 分配给这个虚拟机的 CPU 或内存资源上限值。这个虚拟机不能超过这个值，甚至资源富裕的时候。这个属性的典型应用是来确保连续的性能。假如用户在虚拟机上使用额外的资源，额外的虚拟机被添加到这个主机或资源池，这个虚拟机可能会显著的慢下来。假如设置为-1，在资源使用上没有限制。
- shares – 为在多个虚拟机中分配内存或执行能力的标准。ShareInfo 数据对象由两个属性，level 和 shares。
  - level – 选择 high，low 或 normal 来映射到一个系列预先设定的数值。设置这个属性为 custom 来设置一个明确的共享数值。
  - shares – 允许你设置分配到资源池的共享数值。这个分配量被具有同一级别的资源池之间划分。

10.3.5.2 CPU and Memory Modification for Running Virtual Machines

设置 CpuHotAddEnabled 和 CpuHotRemoveEnabled 来确定当虚拟机在运行时是否虚拟处理器能被添加或移除。设置 MemoryHotAddEnabled 来确定当虚拟机在运行时是否可以添加内存。

10.3.5.3 Number of CPUs

你可以通过 VirtualMachineConfigSpec.numCPUs 属性来为虚拟机设置虚拟处理器个数。对于这个属性合法的值依赖于你设置的 guestosid 的值。

10.3.5.4 CPU Processors and Memory Affinity

假如你的虚拟机在一个 ESX/ESXi 系统上，并且由一个支持 Symmetric Multiprocessors(SMP，对称多处理器)，你可以通过设置 cpuAffinity 和 memoryAffinity 来配置多虚拟 CPUs。你定义一系列整形值来表示处理器(对于 CPU)和 NUMA 节点(对于内存)。假如你重新配置这个密切关联设置并且清空这个队列，任何存在的密切关联将被移除。

10.3.5.5 CPU Features

对于一个虚拟机或客户操作系统你可以使用

VirtualMachineConfigSpec.cpuFeatureMask[].info 属性来表示 CPU 特性需求。参考 HostCpuIdInfo 数据对象。

10.3.6 网络

你配置网络设置，因此一个虚拟机可以通过主机和其他虚拟机通讯。

10.3.6.1 虚拟网络接口

你使用一个 VirtualEthernetCard 的子类来添加一个虚拟网络接口，你可以设置 addressType 为 Manual，Generate，或 Assigned。假如你选择了 Assigned，你可以明确的配置一个 MAC 地址。

虚拟网络接口数量依赖于你为虚拟机指定的硬件版本。硬件版本 7 中虚拟机支持 10 个虚拟 NICs。硬件版本 4 中虚拟机支持 4 个虚拟 NICs

10.3.6.2 虚拟机 MAC 地址

当虚拟机创建时，ESX/ESXi 或 vCenter Server 系统分配给每一个虚拟网络接口一个唯一的 MAC 地址。为每一个虚拟网络适配器产生的 MAC 地址的前 3 个字节由具体一个具体制造商的 Organizationally Unique Identifier(OUI，组织上的唯一标示符)。MAC 地址生成程序产生其他的 3 个字节。vSphere 为生成的 MAC 地址检测冲突。当 MAC 地址生成后，除非虚拟机被移动到其他不同的位置上否则不可以被改变。

所有的分配给正在运行的或挂起的虚拟机的虚拟网络接口的 MAC 地址都被监测。关机的虚拟机不会被检测。有可能当它移动后会获得一个不同的 MAC 地址。

10.3.7 光线通道 NPIV 设置

N-port ID 虚拟化(NPIV)支持在多个虚拟端口中共享一个单独的物理 FC HBA 端口，每一个都具有唯一标示。这个功能让你可以在每个虚拟机上控制其访问 LUNs。

每一个虚拟端口由一对 worle wide names(WWNs)：a world wide port name(WWPN)和 a world wide node name 标示。这些 WWNs 由 vCenter Server 分配。

NPIV 支持受下列限制：

- NPIV 必须在 SAN 交换机上支持。

- NPIV 仅被具有 RDM 磁盘的地虚拟机支持。具有正常的虚拟磁盘的虚拟机继续使用主机物理 HBAs 的 WWNs。
- 在一个主机上的虚拟机有使用它们的 NPIV WWNs 到一个 LUN 的访问假如在 ESX/ESXi 主机上的物理 HBAs 有使用它的 WWNs 到一个 LUN 的访问。确认这个访问同时提供给主机和虚拟机。

你可以通过 VirutalMachineConfigSpec 以 npiv 开头的属性来建立 NPIV。

10.3.8 文件位置

对于一个虚拟机的文件问题有下列属性来确定：

- VirtualMachineConfigSpec.files 是一个 VirtualMachineFileInfo 数据对象，它允许你来指定日志目录，快照目录，挂起目录，配置文件位置。大多数位置都有一个默认值，如果需要你可以修改。
- VirtualMachineConfigSpec.locationID 是一个依赖在虚拟机的配置文件位置和这个虚拟机被分配的主机的 UUID 的 128bit hash。这个属性一般不由开发人员设置；然而，假如你移动这个虚拟机建议通过设置它为一个空字符串来清除这个属性。

假如一个虚拟机的 VirtualMachineCapability.swapPlacementSupported 属性为 true，你可以为 VirtualMachineConfigSpec.swapPlacement 属性设置一个值。这个值必须是 VirtualMachineConfigInfoSwapPlacement 枚举类型中的一个值，字符串类型。

10.4 为虚拟机添加设备

你可以在虚拟机创建时使用 VirtualMachineConfigSpec.deviceChange 属性来添加设备到虚拟机中，这个属性是一个 VirtualDeviceSpec。你通过使用一个 backing 对象指定虚拟设备应该影射的主机设备。一个 backing 对象代表主机设备和一个虚拟设备关联。

- Backing option object – 你可以通过提取相关的 backing option 对象来找到主机支持的设备。
- Backing information object – backing information 对象允许你为虚拟设备设置提供数据。如下访问一个 VirtualDeviceBackingInfo：

  VirtualMachineConfigSpec.deviceChange[].device.backing

为了添加一个设备到一个虚拟机，你首先必须找到 ESX/ESXi 主机支持的一致的设备，

然后指定一个 VirtualDevice 对象，执行下列任务添加一个设备到虚拟机：

1. 通过调用 QueryConfigOption 方法来查找你的 ESX/ESXi 系统支持的设备，你可以通过 VirtualMachine.environmentBrowser 属性来访问。这个方法返回一个 VirtualMachineConfigOoption 数据对象指名 ESX/ESXi 支持的设备。例如，VirtualMachineConfigOption.hardwareOptions 包括了支持的 CPU 和内存和一个 VirtualDeviceOption 数据对象队列的信息。

   注意：你不能使用 QueryConfigOption 方法来创建一个默认设备的其他实例。假如你尝试去添加一个默认设备，例如一个 IDE 控制器，服务器忽略这个操作。
2. 为设备指定 backing 信息对象。实际对于不同的对象操作也不同。例如，对于一个 CD-ROM 通过设备，你使用一个 VirtualCdromPassthroughBackingInfo 设备。VirtualDevice.backing 属性是一个被设备扩张的 VirtualDeviceBackingInfo 对象

   下列代码片段添加一个 CD-ROM 通过设备：

   virtualCdromPassthroughBackingInfo vcpbi = new

   VirtualCdromPassthroughBackingInfo();

   //Does the virtual device have exclusive access to the CD-ROM device?

   Vcpbi.setExclusive(false);

   //Specifies the device name.

   Vcpbi.setDeviceName(‘cdrom0’);
3. 为这个设备确定连接。

   VirtualDevice.connectable 属性是一个 VirtualDeviceConnectInfo 数据对象。这个对象提供了关于当一个虚拟在运行时移除一个设备的限制。假如这个设备不可以移除这个属性为 null.

   VirtualDeviceConnectInfo vdci = new VirtualDeviceConnectInfo();

   //Allow the guest to control whether the virtual device is connected?

   vdci.setAllowGuestControl(false);

   //Is the device currently connected?

   vdci.setConnected(true);

   //Connect the device when the virtual machine starts?

   vdci.setStartConnected(true);
4. 定义控制器 key，虚拟设备 key，和单元编号。

你通过整形属性：controllerKey，key，和 unitNumber 定义这些条目。

5. 指定设备信息。

   deviceInfo 属性是一个 Description 数据对象，具有 name 属性和一个 summary 属性。你可以为每一个属性提供一个字符串，来描述这个设备。

   Description vddesc = new Description();

   vddesc.setLabel(‘CD-ROM Device cdrom0’);

   vddesc.setSummary(‘The CD-ROM device for me’);

6. 制定虚拟设备作为一个 VirtualDeviceConfigSpec 的 device 属性。

7. 指定 VirtualDeviceConfigSpec 作为 VirtualMachineConfigSpec 的 deviceChange 属性，VirtualMachineConfigSpec 传入到一个 Folder.CreateVM_Task 或 VirtualMachine.ReconfigVM_Task 方法中。

下面是关于一个 CD-ROM 通过设备完成的代码片断：

```
VirtualDevice vd = new VirtualDevice();
vd.setBacking(vcpbi);
vb.setConnectable(vdci);
vd.setControllerKey(257);
vd.setDevicdeInfo(vddesc);
vd.setKey(2);
vd.setUnitNumber(25);
```

10.5 执行虚拟机电源操作

类似于物理机器，虚拟机也有电源状态：

- Powered on – 虚拟进正在运行。假如没有安装 OS，你可以像在物理机操作那样为虚拟机安装 OS。
- Powered off – 虚拟机没有在运行。你仍可以更新在虚拟机物理硬盘上的软件，这个 功能在物理机上是不可能的。
- Suspended – 虚拟机被暂停并且可以重新开始；类似于一个物理机在待机后冬眠状态。

注意 在你启动一个虚拟机前，要确定主机有足够的资源。必须拥提供足够的内存为虚

拟机和一些内存 overhead。

虚拟机电源操作允许你来改变电源状态。每一个操作对当前状态都很敏感，例如，当启动一个关机的虚拟机会得到预期的结果，而当为一个运行的虚拟机上电的时候会得到一个错误。在你运行下列任务时你必须检查当前的状态。

- PowerOnVM_Task – 上电一个虚拟机。假如虚拟机是挂起状态，这个方法重新从挂起点执行。
- PowerOffVM_Task – 下电一个虚拟机。
- ResetVM_Task – 重新上电这个虚拟机。假如当前状态时 poweredOn，ResetVM_Task 首先执行一个 powerOff 操作。假如电源状态为 poweredOff，ResetVM_Task 执行 powerOn 操作。
- SuspendVM_Task – 挂起计算机。你可以随后上电这个挂起的虚拟机。

虚拟机经常配置成当他们启动时同时启动客户操作系统，当他们关闭时同时关闭客户操作系统。然而，开始和停止一个虚拟机不同于开始和停止客户操作系统。

注意：电源操作可能影响其它已经分配到一个 DRS 群集或 VMware HA 里面的虚拟机。

你可以使用 Datacenter.PowerOnMultiVM_Task 来上电多个在一个数据中心中的虚拟机。传入一个 VirtualMachine managed 对象引用队列和一个选项值队列给这个方法。假如在这个队列里的任何一个虚拟机是由 VMware DRS 手工管理的，系统会产生一个 DRS 建议用户手工来应用操作。独立的或 DRS 不可用的虚拟机在当前主机上加电。有 DRS 管理的虚拟机会在 DRS 建议的主机上加电。

10.6 注册和注销虚拟机

当你创建一个虚拟机时，它成为清单的一部分(默认的在你调用创建方法的目录里)，并且它被注册了。假如你拷贝虚拟机文件来重新定位虚拟机，或使用 vSphere Client 从清单中移除文件，它改变成未注册并且不可用。你不能上电一个不属于清单里的虚拟机。

为了恢复虚拟机到清单中，是它重新可用，你可以使用 RegisterVM_Task 方法，方法定义在 Folder managed 对象里。你注册这个虚拟机到一个主机或一个资源池中。假如你想从它来克隆其它虚拟机你可以注册这个虚拟机为一个模板。

ColdMigration.java 例子展示了注册和配置一个虚拟机，中心代买是下列的调用，注册这个虚拟机。参数包括虚拟机当前的文件夹，数据存储路径，名称，是否注册为一个模板，

要注册到的资源池或主机。

ManagedObjectReference taskmor = cb.getConnection().getService().registerVM_Task(

vmFolderMor，vmxPaht，getVmName()，false，resourcePool，host)；

注册完成后，虚拟机可以从它注册的资源池或主机上获得它需要的资源(CPU，内存等等)。

RemoveManagedObject.java 例子展现了注销一个虚拟机。

10.7 自定义客户操作系统

你在虚拟机上安装客户操作系统类似于在物理机上安装。以后，假如 VMware Tools 安装到客户操作系统上你可以使用 vSphere API 来检索信息和执行客户化操作。

VirtualMachine 包括下列方法来管理客户机操作系统：

- ShutdownGuset 和 RebootGuest 关闭和重起客户 OS，StandbyGuest 使客户操作系统进入到待机模式。在不同的情况下，你可以在客户 OS 上执行这个操作。例如，你可以关闭 Windows 但是保持虚拟机在运行。
- ResetGuestInformation 清除已经保存的客户信息。客户信息仅当虚拟机关闭电源的情况下可以被清除。假如过时的信息被存储使用这个方法，可以避免重复使用一个 IP 地址或 MAC 地址。
- SetScreenResolution 设置客户操作系统控制台屏幕大小。当你调用这个方法，你在 vSphere Client 里面访问的虚拟机控制台立刻被改变。

你可以通过 CustomizationSpec 数据对象来为客户 OS 自定义身份和网络设置，CustomizationSpec 是 VirtualMachine.CustomizeVM_Task 的一个参数。CustomizationSpec 也是你克隆一个虚拟机时用到的 VirtualMachineCloneSpec 的一个属性。

这个方法主要为虚拟机设置来进行自定义设置，但是因为虚拟机和客户 OS 共享信息，所以你也可以用这个方法来自定义客户 OS。

CustomizationSpec 允许你来设置下列属性：

- encryptionKey – 可以被当作公钥来加密管理员密码的字节数组。
- globalIPSettings – 包括一个 CustomizationGlobalIPSettings 数据对象，它指定了一个 DNS 服务器列表和一个虚拟网路适配器的名称解析后缀列表。
- identity – 你可以指定网络标示和设置，类似于 Microsoft Sysprep 工具。

- nicSettingMap – 自定义 IP 设置到一个特别的虚拟网络适配器上。
- options – 可选操作(LinuxOptions 或 WinOptions)。

10.8 安装 VMware Tools

VMware Tools 是一套提高虚拟机客户操作系统和改善虚拟机管理的工具。为每一个客户 OS，VMware 提供一个相应的二进制的版本。SDK 需要你安装 VMware Tools，否则一些和客户操作系统关联的操作将失败。

注意：你必须在安装 VMware Tools 之前安装客户操作系统。

随着 VMware Tools 安装到客户 OS 上，虚拟机获得它的 DNS(domain name server)和一个 IP 地址，因此可以访问网络。

VirtualMachine 包括三个方法来自动安装和升级 VMware Tools。

- MountToolsInstaller – 为客户操作系统挂载 VMware Tools CD 安装程序。为了监控工具安装的状态，检查 GuestInfo.toolsStatus。检查 GuestInfo.toolsVersionStatus 和 GuestInfo.toolsRunningStatus 相关信息。
- UnmountToolsInstaller – 卸载 VMware Tools 安装程序 CD。
- UpgradeToolsTask – 执行 VMware Tools 更新。这个方法假设 VMware Tools 已经安装并且正在运行。这个方法需要一个参数，InstallerOptions，可以指定命令行选贤传递给安装程序来修改 tools 的安装过程。

使用在 VirtualMachineConfigSpec.toolsInfo 属性里的 ToolsConfigInfo 数据对象来为运行在客户操作系统上的 VMware Tools 软件明确配置。

10.9 虚拟机升级

你可以运行 VirtualMachine.UpgradeVM_Task 方法来升级虚拟机硬件。方法升级这个虚拟机的虚拟硬件到这个虚拟机所在的主机支持的最新版本。你可以指定版本号作为一个参数。假如你想你的虚拟机运行在一个支持新的硬件版本的虚拟化层上时这个方法很有用处。

11、虚拟机管理

虚拟机可以像物理计算机那样操作和配置，虚拟机也支持物理计算机不支持的功能。

11.1 虚拟机迁移

迁移是从一个主机或存储位置移动一虚拟机到另外位置的过程。通过拷贝来创建一个新的虚拟机，不是迁移。vSphere 支持下列迁移类型：

Table 11-1. vSphere Migration Types

Migration Type	Description
Cold migration	Moves a powered-off virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage locations. Cold migration can be used to migrate virtual machines from one datacenter to another.
Migration of a suspended virtual machine	Moves a suspended virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage location. You can migrate suspended virtual machines from one datacenter to another.
Migration with VMotion	Moves a powered-on virtual machine to a new host. Migration with VMotion allows you to move a virtual machine to a new host without interruption in the availability of the virtual machine. Migration with VMotion cannot be used to move virtual machines from one datacenter to another.
Migration with Storage VMotion	Moves the virtual disks or configuration file of a powered-on virtual machine to a new datastore. Migration with Storage VMotion allows you to move a virtual machine's storage without interruption in the availability of the virtual machine.

一个暂停虚拟机的迁移和通过 VMotion 的迁移都称为热迁移，因为他们不需要虚拟机关闭它的电源。

你可以手工移动虚拟机或设置计划任务来执行冷迁移。

11.1.1 冷迁移

假如一个虚拟机关闭了，你可以通过拷贝所有虚拟机文件到一个不同的目录来迁移它到不同的群组，资源池或主机。ColdMigration 例子展示了这个功能。

11.1.2 使用 VMotion 迁移

VMware VMotion 支持在线迁移正在运行的虚拟机从一个服务服务器到另一个。你仅能在同一个数据中心的两台由一个 vCenter 服务器管理的主机间执行这个迁移。当要移动它到不同的数据中心时必须下电这个虚拟机。

当你调用 VirtualMachine 对象的 MigrateVM_Task 方法时，你要指定一个主机或资源池来迁移它。任务的级别和虚拟机电源状态是可选项。VMotion 例子执行下列任务：

- 使用 QueryVMotionCompatibility_Task 来检查主机的兼容性 。
- 使用 CheckMigrate_Task 来检查迁移是否可行。例如，假如来台主机不兼容，虚拟机部不可以从一个迁移到另一台。

- 使用 CheckRelocation_Task 来检查是否重定位可行。

假如主机兼容，例子执行迁移操作。

11.1.3 使用存储 VMotion

存储 VMotion 允许你移动一个正在运行的虚拟机从一个 VMFS 卷到另外一个。虚拟机或它关联的存储不需要离线。所有数据存储都被支持，包括本地存储，VMFS，和 NAS(network attached storage)。

你可以放置虚拟机和它的所有磁盘在一个单一的位置，也可以为虚拟机配置文件和每个虚拟磁盘选择肯开存放。在存储 VMotion 期间虚拟机保留在同一个主机上。

为了执行存储 VMotion，使用 VirtualMachine.RelocateVM_Task 方法。RelocateVMSpec 传入到方法中，RelocateVMSpec 确定了目标存储和目标主机或资源池。

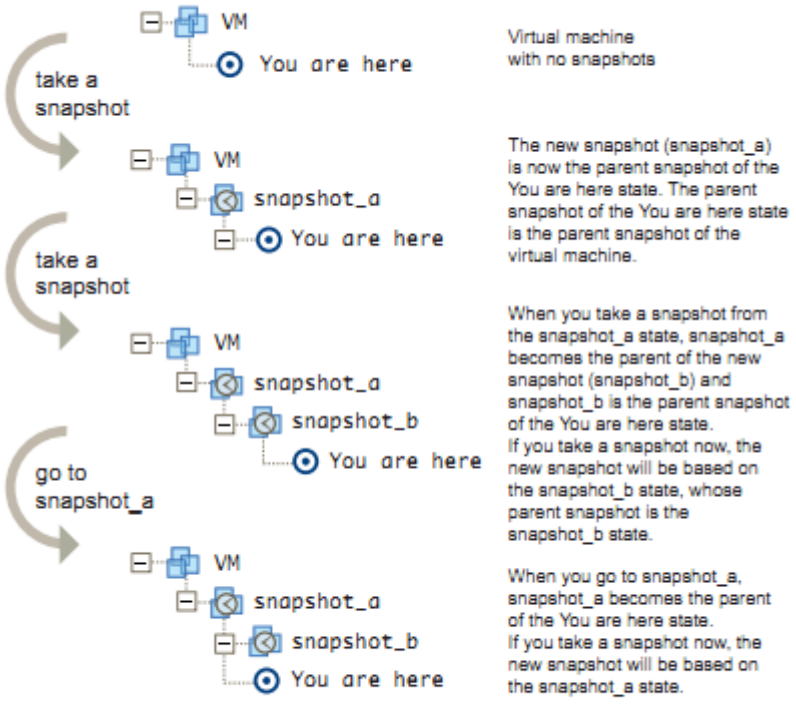
11.2 快照

快照是虚拟机磁盘文件（VMDK）在某个点及时的复本。快照包括所有虚拟磁盘上的数据的状态和这个虚拟机的电源状态(on，off，suspended)。你可以在虚拟机上电，下电，挂起状态下拍快照。

当你创建了一个快照，系统为这个快照在数据存储上创建了一个 delta 磁盘文件并且写入任何变化到这个 delta 磁盘中。你随后可以恢复到虚拟机以前的状态。

VirtualMachine 对象有为创建快照的方法，恢复到树上的任何任何快照点，和移除快照。

Figure 11-1. Virtual Machine Snapshots



When you revert a virtual machine, the virtual machine returns to the parent snapshot of the virtual machine (that is, the parent of the current You are here state).

快照层次可能变得很复杂。例如，假设在图 11-1 的例子中，你恢复到到 snapshot_a。然后开始工作并且改变到 snapshot_a 虚拟机，创建一个新的快照，实际上，创建一个分支树。

11.2.1 创建一个快照

VirtualMachine.CreateSnapshot_Task 方法创建一个虚拟机的新快照。附带着，当前快照变成这个新快照的父级。

该方法允许你为快照指定名称同时需要你设置 memory 和 quiesce 属性。

- memory – 假如为 true，虚拟机内部状态的 dump(一个基本的 memory dump)包含在。快照里。内存快照消耗时间和资源，需要一些时间来创建。当设置为 false，快照电源状态被设置成关闭。
- quiesce – 假如为 true 并且在拍快照时虚拟机处于开机状态，VMware Tools 通常用来静默在虚拟机中的文件系统。确保一个磁盘快照和客户文件系统状态一致。假如虚拟机为关闭状态或 VMware Tools 不可用，quiesce 标志被忽略。

VMSnapshot.java 例子调用这个方法如下：

ManagedObjectReference taskMor = service.createSnapshot_Task(vmMor，

snapshotName，desc，false，false)；

方法返回 MOR 到一个监控这个操作的 Task 对象。如果成功这个任务里的 info.result 属性包括了新创建的 VirtualMachineSnapshot。

11.2.2 恢复到一个快照

当你恢复到一个快照，你恢复一个虚拟机到拍快照时的状态。VirtualMachine.RevertToSnapshot_Task 允许你指定一个目标主机和是否虚拟机开机。

当拍快照时假如虚拟机正在运行，你恢复它，你必须指定一个快照恢复到的主机，或设置 SupressPowerOn 标志为 true。

11.2.3 删除一个快照

你可以通过调用 VirtualMachine.RemoveAllSnapshots 或 VirtualMachineSnapshot.RemoveSnapshot_Task 方法删除所有快照。VirtualMachineSnapshot 对象是前面 CreateSnapshot_Task 方法返回的。

11.3 链接的虚拟机

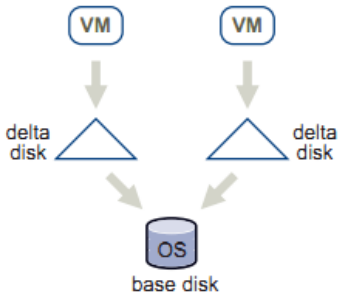
链接虚拟机是两个或多个虚拟机共享存储和支持重复数据有效共享。

11.3.1 链接的虚拟机和磁盘备份

在它最简单的形式里，通过使用 delta disk backings 功能来实现共享存储。一个 delta disk backing 是一个虚拟磁盘文件位于一个标准的虚拟磁盘 backing 文件头部。每一次虚拟机上的客户操作系统在一个虚拟磁盘上写，数据被写入到 delta 磁盘。每一次虚拟机上的客户操作系统从磁盘读，虚拟机首先把在 delta 磁盘上的磁盘快定为目标。假如数据不在 delta 磁盘上，虚拟机会在基本磁盘上查找。

链接虚拟机可以通过一个快照或当前运行点来创建。当你创建了一系列链接虚拟机后，他们共享基本磁盘 backing 并且每个虚拟机拥有自己的 delta disk backing，如图 11-2。

Figure 11-2. Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing



注意 我们建议在一个链接虚拟机组中上限是 8 个主机虚拟机访问同一个基础磁盘。然而，在这个组中的每一个主机虚拟机的链接虚拟机数是没有限制的。

11.3.1.1 HA 群集限制

在一个链接克隆组里的虚拟机可以使 VMware HA(high availability)群集的组成部分。群集里面的主机数量影响 HA 的重新启动一个失败虚拟机的能力。假如群集有 8 个或更少的主机，链接虚拟机可以正常的重新启动。然而，假如群集里有超过 8 个主机，HA 可能不能在虚拟机失败后重新启动它。这是因为 HA 不知道在一个链接克隆组里的虚拟机有 8 个主机限制，所以当错误放生时，HA 可能试着在不属于最大主机限制的主机组里面的一个主机上重新启动虚拟机。HA 将尝试 5 次在不同的主机上。例如，在群集里有 13 台或更多主机，可能 HA 将不会在一个属于链接克隆组里的主机上尝试。在 vSphere4.0，4.1 和 5.0 是一样的。

11.3.2 创建一个链接的虚拟机

你可以通过下列两种方法来创建链接虚拟机：

- 从一个快照克隆虚拟机。
- 从当前虚拟机状态克隆虚拟机。这个状态可能和快照点不同。

11.3.2.1 通过一个快照创建创建链接的虚拟机

首先你创建一个快照，然后从这个快照上创建链接虚拟机。

1. 为创建快照，为这个虚拟机调用 CreateSnapshot_Task 方法。这个虚拟机可以处于任何电源状态。下列代码描述了创建一个名为 snap1 的快照。代码不包括一个 memory dump。VMware Tools 通常会在虚拟机上电的情况下静默虚拟机中的文件系统。

myVm.CreateSnapshot(“snap1”，”snapshot for linked virtualmachine”，False，True)

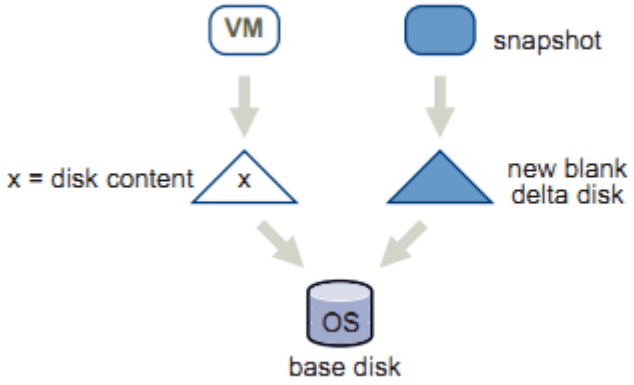
2. 为创建链接虚拟机，指定你创建的快照并且使用 createNewDeltaDiskBacking 的 VirtaulMachineRelocateDiskMoveOptions.diskMoveType，如例子 11-1 展示。通过一个工作在可处于任何电源状态下虚拟机的快照来创建链接虚拟机。

Example 11-1 Creating a Linked VirutalMachine from a Snapshot

```
relSpec = new VirtualMachineRelocateSpec();
relSpec.diskMoveType =
    VirtualMachineRelocateDiskMoveOptions.createNewChildDiskBacking;
cloneSpec = new VirtualMachineCloneSpec();
cloneSpec.powerOn = false;
cloneSpec.template = false;
cloneSpec.location = relSpec;
cloneSpec.snapshot = myVm.snapshot_currentSnapshot;
myVm.Clone(myVm.parent，myVm.name + “-clone”，cloneSpec);
```

结果是一个具有一样基础磁盘的虚拟机，但是是一个新的 delta disk backing。

Figure 11-3. Creating a Linked Virtual Machine from a Snapshot



11.3.2.2 通过当前运行点创建一个链接虚拟机

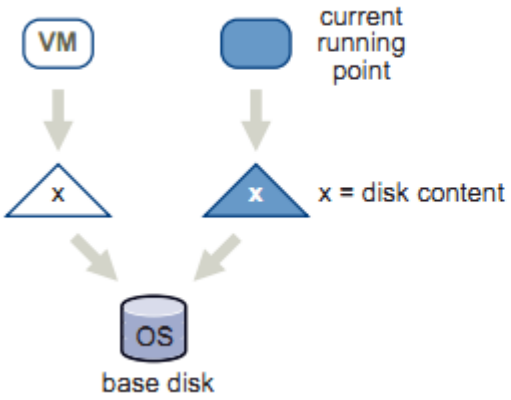
为了从当前运行点创建一个虚拟机，克隆这个虚拟机，如例子 11-1 中，但是要使用一个 moveChildMostDiskBacking 的 diskMoveType。虚拟机可以处于任何电源状态下。

Example 11-2. Creating a Linked Virutal Machine from the Current Running Point

---

```
relSpec = new VirtualMachineRelocateSpec();
relSpec.diskMoveType =
    VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking;
cloneSpec = new VirtualMachineCloneSpec();
cloneSpec.powerOn = false;
cloneSpec.template = false;
cloneSpec.location = relspec;
myVm.Clone(myVm.parent，myVm.name + “-clone”，cloneSpec);
```

Figure 11-4. Creating a Linked Virtual Machine from the Current Running Point



11.3.3 移除快照和删除链接虚拟机

在你创建了一组链接虚拟机后，你可以移除一个作为一个链接虚拟机基础的快照，或删除一个虚拟机。这些操作影响在链接虚拟机组里的磁盘。为磁盘合并或删除需要链接到 vCenter 服务器上来执行这些操作。

- 快照删除 – 在快照删除期间，快照元数据也被删除，由这个快照产生的虚拟机不再显示包括快照了。假如直接链接 ESX/ESXi 主机来移除一个快照，共享的磁盘不会被合并并且不需要的 delta 磁盘可能会出现。假如链接 vCenter 服务器来删除一个快照，共享的磁盘不会被合并，但是不共享的磁盘会被合并。
- 虚拟机删除 – 当你直接链接 ESX/ESXi 主机来删除一个虚拟机时，共享的磁盘不会被删除。当你链接 vCenter 服务器来删除一个虚拟机时，共享的磁盘不会被删除。但是不共享的磁盘会被删除。

注意 在删除主虚拟机前删除所有通过它创建的链接虚拟机，因此你不会有孤立的或错误的磁盘文件出现在你的文件系统上。

11.3.4 在一个链接虚拟机群中迁移一个虚拟机

你可以在数据存储和 save 储器间移动在一个链接虚拟机组里的虚拟机，就像例子 11-3 展示的。delta 磁盘的内容可能没有基础的内容重要，你可以通过移除 delta 磁盘来保存存储。

Example 11-3. Relocating a Linked Virtual Machine

```
---------------------------------------------------------------------------------------------------------
relSpec = new VirtualMachineRelocateSpec();
relSpec.diskMoveType =
        VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking;
relSpec.datastore = localDatastore;
myVm.Relocate(relSpec);
---------------------------------------------------------------------------------------------------------
```

你可以迁移多个链接虚拟机到一个新的数据存储，但是在迁移过程中保持所有的共享的存储不变。为了完成迁移，要求虚拟机一个一个来迁移，给出选项来允许重新链接到一个存在的磁盘上，如例子 11-4。

Example 11-4. Relocating Multiple Linked Virtual Machines

```
---------------------------------------------------------------------------------------------------------
relSpec = new VirtualMachineRelocateSpec();
relSpec.diskMoveType =
        VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking;
relSpec.datastore = localDatastore;
relSpec.datastore = targetDatastore;
myVm.Relocate(relSpec);
---------------------------------------------------------------------------------------------------------
```

11.3.5 提升一个虚拟机磁盘

提升一个虚拟机磁盘来提升性能。

重要的 当链接到 vCenter 服务器时你只能使用 PromoteDisks API。

你可以使用 PromoteDisks 来拷贝磁盘 backings 或合并磁盘 backings。

- 拷贝 – 加入 unlink 参数为 true，任何由多个虚拟机共享的磁盘 backing 为拷贝的，因此这个虚拟机拥有自己的非共享版本。文件被拷贝到虚拟机的 home 目录。这个设置结果提升读取性能，但是需要更多的空间。下列调用来拷贝和共享磁盘，然后消除所有不需要的磁盘。

  myVm.PromoteDisks(true,[]);

- 合并 – 假如 unlink 参数为 false，任何没有在多个虚拟机间共享的磁盘或没有和一个快照关联的磁盘 backing 都会和它的子 backing 合并。这个设置以禁止共享功能来提升读取性能。下列调用消除任何不需要的磁盘：

  myVm.PromoteDisks(false,[]);

假如你以不被快照使用或不和其他虚拟机共享的磁盘 backings 为结束的话升级一个虚拟机磁盘是有用的。

同时使用 PromoteDisks 的第二个参数，允许你进在磁盘的子集上应用该方法。例如，你可以不共享和仅合并 key 为 2001 的虚拟磁盘，如下：

```
for any of my VMs in dev
    if( dev.key == 2001)
        disk2001 = dev
myVm.PromoteDisks(true,[disk2001]);
```

11.3.6 执行 Delta 磁盘的高级操作

为了 delta 磁盘的高级操作，你可以使用 VirtualDeviceConfigSpec 方法例如 VirtualDeviceConfigSpec.create 和 VirtualDeviceConfigSpec.add。

add 和 create 两个方法允许你在一个存在的磁盘顶部创建一个空的 delta 磁盘。你可以在 VirtualDeviceSpec 里明确是 add 还是 create 并且传入一个 parent 属性是一个已存在磁盘的虚拟磁盘。方法创建一个新的 delta 磁盘，它的父辈是已存在的磁盘。

注意 不要使用 VirtualMachine.ReconfigVM_Task 调用来创建或添加一个 delta 磁盘。

例子 11-5 展示了如何为虚拟机的第一个虚拟磁盘添加 delta 磁盘。

Example 11-5. Creating a Virtual Machine

```
---------------------------------------------------------------------------------------------------------
disk = none;
for any of my VMs in dev
      if(VirtualDisk.isinstance == dev)
            disk = dev;
#Remove the disk
removeDev = new VirtualDeviceConfigSpec();
removeDev.operation = “remove”;
removeDev.device = disk;
#Create a new delta disk which has the orginal disk as its parent disk
addDev = new VirtualDeviceConfigSpec();
addDev.operation = “add”;
addDev.fileOperation = “create”;
addDev.device = copy.copy(disk);
addDev.device.backing = copy.copy(disk.backing);
addDev.device.backing.fileName = “[“ + disk.backing.datastore.name + “]”;
addDev.device.backing.parent = disk.backing;

spec = new VirtualMachineConfigSpec();
spec.deviceChange = [removeDev, addDev];
vm.Reconfigure(spec);
---------------------------------------------------------------------------------------------------------
```

12、虚拟应用

一个虚拟应用由一个或多个虚拟机组成。它可以作为一个单一的单元被部署，管理和维护。

12.1 关于虚拟应用

一个虚拟应用指明和封装虚拟机和应用的组件，操作策略和服务级别和这些组件关联。一个虚拟应用可以被简单的当做一个独特的具有确定的操作系统(virtual appliance)虚拟机，或复杂如一个公司 web 站点。每一个在一个虚拟应用中的虚拟机包含一个预装的，预先配置的操作系统和可能还包还一个优化的应用程序栈来提供一系列服务。

在 vSphere Web Services SDK 中，VirtualApp managed 对象代表一个虚拟应用。一个 VirtualApp 对象通过下列能力来扩展 ResourcePool：

- 在 vAppConfigInfo 中存放产品信息如产品名称，提供商，属性，许可证。
- 详细定义了开机和关机的顺序。
- VirtualApp 对象可以作为 OVF 包导入和导出。
- 使用 OVF 环境执行应用级自定义。

12.1.1 管理概述

你可以用 Web Services SDK 通过下列步骤来创建和管理虚拟应用：

1. 调用 CreateVApp 方法来创建一个没有孩子的虚拟应用。
2. 添加 child 对象。
3. 导出 VirtualApp 为 OVF(ExportVApp 方法)。

你随后可以导入这个 OVF 来创建和自定义虚拟应用。

12.1.2 直接的和链接的子项

一个虚拟应用由一个或多个子虚拟机或虚拟应用构成。VirtualApp 子项具有下列特征：

- 每一个子项有确定的父 VirtualApp。
- 每一个子项参与到开机和关机顺序里。
- 每一个子项的生命周期由父 VirtualApp 对象决定。

VirtualApp 子项可以是直接的或链接的，取决于这个子项来源于它的资源。

- Direct Children.一个虚拟应用的 direct child 是你显示的添加的一个虚拟机或虚拟应用对象。Direct child 和它的父 VirtualApp 对象共享资源。虚拟机和虚拟应用都可以是 direct child。

- Linked Children. 一个虚拟应用的一个 linked child 是你通过调用 UpdateLinkedChild 方法来添加的一个虚拟机或虚拟应用。Linked child 通过允许 child 实体来使用父 VirtualApp 对象不同的资源来提高 VirtualApp 的伸缩性。Linked child 可以使不同群集的一部分，但是一个虚拟应用和它的孩子必须在同一个数据中心中。虚拟机和虚拟应用都可以是 linked child。

Linked children 提供了更好的弹性。尤其是，你可以跨群集建立虚拟应用。vSphere Client 不支持添加或移除链接，它只是显示链接。

当你添加一个 linked child 到一个虚拟应用中时，遵守下列规则：

- 假如 UpdateLinkedChildren 方法在其他虚拟应用的一个直接的孩子上调用的话，一个 InvalidArgument 错误将被抛出。
- 当你添加一个已经是其他虚拟应用的 linked child 的虚拟机或虚拟应用时，已经存在的链接将被移除并且被新联替代。
- 一个 linked child 的生命周期由 VAppEntityConfigInfo 数据对象的 destroyWithParent 属性来决定。假如设置为 true，当父 VirtualApp 被销毁时 child 也将被销毁。否则，当 VirtualApp 被销毁时链接被移除。

假如你添加包含多个实体的一个虚拟应用，例如多个虚拟机，实体按顺序移动和在同一个时间保证一个，在列表里定义。假如一个错误出现，方法终止并抛出一个异常。

12.1.3 OVF 包

打开 Virtualization Format(OVF)是一个虚拟应用的发布格式。vSphere 使用 OVF 包作为发布和存储虚拟应用的一个单元。因为这些实体的上传，下载，存储在 OVF 包格式里，vSphere 支持多种的访问和部署虚拟应用。

一个典型的虚拟应用由一个或多个虚拟磁盘文件和一个配置文件组成。

- 虚拟磁盘包括运行在虚拟应用中的虚拟机上的操作系统和应用。
- 配置文件包括描述虚拟应用如何配置和部署的原数据。

一个 OVF 包也可以包括证书和清单文件。

OVF 包包还描述能力和虚拟应用基础架构需求，包还到虚拟磁盘和其他存储虚拟机状态文件的引用。这个信息大多保存在一个 XML 文档里被称为 OVF 包裹。当一个 OVF 包被实例化到一个虚拟应用或一个虚拟机对象中(依赖于包裹中的原数据)，然后存储在 OVF 包

裹中的配置信息将被应用到这个虚拟应用或虚拟机对象上。

一些在 OVF 文件中的信息是不变的，全部的包还在 VirtualApp 对象体中 ovf:Section_Type 元素。其他部分被实体改变或扩展。你不需要了解全部的 OVF 包元素知识，只要了解基础关键部分明白他们如何关联到虚拟应用的就可以。

12.2 创建一个 VirtualApp

你通常会创建一个不具有 children 的 VirtualApp。CreateApp 方法包含下列参数：

- resSpec – 指定一个资源池的属性。
- configSpec – VappConfigSpec 数据对象为了确定虚拟应用详细信息。
- vmFolder – 依赖于 VirtualApp 结构：
  - 当创建 top-level 虚拟应用时，就是，虚拟应用不具有上级虚拟应用，你必须指定一个文件夹。
  - 假如 VirtualApp 在上级列表里拥有其他的虚拟应用，当你创建这个 VirtualApp 时 folder 参数必须为 NULL

12.3 管理 VirtualApp 子项

你可以已 direct 或 linked children 的方式添加虚拟机和虚拟应用到你你的虚拟应用中。使用不同的方法添加或移除 direct 或 linked children，如下：

- Direct children.使用下列方法之一：
  - CreateChildVMTask 添加一个虚拟机。
  - CreateVApp 添加一个新的虚拟应用。
  - MovIntoResourcePool 添加或移除一个存在虚拟机或虚拟应用。
- Linked children.使用 UpdateLinkedChildren 来添加或移除虚拟机或虚拟应用。

你可以调用 UpdateVappConfig 方法来明确如何使每一个虚拟机适合到虚拟应用中。

Table 12-1. VAppEntityConfigInfo Properties

Property	Enumeration
`destroyWithParent`	True if the entity should be removed when the `VirtualApp` is removed.
`key`	Key for the virtual machine or virtual application, a managed object reference to the child.
`startAction`	One of the strings in the `VAppAutoStartAction` enumeration.
`startDelay`	Delay, in seconds, before continuing with the next entity.
`startOrder`	Specifies the start order for this entity. Entities are started from lower numbers to higher-numbers and reverse on shutdown. Multiple entities with the same start order are started in parallel and the order is unspecified. This value must be 0 or higher.
`stopAction`	Defines the stop action for the entity. Can be set to none, `powerOff`, `guestShutdown`, or `suspend`. If set to none, then the entity does not participate in auto-stop.
`stopDelay`	Delay, in seconds, before continuing with the next entity.
`tag`	Tag for the entity.
`waitingForGuest`	Determines if the virtual machine should start after receiving a heartbeat, from the guest.

12.4 导出一个虚拟应用

你可以把一个虚拟应用导成一个 OVF 模板。为了导出虚拟应用，其中的所有虚拟机必须关机。

12.4.1 获得导出租约

调用 VirtualApp.ExportVapp 方法来获得一个在这个 VirtualApp 对象上的导出租约。导出租约包含一个 VirtualApp 对象里面的虚拟机磁盘的 URLs 列表，就像访问这些 URLs 的入场卷。

方法返回一个 HttpNfcLease managed 对象引用。你可以检索关于状态的信息，过后，检索租约的内容从这个 managed 对象应用中。

注意 取得一个租约需要时间，周期性的检测获取的状态(HttpNfcLeaseState 类型)来决定什么时间可以继续操作和下载磁盘。

当租约已经取得，在 VirtualApp 对象中的虚拟机被锁住。锁住的虚拟机不能被开机，删除，或任何可能打断导出操作的状态改变。

12.4.2 下载磁盘

当租约准备好后，HttpNfcLeaseInfo.deviceUrl 属性是一个 HttpNfcLeaseDeviceUrl 数据对

象队列。每个对象提供一个影射到逻辑设备 IDs 上传和下载的 URLs。从租约对象返回的 URLs 通常直接发布或取得虚拟磁盘。虚拟磁盘必须是 stream-optimized VMDK 格式。在虚拟磁盘格式声明中描述了这个格式。

当下载每一个磁盘时，你可以为结果磁盘文件创建一个文件名称。你必须选择一个与租约里的设备 URL 一致的 key 的名称(HttpNfcLease.info.deviceURL.key)。生成 OVF 描述符需要这个 key。

客户端必须在导出租约期间周期性的调用 HtppNfcleaseProgress 方法来阻止租约失效。

你可以调用 HttpNfcLeaseAbort 方法来终止租约。调用这个方法释放虚拟应用上的锁。

12.4.3 产生 OVF 描述符

在最后导出步骤期间，你通过调用 OvfManager.createDescriptor 方法生成一个 OVF 描述符。你必须为每一个下载的文件指定名称和大小。使用 CreateDescriptorParams.ovfFiles 对列来描述文件。假如这个队列为空，占位符被放入到 OVF 描述符里并且客户端可以更新占位符。

通过创建描述符，你通过客户端提供现在的磁盘文件信息，选择的文件名和需要的大小，到服务器。服务器在下载期间根据信息给与压缩。

12.4.4 完成租约

在生成 OVF 描述符后，调用 HttpNfcLease.HttpNfcLeaseComplete 来更新任务信息并且释放在虚拟应用上的锁。

12.5 导入一个 OVF 包

为了导入虚拟应用 OVF 模板，按照如下几个基本步骤。对于包含单一虚拟机的 OVF 包或包含多个复合的虚拟应用的 OVF 包步骤是一致的。

1. 调用 OvfManager.parseDecriptor 解析 OVF 描述符。
2. 调用 OvfManager.validateHost 来验证目标 ESX/ESXi 主机。
3. 调用 OvfManager.createImportSpec 来创建 VirtualAppImportSpec。

   这个结构包含了在 vCenter 服务器上创建实体的所需的所有信息，包括 children。

   客户端不需要读或编辑 VirtualAppImportSpec 来执行基本 OVF 操作。

4. 调用 Resource.importVApp 创建 vCenter 服务器实体。

   这个方法适用一个解析了 OVF 描述符来创建 VirtualApp 和 VirtualMachine 对象在 vSphere 环境中。

导入自身的进程包含下列两步：

- 服务器创建虚拟机和虚拟应用。

  你必须等待服务器创建好所有清单对象，在对象创建期间，服务器监控 ImportVapp 调用返回的 HttpNfcLease 对象的 state 属性。当服务器完成对象创建，服务器更改租约为 ready 状态，你可以开始上传虚拟磁盘内容。假如在服务器创建清单对象期间出现一个错误，租约改变为 error 状态，并且导入进程中止。

- 客户端应用上传虚拟机盘内容以具有磁盘内容 HTTP POST 请求方式提供 URLs。自盘是 stream-optimized VMDK 格式。

  当所有清单对象已经创建完成并且 HttpNfcLease 已经改变到 ready 状态，你可以上传磁盘文件，通过 HttpNfcLease 对象的 info 属性提供的 URLs。你必须在租约保持租约活动期间调用 HttpNfcLeaseProgress 方法报告进度到服务器。失败将导致租约过期，中止导入进程。

当你上传完磁盘，调用 HttpNfcLeaseComplete 方法完成租约。你可以调用 HttpNfcLeaseAbort 方法来终止导入进程。

假如导入进程失败，被终止，或超时，所有被创建的清单对象将被移除，包括所有虚拟磁盘。

12.6 虚拟应用生命周期

你可以给一个虚拟应用开机或关机，还可以执行其生命周期性的他操作。

12.6.1 上电和下电一个虚拟应用

你可以使用 PowerOnVApp_Task 方法来上电一个 VirtualApp 对象。这个方法按照在虚拟应用配置中设置好的启动顺序来启动虚拟机或子虚拟应用。

当一个虚拟应用启动时，所有子实体上的电源操作将不可用。

假如一个在一个虚拟应用中的虚拟机启动失败，返回一个异常并且上电顺序终止，在失败的情况下，已经启动的虚拟机将保持上电状态。

你可以使用 PowerOffVAPP_Task 方法来下电一个虚拟应用。这个方法按照在虚拟应用配置中设置好的启动顺序来停止虚拟机或子虚拟应用。假如 force 设置为 true，该方法停止所有虚拟机(没有明确的顺序，有可能并行)不管 VirtualApp 对象自启动配置。

当一个虚拟应用正在停止时，所有子实体上的电源操作将不可用。

12.6.2 注销一个虚拟应用

你可以调用 UnregisterVApp_Task 方法来从清单中移除一个 VirtualApp 对象而不移除所有在磁盘上的任虚拟机文件。所有具有管理服务器(ESX/ESXi 或 vCenter Server 系统)的 high-level 信息存储将被移除，包括 VirtualApp 对象配置信息，统计信息，权限信息和警告信息。

12.6.3 挂起一个虚拟应用

你可以调用 SuspendVApp_Task 方法来挂起一个虚拟应用中的所有正在运行的虚拟机，包括子虚拟应用中的虚拟机。虚拟机挂起通常使用下电操作的顺序，和上电顺序相反。

当一个虚拟应用正在挂起时，所有的在子实体上的电源操作将不可用。假如你尝试执行一个电源操作，将产生一个 TaskInProgress 错误。

12.6.4 销毁一个虚拟应用

当一个 VirtualApp 对象被销毁时，所有它的虚拟机和任何子虚拟应用都将被销毁。

VirtualAppVAppState 类型定义了一个 VirtualApp 对象可以处于的一组状态。在 started 和 stopped 之间短暂的状态也被显示的定义。从一个虚拟应用 starting 或 stopping 到完成可能会花费几分钟时间。

一个 linked child 的 life-time 由 VAppEntityConfigInfo 数据对象上的 destoryWithParent 属性来定义。假如设置为 true，当父虚拟应用被销毁时 child 也将被销毁。否则，当这个虚拟应用被销毁时仅连接被移除。

13、资源管理

构成所有虚拟组建基础的是主机系统上实际的物理资源，例如 CPU，RAM，存储，网

络基础设施，等等。vSphere 支持在一个独立的主机或跨主机的资源池上共享资源。vSphere 也支持群集的失效备援或负载均衡。

13.1 资源管理对象

ComputeResource 或 ClusterComputeResource managed 对象是所有环境中资源管理的中心。

- ComputeResource managed 对象代表一组虚拟机的一系列资源。一个 ComputeResource 总是和一个 root ResourcePool 对象关联。
- ResourcePool managed 对象代表一个单独主机的一系列物力资源，一个主机资源子集，或跨多个主机的资源。资源池可以通过创建子资源池来进一步细分。虚拟机只有和资源池管理上才可以上电。
- ClusterComputeResource 数据对象聚合多个关联的 HostSystem 对象计算资源到一个可以被虚拟机使用的单一计算资源中。假如你计划使用 VMware 群集服务例如 HA(High Avalability)，DRS(Distributed Resource Scheduling)，或使用 EVC(Enhanced vMotion Compatibility)，使用 ClusterComputeResource。

重要提示 HA，DRS 和 EVC 需要许可证，假如任何群集功能不正常工作了，检查是否为它添加了许可证。

13.2 资源管理介绍

一个 ESX/ESXi 主机基于以下几个因素来为每个虚拟机分配基于硬件的部分资源。

- ESX/ESXi 主机，资源池，虚拟机所属群集的总的可用资源。
- 上电虚拟机的个数和这些虚拟机使用的资源。
- 管理虚拟化所需的 Overhead。
- 用户定义的限制。

资源管理允许你为虚拟机动态分配资源，因此你可以更有效的使用可用容量。你可以通过下列方式来改变资源分配。

- 为单独的虚拟机指定资源。
- 创建一个资源池层次并且添加虚拟机到一个适合的资源池中。

- 添加主机和虚拟机到一个群集这样你能使用 VMware DRS 的高级特性来建议或自动资源重新分配。

13.3 资源分配

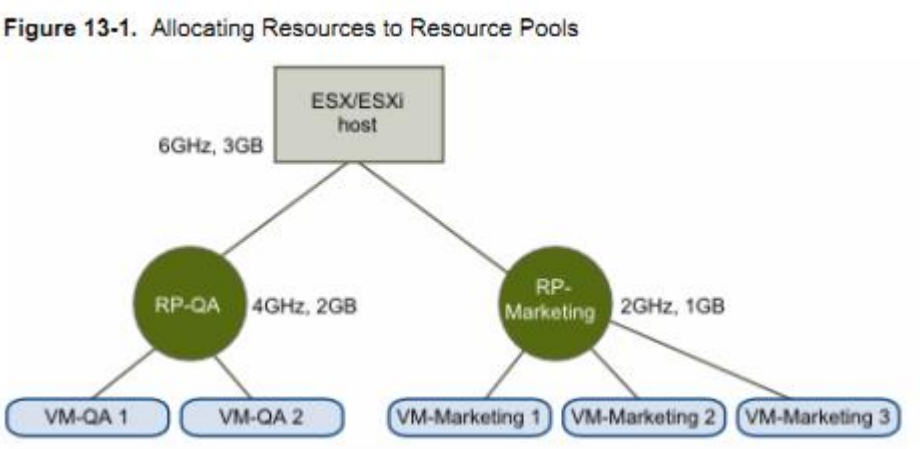
当你创建了一个虚拟机时，通常你明确分配给虚拟机一个可以取得资源的资源池和一个可选的虚拟机因该运行在其之上的主机。你可以如下访问资源池：

- Standalone host – 当你调用 Folder.AddStandaloneHost_Task 时，调用返回一个 Task 对象包含了 ComputeResource。ComputeResource.resourcePool 属性是和计算资源(和主机)关联的根资源池。
- Cluster – 当你调用 Folder.CreateClusterEx 时，方法返回一个 ClusterComputeResource 实例的 managed 对象引用。因为 ClusterComputeResource 继承了 ComputeResource 的所有属性，所以你可以通过 ClusterComputeResource.resourcePool 属性来访问根目录。

13.3.1 资源池层次结构

资源池层次结构可以精细的控制虚拟机实用多少资源。

例如，假设一个单独的拥有几个虚拟机的主机。市场部门使用三个虚拟机，QA 部门使用两个虚拟机。因为 QA 部门需要大量的 CPU 和内存，管理员为每一个组创建了一个资源池。管理员为 QA 部门资源池设置 CPU 共享级别为 High 而为市场部门设置为 Normal，因此 QA 部门的人员可以进行自动化测试。第二个资源池具有少量的 CPU 和内存资源就足以承载市场部门的轻负责任务。当 QA 部门没有全部使用完它的资源时，市场部门可以使用这些可用的资源。



13.3.2 资源池管理指导原则

下列规则控制资源池创建：

- 一个 Root 资源池所拥有的资源至少必须和它当前所有 children 的资源相等。
- 不要过量分配资源池资源。所有 child 资源池的和应该少于(不是等于或大于)它的父资源池。例如，假如四个子资源池预留总数 40G 并且父资源池有一个 60G 的预留值，你有一些空间来创建其他的资源池。然而，假如四个子资源池预留总数为 60G，你就没有空间了。对于虚拟机，一些资源的过量分配是支持的。
- 在创建新的子资源池前，请检查父资源池中可用的资源。ResourcePool.runtimeInfo 属性是一个 ResourcePoolRuntimInfo 数据对象。ResourcePoolRuntimeInfo.cpu 和 ResourcePoolRuntimeInfo.memory 属性是具有资源使用信息的 ResourcePoolResourceUsage 对象，包括一个 unreservedForPool 属性。假如父资源池没有足够可用的资源，在添加新资源池前请从新设置子资源池的预留值。
- 首先配置子资源池，确定每一个子资源池的预留属性吞并了所有父资源池的所有资源。

13.3.3 群集概括

vSphere 支持把 ESX/ESXi 主机编到由同一个 vCenter 服务器管理的群集。群集可以具有更高级的特性如 VMware DRS 和 VMware HA。

- VMware HA(VMware High Availability 高可用性)从一个群集中的一个主机上迁移虚拟机到例外一台主机上，在这个主机失效的时候。

- VMware DRS(VMware Distributed Resource Scheduler 分布式资源管理)，提供动态重新分布资源。DRS 也支持 Distributed Power Management(DTM)，它提出建议或决定关闭主机和需要时开启主机，来节省电量。

  你可以设置 VMware DRS 为自动迁移虚拟机，或显示建议在数据中心中资源利用率不高的时候。

13.4 创建和配置资源池

根资源池关联每一个 ComputeResource，ClusterComputeResource。

通过调用 ResourcePool.CreateResourcePool 方法来创建资源池层，传入一个 ResourceConfig 方法作为参数，ResourceConfig.cpuAllocation 和 ResourceConfig.memoryAllocation 属性指向一个 ResourceAllocationInfo 对象，它允许你来明确信息。

- reservation –保证资源池可用的 CPU 或 memory 的总数。预留的资源在它们不被使用的时候也不会被浪费。假如使用量少于预留量，资源可以被其他资源池或正在运行的虚拟机使用。
- expandableReservation – 在一个具有扩展预留值得资源池里，预留量可以扩展超过前面指定的值。假如父资源池没有设置预留资源。一个 non-expandable 预留被称为固定预留。这个属性被虚拟机忽略。
- limit – 分配给这个资源池的 CPU 和内存资源的上限。虚拟机或资源池不能超过这个限制，即使资源富余。这个属性用来确保一致的性能。设置这个属性为-1 表示资源使用没有限制。
- shares – 在多个资源池中分配内存或处理能力的关联单位。SharesInfo 数据对象有两个属性，level 和 shares，你可以明确资源分配。
  - level – 选择 high，low 或 normal 来映射到预先定义的一系列共享数字值。设置这个属性为 custom 则需要用明确的数字来取代。
  - shares – 允许你明确你希望分配给这个资源池的共享数值。这个分配在具有同一个级别的资源池间平分。

调用 ResourcePool.UpdateConfig 或 ResourcePool.UpdateChildResourceConfiguration 方法来改变配置。

13.4.1 了解扩展预留

举例来说明扩展预留。

13.4.1.1 扩展预留举例 1

假设一个管理员管理资源池 P，定义了两个子资源池,S1 和 S2，对两个不同的用户(或组)。

管理员知道用户想要上电的具有预留值的虚拟机，但是不知道每个用户预留了多少。为 S1 和 S2 设置预留值扩展允许管理员更灵活的分配和继承资源池 P 的公共预留量。

没有扩展的预留，管理员必须明确的分配 S1 和 S2 一个确定的量。明确的分配带来不灵活，尤其在深层次的资源池层次中并且可能使在资源池层中设置预留更为复杂化。

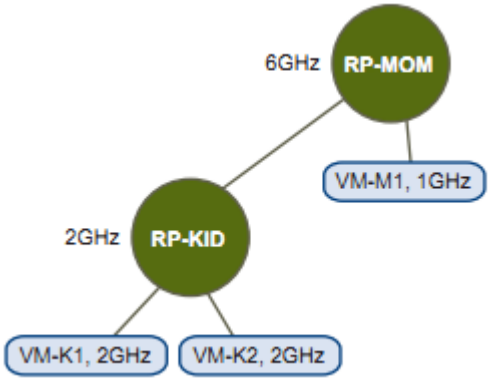
扩展预留引起一个明确隔离的损失，当 S1 可以使用全部的 P 的预留资源时，没有直接可以给 S2 使用的内存或 CPU。

14.4.1.2 扩展预留举例 2

假设下列方案(在图 13-2 中显示)

- 父资源池 RP-MOM 由一个 6GHz 的预留和一个运行的虚拟机 VM-M1 具有 1GHz 的预留。
- 你创建一个子资源池 RP-KID 具有一个 2GHz 的预留和选择了 **Expandable Reservation** 选项。
- 你添加两个虚拟机，VM-K1 和 VM-K2，对这个子资源池拥来说它们都具有 2GHz 预留并且试着为他们上电。
- VM-K1 可以直接从 RP-KID(具有 2GHz)中预定资源。
- 对于 VM-K2 没有本地资源可用，因此它从父资源池 RP-MOM 中借资源，RP-MOM 有 6GHz 和 1GHz(被虚拟机预定的)减 2GHz(被 RP-KID 预留的)，还有 3GHz 没有预定。3GHz 可用，你可以启动 2GHz 的虚拟机。

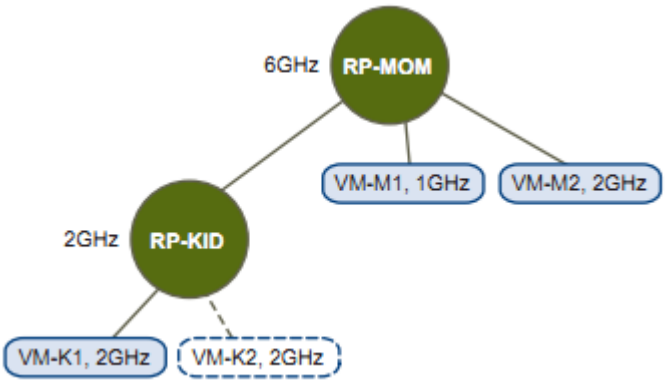
图 13-2. Admission Control with Expandable Resource Pools，Scenario 1



现在，考虑另外一个具有 VM-M1 和 VM-M2 的方案，在图 13-3 中显示。

- 启动两个在具有总预留为 3GHz 的 RP-MOM 中的虚拟机。
- 你仍然可以启动在 RP-KID 中的 VM-K1 因为本地有 2GHz 可用。
- 当你尝试启动 VM-K2 时，RP-KID 没有可预定得 CPU 所以它检测它的上级，RP-MOM 仅有 1GHz 可用(RP-MOM 的 5GHz-本地虚拟机预定的 3GHz-RP-KID 预定的 2GHz)。因此，你不能启动 VM-K2，它需要 2GHz 的预留。

图 13-3. Admission Control with Expandable Resource Pools，Scenario 2



13.4.2 删除子资源池

ResourcePool.DestroyChildren 方法递归删除一个资源池的所有子资源池。这个操作需要一个单独的参数，一个到父 ResourcePool managed 对象的引用。任何关联子资源池的虚拟机被重新分配到父资源池。

13.4.3 移动资源池或虚拟机到一个资源池中

你可以在一个资源池层中移动一个资源池和它的子资源池。

ResourcePool.MoveIntoResourcePool 方法让你移动虚拟机，虚拟应用，或资源池层次到

一个新的资源池。你调用这个方法需要具有你想要移动的一个 ResourcePool 队列或 VirtualMachine managed 对象引用。这个资源池层，包括子资源池和虚拟机，将被移动当你移动一个资源池时。

当前子资源池的最小可用的资源必须少于或等于当前父资源池的资源。根资源池不能被移动。

13.5 VMware DRS 和 VMware HA 群集介绍

群集主要用于 VMware DRS 和 VMware HA.

13.5.1 VMware DRS

一个 VMware DRS 群集是一个 ESX/ESXi 主机的集合，关联的共享资源的虚拟机和一个共享的管理接口。在你获得群集级别的资源管理好处前你必须创建一个 DRS 群集。

当你添加一个主机到一个 DRS 群集，这个主机的资源成为群集资源的一部分。除了聚合资源，一个 DRS 群集支持 cluster-wide 资源池和实施群集级别的资源分配策略。下列群集级被资源管理能力是可用的：

- **负载均衡.** vCenter 系统监控这个群集中所有主机和虚拟机的分布和 CPU 和内存资源的使用。DRS 拿这些数字和这个群集资源池和虚拟机，当前需求，和不平衡的目标属性提供的理想的资源使用量做比较。随后 DRS 执行(或建议)虚拟机迁移。当你首次在这个群集中启动一个虚拟机，DRS 试图保持正确的负载均衡或者放置虚拟机到一个正确的主机上或提出一个推荐。
- **电源管理.** 当 VMware DTM(Distributed Power Management 分布电源管理)功能可用时，DRS 在群集和主机级别能力和群集的虚拟机需求间做比较，包括最近的历史的需求。DTM 放置(或建立放置)主机进入待机电源模式假如足够超量的性能被发现。DTM 启动(或建议启动)主机假如性能需要。依赖于主机电源状态推荐结果，虚拟机可能需要从这个主机迁移进入或移出。
- **虚拟机安置.** 你可以控制一个群集中虚拟机布防在那个主机上，通过分配 DRS 亲和力和抗亲和力规则。

13.5.2 VMware HA

VMware HA 支持虚拟机高可用通过集中它们和主机，它们归属于一个群集。VMware HA 监控主机。在主机失败事件中，VMware HA 具有迁移虚拟机到主机能力。当你添加新虚拟机到一个 VMware HA 群集，VMware HA 检测是否有足够的性能来在不同的主机上启动它，以防万一主机失败不可用的情况。

13.6 创建和配置群集

vSphere Web Services SDK 包括所有群集管理任务的对象和方法。

13.6.1 创建一个群集

假如你的环境包括一个 vCenter 服务系统和多个 ESX/ESXi 主机，你可以通过调用 Folder.CreateCluster 方法来创建一个群集。传入新群集的名称和一个 ClusterConfigSpec 数据对象。在这个数据对象里，你确定下列属性：

- VMware HA
  - dasConfig – ClusterDasConfigInfo 数据对象确定群集上的 HA 服务。在这个对象上的属性决定是否严格的准入控制可用，在群集中的默认虚拟机设置是，是否 VMware HA 在主机失效后从新启动虚拟机。
  - dasVMConfigSpec – ClusterDasVMCnofigSpec 对象。ClusterDasVMConfigSpec.info 是一个 ClusterDasVmConfigInfo 数据对象，它明确了一个单个虚拟机的 HA 配置。你可以对不同的虚拟机应用不同的设置，或使用 dasConfig 属性中的默认设置。
- VMware DRS
  - drsConfig – ClusterDrsConfigInfo 数据对象包含 VMware DRS 服务的配置信息。对象中的属性明确了 cluster-wide(默认)行为对于虚拟机和产生群集门槛的建议。你可以启用和禁用 VMware DRS 通过 ClusterDrsConfigInfo.enabled 属性。
  - drsVmConfigSpec – ClusterDrsVMConfigSpec 数据对象指向一个为单个虚拟机确定 DRS 配置的 ClusterDrsVmConfigInfo 数据对象。ClusterDrsVmConfigInfo 覆盖单独的虚拟机的默认 DRS 配置并且允许你明确 DRS 行为和是否 DRS 可

以执行迁移操作或建议初始放置虚拟机的位置。

当你更新一个DRS配置时，你调用ComputeResource.ReconfigurComputeResource_Task并且传入一个ClusterConfigSpecEx对象。在这个ClusterConfigSpecEx.drsVmConfigSpec属性中，你可以确定一个ClusterDrsVMConfigSpec对象来为单独的虚拟机定义配置。

- rulesSpec – ClusterDrsRuleSpec数据对象指向一个明确了密切的和非密切的规则的ClusterRuleInfo数据对象。

13.6.2 添加一个主机到一个群集中

在不同的环境下有不同的方法来添加一个主机到一个群集中。每一个方法返回一个指向一个任务的managed对象引用。

- ClusterComputeResource.AddHost_Task – 添加一个主机到一个群集中。主机名称必须是一个IP地址，例如192.168.0.1，或一个可解析的DNS名称。假如群集支持嵌套的资源池，你必须指定可选的resourcePool参数，主机的根资源池变为指定的资源池，资源池和它所关联的层被添加到群集。

  假如一个群集不支持嵌入的资源池，你添加一个主机到这个群集时，单独的主机资源池层被舍弃并且所有在这个主机上的虚拟机被添加到群集根资源池中。
- ClusterComputeResource.moveHostInto_Task 在同一个数据中心中把一个主机当作群集来移动到一个群集中。假如主机已经是一个不同群集的一部分时，主机必须进入维护模式。
- ClusterComputeResource.moveInto_Task像moveHostInto_Task那样工作，但是支持一个主机队列。当时使用这个方法时，你不能保持这些主机的原始的资源池层次原有结构。

13.6.3 重新配置一个群集

你可以从新配置一个群集通过调用ClusterComputeResource.ReconfigureCluster_Task方法。这个方法允许你启用或禁用VMware DRS或VMware HA和设置属性。

为了从一个群集中移出主机，你可以使用下列方法：

- ClusterComputeResource.MoveHostInto_Task 或 MoveInto_Task – 从一个群集中移出一个主机并且移动这个主机到另一个群集中。
- Folder.MoveIntoFolder_Task – 从一个群集中移出一个主机并且使其成为独立的主机。
- Host.Destroy_Task – 从清单列表中移出一个主机。

13.7 管理 DRS 群集

vSphere 客户端 UI 允许你研究 DRS 群集行为，当 DRS 在运行，它产生建议和关联信息对于群集有个好的平衡性能很重要。

- 初始虚拟机放置位置
- 负载均衡的虚拟机迁移。每一个迁移建议由一个评估你可以在 ClusterRecommendation.rating 属性中找到。客户端应用可以选择仅仅考虑高优先权的迁移或具有多优先权级别的迁移。
- 检测是否 DRS 群集有效 – 充足的可用资源来启动额外的虚拟机 – 或不可用。

DRS 建议存储在 ClusterComputeResource.recommendation 属性中，它是一个 ClusterRecommendation 数据对象队列。每一个 ClusterRecommendation 包括执行动作的信息和显示给最终用户的信息或日志信息。

- 客户端应用可以调用 ClusterComputeResource.ApplyRecommendation 来应用一个或多个建议。
- 为了更细致的控制，客户端应用可以仅仅执行独立的动作。ClusterRecommendation.action 属性是一个 ClusterAction 对象队列。每一个 ClusterAction 包括一个该动作的目标和类型，类型是 ActionType 枚举类型(HostPowerV1，MigrationV1，VmPowerV1)中的一个字符串值。客户端应用可以使用 ActionType 信息来按照 DRS 建议来启动主机，迁移虚拟机，或启动虚拟机通过调用 Datacenter.PowerOnMultiVM_Task。

13.8 管理 HA 群集

你可以通过调用移动主机到一个群集的方法来添加一个主机到一个 HA 群集。你不得不调用 HostSystem.ReconfigureHostForDAS_Task 来为 HA 重新配置主机当自动 HA 配置失败

的时候。

13.8.1 首要和次要主机

你可以调用 ClusterComputeResource.AddHost_Task 方法来添加一个主机到一个群集中，需要你确定主机名称，端口，和主机密码作为一个 HostConnectSpec。

当你添加一个一个主机到一个 VMware HA 群集时，一个代理被上传盗主机并且配置成可以和在群集中的其他代理通讯。前五个被添加到群集的主机被设计成为主要的主机，后面的所有主机被设计成为次要主机。主要主机保持和复制全部群集状态并且用来启动故障转移操作。假如一个主要主机被从群集中移出，VMware HA 推选其他的主机成为主状态。

任何假如群集的主机必须和已经存在的主要主机通讯来完成自己的配置(除了当你添加第一台主机到群集时)。为了 VMware HA 操作正确至少由一个主要主机具有功效。假如全部主要主机都失效(无相应)，没有主机可以成功完成 VMware HA 的配置。

其中一个主要主机被设计成活动主要主机并且它的职责包括：

- 决定在那里重新启动虚拟机。
- 保持失败重新启动尝试追踪。
- 决定什么时候合适尝试重新启动一个虚拟机。

13.8.2 故障检测和主机网络隔离

不同主机上的代理彼此相互联系和监控通过一个交换心跳，默认是一秒钟。假如在 15 秒内没有从一个主机接收到心跳，并且这个主机被能 ping 通，这个主机被声明为失败。运行在这个失败的主机上的虚拟机将在具有最多可用的未被预定的能力(CPU 和内存)的候补主机上重新加载。

发生主机网络隔离时主机仍在运行时，但是它不再和其他在群集中的主机通讯。按照默认设置，假如一个主机从所有在群集中的其他主机停止接收心跳超过 12 秒，它尝试 ping 它的隔离地址，假如还是失败，主机声明自己从网络中隔离。

当隔离的主机网络连接在 15 秒或更久的时间内没有恢复，在群集中的其他主机把这个主机当作失败来处理并且尝试故障转移它的虚拟机。然而，当一个隔离的主机仍然可以访问共享存储，它也保留着虚拟机文件的磁盘锁。为了避免减灾的数据损毁，VMFS 磁盘锁定禁止同时对虚拟机磁盘文件上的写操作并且尝试故障转移隔离的主机的虚拟机失败。默认，隔

离的主及保持它的虚拟机开机，但是你可以更改主机隔离响应。

13.8.3 同时使用 VMware HA 和 DRS

使用具有 VMware DRS 的 VMware HA 结合了自动故障转移和负载均衡。这个组合可以使在 VMware HA 移动虚拟机到不同的主机后快速的达到虚拟机的重新均衡。

当 WMware HA 执行故障转移和重新在不同的主机上启动虚拟机时，它首要优先权是所有虚拟机立即有效。在虚拟机已经重新启动后，这些主机可能高负荷运行，而其他的主机相对的出于低负荷运行。VMware HA 使用 CPU 和内存预留值来决定故障迁移，而真正的使用值可能更高。

在一个群集中打开了 DRS 和 VMware HA 准入开关，虚拟机可能不会从主机进入维护模式时撤离，因为要为维持故障转移级别预留资源。你必须手动使用 VMotion 来迁移虚拟机。

当 VMware HA 准入不可用时，故障转移资源限制不会作用到 DRS 和 VMware Distributed Power Management(DMP)上。这个限制不是强迫的。

- DRS 从主机撤离虚拟机并且使主机进入维护模式或待机模式不管可能影响到故障迁移需求。
- VMware DPM 关闭主机(使他们进入待机模式)即使违反了故障迁移需求。

14、任务和计划任务

VMware vSphere 使用一个异步的客户-服务器通讯模式。以_Task 结尾的方法是非阻塞的，返回一个任务 managed 对象的引用。你可以使用 Task 和 ViewManager managed 对象来监控任务，取消当前任务，和创建自定义任务。

假如你正在使用 vCenter 服务系统，ScheduledTaskManager 允许你规划你自己的任务在某一个时间点重复运行。

14.1 创建任务

每次 vSphere 服务器运行一个方法，它创建一个 Task 和一个对应的 TaskInfo 数据对象。一些方法同步执行和当任务完成返回数据。但是以_Task 结尾的方法是异步执行的，返回一

个指向任务的引用，这个任务将被创建和完成当一个处理器可用的时候。他们被创建来以非阻塞方式执行这个功能。因此，你必须使用这个任务的引用来监控状态和任务的结果。vSphere 操作包括他们名字带有_Task 后缀是异步的和返回 Task 引用。

Task 对象通过它的 TaskInfo 数据对象提供关于调用操作的状态信息。在运行时一个 TaskInfo 的常用的入口是 Task managed 对象的 info 属性。通过监控 TaskInfo 对象的属性，客户端应用可以在 Task 任务完成时执行合适的动作，或在任务没有成功执行完成时处理错误。

14.1.1 Session 驻留

一个 Task 和它相关的对象是特定的会话，因此他们在会话关闭后不能永久的存在。当你的客户端打开一个会话时，你仅能获得关于你的客户端被授权察看的 Task 对象信息。

14.1.2 取消任务

为了关闭一个正在运行的任务，调用 Task.CancelTask 方法，传入你想取消任务的 managed 对象引用，如下所示：

```
my_conn.cancelTask(taskMoRef);
```

你仅可以取消它的 cancelable 属性为 true 并且它的 state 属性为 running 的任务。发起任务的操作在它创建这个任务时设置 cancelable 的值。例如，CreateVM_Task 不能被取消。在尝试取消一个正在运行的任务时，你可以检查 Task 关联的 TaskInfo 数据对象的 cancelabel 属性和 state 属性的值。

14.1.3 使用 TaskInfo 决定任务状态

Task 对象通过它的 TaskInfo 数据对象提供关于调用操作的状态信息。在运行时一个 TaskInfo 的常用的入口是 Task managed 对象的 info 属性。通过监控 TaskInfo 对象的属性，客户端应用可以在 Task 任务完成时执行合适的动作，或在任务没有成功执行完成时处理错误。

Task.info 属性包含一个 TaskInfo 对象，TaskInfo 对象包含服务器返回给你的客户端关于这个任务的信息。

当一个任务被服务器实例化，TaskInfo.result 属性被初始化为 UnSet。当一个操作成功完

成时，result 属性被设置为明确的类型到这个操作。result 可以是一个数据对象，一个到 managed 对象的引用，或任何其他被操作定义的数据结构。

例如：

1. ClusterComputeResource.AddHost_Task 方法返回一个任务对象，这个任务对象的 info 属性包含一个 TaskInfo 数据对象。
2. 在任务开始时，result 属性是 Unset。
3. 在操作成功完成时，TaskInfo 的 result 属性包含新添加的 HostSystem 的 managed 对象引用。

表 14-1 列出了一些 TaskInfo 数据对象在 CreateVM_Task 方法实例化的任务从开始到任务结束能取得值。

Table 14-1. Sample TaskInfo Values

Property	Datatype	Start of Task Sample Values	End of Task Sample Values
cancelable	boolean	false	false
...			
completeTime	dateTime	Unset	"2009-02-19T22:53:35.015338Z"
progress	int	36	100
queueTime	dateTime	"2009-02-19T22:50:39.111604Z"	"2009-02-19T22:50:39.111604Z"
reason	TaskReason	reason	reason
result	anyType	Unset	64
....			
state	TaskInfoState	"running"	"success"

14.1.4 监视 TaskInfo 属性

为了监控任务的状态，使用 PropertyCollector.WaitForUpdatesEx 方法。你可以监控 TaskInfo 属性的值，当任务运行完成时它会改变。例如，你可以在操作过程中检测 startTime，queueTime，completeTime，progress，result，和 states 的值。当主代码在执行其他活动时，请在一个单独的线程中监控这些属性直到任务完成。

你的代码必须管理任务完成时放回的数据类型(managed 对象引用，数据对象等等)。除了 success，queued 和 running，一个操作也可以进入 error 状态，你的代码必须要处理的。

任务对象的生命周期不依赖于创建它的 TaskManager 也不依赖于和它关联的实体。为了传递操作的状态而存在。当你的应用不在需要这个信息的时候你可以丢弃到它的引用。

例子 14-1 下列代码展示了从每个在队列中的任务对象上取得 info 属性值。

Example 14-1. Displaying TaskInfoState Values for Tasks in recentTask Array

```
private void displayTasks(ObjectContent[] oContents) {
    for(int oci = 0; oci < oContents.length; ++oci) {
        System.out.println("Task");
        DynamicProperty[] dps = oContens.getPropSet();
        if(dps != null) {
            String op = ""; type = ""; state = ""; error = "";
            for(int dpi =0; dpi < dps.length; ++dpi) {
                DynamicProperty dp = dps[dpi];
                if("info.entity".equals(dp.getName()) {
                    type = ((ManagedObjectReference)dp.getVal()).getType();
                } else if("info.entityName".equals(dp.getName()) {
                    name = (String)dp.getVal();
                } else if("info.state".equals(dp.getName())) {
                    TaskInfoState tis = (TaskInfoState)dp.getVal();
                    if(TaskInfoState.error.equals(tis)) {
                        state = "-Error";
                    } else if(TaskInfoState.queued.equals(tis)) {
                        state = "-Queued";
                    } else if(TaskInfoState.running.equals(tis)) {
                        state = "-Running";
                    } else if(TaskInfoState.success.equals(tis)) {
                        state = "-Success";
                    }
                } else if("info.cancelled".equals(dp.getName())) {
                    Boolean b = (Boolean)dp.getVal();
                    if(b != null && b.booleanValue()) {
                        state = "-Cancelled";
                    }
                }
            }
        }
```

```
    }
}
```

例子 14-2 展示了程序的输出

Example 14-2. Sample Run of the TaskList Java Application

```
java com.vmware.samples.general.TaskList --url https://srv/sdk --username root --password *******

Started
Task
Operation AcquireCimServicesTicket
Name srv
Type HostSystem
State -Success
Error
======================
Ended TaskList
```

14.2 访问和处理多任务

使用 ViewManager 的 ListView 方法来确定你要监控的一系列任务。

你可以使用 ViewManager 中具有 Property Collector 的 views 集合中的一个 view 来的指定一个小的和更有效率的数据集。每一个 view 代表你在服务器上选择的对象。Views 更有效率因为你仅仅需要一个单独的 PropertyCollector 对象的实例，来代替具有多个制定的过滤器的实例。

14.2.1 使用 ViewManager 对象收集数据

使用 ViewManager 方法之一来获得关于任务对象和引用的信息在 session 运行的时候。ViewManager 的 ListView 方法允许你自定义你的具有一个输入对象列表的 view，ContainerView 方法让你浏览所有在一个文件夹，数据中心，资源池，或其他数据容器中的对象，InventoryView 方法让你监控整个清单。你能创建的最小的 view 是最有效率来检索任务数据的方式。

ViewManager 有下列属性：

viewList – 一个 view 引用队列。每一个队列对象是一个指向一个右 View Manager 创建的 view 的 managed 对象引用。

14.2.1.1 使用 ListView 对象监控任务的例子

使用 ViewManager 的 ListView 方法来指定你想要监控的一系列任务。

下列例子使用 ViewManager 具有一个 ListView 方法的服务接口来访问和管理任务对象。这个例子使用属性收集器来监控在虚拟机克隆过程中创建的任务。这个程序创建两个指定虚拟机的克隆然后监控这个任务并打印出状态和引用值在任务完成时。

下列步骤描述了使用 ListView managed 对象创建一个程序的过程。

To Create a program that uses ListView managed object

1. 导入 vSphere Web Services API 库：

   import com.vmware.vim25*;

2. 导入必须的 Java(JAX-WS connection,bindings,SOAP)库：

   import java.util.*;

   import javax.net.ssl.HostnameVerifier;

   import javax.net.ssl.HttpsURLConnection;

   import javax.net.ssl.SSLSession;

   import javax.xml.ws.BindingProvider;

   import javax.xml.ws.soap.SOAPFaultException;

3. 创建 cloneVMTask 类来在一个主机上创建克隆虚拟机任务，因此我们可以演示如何监控这些任务。

   public class cloneVMTask {

4. 为服务实例和方法声明变量：

   // Sevices and methods

   static ManagedObjectReference pCollector;

   static ManagedObjectReference viewMgr;

   static ManagedObjectReference serviceContent;

   static VimPortType methods;

5. 创建一个函数检索数据中心中所有虚拟机的引用查找指定的名称的虚拟机。例子中的函数使用了 getVMRef(String vmName)。

   private static ManagedObjectReference getVMRef(String vmName) throw Exception

{

ManagedObjectReference vmRef = null;

6. 使用一个 container view 来收集数据中心中所有虚拟机的引用。

List<String> vmList = new ArrayList<String>;

vmList.add(“VirtualMachine”);

ManagedObjectReference cViewRef = method.createContainerView(viewMgr,

serviceContent.getRootFolder(),vmList,true);

7. 创建一个 ObjectSpec 来定义遍历的开始位置。使用 setObj 方法来指定 container view 是一个为这次遍历的根对象。设置 setSkip 方法为 true 来明确你不想在结果中包含这个 container。

ObjectSpec oSpec = new ObjectSpec();

oSpec.setObj(“traverseEntities”);

oSpec.setSkip(true);

8. 创建一个 traversal spec 来选择所有在 view 中的对象。

TraversalSpec tSpec = new TraversalSpec();

tSpec.setName(“traversalEntities”);

tSpec.setPath(“view”);

tSpec.setSkip(false);

tSpec.setType(“ContainerView”);

9. 添加这个 traversal spec 到 object spec。

oSpec.getSelectSet().add(tSpec);

10. 为检索(虚拟机名称)指明属性。

PropertySpec pSpec = new ProperSpec();

pSpec.setType(“VirtualMachine”);

pSepc.getPathSet().add(“name”);

11. 创建一个 PropertyFilterSpec 并且添加对象和属性 spec 到其中。

PropertyFilterSpec fSpec = new PropertyFilterSpec();

fSpec.getObjectSet().add(oSpec);

fSpec.getPropSet().add(pSpec);

12. 创建一个过滤器队列并添加这个 spec 到其中。

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

13. 从服务器取得数据。

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector,fSpecList,ro);
```

14. 通过返回的队列查找匹配的指定的 vmName。

```
if(props != null) {
    for(ObjectContent oc : props.getObjects()) {
        String vmname = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if(dps != null) {
            for(DynamicProperty dp : dps) {
                vmname = (String) dp.getVal();
                if(vmname.equals(vmName)) {
                    vmRef = oc.getObj();
                    break;
                }
            }
            If(vmRef != null) {break;}
        }
    }
}
if(vmRef == null) {
    System.out.println("Specified Virtual Machine not Found");
    throw new Exception();
}
return vmRef;
```

15. 取得包含指定虚拟机的文件夹(VirtualMachine.parent)。

```
private static ManagedObjectReference getVMParent(ManagedObjectReference vmRef)
throw Exception {
```

16. 创建一个 Object Spec 来定义属性集合。使用 setObj 方法来指定 vmRef 为这次遍历的根对象。设置 setSkip 方法为 true 来明确你不想在结果中包含这个虚拟机。

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(true);
```

17. 指定要检索的属性(VirtualMachine.parent)。

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType(“VirtualMachine”);
pSpec.getPathSet().add(“parent”);
```

18. 创建一个 PropertyFilterSpec 并且添加对象和属性 spec 到其中。

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

19. 创建一个过滤器队列并添加这个 spec 到其中。

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

20. 从服务器取得数据。

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector,fSpecList,ro);
```

21. 取得父文件夹的引用。

```
ManagedObjectReference folderRef = null;
if(props != null) {
    for(ObjectContent oc : props.getObjects()) {
        List<DynamicProperty> dps = oc.getPropSet();
        if(dps != null) {
            for(DynamicProperty dp : dps) {
                folderRef = (ManagedObjectReference) dp.getVal();
            }
        }
    }
```

```
}
if(folderRef == null) {
    System.out.println("Folder not found");
    throw new Exception();
}
return folderRef;
```

现在我们已经有了你在命令行指定的虚拟机(vmRef)的引用信息和虚拟机父目录的引用(folderRef)，我们准备来创建克隆虚拟机。

22. 创建克隆，使用 cloneVM 方法并传入我们在前面检索到的 vmRef。

```
Private static void cloneVM(ManagedObjectReference vmRef) throw Exception {
```

23. 创建了克隆 managed 对象后，创建一个克隆声明。如果可能使用默认值。

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineRelocateSpec vmrs = new VirtualMachineRelocateSpec();
cloneSpec.setLocation(vmrs);
cloneSpec.setPowerOn(true);
cloneSpec.setTemplate(false);
```

24. 得到克隆虚拟机(VirtualMachine.parent)目标文件夹。克隆将被创建在包含指定的虚拟机(vmName)的相同的文件夹中。

```
ManagedObjectReference folder = getVMParent(vmRef);
```

25. 创建两个克隆虚拟机。

```
ManagedObjectReference cloneTask = methods.cloneVMTask(vmref,folder,"clone_1",
        cloneSpec);
ManagedObjectReference cloneTask2 = methods.cloneVMTask(vmref,folder,"clone_2",
        cloneSpec);
```

26. 为克隆任务创建一个列表 view。

```
List<ManagedObjectReference> taskList = new
ArrayList<ManagedObjectReference>();
taskList.add(cloneTask);
taskList.add(cloneTask2);
ManagedObjectReference cloneTaskList = methods.createListView(viewMgr,taskList);
```

接下来我们为 WaitForUpdateEx 将建立一个属性过滤器。这包括创建一个对象 spec，一个 遍历 spec，一个属性 spec，和最终的属性过滤器。下面 6 步讲描述这些过程。

27. 创建一个对象 spec 来开始遍历。

    ObjectSpec oSpec = new ObjectSpec();

    oSpec.setObj(cloneTaskList);

    oSpec.setSkip(true);

28. 创建一个遍历 spec 来选择 view 中的任务列表。

    TraversalSpec tSpec = new TraversalSpec();

    tSpec.setName(“traversalTasks”);

    tSpec.setPath(“view”);

    tSpec.setSkip(false);

    tSpec.setType(“ListView”);

29. 添加遍历 spec 到对象 spec 中。

    oSpec.getSelectSet().add(tSpec);

30. 创建属性 spec 来得到 Task.state 和 Task.info.result。

    PropertySpec pSpec = new ProperySpec();

    pSpec.setType(“Task”);

    pSpec.setAll(false);

    pSpec.getPathSet().add(“info.state”);

    pSpec.getPathSet().add(“info.result”);

31. 创建一个过滤器 spec。

    PropertyFilterSpec fSpec = new PropertyFilterSpec();

    fSpec.getObjectSet().add(oSpec);

    fSpec.getPropSet().add(pSpec);

32. 创建过滤器。

    ManagedObjectReference pFilter = methods.createFilter(pCollector,fSpec,true);

    在接下来的部分，我们使用 waitForUpdatesEx 方法来寻找在 cloneTask.info.state 和 cloneTask.info.result 的改变。当状态为”success”，cloneTask.info.result 是克隆 managed 对象的引用。注意检索的属性顺序是不固定的，它可以调用多次 waitForUpdatesEx 来检索一个人的得两个属性。

这个代码没有设置超时时间(WaitOptions.maxWaitSecondes 没有设置)，因此在它检索完所有属性值后，waitForUpdatesEx 将阻塞线程，直到和 vSphere 服务器的 TCP 连接超时。

一个客户端如何处理 session 依赖于特定的上下文。(客户端可以在自己的线程中调用 WaitForUpdatesEx，查找指定的更新然后停止调用方法。)

33. 初始化等待循环(?)

```
String version = “”;
Boolean wait = true;
waitOptions waitOptions = new waitOptions();
while(wait) {
```

34. 调用 WaitForUpdatesEx。

```
updateSet uSet = methondes.waitForUpdatesEx(pCollector,version,waitOptions);
if(uSet == null) {
    wait = false;
} else {
```

35. 为随后调用 WaitForUpdatesEx 取得版本。

```
version = uSet.getVersion();
```

36. 得到属性更新列表。

```
List<PropertyFilterUpdate> pfUpdates = uSet.getFilterSet();
for(PropertyFilterUpdate pfu : pfUpdates) {
```

37. 通过这个过滤器得到过程更新的对象的列表。

```
List<ObjectUpdate> oUpdates = pfu.getObjectSet();
for(ObjectUpdate ou : oUpdates) {
```

38. 查找 ObjectUpdate.kind=MODIFY(属性被编辑了)。

```
if(ou.getKind() == ObjectUpdateKind.MODIFY) {
    String name = “”;
    TaskInfoState state;
    ManagedObjectReference cloneRef = new ManageObjectReference();
}
```

39. 得到改变的数据。

List<PropertyChange> pChange = ou.getChangeSet();

40. 检索属性名称。

```
for(PropertyChange pc : pChanges) {
    name = pc.getName();
    //The task property names are info.state or info.result;
    //pc.val is an xsd:anyType:
    //-- for info.state, it is the state value
    //-- for info.result, it is the clone reference
    if (name.equals("info.state")) {
        state = (TaskInfoState)pc.getVal();
        System.out.println("State is "+state.value());
    }
    else if (name.equals("info.result")) {
        cloneRef = (ManagedObjectReference)pc.getVal();
        System.out.println("Clone reference is "+cloneRef.getValue());
        }
    }
}
```

使用一个 TrustManager 来处理认证并且提供一个主机名核实方法。(主机名核实声明在主函数中。)

为了达到例子的目标，TrustManager 将接受所有证书。这个仅仅出现在开发环境。产品代码中应该保证证书的支持。

```
private static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
javax.net.ssl.X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }
    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }
```

```
        public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }
        public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
            String authType) throws java.security.cert.CertificateException {
            return;
        }
        public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
            String authType) throws java.security.cert.CertificateException {
            return;
        }
    }
```

现在我们来设置检索任务信息，我们引用主方法。

```
// cloneVMTask( server, user, password, virtual-machine )
public static void main(String [] args) throws Exception {
```

41. 我们创建变量来保存从命令行传入的参数。

```
String serverName = args[0];
String userName = args[1];
String password = args[2];
String vmName = args[3];
String url = "https://"+serverName+"/sdk/vimService";
```

42. 添加访问 API 方法和服务的变量。

```
// -- ManagedObjectReference for the ServiceInstance on the Server
// -- VimService for access to the vSphere Web service
// -- VimPortType for access to methods
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
```

43. 声明一个主机名称核实保证自动连接。主机名核实在 SSL 握手期间被调用。

```
HostnameVerifier hv = new HostnameVerifier() {
```

```
        public boolean verify(String urlHostName, SSLSession session) {
            return true;
        }
    };
```

44. 创建信任管理。

```
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

45. 设置默认主机名验证来保证连接。

```
HttpsURLConnection.setDefaultHostnameVerifier(hv);
```

46. 设置 ServiceInstance 的 managed 对象引用。

```
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
```

47. 创建一个 VimService 对象来获取一个 VimPort binding provider。BindingProvider 提供访问 request/response 信息中的协议域。检索请求上下文，将被用来处理信息请求。

```
vimService = new VimService();
methods = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) methods).getRequestContext();
```

48. 保存请求上下文中的服务器 URL 并且明确 true 来保持客户端和服务器的连接。客户端 API 包括请求中的服务器的 HTTP cookie 来维持 session。假如你不设置为 true，

服务器将为每次请求开始一个新的 session。

ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);

ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

49. 检索 ServiceContent 对象并且登陆。

    serviceContent = methods.retrieveServiceContent(SVC_INST_REF);

    methods.login(serviceContent.getSessionManager(),userName,password,null);

50. 取得属性集合和 view 管理器的引用。

    pCollector = serviceContent.getPropertyCollector();

    viewMgr = serviceContent.getViewManager();

51. 取得指定虚拟机的引用。

    ManagedObjectReference vmRef = getVmRef(vmName);

52. 克隆虚拟机并等待结果。

    cloneVM(vmRef);

53. 关闭连接

    methods.logout(serviceContent.getSessionManager());

14.2.2 使用一个 TaskManager 接口来收集数据

TaskManager 是一个服务接口，你可以使用它来访问和处理任务对象。这个方法使用一个 PropertyCollector，它包含 TaskManager managed 对象的 recentTask 属性，其相当于 vSphere 客户端程序底部的最近任务面板。

你可以在你的客户端应用中使用下列 TaskManager 属性。

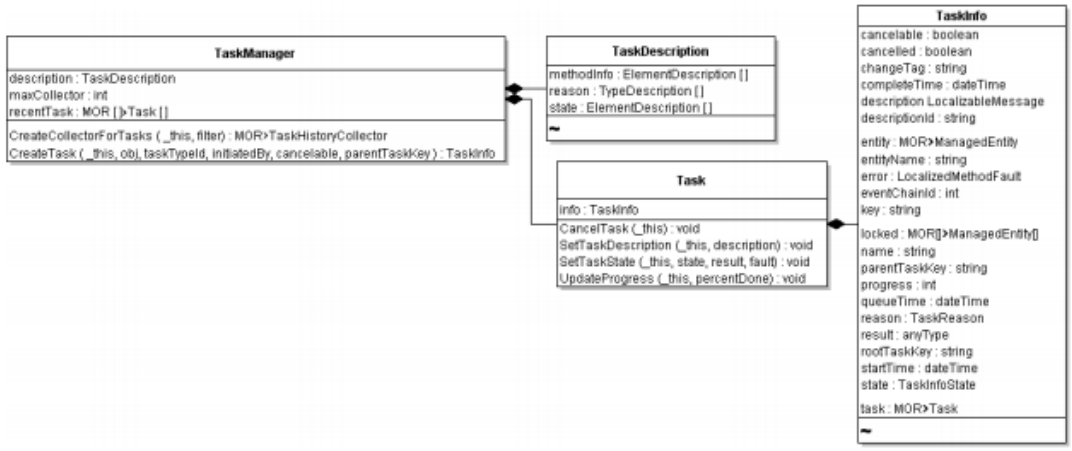
- description – TaskDescription 对象包括一个 methodInfo 属性。methodInfo 包含一个 key-based 队列，TaskManager 使用操作的名称来定位一个 TaskInfo 数据对象的 descriptionId 属性值。例如 key-based 队列的两个元素 methodInfo[“Folder.createVM”] 和 methodInfo[“Folder.createClusterEx”]。
- recentTask – Task managed 对象引用队列，他们按顺序准备运行，运行，或完成在过去的 10 分钟里。在有 vCenter 服务器管理的 ESX/ESXi 主机上，一个完成得任务任务必须包含在最近 200 个任务列表中。一个连接到 vSphere 服务器的 vSphere 客户端显示排队等待的，正在运行的，已经完成的任务在最近任务面板中。

除了这些属性之外，TaskManager 有下列方法：

- CreateTask – 由其他方法使用来创建一个自定义任务对象。开发者创建 extensions 可以使用这个方法来创建自定义任务对象。
- CreateCollectorForTasks – 创建一个对象包含所有满足特定标准的 vCenter 服务器数据库中的任务。你不能运行这个方法在 ESX/ESXi 系统上。

图 14-1 显示了一个 TaskManager 和关联对象的 UML 类图。

Figure 14-1. TaskManager and Task Managed Objects



14.2.2.1 适用 TaskManager 测试最近任务

为了得到最近任务列表，使用一个 PropertyCollector 来获得 TaskManager 和所有 TaskManager 的 recentTask 属性中的任务对象引用。例子摘录了部分 TaskList.java 例程，创建 ObjectSpec，PropertySpec，和一个 TraversalSpec 来获得服务器上 TaskList 所有任务对象的引用。

Example 14-3. PropertyFilterSpec Definition to Obtain recentTask Property Values

```
private PropertyFilterSpec[] createPFSForRecentTasks(ManagedObjectReference taskManagerRef) {
    PropertySpec pSpec = new PropertySpec();
        pSpec.setAll(Boolean.FALSE);
        pSpec.setType("Task");
        pSpec.setPathSet(new String[] {"info.entity", "info.entityName", "info.name",
            "info.state", "info.cancelled", "info.error"});
    ObjectSpec oSpec = new ObjectSpec();
        oSpec.setObj(taskManagerRef);
        oSpec.setSkip(Boolean.FALSE);
    TraversalSpec tSpec = new TraversalSpec();
        tSpec.setType("TaskManager");
        tSpec.setPath("recentTask");
        tSpec.setSkip(Boolean.FALSE);
    oSpec.setSelectSet(new SelectionSpec[]{tSpec});
    PropertyFilterSpec pfSpec = new PropertyFilterSpec();
        pfSpec.setPropSet(new PropertySpec[]{pSpec});
        pfSpec.setObjectSet(new ObjectSpec[]{oSpec});
    return new PropertyFilterSpec[]{pfSpec};
}
```

对于由一个 vCenter 服务器管理的 ESX/ESXi 系统，使用 TaskHistoryCollecto。

14.2.3 了解 ScheduledTaskManager 接口

你可以使用 ScheduledTaskManager 来安排任务。在 vSphere 客户端里，预定好的任务显示在 Task&Events 标签中。

你可以在 vCenter 服务器上不同的时间里定义要执行的活动。

- 当一个 vCenter 服务器开始操作时，例如在重新启动后。
- 在一个特定的时间点。
- 在每小时，每天，每周或每月的间隔。

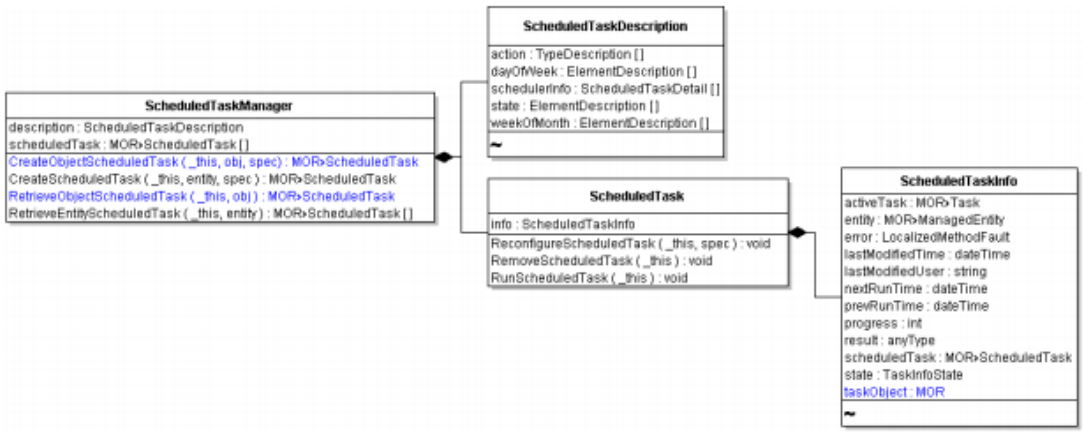
你可以规划脚本来运行或编写方法来由服务器调用。你应用活动在清单中的实体上，例如一个虚拟机或一个主机。

你可以通过 ScheduledTaskManager 执行下列活动。

- 调用 ScheduledTaskManager.RetrieveEntityScheduledTask 方法来检索一个特定 managed 实体的已安排的任务。
- 调用 ScheduledTaskManager.CreateScheduledTask 方法来创建一个规划任务。

图 14-2 显示了 ScheduledTaskManager 服务接口和相关联的数据对象。

Figure 14-2. ScheduledTaskManager and ScheduledTask Managed Objects



ScheduledTaskManager.scheduledTask 属性包含一个为服务器配置的 ScheduledTask 对象列表。假如你没有活动安排，这个属性是空的。对于在队列中的任何一个 ScheduledTask 对象，你可以使用 ShceduledTask 对象的 info 属性来获得关于计划好的活动的状态信息。信息内容包括任务进度，状态，前一个和下一个运行时间，其他细节包含在 ScheduledTaskInfo 数据对象里。

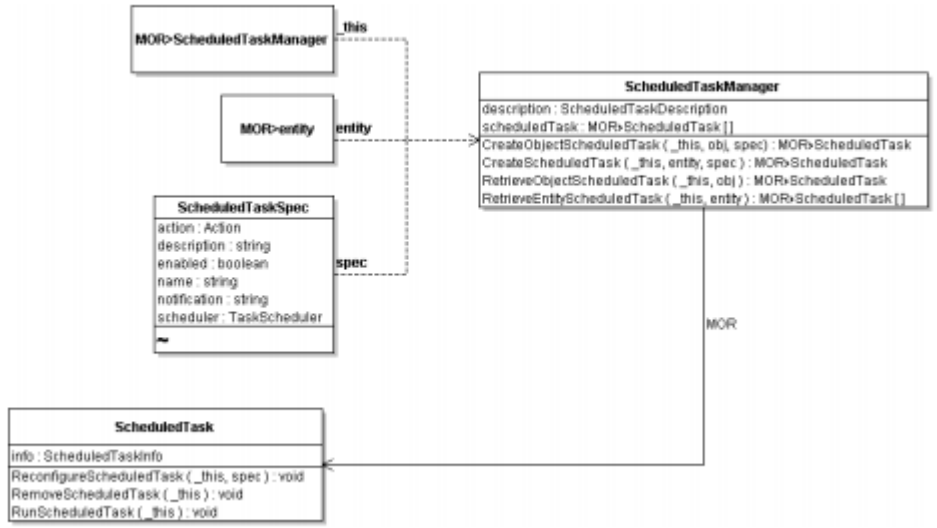
假如一个 ScheduledTask 指定的活动创建了属于它自己的任务(例如具有任何异步的操作)，Task 的 managed 对象引用位于 ShceduledTaskInfo 的 activeTask 属性里。

14.2.3.1 计划任务

你通过调用 ScheduledTaskManager.CreateScheduledTask 方法来创建一个 ScheduledTask。当你调用方法时，包括一个 ScheduledTaskSpec 对象，它定义了计划和活动运行的具体时间。一个计划的活动基于下列规则应用到一个对象：

- 假如你为计划活动指定一个容器对象作为实体，计划会应用到所有该容器直接派生的实体上。你可以在 Folder，Datacenter，或 VirtualApp 级别上设置一个 ScheduledTask 并且使计划活动应用到所有和 Folder，Datacenter，或 VirtualApp 关联的实体上。
- 假如你指定清单中一个节点对象，例如一个虚拟机，活动仅仅应用到虚拟机。

Figure 14-3. Using ScheduledTaskManager to Create a ScheduledTask



14.2.3.1.1 定义计划和活动

ScheduledTaskSpec 数据对象包含所有创建一个 ScheduledTask 的信息。

- action – 当 ScheduledTask 运行时活动被获得。指定一个 Action 数据对象，它是一个由几个指定的活动类型扩展出来的抽象类型。Action 数据对象也被用于 Alarm 架构。
- notification – 为发送关于 ScheduledTask 的通知信息指定 email 地址。为使用通知，vCenter 服务器必须有一个配置好的 SMTP email 网关，默认的，notification 被设置为空字符串。
- schedule – 指定时间，频率，和其他计划细节。TaskScheduler 数据对象是几个计划对象的基础类型。

14.2.3.1.2 计划循环操作

你可以创建合适的 TaskSchedulr 子类型实例来指定时间，日期，或频率和设置 ScheduledTaskSpec 的 scheduler 属性。

TaskScheduler 基础类型有两个属性：

- activeTime 是活动应该发生的时间。假如你没有设置这个属性，默认设置为提交计划任务到服务器的时间。
- expireTime 是在过了这个点计划活动将不再发生。默认，这个属性没有被设置，因

此计划任务不会失效。

表 14-2 提供了一些关于 TaskScheduler 子类型的使用信息。表中的例程是 java 代码片断。

Table 14-2. TaskScheduler Data Object Subtypes

TaskScheduler Subtype	Usage

`AfterStartupTaskScheduler`	Schedule a task to start as soon as the vCenter Server system is started, or at a defined time after startup. The value must be zero (task triggered at startup) or higher.
	Example: Schedule a task to run 10 minutes after vCenter Server startup.
	`AfterStartupTaskScheduler asts = new AfterStartupTaskScheduler();`
	`asts.setMinute(10);`

`OnceTaskScheduler`	Schedule an action to run once only at the specified date and time.
	Example: Schedule a task to run 30 minutes after the schedule is submitted to the server.
	`Calendar runTime = Calendar.getInstance();`
	`runtime.add(Calendar.MINUTE, 30);`
	`OnceTaskScheduler ots = new OnceTaskScheduler ();`
	`ots.setRunAt(runTime);`

`RecurrentTaskScheduler`	Base type for `HourlyTaskScheduler`, `DailyTaskScheduler`, `WeeklyTaskScheduler`, and `MonthlyTaskScheduler` objects. Set the interval property to define how frequently a task should run. For example, setting the interval property of an hourly task to 4 causes the task to run every 4 hours.

`HourlyTaskScheduler`	Schedule a task to run once every hour (or every specified number of hours) at a specified time. Set the interval property to run the task after a specified number of hours.
	Example: Schedule a task to run every 4 hours at half-past the hour.
	`HourlyTaskScheduler hts = new HourlyTaskScheduler();`
	`hts.setMinute(30);`
	`hts.setInterval(4);`

`DailyTaskScheduler`	Schedule a task to run daily or a specified number of days at a specified time (hour and minutes). Use in conjunction with the interval property to run the task after a specified number of days.
	Example: Schedule a task to run daily at 9:30 am (EST).
	`DailyTaskScheduler dts = new DailyTaskScheduler();`
	`dts.setMinute(30);`
	`dts.setHour(14);`

`WeeklyTaskScheduler`	Schedule a task to run every week (or every specified number of weeks) on a specified day (or days) at a specific time. The hours and minutes are set as UTC values. At least one of the boolean values must be set to `true`. You can also set the interval property to run the task after a specified number of weeks.
	Example: Schedule a task to run every Tuesday and Sunday at 30 minutes past midnight.
	`WeeklyTaskScheduler wts = new WeeklyTaskScheduler();`
	`wts.setMonday(true);`
	`wts.setTuesday(true);`
	`...`
	`wts.setSaturday(false);`
	`wts.setSunday(true);`
	`dts.setMinute(30);`
	`dts.setHour(4);`

Table 14-2. TaskScheduler Data Object Subtypes (Continued)

TaskScheduler Subtype	Usage

MonthlyByDayTaskScheduler	Schedule a task to run every month (or every specified number of months) on a specified day at a specified time (hour and minutes). You can also set the interval property to run the task after a specified number of months.
	Example: Schedule a task to run every 3 months (on the last day of the month) at 12:30 p.m.
	`MonthlyByDayTaskScheduler mbdts = new MonthlyByDayTaskScheduler();`
	`mbdts.setDay(31);`
	`mbdts.setInterval(3);`
	`mbdts.setMinute(30);`
	`mbdts.setHour(14);`

MonthlyByWeekdayTaskScheduler	Schedule a task to run every month (or every specified number of months) on a specified week, weekday, and time (hour: minutes). You can also set the interval property to run the task after a specified number of months.
	Example: Schedule a task to run on the last Wednesday of each month at 12:30 a.m.
	`MonthlyByWeekdayTaskScheduler mbwts = new MonthlyByWeekdayTaskScheduler();`
	`mbwts.setOffset(WeekOfMonth.last);`
	`mbwts.setWeekday(DayOfWeek.wednesday);`
	`mbwts.setHour(4);`
	`mbwts.setMinute(30);`

所有对象的小时和分钟属性扩展在 Coordinated Universal Time(UTC)中定义的 RecurrentTaskSchedule 数据对象而不是服务器上的本地时间。当你定义计划时，装换你的本地时间到一个 UTC 值。

例子 14-4 中的代码段定义了一个 ScheduledTask，在每天 4：15a.m 启动一个虚拟机，假如服务器本地时间是 Pacific Standard Time(PST)时区。为了一个在 Eastern European Summer (EEST)时区，已经被系统设置为 3：15pm。

Example 14-4. Scheduled Task for Powering-on Virtual Machines

```
...
// Set the schedule using the DailyTaskScheduler subtype.
DailyTaskScheduler dTScheduler = new DailyTaskScheduler();
dTScheduler.setHour(12);
dTScheduler.setMinute(15);
ScheduledTaskSpec tSpec = new ScheduledTaskSpec();
tSpec.setDescription("Start virtual machine as per schedule.");
tSpec.setEnabled(Boolean=TRUE);
tSpec.setName("Power On Virtual Machine");
tSpec.setAction(ma);
tSpec.setScheduler(dTScheduler);
tSpec.setNotification("admin@vmware.com");
my_conn.createScheduledTask(_sic.getScheduledTaskManager, vmRef, tSpec);
...
```

14.2.3.2 取消一个计划任务

你可以通过下列方法来取消一个计划的任务。

- 为了取消一个正在运行的计划任务，调用 ScheduledTask.RemoveScheduledTask。这个方法不能取消随后运行的 ScheduledTask。
- 为了取消即将运行的 ScheduledTask，调用具有一个新的 SchedulededTaskSpec 数据

对象其包含新的计划详细配置的 ScheduledTask.ReconfigureScheduledTask。

- 为了取消一个引起第二个任务的 ScheduledTask，创建一个 PropertyCollector 来获得这个任务的引用然后调用它自身的 CancelTask 方法。这个任务必须是可以被取消的。

14.2.4 使用 TaskHistoryCollector

TaskHistoryCollector 让你收集关于任务的信息。你通过 TaskManager.CreateCollectorForTasks 方法来创建一个 TaskHistoryCollector

创建一个 TaskHistoryCollector

1. 确定你想要收集的任务对象的类型，创建一个 TaskFilterSpec 数据对象实例来明确你的过滤器标准。

   TaskFilterSpec 包括一个 taskTypeId 属性，你可以用来限制收集到任务对象的类型。你也可以定义一个 TaskFilterSpecByTime 数据对象的 time 属性来在 TaskFilterSpec 中提供一个时间范围。
2. 获得你的服务器实例上的 TaskManager 的 managed 对象的引用。
3. 通过 CreateTaskHistoryCollector 方法提交过滤器和引用到服务器。服务期返回一个 TaskHistoryCollector 对象引用。

一个 HistoryCollector 被创建好后，服务器添加新的符合过滤器标准的对象到集合中在他们发生时。系统通过放置这个对象在 LatestPage 的第一个位置来添加新的对象到集合和从集合中移除老的对象。TaskHistoryCollector 对象的 LatestPage 属性有一个包含在集合中得最近 1000 个对象的属性。使用 PropertyCollector 来从 LatestPage 属性中获得条目。

HistoryCollector 仅存在于 session 实现它的期间。调用 HistoryCollector.DestroyCollector 方法来删除收集器在 session 结束前。

14.2.4.1 创建一个 TaskHistoryCollector 过滤器

当你创建了一个 TaskHistoryCollector 时，你可以定义过滤器。例如，与其返回所有和虚拟机关联的任务对象，你可以创建过滤器仅仅收集由备份管理员在 2:00 和 4:00 间执行的虚拟机相关的任务对象。

TaskFilterSpec 对象允许你制定集合标准。大多数属性是可选的和可以当作 null 值来提

交的。TaskFilterSpec 允许你基于用户名，实体类型，和任务状态来收集信息。

14.2.4.2 管理 HistoryCollector

HistoryCollector managed 对象提供管合的生命周期和滚动 views 的管理。

- DestroyCollector – HistoryCollector 仅仅存在于当前 session。调用 DestroyCollector 操作来显式销毁集合在 session 结束前。
- ResetCollector – 调整集合对象的子集的开始位置使对象立刻位于当前 latestPage 前面。
- RewindCollector – 定位 latestPage 到队列中老的条目上。当 HistoryCollector 被创建，它是默认位置。
- SetCollectorPageSize – 接受一个整形参数来设置 HistoryCollector 的 latestPage 属性的大小。HistoryCollector 的默认大小是一个具有最大 1000 个合适类型(Task,Event)的对象队列。队列根据对象的创建时间排序。

15、事件和警告

事件是 vSphere 传递关于发生在系统里的事情的信息。你可以直接监控事件或使用一个 EventHistoryCollector 在特定的时间段检索时间。

警告是 vSphere 提醒用户出现的问题。你也可以创建自定义的警告来监控系统和建立后续的动作。警告建立包括明确触发条件和定义结果触发的动作。

15.1 事件和警告管理对象

EventManger 是事件架构的服务接口。

Event 子类型定义了系统产生的事件。

EventHistoryCollector 允许你来监控事件。你可以创建一个过滤器来限制检索的事件号。你可以同时监控系统事件和你自定义的事件。

AlarmManager 是创建，设置，和管理警告的服务接口。你创建一个警告，明确触发条件和处理动作。当满足为警告定义的条件再系统上发生时，为警告定义的动作将开始。警告产生一个可以通过 EventHistoryCollector 检索到的事件。

15.2 了解事件

事件是一个数据对象类型包含了关于 managed 实体和其它再服务器上的对象状态改变信息。事件包括发生在数据中心，数据存储，群集，主机，资源池，虚拟机，网络，和分布式交换机上的用户动作和系统动作。例如，这些通常的系统活动产生一个或多个事件数据对象。

- 虚拟机的开机和关闭
- 创建一个虚拟机
- 在一个虚拟机的客户机操作系统上安装 VMware Tools
- 重新配置一个计算资源
- 添加一个新的已配置的 ESX/ESXi 系统到 vCenter 服务器上

在 vSphere 客户端软件中，在单独的 ESX/ESXi 系统上的事件对象信息显示在事件标签中，对于被管理的主机，事件对象信息显示在任务&事件标签中。

事件对象的保留时间依赖于系统安装。

- **独立的 ESX/ESXi 系统** – 事件对象不是可持续的。事件仅仅保存在主机系统的内存中。重新启动一个独立的 ESX/ESXi 主机或关闭一个虚拟机将从本地内存中移除事件对象。

  一个独立的 ESX/ESXi 系统可以保留事件对象 15 分钟，但是这非常依赖主机的进程负载，虚拟机的个数，和其它因素。
- **被管理的 ESX/ESXi 系统** – 事件对象是持续的。被管理的 ESX/ESXi 系统发送事件对象到管理他们的 vCenter 服务器上，vCenter 服务器保存这些信息到数据存储中。

使用 EventHistoryCollector，你可以获得在指定 ESX/ESXi 系统，或指定数据存储历史时期上搜集的对象的信息。

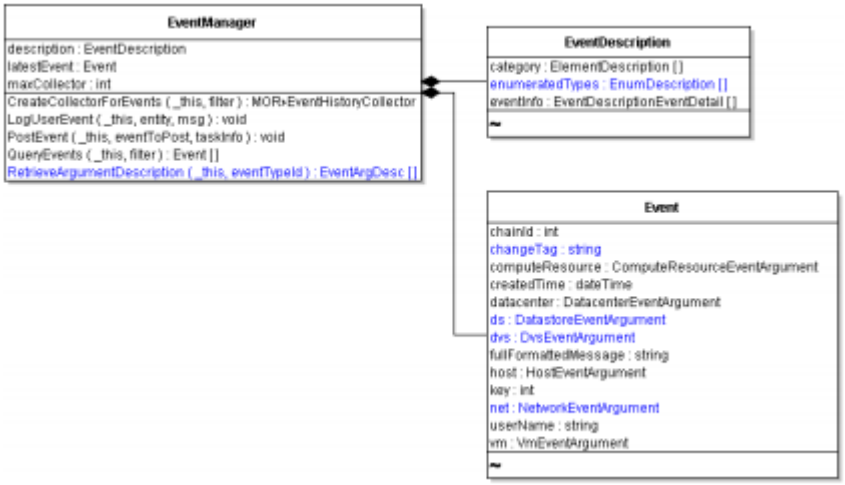
15.2.1 使用 EventManager 管理事件

EventManger 是事件架构的服务接口。图 15-1 显示了 EventManager 和关联的对象。一个 EventManager 有下列属性：

- 一个 description 属性，定义为一个 EventDescription 数据对象的实例，包含一个事件类型和其它信息。

- 一个 latesEvent 属性，它包含最近在内存中的事件数据对象。
- 一个 maxCollector 属性，指定每一个客户端 session 可以创建的 EventHistoryCollector 对象个数。这个值由 vCenter 服务器设置。

Figure 15-1. EventManager Managed Object and Associated Objects



15.2.2 事件数据对象

事件子类型定义了系统产生的事件。图 15-2 显示仅仅显示了扩展于事件数据对象的几个子类型。例如，TaskEvent 继承了所有 Event 属性并且包含一个 info 属性是一个 TaskInfo 对象的实例。

下列通常由控制台风格的客户端应用产生的事件对象：

com.vmware.vim.VmPoweredOnEvent

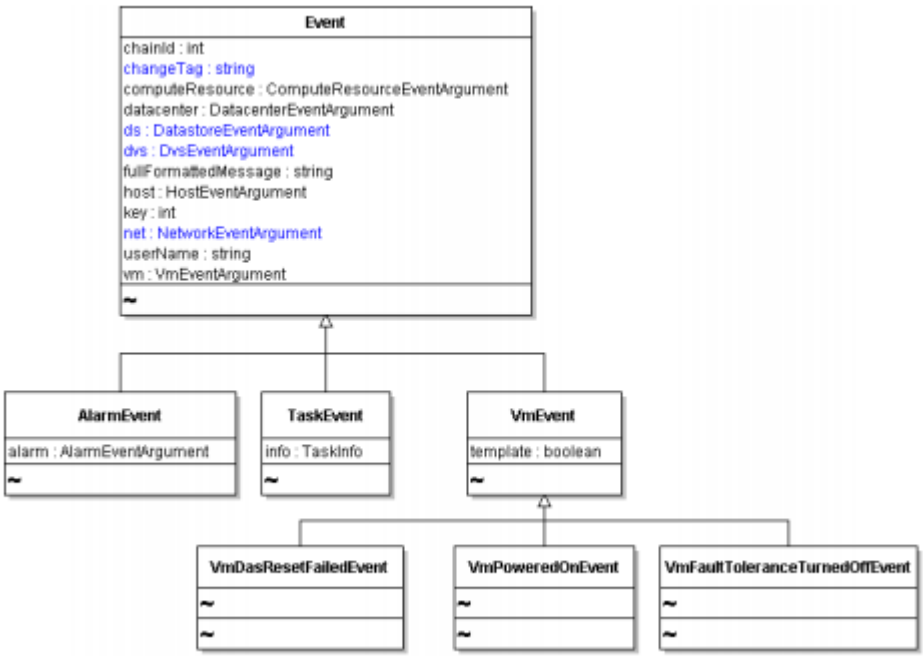
com.vmware.vim.VmStartingEvent

com.vmware.vim.VmReconfiguredEvent

com.vmware.vim.VmCreatedEvent

com.vmware.vim.VmBeingCreatedEvent

Figure 15-2. Event Data Object and Sample Subtypes



15.2.3 设计事件信息内容格式

当在控制台显示的时候，事件对象没有被格式化也没有提供上下文信息。你可以使用预定义在 Event.fullFormatedMessage 属性中的字符串来格式化一个事件信息。

你也可以基于上下文的信息来格式化一个事件信息。在运行时，事件数据对象是包含关联事件源信息的值。例如，事件数据对象的 computeResource，datacenter，ds，dvs，host，net，和 vm 属性。

你可以使用具有 EventManager.description.eventInfo 中的 EventDescriptionEventDetail 中信息的事件对象的属性来格式化事件信息。

15.2.4 创建自定义事件

EventManager.logUserEvent 方法允许你来创建自定义事件对象。你可以将任何管理的实体和你自定义的事件相关联。

定义一个自定义事件：

1. 获得到 EventMangager 的 managed 对象引用。

   ..

   ManagedObjectReference _svcRef = new ManagedObjectReference();

```
ServiceContent _sic = my_conn.retrieveServiceContent(_svcRef);
ManagedObjectReference eMgrRef = _sic.getEventManager();
...
```

2. 获得你要关联事件的实体的 managed 对象引用。
3. 调用 logUserEvent 方法，传入 EventManger 和 Event 的引用和一个包含事件信息的字符串给 msg 参数。

用户定义的事件对象显示在 vSphere 客户端的系统其他事件中，具有前缀 User logged event：接着是传入到 msg 参数的文本。在其他客户端中，例如在基于控制台的事件例程，自定义事件作为 com.vmware.vim.GeneralUserEvent 对象显示。

15.3 使用 EventHistoryCollector

EventHistoryCollector 让你可以收集系统已经产生的事件信息。你可以使用 EventManager.CreateCollectorForEvent 方法创建一个 EventHistoryCollector。

创建一个 EventHistoryCollector

1. 确定你要收集事件对象的类型，创建一个明确你过滤规则得 EventFilterSpec 数据对象实例。

   EventFilterSpec 包括一个 eventTypeId 属性，可以通过制定类型来限制你要收集的事件对象集合。你也可以在 EventFilterSpec 中提供一个时间范围，通过为它的 time 属性定义一个 EventFilterSpecByTime 数据对象来实现。
2. 获得在你服务器实例上的 EventManager 的 managed 对象引用。
3. 通过 CreateEventHistoryCollector 操作提交过滤器和和引用到服务器。

在你创建了 HistoryCollector 后，当他们发生时服务器添加符合过滤器规则的新对象到集合中。系统添加新对象到集合位于 latesPage 的第一个位置和移除集合最后一个对象。EventHistoryCollector 的 latesPage 属性具有一个拥有 1000 个最近的对象集合。使用一个 PropertyCollector 来从 latesPage 属性中获得具体的条目。

HistoryCollector 仅仅存在 seesion 实例化它的期间。你调用 DestroyCollector 操作来显示的来消除集合在 session 结束前。

15.3.1 创建一个 EventHistoryCollector 过滤器

当你创建一个 EventHistoryCollector 时，你可以定义过滤器。例如，不用返回所有与虚拟机关联的事件对象，你可以创建过滤器仅仅收集由备份管理员在 2:00 和 4:00 间执行的虚拟机相关的事件对象。

EventFilterSpec 对象允许你指定集合规则。大多数属性是可选的和可以提交 null 值。EventFilterSpec 让你可以收集基于用户名，实体类型，时间，和事件状态的事件。

15.3.2 管理 HistoryCollector

HistoryCollector managed 对象提供管理生命周期和一个集合的滚动 view。

- DestroyCollector – HistoryCollector 仅仅存在于当前 session。调用 DestroyCollector 操作来显式销毁集合在 session 结束前。
- ResetCollector – 调整集合对象的子集的开始位置使对象立刻位于当前 latestPage 前面。
- RewindCollector – 定位 latestPage 到队列中老的条目上。当 HistoryCollector 被创建，它是默认位置。
- SetCollectorPageSize – 接受一个整形参数来设置 HistoryCollector 的 latestPage 属性的大小。HistoryCollector 的默认大小是一个具有最大 1000 个合适类型(Task,Event)的对象队列。队列根据对象的创建时间排序。

15.4 使用警告

vSphere 警告架构支持自动行动和发送不同类型的通知来响应特定的服务器条件。大多数警告默认存在于 vCenter 服务器系统上。你也可以自己创建警告。例如，一个警告可以在 CPU 使用率在一个特定的虚拟机上超过 30 分钟使用率超过 99%时发送电子邮件。

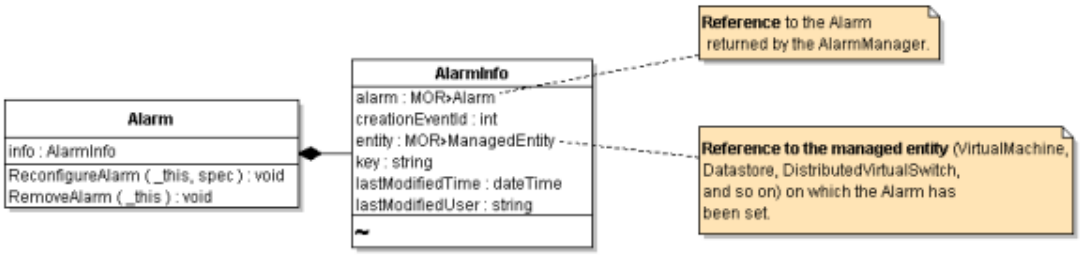
警告架构可以和其他服务组件集成在一起，例如事件和性能计数器。

AlarmManager 是创建，设置和管理警告的服务接口。你创建一个警告，明确出发条件和执行的动作。当定义在系统上的警告条件发生时，为警告确定的动作将启动。警告也会长生一个事件被传递给事件历史数据库。另外，由警告开启的动作也可以发送第二个事件到数据库，依赖于动作类型。

15.4.1 获取警告列表

使用 AlarmManager.GetAlarm 方法来获取一个为确定的 managed 实体定义的所有警告 managed 对象引用的队列。当你调用方法，你可以传入一个可选的 managed 实体引用。如果没有一个 managed 实体引用，GetAlarm 操作返回所有和 session 调用这个操作关联的可见的实体的所有警告对象。

Figure 15-3. Alarm Managed Object



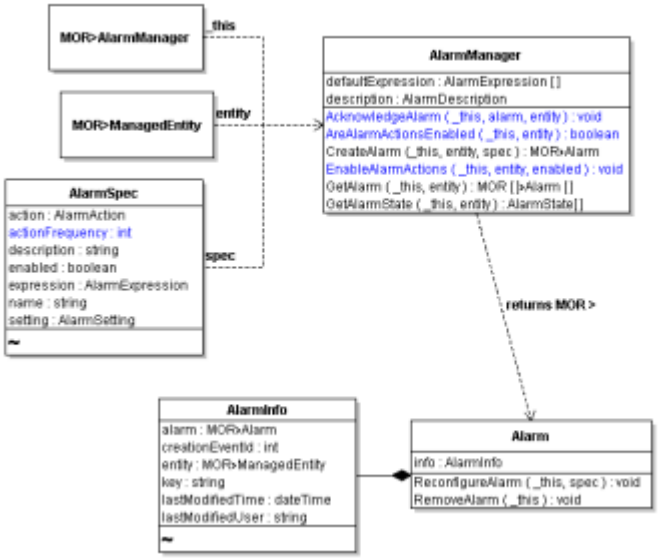
Alarm.info 属性是一个 AlarmInfo 数据对象。你可以通过收集 AlarmInfo 数据对象来获得关于活动的警告信息。

15.4.2 创建警告

你通过 AlarmManager.CreateAlarm 方法来创建一个警告。最简单的情况，你在 AlarmSpec.expression 属性中明确触发条件和在 AlarmSpec.action 属性中明确执行动作。当 expression 评估为 true 时，警告执行动作。

图 15-4 显示 CreateAlarm 方法

Figure 15-4. CreateAlarm Method Inputs and Outputs



创建一个警告

1. 获得一个位于 vCenter 服务器上的 AlarmManager 的 managed 对象引用。
2. 获得你想要设置警告的实体的 managed 对象引用。
3. 创建一个 AlarmSpec 数据对象和在属性中为这个警告制定细节。
4. 调用 AlarmManager.CreateAlarm，传入引用和 AlarmSpec 数据对象。系统返回一个到这个警告的 managed 对象引用。

15.5 使用 AlarmSpec 数据对象定义警告

AlarmSpec 数据对象具有警告的所有方面的属性，包括表达式和采取的行动当表达式评估为 true 时。下列属性定义警告。

- action – 当警告变为活动状态时动作被调起。Action 子类型之一。
- actionFrequency – 警告需要启动制定动作剩余的秒数。
- expression – 一个或多个 AlarmExpression 数据对象和在一起评估出一个 true-false 表达式。
- setting – 在 AlarmSetting 数据对象中为警告定义容错和频率限制。

  AlarmSetting 包含两个整形属性：

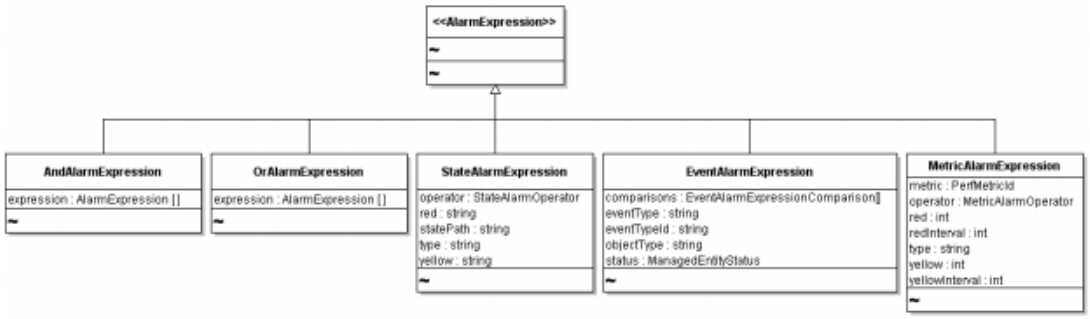
  - reportingFrequence，明确一个警告中两个活动间的时间间隔。使用 0 来明确当需要警告可以激活。
  - toleranceRange，指定可接受的在一个 MetricAlarmExpression 中定义的值的上

下范围(百分比)

15.5.1 使用 AlarmExpression 指定警告触发条件

你使用 AlarmExpression 数据对象来指定你想要警告触发的条件。AlarmExpression 数据对象是一个具有几个子类型的抽象类型，其允许你在对象上设置临界值，对象状态，或指定事件监控。

Figure 15-5. AlarmExpression and Its Subtypes



15.5.1.1 AlarmExpression 类型

通过使用合适的 AlarmExpression 类型，你可以为警告设置不同的条件，状态，或事件。

Table 15-1. Alarm Expressions and Examples

AlarmExpression	Description	Example
`StateAlarmExpression`	Specifies thresholds that trigger the alarm.	Triggered by a power state change of a virtual machine or state change of a distributed virtual switch.
`MetricAlarmExpression`	Specifies levels at which the alarm changes state. See "Using MetricAlarmExpression" on page 175.	Triggered when resource utilization metrics exceed a specified limit.
`EventAlarmExpression`	Specifies a type of event as the basis for the alarm.	Triggered by power on or power off events of primary or secondary virtual machines in a fault-tolerant cluster.
`EventAlarmComparison`	Specifies the property of the `Event` that should trigger the alarm and the operator to use as the basis for comparison.	
`AndAlarmExpression` `OrAlarmExpression`	Combines one or more instances of the `AndAlarmExpression` and the `OrAlarmExpression` data objects into an expression that evaluates to `true` or `false`.	

15.5.1.2 使用 MetricAlarmExpression

MetricAlarmExpression 数据对象让你设置一个警告来监控性能度量。vSphere 客户端使

用数据对象来指示当在一个 DAS 或 DRS 群集环境中的主机或群集没有足够的资源。

你设置 metric 属性到一个你想要在系统上监控的性能度量的 PerfMeticId。设置 red 或 yellow 属性来确定 metric 的值从 green，到 yellow，到 red。你必须定义 red，yellow，或全部两个属性。使用 MetricAlarmOperator 枚举类型的 isAboue 或 isBelow 来完成阀值得定义。

使用 red 和 yellow 属性，你可以使用 redInterval 或 yellowInterval 属性。这些属性使你可以设置性能度量必须停留在 red 或 yellow 状态的秒数在表达式变为 true 和触发定义的动作之前。

15.5.2 指定警告动作

你通过设置 AlarmSpec 数据对象的 action 属性到为目的而定义的 AlarmAction 数据对象上来指定系统调用的动作。

AlarmAction 数据对象是一个具有两个派生对象的抽象类型。

- AlarmTriggeringAction 数据对象有一个 action 属性和一个 transitionSpec 属性。AlarmTriggeringActionTransitionSpec 允许你为警告定义一个开始状态和结束状态。你可以通过指定 AlarmTriggerActionTransitionSpec 的 repeats 属性为 false 来限制警告实际的触发数。
- GroupAlarmAction 数据对象是一个 AlarmAction 基础类型版本队列。你可以创建一个单一的 AlarmAction 实例或一个 AlarmAction 队列实例当你的警告定义在系统上的条件触发时。

系统通过下列方式响应警告：

- 调用一个操作。为调用一个操作，创建一个 MethodAction 数据对象。
- 运行一个脚本。为运行一个脚本，创建一个 RunScriptAction 数据对象的实例，指定其在 vCenter 服务器上的绝对路径来运行 shell 脚本。
- 发送一个 email 信息。为发送一个 email 信息给系统管理员，使用 SendEmailAction 数据对象。

Figure 15-6. AlarmAction and Related Objects

举例，你可以使用 MethodAction 数据对象类型在服务器上调用一个操作。

MethodAction 数据对象包含下列属性：

- name – 你想要在计划的时间调用的操作名称。
- argument – 明确需求要的参数，如果有的话，作为一个 MethodArgumentAction 数据对象队列。

  依赖于和警告相关连的实体，MethodAction.argument 属性不是必须的。

15.6 删除或禁用一个警告

一个警告保持活动直到你删除它或禁用它。为了删除警告，需要取得 Alarm 的 managed 对象引用并且调用它的 RemoveAlarm 操作。

为了禁用这个警告，需要获得到 AlarmManager 的 managed 对象引用和警告设置的实体的 managed 对象引用。调用 AlarmManager.EnableAlarmActions 操作，传入 false。

16、vSphere 性能

VMware vSphere 服务器使用性能计数器来跟踪资源的使用。在运行时，vSphere 组件产生性能数据，vSphere 存储在性能计数器中。你可以使用 PerformanceManager 接口来检索这些数据。

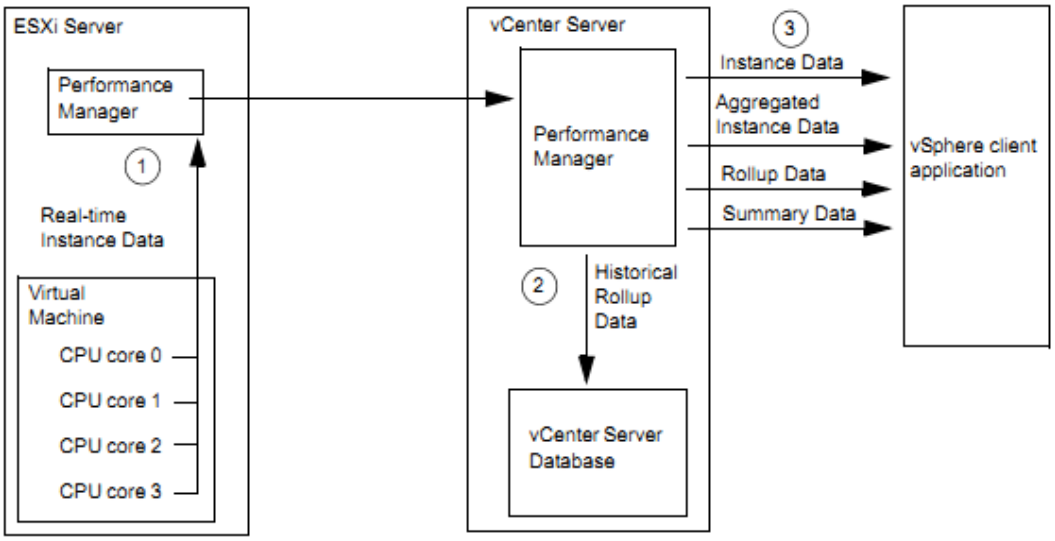
16.1 vSphere 性能数据收集

在 vSphere 环境中，虚拟机和物理组件产生性能数据。为了跟踪资源使用情况，ESXi 服务器执行实时数据收集 vCenter 服务器存储这些数据到 vCenter 数据库中。vCenter 服务器也可以存储符合性能间隔的一个历史数据汇总。

- 实时数据收集 – 一个 ESXi 服务器为每一个性能计数器每 20 秒收集一次数据并且保留数据 1 个小时。
- 历史数据汇总 – 一个 vCenter 服务器从所有的由其管理的主机上收集数据。PerformanceManager 定义了性能数据汇总的时间间隔，一套合并数据值得方法。服务器存储汇总后的性能计数器数据到 vCenter 数据库中。

下列图解展示了 vSphere 性能数据收集和检索。

Figure 16-1. vSphere Performance Data Collection and Retrieval



1. ESXi 服务器每 20 秒抽查性能计数器实例并且保留实时实例数据 1 个小时。例如，图示显示收集 4 个 CPU 内核的 CPU 统计数据。
2. vCenter 服务器从它管理的主机上检索和存储数据。vCenter 服务器产生符合设定的历史间隔的汇总数据。
3. vSphere 客户端应用可以检索实时实例数据，合计实例数据，历史汇总数据，和摘要数据。

下列表定义的条目用来描述 vSphere 性能管理。

Table 16-1. Performance Management Terminology

Term	Definition
performance providers	Performance providers include managed entities, such as hosts, virtual machines, compute resources, resource pools, datastores, and networks.
performance counter	Unit of statistical data collected on a vSphere server. For example, a vCenter server collects the average CPU utilization for hosts, virtual machines and clusters (the counter `cpu.usage.average`).
counter ID	System-generated identifier for a performance counter.
instance	An identifier derived from device configuration names. Examples of counter instances are the name of a virtual Ethernet adapter such as "vmnic0:", or a number that identifies a CPU core, such as 0, 1, 2, or 3. Performance data is retrieved as specific instances of performance counters.
instance data	Performance data collected at 20-second intervals.
metric ID	Combination of a counter ID and an instance. You use metric IDs – `PerfMetricId` objects – when you construct a performance query specification to identify the data to be collected. There are two system-defined instances that you can use to specify aggregate retrieval. See the description of aggregate performance data below. ■ "*" – An asterisk directs the vSphere Server to return all instances plus rollup data. This is not supported for some disk-related counters. ■ "" – A string of length zero directs the vSphere Server to return only aggregated instance data or rollup type data. The vSphere Server returns metric IDs embedded in the data objects that it returns as a response to performance queries.

Table 16-1. Performance Management Terminology

Term	Definition
performance interval	Data object (`PerfInterval`) which defines the time interval between collection events, the collection level, and the time period that the data will be stored on the Server. ■ ESXi Servers define a built-in performance interval that specifies data collection every 20 seconds for each performance counter. ESXi Servers also define a single historical interval (`PerformanceManager.historicalInterval`) that defines aggregate performance data. This system-defined performance interval specifies aggregate data collection every 300 seconds for each counter. You cannot modify the performance intervals on an ESXi Server. ■ vCenter Servers define four performance intervals that determine how collected instance data is aggregated and stored. You can modify the system-defined intervals on a vCenter Server to a limited extent.
collection level	Number between one and four that is assigned to a performance interval (`PerformanceManager.historicalInterval[].level`). The interval collection level corresponds to the level specified for individual performance counters (`PerfCounterInfo.level`). A vCenter Server uses a performance interval to perform performance data aggregation, using data for the counters with levels that match the performance interval collection level.
rollup type	Methodology for producing a single value from a set of statistical values (`PerformanceManager.perfCounter[].rollupType`). Examples of rollup types are average, latest, and summation.
aggregate performance data	A single value that represents a set of instance data values collected for a performance counter. The single value is derived using one of the rollup types.

16.2 PerformanceManager 对象和方法

PerformanceManager 提供获取关于系统各个方面的性能统计数据，由性能 provider 产生和维持的。它也可以定义历史性能间隔和确定一系列你可以是用来获得性能数据的性能计数

器。下列表展示了 PerformanceManager 属性。

Table 16-2. PerformanceManager Properties

Property	Description
description	Composite object that includes information about the types of counters (`counterType`) and statistics (`statsType`) available on the system.
historicalInterval	Array of system-defined performance intervals (`PerfInterval` data objects). Each object defines the interval between rollup events, the collection level, and the time period that the data is stored on the system. ■ For an ESXi system, the array contains a single performance interval. You cannot modify the ESXi performance interval. ■ For vCenter Server systems, the `PerfInterval` objects control how ESXi performance data are rolled up and stored in the database. You can modify some of the `PerfInterval` properties on a vCenter Server.
perfCounter	Array of `PerfCounterInfo` data objects. The array identifies all of the performance counters known to the vCenter Server at the time a client accesses the array. The set of counters may change as ESXi hosts are added or removed from vCenter management. Each `PerfCounterInfo` object contains metadata associated with a performance counter.

PerformanceManager 方法允许你来检索性能统计和定义统计的原数据。下表把方法归类和描述了他们的作用。

Table 16-3. PerformanceManager Methods

Method Type	Method	Purpose
Performance data availability	QueryAvailablePerfMetric	Returns `PerfMetricId` objects which identify the counter data available on the specified entity. For example, a virtual machine provides the memory counter `granted`, which indicates the amount of physical memory that is mapped for the virtual machine. The `PerfMetricId` object for the `mem.granted.average` counter specifies the system-defined counter ID. Since this is a memory counter, the `PerfMetricId.instance` property is empty.
Performance data retrieval	QueryPerf	Returns statistics for a specific list of managed entities that provide performance data.
	QueryPerfComposite	Returns statistics for a host and its virtual machines. This method accepts the `refreshRate` for current statistics or the `intervalId` of one of the historical intervals as a parameter. Supported for the `HostSystem` managed entity only.
Performance counter metadata retrieval	QueryPerfCounter	Returns `PerfCounterInfo` data objects for the specified list of counter IDs.
	QueryPerfCounterByLevel	Returns `PerfCounterInfo` data objects for the specified collection level.
Performance provider information	QueryPerfProviderSummary	Returns the `PerfProviderSummary` data object for the specified managed object.
Collection parameters	ResetCounterLevelMapping	Restores a set of performance counters to their default collection levels.
	UpdateCounterLevelMapping	Changes the collection level for a set of performance counters.
	UpdatePerfInterval	Modifies the system-defined performance intervals.

16.3 检索 vSphere 性能数据

为了检索已收集的数据，你的客户端应用需要创建一个查询声明并且把查询声明传入到一个性能查询方法中。这个查询声明由一个或多个 PerfQuerySpec 对象组成。每一个对象身份如下：

- Performance provider – 服务器返回性能数据的 managed 实体(PerfQuerySpec.entity)
- Performance counter – PerfMetricId 对象确定性能计数器实例(PerfQuerySpec.metricId)
- Performance interval – 数据汇总取样期间(PerfQuerySpec.interval)
- 返回数据总数 – 开始和结束所需时间(PerfQuerySpec.startTime,PerfQuerySpec.endTime) 和最大取值个数(PerfQuerySpec.maxSample)来限制返回的数据总数。
- 输出数据格式(PerfQuerySpec.format) – 两种方式可选：
  - 值包含在数据对象中的正常输出
  - 包含逗号分割符的值格式化成字符串输出

Entity 和 metricID 组合确定一组服务器将返回性能数据的计数器。Interval，startTime，endTime 属性组合产生实例，聚合实例，汇总，或总结数据。下表总结了你可以从 vCenter 服务器检索到的不同的性能数据分类。

Table 16-4. Classification of Performance Data By Performance Interval

Performance Data	Description
Instance	ESXi Servers sample performance data every 20 seconds. 20-second interval data is called instance data or real-time data. To retrieve instance data, specify a value of 20 seconds for the `PerfQuerySpec.intervalId` property.
Aggregated Instance	A vSphere client can retrieve aggregated instance data. To obtain aggregated instance data, specify the following `PerfQuerySpec` properties. ■ `intervalId` – Specify 20 seconds to indicate instance data. ■ `metricId[].instance` – specify a zero-length string ("") for aggregated instance data.
Rollup	The vCenter Server uses the historical intervals to rollup performance data from the servers that it manages. To retrieve historical performance data, specify the following `PerfQuerySpec` properties. ■ `intervalId` – Specify a value that corresponds to one of the historical intervals (`PerformanceManager.historicalInterval[].samplingPeriod`). ■ `startTime/endTime` – If specified, use time values that are not within the last 30 minutes of the current time. If you do not specify a starting time, the Server will return values starting with the earliest data. If you do not specify an end time, the Server will return values that include the latest data.
Summary	When you call the `QueryPerf` method and specify a performance interval (`PerfQuerySpec.intervalId`) that does not match one of the historical intervals (`PerformanceManager.historicalInterval[].samplingPeriod`), the Server will attempt to summarize the stored data for the specified interval. In this case, the Server may return values that are different from the values that were stored for the historical intervals.

16.3.1 性能计数器例程(QueryPerf)

下列代码片断是一个使用 PerformanceManager.QueryPerf 方法来获得一个虚拟机性能统计的例子的一部分。

这个例子检索下列统计数据：

- disk.provisioned.LATEST – 虚拟机存储能力。
- mem.granted.AVERAGE – 映射到虚拟机上的物理内存总数。
- power.power.AVERAGE – 当前电源使用情况。

例子创建一个查询声明(PerfQuerySpec)来确定要检索的数据，调用 QueryPerf 方法，并且打印出检索的性能数据和相应的性能计数器元数据。下列章节描述了检索性能统计的基本步骤。

- 映射性能计数器
- 创建一个性能查询声明并且调用 QueryPerf 方法
- 处理返回的数据。

16.3.1.1 映射性能计数器(计数器 ids 和元数据)

性能计数器通过字符串名称来被表示出来，例如，disk.provisioned.LATEST 或 mem.granted.AVERAGE。一个 vSphere 服务器使用 system-generated 计数器 IDs 来跟踪性能计数器。当你创建了一个性能查询，你使用计数器 IDs 来确定要检索的统计信息，因此它用来映射名称到 IDs。

例子必须在调用 QueryPerf 时指定计数器 IDs，并且在它打印出返回数据时使用性能计数器元数据。为了获得性能计数器 IDs 和相应的性能计数器元数据，例子创建了两个哈西 maps。例子映射全部性能计数器来支持任何计数器的检索。

HashMap 声明

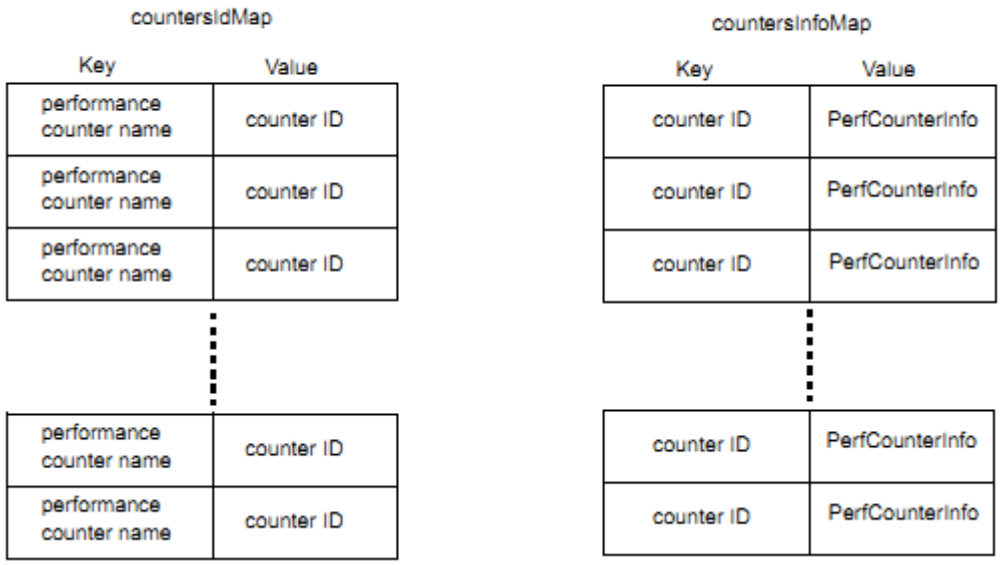
下列代码片断声明了两个哈西 maps。

- countersIdMap – 使用完整的计数器名称来索引性能计数器 IDs。一个完整的计数器名称是由计数器组，名称，和汇总类型合并组成。例子中使用这个 map 来获得计数器 IDs 在它创建性能查询声明时。
- countersInfoMap – 使用性能计数器 IDs 来索引 PerformanceCounterInfo 数据对象。例子中使用这个 map 来获得元数据在它打印返回的性能数据时。

```
/*
 * Map of counter IDs indexed by counter name.
 * The full counter name is the hash key - group.name.ROLLUP-TYPE.
 */
private static HashMap<String, Integer> countersIdMap = new HashMap<String, Integer>();
/*
 * Map of performance counter data (PerfCounterInfo) indexed by counter ID
 * (PerfCounterInfo.key property).
 */
 private static HashMap<Integer, PerfCounterInfo> countersInfoMap =
          new HashMap<Integer, PerfCounterInfo>();
```

下列图示显示了哈西 maps 的表现形式。

Figure 16-2. Performance Counter Hash Maps



创建 Map

例子是用 Property Collector 来检索 vCenter 服务器(Performance.perfCounter[])上已知的性能计数器(PerfCounterInfo)队列。然后使用这些数据创建 maps。代码段使用 apiMethods 变量，它是一个可以访问 vSphere API 方法的 VimPortType 对象。

下面是代码段执行步骤：

1. 创建一个 ObjectSpec 来定义属性收集器的上下文。这个例子指定了性能管理。
2. 创建一个 PropertySpec 来定义要检索的数型，这个例子检索 perfCounter 属性，它

是一个 PerfCounterInfo 对象的队列。

3. 为调用 PropertyCollector 创建一个 PropertyFilterSpec。ProprtyFilterSpec 创建了 ObjectSpec 和 PropertySpec 之间的关联。
4. 调用 PropertyCollector.RetrievePropertyEx 方法。这个方法阻塞直到服务器返回请求的属性数据。
5. 转换返回的 xsd::anyType 值到 PerfCounterInfo 对象队列。
6. 通过循环返回的队列装载入 map。计数器名称到计数器 ID 的映射使用一个完整有效的计数器名称。有效的名称是一个由计数器组，计数器名称，和汇总类型-group.counter.ROLLUP-TYPE 组成的路径。汇总类型必须为大写。有效的名称例如 disk.provisioned.LATEST 和 mem.granted.AVERAGE。

```
/*
 * Create an object spec to define the context to retrieve the PerformanceManager property.
 */
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(performanceMgrRef);
/*
 * Specify the property for retrieval
 * (PerformanceManager.perfCounter is the list of counters of which the vCenter Server is aware.)
 */
PropertySpec pSpec = new PropertySpec();
pSpec.setType("PerformanceManager");
pSpec.getPathSet().add("perfCounter");
/*
 * Create a PropertyFilterSpec and add the object and property specs to it.
 */
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
/*
```

```
 * Create a list for the filter and add the spec to it.
 */
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
/*
 * Get the performance counters from the server.
 */
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = apiMethods.retrievePropertiesEx(pCollectorRef,fSpecList,ro);
/*
 * Turn the retrieved results into an array of PerfCounterInfo.
 */
List<PerfCounterInfo> perfCounters = new ArrayList<PerfCounterInfo>();
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                /*
                 *  DynamicProperty.val is an xsd:anyType value to be cast
                 *  to an ArrayOfPerfCounterInfo and assigned to a
List<PerfCounterInfo>.
                 */
                perfCounters                                                =
((ArrayOfPerfCounterInfo)dp.getVal()).getPerfCounterInfo();
            }
        }
    }
}
/*
```

```
 * Cycle through the PerfCounterInfo objects and load the maps.
 */
for(PerfCounterInfo perfCounter : perfCounters) {
    Integer counterId = new Integer(perfCounter.getKey());
    /*
     * This map uses the counter ID to index performance counter metadata.
     */
    countersInfoMap.put(counterId, perfCounter);
    /*
     * Obtain the name components and construct the full counter name,
     * for example – power.power.AVERAGE.
     * This map uses the full counter name to index counter IDs.
     */
    String counterGroup = perfCounter.getGroupInfo().getKey();
    String counterName = perfCounter.getNameInfo().getKey();
    String counterRollupType = perfCounter.getRollupType().toString();
    String fullCounterName = counterGroup + "." + counterName + "." +
counterRollupType;
    /*
     * Store the counter ID in a map indexed by the full counter name.
     */
    countersIdMap.put(fullCounterName, counterId);
}
```

16.3.1.2 检索统计信息

下列代码段调用 QueryPerf 方法来检索统计数据。它执行系列任务：

1. 创建一个有效的性能计数器名称队列。名称是一个路径由 group-name.counter-name.ROLLUP-TYPE 组成，例如 mem.granted.AVERAGE。汇总类型必须是大写要匹配性能计数器元数据中的汇总类型

(PerfCounterInfo.roolupType)。

2. 创建一个 PerfMetricId 对象的队列，每一个计数器都被检索。Metric ID 是一个由计数器 ID 和实例组合而成的。为了填充 PerfMetricId 属性，例程做了如下工作：
   - 使用 countersIdMap 来转换一个完整的计数器名称成为一个计数器 ID。
   - 为 PerfMetricId.instance 属性指定星号(*)。星号是系统定义的包含实例和汇总检索的实例声明。
3. 为方法调用建立一个查询声明。这个查询明确了如下信息：
   - 要检索性能数据的虚拟机(entityMor)。
   - 300 的间隔 ID 来收集 5 分钟的汇总数据。
   - 为检索的数据采用逗号分割值(CSV)格式。
4. 调用 QueryPerf 方法。

```
/*
 * Use <group>.<name>.<ROLLUP-TYPE> path specification to identify counters.
 */
String[] counterNames = new String[] {"disk.provisioned.LATEST",
                                      "mem.granted.AVERAGE",
                                      "power.power.AVERAGE"};
/*
 * Create the list of PerfMetricIds, one for each counter.
 */
List<PerfMetricId> perfMetricIds = new ArrayList<PerfMetricId>();
for(int i = 0; i < counterNames.length; i++) {
    /*
     * Create the PerfMetricId object for the counterName.
     * Use an asterisk to select all metrics associated with counterId (instances and rollup).
     */
    PerfMetricId metricId = new PerfMetricId();
    /* Get the ID for this counter. */
    metricId.setCounterId(countersIdMap.get(counterNames[i]));
    metricId.setInstance("*");
```

```
        perfMetricIds.add(metricId);
    }
    /*
     * Create the query specification for queryPerf().
     * Specify 5 minute rollup interval and CSV output format.
     */
    int intervalId = 300;
    PerfQuerySpec querySpecification = new PerfQuerySpec();
    querySpecification.setEntity(entityMor);
    querySpecification.setIntervalId(intervalId);
    querySpecification.setFormat("csv");
    querySpecification.getMetricId().addAll(perfMetricIds);
    List<PerfQuerySpec> pqsList = new ArrayList<PerfQuerySpec>();
    pqsList.add(querySpecification);
    /*
     * Call queryPerf()
     *
     * QueryPerf() returns the statistics specified by the provided
     * PerfQuerySpec objects. When specified statistics are unavailable -
     * for example, when the counter doesn't exist on the target
     * ManagedEntity - QueryPerf() returns null for that counter.
     */
    List<PerfEntityMetricBase> retrievedStats = apiMethods.queryPerf(performanceMgrRef,
    pqsList);
```

16.3.1.3 vSphere 服务器返回的性能数据

查询方法返回抽样的信息和性能数据。抽样信息预示着收集的间隔时间和数据收集的时间点。当你调用性能查询方法时，你传入查询声明(PerfQuerySpec)来指定要检索的性能数据。为了格式化输出数据，为 PerfQuerySpec.format 属性指明”normal”或”csv”。

查询方法返回 PerfEntityMetricBase 对象，你必须转换成为合适的类型来和在方法调用时声明的 PerfQuerySpec.format 相一致。

- QueryPerf 方法返回一个 PerfEntityMetricBase 对象队列。
- QueryPerfComposite 方法返回一个 PerfCompositeMetric 对象，包含 PerfEnttityMetricBase 对象。

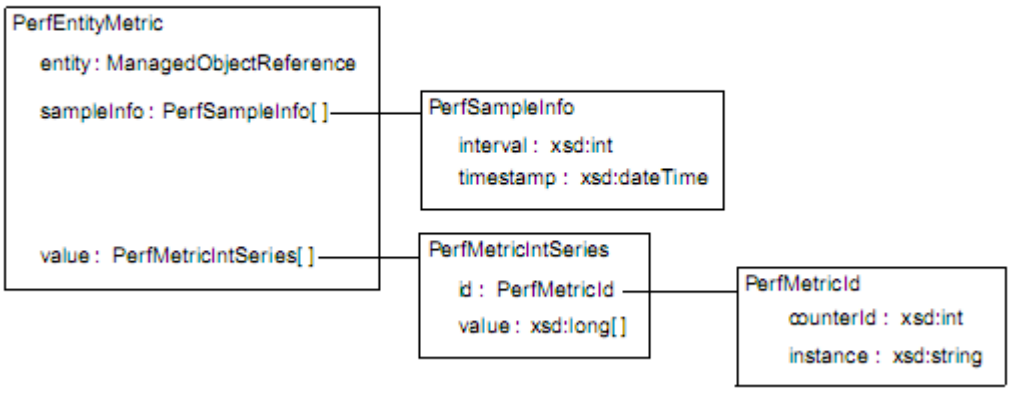
Normal Outup Format

当你指定”normal”格式时，你必须转换返回的 PerfEntityMetricBase 对象成 PerfEntityMetric 对象。每一个 PerEntityMetric 对象包含下列属性：

- entity – 性能提供者的引用。
- sampleInfo – 抽样信息的队列(PerfSampleInfo 数据对象)，编码为 xsd:int 和 xsd:dateTime 类型值。
- value – 数据值队列(PerfMetricIntSeries 数据对象)。每一个队列中的对象包含下列属性：
  - id – 性能度量 ID 代表计数器实例。
  - value – 整形数值队列对应于抽样信息队列(PerfEntityMetric.sampleInfo)。

下图展示了查询方法对于 normal 格式返回的数据对象层次结构。

Figure 16-3. PerfEntityMetric Object Hierarchy



CSV 输出格式

当你指定”csv”格式，你必须转换返回的 PerfEntityMetricBase 对象成为 PerfEntityMetricCSV 对象。抽样信息和收集的数据按照逗号分隔值编码适合表格格式。

PerfEntityMetricCSV 对象包含下列属性：

- entity –性能提供者的引用。
- sampleInfoCSV – 字符串包含一组时间间隔和日期数值。属性包含 PerfSampleInof

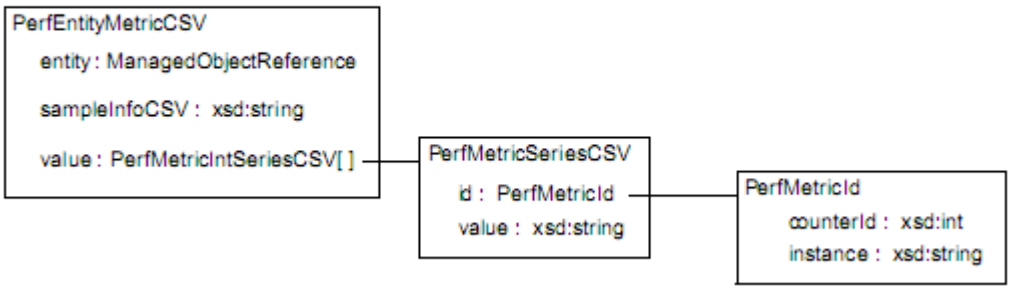
xsd:int 和 xsd:dateTime 数值的字符串表示。字符串值被编码为下列 CSV 格式：

interval1,date1,interval2,date2

- value -数据值队列(PerfMetricSeriesCSV 数据对象)。每一个队列中的对象包含下列属性：
  - id – 性能度量 ID 指定了计数器实例。
  - value – 一组 CSV 格式的抽样数据值，和抽样信息队列一致(PerEntityMetricCSV.sampleInfoCSV)。

下图展示了查询方法对于 csv 格式返回的数据对象层次结构

Figure 16-4. PerfEntityMetricCSV Object Hierarchy



16.3.1.4 处理返回的性能数据

下列代码段打印出返回的性能数据。这个例子使用 CSV 格式的数据。代码段执行下列任务：

- 循环 QueryPerf 方法(retrieveStats)返回的 PerEntityMetricBase 对象队列。
  - 转换 PerfEntityMetricBase 对象成一个 PerfEntityMetricCSV 对象来处理在 PerfQuerySpec 中声明的 CSV 输出。
  - 检索取样数据的值。
  - 检索间隔信息(csvTimeInfoAboutStats)。sampleInfoCSV 字符串(PerfEntityMetricCSV.sampleInfoCSV)是 PerfSampleInfo 数据被格式化为由逗号分隔的间隔，时间对—interval-1，time-1，interval-2，time-2。对队列嵌入在字符串中对应取样数据的值队列(PerfEntityMetricCSV.value[])。
  - 打印时间和间隔信息。
  - 循环取样数据的值(metricsValues)。
    - 使用 countersInfoMap 来转换 PerfMetricSeriesCSV 返回的计数器 ID 成为

相对应的 PerfCounterInfo 对象。

- 使用计数器元数据来打印出关于返回计数器的抽样数据值得确认信息。

```
/*
 * Cycle through the PerfEntityMetricBase objects. Each object contains
 * a set of statistics for a single ManagedEntity.
 */
for(PerfEntityMetricBase singleEntityPerfStats : retrievedStats) {
    /*
     * Cast the base type (PerfEntityMetricBase) to the csv-specific sub-class.
     */
     PerfEntityMetricCSV entityStatsCsv = (PerfEntityMetricCSV)singleEntityPerfStats;
     /* Retrieve the list of sampled values. */
     List<PerfMetricSeriesCSV> metricsValues = entityStatsCsv.getValue();
     if(metricsValues.isEmpty()) {
          System.out.println("No stats retrieved. " +
                                  "Check whether the virtual machine is powered on.");
          throw new Exception();
     }
    /*
     * Retrieve time interval information (PerfEntityMetricCSV.sampleInfoCSV).
     */
    String csvTimeInfoAboutStats = entityStatsCsv.getSampleInfoCSV();
    /* Print the time and interval information. */
    System.out.println("Collection: interval (seconds),time (yyyy-mm-ddThh:mm:ssZ)");
         System.out.println(csvTimeInfoAboutStats);
         /*
          * Cycle through the PerfMetricSeriesCSV objects. Each object contains
          * statistics for a single counter on the ManagedEntity.
          */
         for(PerfMetricSeriesCSV csv : metricsValues) {
```

```
                /*
                 * Use the counterId to obtain the associated PerfCounterInfo object
                 */
                PerfCounterInfo pci = countersInfoMap.get(csv.getId().getCounterId());
                /* Print out the metadata for the counter. */
                System.out.println("---------------------------------------");
                System.out.println(pci.getGroupInfo().getKey() + "."
                        + pci.getNameInfo().getKey() + "."
                        + pci.getRollupType() + " - "
                        + pci.getUnitInfo().getKey());
                System.out.println("Instance: "+csv.getId().getInstance());
                System.out.println("Values: " + csv.getValue());
            }
        }
```

16.3.2 大规模的性能数据检索

前一个章节中描述了如何检索单一实体的性能数据。当你设计大规模的客户端检索性能数据时，请考虑下列信息以获得更有效地执行。

- 使用 CSV 格式输出。CSV 格式为输出数据提供更多的压缩使其可以过量的保存在元数据上。
- 创建查询声明来引用一组 vSphere 实体。
  - 使用一个 QueryPerf 方法调用每一个实体是不高效的。
  - 使用一个单一的调用 QueryPref 来检索所有的性能数据是不高效的。
  - 一般的规则是，在一个单一的 QueryPref 方法调用中指定 10 到 50 个实体。这是个通用的建议因为你的系统配置可以强加不同的约束。
- 检索统计信息的频率不要超过他们的刷新频率。例如，当你检索数据的时间价格为 20 秒，数据将直到下一个 20 秒的数据收集事件时才会改变。
- 仅在你打算为一个指定的使用一个明确的性能间隔的计数器发送一个查询时使用 QueryAvailablePerfMetric。这个方法将返回你可以在查询中使用的 PerfMetricId 对

象。

在其他所有情况时，为查询创建 PerfMetricId 对象。

- 对于 counterId 属性，使用从 PerformanceManager 计数器对列中取的(PerformanceManager.perfCounter[].key)。
- 对于 instance 属性，指定一个星号(“*”)来检索实例和合计数据或一个零长度的字符串(“”)来仅仅检索合计数据。

16.3.3 使用 QueryPerf 方法作为原始数据

QueryPerf 方法可以作为原始数据来绕过 vCenter 数据库和替代从一个 EXSi 主机检索的性能数据。你可以使用原始数据来获得实时的 20 秒间隔收集的实例数据和 5 分钟间隔的合集数据。

你可以在 vCenter 服务器 2.5 和以后版本中使用原始数据。

Table 16-5. Raw Data Feed

Performance Interval	Description
20-second	ESXi servers collect data for each performance counter every 20 seconds and maintain that data for an hour. When you specify a 20-second interval in the query specification for the `QueryPerf` method (`PerfQuerySpec.intervalId`), the method operates as a raw data feed. The Server ignores the historical interval collection levels and retrieves data for all of the requested counters from the ESXi servers. When you send a query for 20-second instance data, the server returns the most recent data collected for the 20-second interval. The server does not perform additional, unscheduled data collection to satisfy the query.
5-minute	ESXi servers aggregate performance data according to the system-defined performance interval which specifies data collection every 300 seconds. To use a raw data feed for this data, specify the following `PerfQuerySpec` properties in the call to the `QueryPerf` method. ■ `intervalId` – Specify 300 seconds to match the system-defined performance interval. ■ `startTime`/`endTime` – Specify time values within the last 30 minutes of the current time. The `QueryPerf` method checks the performance interval collection level on the vCenter Server. The method returns aggregated statistics for performance counters that specify a collection level (`PerfCounterInfo.level`) at or below the vCenter Server performance interval for the 300 second sampling period (`PerfInterval.level`). For example, if the vCenter Server performance interval is set to level one, and your query specification requests only performance counters that specify level four, the `QueryPerf` method will not return any data.

16.3.4 查询方法比较

下表展示了性能查询方法的比较。

Table 16-6. Performance Query Methods

Method	Notes
QueryPerf	■ Specify an array of PerfQuerySpec objects. ■ An unset PerfQuerySpec.metricId property produces results for all counters defined for PerfQuerySpec.entity. ■ PerfQuerySpec.maxSample is ignored for historical statistics. You can use this method to retrieve historical statistics; you can also use it as a raw data feed. For information about retrieving the raw data collected on ESXi servers, see "Using the QueryPerf Method as a Raw Data Feed" on page 190.
QueryPerfComposite	■ Method works only at the host level. You can use a single call to the QueryPerfComposite method to retrieve performance data for a host and its virtual machines. ■ Specify a single PerfQuerySpec object. ■ You must specify a list of performance metrics to identify the data to be retrieved (PerfQuerySpec.metricId). ■ You cannot specify PerfQuerySpec.maxSample. This method is designed for efficient client-server communications. QueryPerfComposite usually generates less network traffic than QueryPerf because it returns a large-grained object, a PerfCompositeMetric data object, that contains all the data.

16.3.5 检索总计性性能数据

你可以获得接近实时的性能或利用率的总计性信息不通过使用 PerfomanceManager 方法。vSphere 服务器为主机(HostListSummaryQuickStats)，虚拟机(VirtualMachineQuickStats)，和资源池(ResourcePoolQuickStats)维持”quick.stats”数据对象。

16.4 性能计数器元数据

性能计数器由系统资源组组织管理，例如性能计数器组是内存，CPU，硬盘。计数器组和指定的使用在任何 vSphere 服务器上的计数器依赖于服务器配置。vSphere API 引用为每一个计数器组包含一个表，这个表包括了计数器名称，被收集的统计信息类型，度量单位，级别等等。

PerformanceManager.perfCounter 属性是一个 PerfCounterInfo 数据对象的队列。每一个对象为收集的数据提供元数据。一个 PerfCounterInfo 对象有一个唯一的 key，计数器 ID。实际运行时被收集性能数据有这个计数器 ID 确定。下表列出了 PerfCounterInfo 的属性。

Table 16-7. PerfCounterInfo Data Object Properties

Property	Description
groupInfo	Name of the resource group to which this counter belongs, such as disk, cpu, or memory.
key	Unique integer that identifies the counter. Also called the counter ID. The value is unique and it is not static—it might, for example, change between system reboots. The counter key on an ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESXi system. However, the system maps the keys from ESXi to vCenter Server systems automatically.
level	Number from 1 to 4 that identifies the level at which data values for this counter are aggregated.
nameInfo	Descriptive name for the counter. The name component of a fully qualified counter name, for example "granted" is the nameInfo property for the mem.granted.AVERAGE counter.
rollupType	Indicates how multiple samples of a counter are transformed into a single statistical value. Examples of rollup types are average, summation, and minimum. No conversion of values occurs for counters that specify absolute values, such as the total number of seconds that the system has been running continuously since startup. The PerfSummaryType is an enumeration containing valid constants for this property.
statsType	Type of statistical data that the value represents over the course of the interval, such as an average, a rate, the minimum value, and so on. The PerfStatsType is an enumeration containing valid constants for this property.
unitInfo	Unit of measure, such as megahertz, kilobytes, kilobytes per second, and so on. The ElementDescription's key property is populated using one of the constants available in the PerformanceManagerUnit enumeration.

16.5 性能时间间隔

PerformanceManager 定义了性能的时间间隔明确了两个收集时间的间隔期间，多少数据将被收集，收集的数据将保持多久。

- 一个 ESXi 服务器有一个内置的性能时间间隔来从计数器实例中产生非连续的数据例如每 20 秒钟。服务器将保持这个实例的数据一个小时。
- 额外的数据收集由历史性能时间间隔来确定，其从计数器实例产生合计数据来符合个别的时间间隔。

PerformanceManager.historicalInterval 属性是一个 PerfInterval 对象的队列。下表列出了 PerfInterval 的属性。

Table 16-8. PerfInterval Properties

Property	Description
samplingPeriod	Number of seconds for the interval. You can modify this property on a vCenter Server only.
length	Period of time for which the server will save the data that it collects. You can modify this property on a vCenter Server only.

level	Level at which the Server collects data. The interval level corresponds to the performance counter level (`PerfCounterInfo.level`). The Server will collect data for all counters with levels that match `PerfInterval.level`, and for all counters with levels lower than `PerfInterval.level`. You can modify this property on a vCenter Server only.
enable	Enable/disable performance data collection. You can modify this property on a vCenter Server only.
key	Unique identifier for the interval. You cannot modify this property.
name	Label for the historical interval; one of the following strings: ■ "Past Day" ■ "Past Week" ■ "Past Month" ■ "Past Year" The `PerformanceManager` uses the `samplingPeriod`, `level`, and `length` properties to determine its collection behavior. It does not interpret the name string. You cannot modify this property.

16.5.1 ESXi 服务器性能时间间隔

一个 ESXi 服务器为每一个性能计数器每 20 秒收集一次性能数据。每一个 ESXi 上的 PerformanceManager.historicalInterval 队列包含一个单一的，只读的的每 5 分钟收集一次统计数据的 PerfInterval 对象。你不能直接从一个 ESXi 服务器上直接检索 5 分钟的统计数据。你可以使用一个管理这个 ESXi 服务器的 vCenter 服务器连接来获得 5 分钟的统计数据。下表显示了在 ESXi 服务器上的历史间隔属性值。你不能编辑这个性能时间间隔。

Table 16-9. Values of PerfInterval Data Object from an ESXi System

Property	Value	Description
`key`	`1`	Numeric identifier for the `PerfInterval`.
`name`	`PastDay`	Name of the `PerfInterval`.
`samplingPeriod`	`300`	Time interval between data sampling events.
`length`	`129600`	Number of seconds that statistics associated with the interval are kept by the vCenter Server.
`enabled`	`true`	This `PerfInterval` is enabled on the system.
`level`	`null`	Statistics collection level. For an ESXi system, this property is `null`. The `PerfInterval` object on an ESXi system defines the baseline interval.

16.5.2 vCenter 服务器性能时间间隔

一个 vCenter 服务器汇总它管理的 ESXi 系统上的性能数据。汇总的数据量依赖于 vCenter 服务器上的级别设定。级别设定反应在 vCenter 服务器系统上的 PerforManager.historicalInterval 属性上。historicalInterval 是一个定义了 4 个不同级别的 PerfInterval 数据对象队列，1 到 4。

表 16-10 列出了一个 vCenter 服务器系统上性能时间间隔的默认值。

Table 16-10. Values of PerfInterval Data Objects from a vCenter Server System

Key	Name	Sampling Period	Length	Enabled	Level
1	Past Day	300	86400	TRUE	1
2	Past Week	1800	604800	TRUE	1
3	Past Month	7200	2592000	TRUE	1
4	Past Year	86400	31536000	TRUE	1

默认的，为 4 个时间间隔的收集级别设置为 1。使用默认级别，vCenter 服务器将使用指定的收集级别 1 来收集所有性能计数器。使用默认长度值，vCenter 服务器将未收集的数据保存如下时间：

- 过去天的 5 分钟取样数据。
- 过去周的 20 分钟取样数据。
- 过去月的 2 小时取样数据。
- 过去年的 1 天取样数据。

超过 1 年的数据被从 vCenter 服务器的数据库中清除。

16.6 vSphere 性能和数据存储

本章提供了关于修改关于 PerformanceManager 和 vSphere 服务器性能数据收集和存储的操作信息。

16.6.1 编辑历史时间间隔

改变 vCenter 性能时间间隔是全局的并且影响系统里面的所有实体。VMware 建议你不要编辑历史时间间隔。在 PerformanceManager.historicalInterval 队列里的 PerfInterval 数据对象都是相关联的。对性能时间间隔的修改影响整个系统并且可能带来问题。

假如你必须修改一个性能时间间隔，使用 PerformanceManager.UpdatePerfInterval 方法并且按照下列指导原则：

- 性能数据保留时间 (PerfInterval.length) 必须是收集时间间隔的倍数 (PerfInterval.samplingPeriod)。
- 在每一个时间间隔的性能数据保留长度必须比它前面的有所增加。对于每一个连续的性能时间间隔的 PerfInterval.length 值必须比在历史时间间隔队列里面前一个时间间隔的 length 值大。

- 你不能在 ESXi 系统上编辑 PerfInterval.samplingPeriod 属性值。

16.6.2 编辑性能计数器收集级别

PerformanceManager 提供 UpdateCounterLevelMapping 方法来改变非连续的性能计数器(PerfCounterInfo.level)收集级别。认真考虑使用 UpdateCounterLevelMapping 方法对于性能和存储的影响。假如你使用这个方法，你有可能引起数据收集和存储上显著的增加，并且在性能上的下降。vCenter 服务器性能和数据库存储需求建立在为性能时间间隔定义的收集级别(PerformanceManager.historicalInterval)和为非连续的性能计数器指定的收集级别(PerCounterInfo.level)。

16.6.2.1 性能计数器数据收集

vSphere 为性能计数器定义了 4 个数据收集级别。每一个性能计数器为收集指定一个级别。历史性能时间间隔为特殊的收集级别(PerformanceManager.historicalInterval)定义了取样期间和长度。

对于一个性能计数器收集的数据总数依赖于性能时间间隔和计数器被定义在的实体类型。例如，一个数据存储计数器譬如 datastorelops(合计在数据存储上的 IO 操作数)将产生一个符合在一个主机上的数据存储的数的数据集。假如一个 vCenter 服务器管理了具有大量数据存储的大量主机，服务器将收集到海量的数据。

vCenter 服务器上还有相对小的其他计数器。例如，内存计数器被每个虚拟机和主机当作单独的计数器。

16.6.2.2 性能计数器数据存储

性能时间间隔收集级别(PerfInterval.level)为 vCenter 服务器存储性能数据定义了一组计数器。服务器将为指定收集级别和低级别的计数器保存数据。

默认，所有性能时间间隔指定收集级别为 1。使用默认设置，vCenter 服务器为所有设置为收集级别为 1 的计数器保存数据在 vCenter 数据库里。不保存级别 2 到 4 的数据。

16.6.2.3 性能管理方法互动

你可以使用 UpdateCounterLevelMapping 方法为个别的计数器来改变收集级别。你也可

以使用 UpdatePerfLevel 方法来为系统定义的性能时间间隔改变收集级别。这些方法可能会引起数据的大量增加。

- 默认系统定义的性能时间间隔使用收集级别 1， 为所有级别为 1 的计数器保存数据。假如你使用 UpdateCounterLevelMapping 方法来改变性能计数器的收集级别到级别 1。你将导致存储的性能数据量增加。
- 假如你使用 UpdatePerfLevel 方法来增加系统定义的性能时间间隔收集级别，你将导致存储的性能数据量增加。

为了恢复计数器级别到默认级别，请使用 ResetCounterLevelMapping 方法。

16.6.2.4 vSphere 客户端性能统计管理

vSphere 客户端显示性能管理历史时间间隔收集级别在 vCenter 管理统计里显示。vSphere 客户端也显示在显示级别的数据收集所需要的存储量的预测。假如个别的计数器级别是通过 vSphere API(UpdateCounterLevelMapping 方法)编辑的，vSphere 客户端将显示一个可编辑的预测。无论怎样，vSphere 客户端不能检测已经调用过的这个方法和不能显示当前级别对于个别的计数器。假如你看到预测的存储有显著的增加，应该意识到有人可能通过 vSphere API 编辑了数据收集。

17、诊断和处理故障

vSphere 包含几个日志，你可以访问和自定义。你也可以为解决问题使用 DiagnosticManager 服务接口。

17.1 处理故障最佳实践

为了系统的处理故障和解决问题，你需要跟踪你的操作。下列是通过你的客户端应用来解决问题的指导原则。

- 不要在同一时间改变超过一件事情，记录下每次改变和它的结果。试着隔离这个问题：是不是本地的，客户端的？服务器产生的错误信息？客户端和服务器间的网络问题？
- 使用您的编程语言的日志能力来捕获客户端运行时信息。查看 Log.cs 示例程序。

- C#客户端日志例子：\SDK\vsphere-ws\dotnet\cs\samples\AppUtil\Log.cs

- 使用下列 VMware 工具来分析和帮助解决问题。
  - vSphere Web Services API。DiagnosticManager 服务接口允许你从服务器日志文件中获取信息，创建一个包含所有系统日志文件和所有服务器配置信息的诊断包。vSphere 客户端和 MOB 提供图形化和 Web 来访问 DiagnosticManager。PerformanceManager 支持瓶颈查询。
  - Managed Object Browser(MOB)。MOB 提供支持直接访问运行时的服务器端对象。你可以使用 MOB 来浏览对象层级，获得属性值，调用方法。
  - VMware vSphere Client。vSphere 客户端允许你检查 ESX/ESXi，vCenter 服务器，和虚拟机的日志文件，并且改变日志级别设置。使用 vSphere Client 菜单命令来创建概述配置信息，性能，和其他细节的报告，并且导出诊断包。vSphere 客户端维护自己的本地日志文件。

17.2 配置文件和日志文件概述

ESX/ESXi 和 vCenter 服务器配置文件控制着系统行为。大多数配置文件在安装时设置，但是可以在安装后修改。日志文件捕获由内核，不同的子系统，和服务产生的信息。ESX/ESXi 和 vCenter 服务器服务维护独立的日志文件。表 A-1 列出了日志文件和报告，他们的位置和相关的配置文件。

Table A-1. Server and System Logs

Description	Log Location	Filename or Names	Configuration File
ESX/ESXi service log	/var/log/vmware/	hostd.log [hostd-0.log, ...hostd-9.log]	config.xml
vCenter Server agent log	/var/log/vmware/vpx/	vpxa.log	
Virtual machine kernel core file	/root/	vmkernel-core.<date> vmkernel-log.<date>	syslog.conf, logrotate.conf, various other
syslogd log	/var/log/	messages [messages.1,... messages.4]	syslog.conf, logrotate.conf
Service console availability report	/var/log/	vmkernel [vmkernel.1, ... vmkernel.8]	syslog.conf, logrotate.conf
VMkernel messages, alerts, and availability reports	/var/log/vmkernel		syslog.conf, logrotate.conf
VMkernel warning	/var/log/	vmkwarning [vmkwarning.1 ... 4 for history]	syslog.conf, logrotate.conf
Virtual machine log file	vmfs/volume/<vm_name>	vmware.log	<vm_name>/<vm_name>.vmx

对于开发者，下列文件有重大意义：

- hostd.log – 主机守护日志。当设置为 trivia 级别时可以被当作一个 SOAP 监控器使用。
- vpxa.log – 代理日志文件，在每台被管理的 ESX/ESXi 系统上。
- vmware.log – 虚拟机日志。

17.2.1 ESX/ESXi 日志文件

ESX/ESXi 日志(hostd.log)捕捉变化的特征和细节信息，依赖于日志级别。每一个到服务器的请求将被记录。你可以通过 vSphere 客户端浏览这个文件。例子 A-1 中显示了未加工的 ESX/ESXi(hostd)日志文件格式。

Example A-1. Sample ESX/ESXi Log (hostd.log) Data

```
...
[2008-05-07 09:50:04.857 'SOAP' 2260 trivia] Received soap response from
              [TCP:myservername.vmware.com:443]: GetInterfaceVersion
[2008-05-07 09:50:04.857 'ClientConnection' 2260 info] UFAD interface version is
              vmware-converter-4.0.0
[2008-05-07 09:50:04.857 'SOAP' 2260 trivia] Sending soap request to
              [TCP:myservername.eng.vmware.com:443]: logout
[2008-05-07 09:50:04.857 'ProxySvc Req00588' 3136 trivia] Client HTTP stream read error
[2008-05-07 09:50:04.872 'ProxySvc Req00612' 3136 trivia] Request header:
POST /vmc/sdk HTTP/1.1
User-Agent: VMware-client
Content-Length: 435
Content-Type: text/xml; charset=utf-8
```

```
Cookie: vmware_soap_session="F127B435-56C7-4580-BAC4-3034DA1E67B6"; $Path=/
Host: myservername.vmware.com


[2008-05-07 09:50:04.872 'ProxySvc Req00588' 3816 trivia] Closed
[2008-05-07 09:50:08.450 'App' 3560 verbose] [VpxdHeartbeat] Invalid heartbeat from 10.17.218.46
[2008-05-07 09:50:10.013 'App' 3560 verbose] [VpxdHeartbeat] Queuing 10.17.218.45:829 (host-55)
[2008-05-07 09:50:10.013 'App' 1928 verbose] [HeartbeatHandler]
               50208862-2752-d94c-2a73-fa2ec9e38ecc:829 (host-55)
```

17.2.2 虚拟机日志文件

每一个运行的虚拟机拥有自己的日志文件，vmware.log，存储在 VMFS 卷上。默认的，日志文件当虚拟机开机的时候交替轮换，文件循环式可配置的。

- ESX/ESXi 保留 6 个日志文件在每一个电力循环中(默认)循环或在配置的文件大小中循环。
- ESX/ESXi 可以配置来保留指定的日志个数。当这个限制达到时，最老的文件将被删除。
- VMware 建议一个日志文件大小为 500KB。
- 由 VMware Tools 产生的信息被分开记录。

Example A-2. VMkernel Availability Report

```
Availability Report for <servername>
Feb 27, 2008 - May  7, 2008

Availability: 99.949%
    Total time: 69 days, 15 hours
    Uptime: 69 days, 14 hours
    Downtime: 51 minutes

Note: Downtime is any time the system isn't capable of running
Virtual Machines.  This includes reboots, crashes, configuration and running linux

Downtime Analysis:
    0.1% (51 minutes) downtime caused by:
    13.1% (6 minutes) scheduled downtime
    86.9% (44 minutes) unscheduled downtime

Reasons for scheduled downtime:
    84.9% server rebooting (1 instance)
    9.4% VMkernel unloaded (1 instance)
    5.7% server booting (3 instances)

Reasons for unscheduled downtime:
    100.0% unknown (powerfail / reset?) (1 instance)

Stats:
    Current uptime: 8 days, 11 hours
    Longest uptime: 61 days, 2 hours
    Shortest uptime: 38 minutes
    Average uptime: 23 days, 4 hours
    Longest downtime: 44 minutes
    Shortest downtime: 7 seconds
    Average downtime: 8 minutes
    Maximum VMs Sampled:  1
    Average VMs Sampled: 0.94

Server Information:   Number of CPUs:  4 logical 4 cores
     2 packages,  Intel(R) Xeon(R) CPU            5150  @ 2.66GHz
       Installed Memory:     2096416 kB
       Current Build:  78591
Report generated Wed May  7 04:02:04 PDT 2008
```

17.2.3 vCenter 服务器日志

vCenter 服务器日志文件默认放置在 Documents and Settings 下的安装软件所使用的账号子目录下。例如：

c:\Documents and Settings\Administrator\Local Settings\Application Data\VMware\

注意：VMware 建议为 vCenter 服务器安装建立专门的账户。

默认，日志文件是隐藏文件。

17.3 编辑日志级别来获得详细的信息

捕捉的变换信息总量依赖于级别设置。

Table A-2. Log Level Settings

Log Level Setting	Description
None	Disables logging.
Error	Logging limited to error messages.
Warning	Error messages plus warning messages are logged.
Info	Default setting on ESX/ESXi and vCenter Server systems. Errors, warnings, plus informational messages about normal operations are logged. Acceptable for production environments.
Verbose	Can facilitate troubleshooting and debugging. Not recommended for production environments.
Trivia	Extended verbose logging. Provides complete detail, including content of all SOAP messages between client and server. Use for debugging and to facilitate client application development only. Not recommended for production environments.

例如，运行在 ESX/ESXi 系统上的 hostd 服务有一个默认日志级别设置为 info。vCenter 服务器日志可以通过 vSphere Client 来控制。

17.3.1 在 ESX/ESXi 系统上设置日志级别

ESX/ESXi 日志由 config.xml 文件中的一个设置来控制，文件位于/etc/vmware/hostd 子目录下。

Example A-3. ESX/ESXi Config.xml File Excerpt Showing Default Log Level Setting

```
<config>
<vmacore>
<threadPool>
<MaxFdsPerThread>2048</MaxFdsPerThread>
</threadPool>
<ssl>
<doVersionCheck> false </doVersionCheck>
</ssl>
<vmdb>
<maxConnectionCount>8</maxConnectionCount>
</vmdb>
<loadPlugins> true </loadPlugins>
</vmacore>
<workingDir> /var/log/vmware/ </workingDir>
<log>
<directory>/var/log/vmware/</directory>
<name>hostd</name>
<outputToConsole>false</outputToConsole>
<level>info</level>
</log>
...
</config>
```

默认，日志级别设置为 info。假如你在开发中运行出错，你可以设置日志级别为 verbose，或 trivia 来获得 SOAP 信息内容以便处理问题。

下列过程是是对于 ESX 系统的。在 ESXi 上，使用 vifs 来移动 config.xml 文件到你可以编辑它的服务器。

为在 ESX 系统上的 hostd 改变日志级别

1. 连接到 ESX 系统上，使用 putty 或其他安全 shell。
2. 打开 config.xml 文件，位于/etc/vmware/hostd。
3. 改变<level>info</level>成为<level>trivia</level>，保存并关闭文件。
4. 导航到 init.d 目录然后重新启动 host 代理。

   cd /etc/init.d

   ./mgmt.-vmware restart

服务重新启动后，新的日志级别将生效。

17.3.2 产生日志

假如你连接到 ESX，你可以使用 tail 命令来明确的创建一个日志文件来捕获关于动作的细节信息。例如，你可以使用 vSphere 客户端来创建一个新的虚拟机然后使用日志的内容作为一个模型来创建自己的代码。

开始日志进程并且捕获内容到文件中。

1. 导航到 hostd.log 文件的位置：

   cd /var/log/vmware

2. 运行 tail 命令，传入一个文件名，捕捉输出的文件：

   tail –f hostd.log > yourfilenamehere

3. 使用 vSphere 客户端执行无论什么在你代码中不好实现的模型操作。例如，创建一个新的虚拟机并且当操作完成时使用 Ctrl-C 停止 tail 进程。

这个文件包含在执行过程中由 hostd 发送和接受的 SOAP 信息内容和其他日志信息。

17.3.3 在 vCenter 服务器系统上设置日志级别

为了在 vCenter 服务器上更改日志级别设置，你必须使用 vSphere 客户端。

使用 VMware vSphere 客户端为 vCenter 服务器设置日志级别

1. 登陆到 vSphere 客户端并且连接到 vCenter 服务器实例。
2. 选择 **Administration** 并且点击 **Server Settings > Logging Options**。
3. 选择从弹出菜单中 **Trivia** 并且点击 **OK**。

17.4 使用 DianosticManager

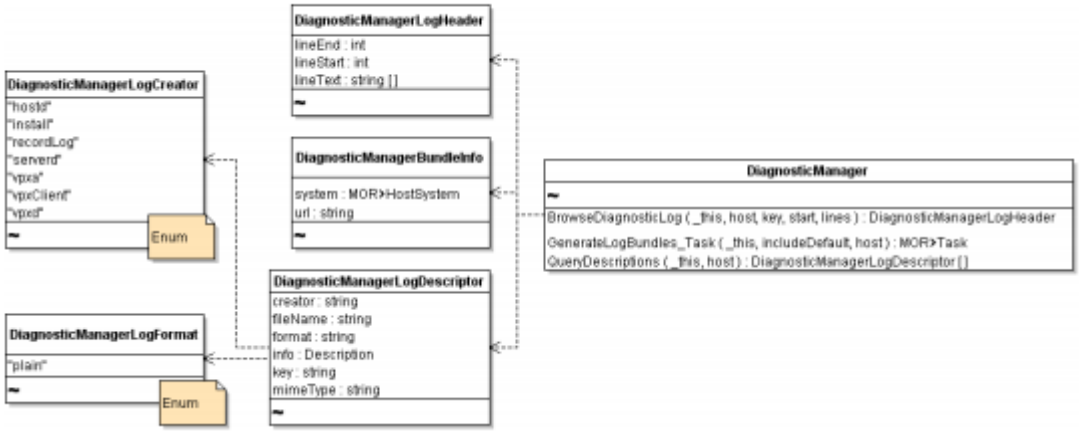
vSphere API 提供访问 DianosticManager，这个服务接口可以从日志文件获取信息和创建诊断包。日志存放点有配置设定决定。例如 info，trivia，等等。

DiagnosticManager 是一个工作在服务器端的 managed 对象，而不是每一个 session 上。DiagnosticManager 没有属性，但是提供下列任务的操作。

- 获取日志的信息和如何定义他们。
- 产生可以送给 VMware 支持分析的诊断包。

图 A-1 显示一个 UML DiagnosticManager 的类图，可在 ESX/ESXi 和 vCenter 服务器系统上使用。

Figure A-1. DiagnosticManager Managed Object and Associated Data Objects



如图 A-1 中显示，DiagnosticManager 支持下列方法：

- BrowseDiagnosticLog
- GenerateLogBundleTask
- QueryDescriptions

DiagnosticManagerLogDescriptor.creator 属性包含这个日志的 creator，其是控制一个指定日志的系统或子系统。

creator 值取自于 DiagnosticManagerLogCreator 枚举类型。表 A-3 列出了所有当前在 DiagnosticManagerLogCreator 枚举类型中的可用的字符串值，可以定位 DiagnosticManagerLogDecriptor 数据对象的 creator 属性。

Table A-3. DiagnosticManagerLogCreator Enumeration

Name	Description
hostd	Host daemon
install	Installation
recordLog	System record log
serverd	Host server agent
vpxa	vCenter agent
vpxClient	vSphere Client
vpxd	vCenter service

Table A-4. ESX/ESXi Sample DiagnosticManager Log Descriptor Values

Creator	File Name	Format	Info.label	Info.summary	Key	Mime Type
hostd	/var/log/vmware/hostd.log	plain	ESX Log	ESX log in plain format	hostd	text/plain
hostd	/var/log/messages	plain	ESX Log	ESX log in plain format	messages	text/plain
hostd	/var/log/vmkernel	plain	ESX Log	ESX log in plain format	vmkernel	text/plain
hostd	/var/log/vmksummary.txt	plain	ESX Log	ESX log in plain format	vmksummary	text/plain
hostd	/var/log/vmkwarning	plain	ESX Log	ESX log in plain format	vmkwarning	text/plain
vpxa	/var/log/vmware/vpx/vpxa.log	plain	VirtualCenterAgent Log	VirtualCenter agent log in plain format	vpxa	text/plain

17.5 使用 MOB 来查看 DiagnosticManager

你可以使用 MOB 来访问 DiagnosticManager。

查看 DiagnosticManager

1. 在一个 Web 浏览器中输入 MOB URL(https//hostname.yourcompany.com/mob)开始。
2. 在 ServiceContent 数据对象中，在 Value 栏点击链接(ha-diagnosticmanger 或 DiagMgr)来查看 diagnosticManager 属性，导航到系统的 DiagnosticManager。
   - 对于 ESX/ESXi，ha-diagnosticsmanager 是 managed 对象 ID。
   - 对于 vCenter 服务器，DiagMgr 通常是 managed 对象 ID。
3. 点击 refernce 来显示在 MOB 中 DiagnosticManager 的 managed 对象引用。

   DiagnosticManager 提供三个操作来允许你获得当前在日志文件中和日志文件内容中可用的描述信息。

   因为 DiagnosticManager 可以跟踪多个 ESX/ESXi 系统，你可以使用

QueryDescriptions 操作来返回所有主机使用的 keys 的名称。从这个队列中，选择你想要获得日志文件的主机的 key。

4. 在 QueryDescriptions 上，点击 **Invoke Method** 链接。

   vCenter 服务器系统返回已选择作为 DiagnosticManagerLogHeader 的 lineTest 属性的字符串队列中的主机的日志内容。

这个方式通过 MOB 返回的字符串队列是日志文件的内容。包含在日志文件中的内容和通过下列其他途径获得的内容是一致的。

- 在 vSphere 客户端显示的。
- 通过 DiagnosticManager.GernerateLogBundles_Task 方法创建的诊断包中。
- 在可用的 hostd.log 文件中。
- 返回给你编写的客户端。

17.6 产生诊断包

通常，客户在 VMware 技术支持的要求下创建诊断包。诊断包也允许开发人员在一个完整的包中快速获得所有配置信息和日志文件。

产生的压缩文件被打包成一个文件，如下格式：

<fqdn-hostname>-esxsupport-yyyy-mm-dd@hh-mm-ss.taz

使用 vSphere 客户端导出诊断数据

1. 启动 vSphere 客户端并链接到 ESX/ESXi 或 vCenter 服务器系统。
2. 选择 Adminstration > Export Diagnostic Data。
   - 对于 vCenter 服务器系统，你可以指定你想要导出的主机的日志和要保存它的位置。
   - 对于 ESX/ESXi 主机系统，你可以为诊断包指定位置。
3. 点击 OK。

使用命令行来收集 ESX 日志文件

1. 使用 putty 或其他 SSH 工具来链接到服务控制台。在 ESXi 系统上，假如 VMware 技术支持要求你使用不支持的 shell，使用它。
2. 导航到/usr/bin 子目录。
3. 运行 mv-support 脚本：

/usr/bin/vm-support

这个脚本收集所有有关的 ESX 系统日志文件，配置文件，和其他服务器细节到一个压缩的文件中(诊断包)，文件名称中包含日期和时间。

18、Managed 对象浏览

Managed 对象浏览(MOB)是一个图形接口允许你在服务器上定位对象和调用方法。你通过 MOB 做的任何改变将在服务器上生效。

本章解释了如何使用 MOB。例子调用 PerformanceManager 查询方法来演示如何使用 MOB 来传入原始数据类型，队列，和复合数据类型(数据对象，包括 managed 对象引用)。

18.1 使用 MOB 来查看对象模型

Managed 对象浏览器，或 MOB，对于 ESX/ESXi 和 vCenter 服务器系统来说是一个基于 Web 服务器的应用。MOB 让你检查存在于服务器上的对象和通过点击来在活动的对象层中导航。MOB 用实际运行的信息更新浏览器，例如，属性名称。

注意：虽然在它的名字中有”browser”字眼，MOB 不是只读的。MOB 允许你通过点击 **InvokeMethod** 连接关联的方法在服务器上做改变。

18.1.1 访问 MOB

MOB 运行在一个 web 浏览器中，可以通过使用 ESX/ESXi 或 vCenter 服务器系统的完整的域名或 IP 地址来访问。

1. 打开一个 Web 浏览器。
2. 键入 ESX/ESXi 或 vCenter 服务器系统的完整的域名(或 IP 地址)：

   https://hostname.yourcompany.com/mob
3. 键入用户账号和密码。

   假如关于 SSL 证书的警告信息出现，你可以忽略他们并且继续登陆到 MOB，假如 VMware 是认证授权的和你不在一个生产环境中。

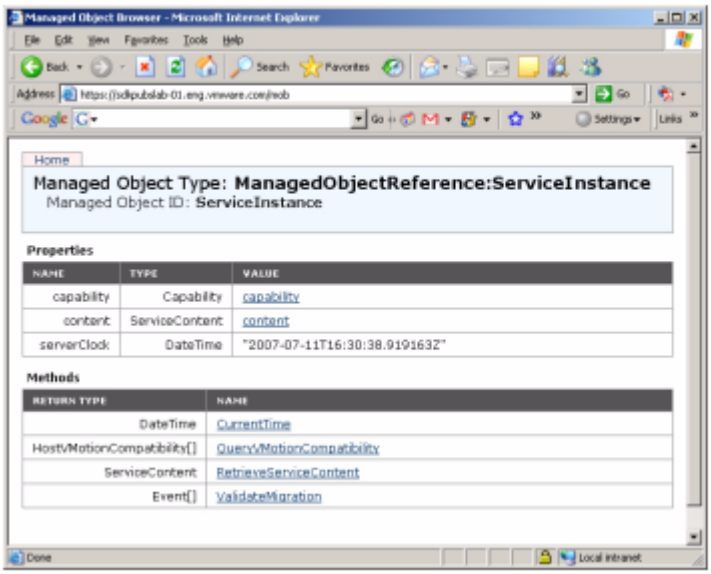
MOB 显示了对象模型的底层架构。

18.1.2 使用 MOB 导航 VMware 架构对象模型

当成功连接到 MOB 时，浏览器显示 ServiceInstance 的 managed 对象引用(请看图 B-1)。客户端应用不能直接使用 managed 对象引用，但是通过使用创建的 ManagedObjectReference 数据对象实例可以和服务器端 managed 对象相互影响。

页面列出了一个 ServiceInstance 对象可用的属性和方法。ServiceInstance 方法和属性提供访问服务器上全部的服务和清单对象。

Figure B-1. Managed Object Browser Connected to a VirtualCenter Server System



MOB 让你可以通过查找属性和他们的值来检查对象间的关系，并且进入到对象。为了浏览服务器上的对象，点击 **Value** 栏的连接来导航到显示对象页面。

例如，为了找到更多关于 ServiceContent 的信息，点击 **content** 连接来显示 ServiceContent 数据对象实例。

18.2 使用 MOB 来调用方法

你可以使用 MOB 如下来调用方法：

1. 在对象显示页面，点击方法的名称。

   一个浏览器窗口显示了关于参数名称和类型的信息并且允许你指定参数值。

2. 指定参数值，为方法使用适当的参数类型，然后点击 **Invoke Method**。

余下的章节讨论如何传入不同的参数到 MOB

18.2.1 传入原始的数据类型给方法

vSphere Web Services SDK 数据类型使用 XML 标记定义在 WSDL 里。原始数据类型由 xsd 命名空间指定。例如，一个属性的一个字符串值定义如数据类型 xsd:string。以纯文本形式在 MOB 中输入一个原始值，没有引号或其他标记。例如，想要输入一个值为 10 的整形，在字段中键入 10。

为了服务器上级别为 4 的性能计数器信息，在 PerformanceManager.QueryPerfCounterByLevel 方法的级别字段中输入(这个方法只能在 vCenter 服务器 PerformanceManager API 可用，不支持 ESX/ESXi 系统)。

在查询的回应中，PerfCounterInfo 数据对象队列和内嵌对象，由服务器上的数值填充，显示在 Web 浏览器上。

18.2.2 传入原始数据队列给方法

对于一个队列，使用参数名称作为属性名称。例如 PerformanceManager.QueryPerfCount 方法需要一个整形队列作为 counterId 参数，如下：

<counterId>58</counterId><counterId>65603</counterId><counterId>65604</counterId>

即使你想要提交一个 single 值给一个 single 队列元素，你必须按这种方式使参数名包在值周围。

18.2.3 传入复合结构给方法

对于复合数据类型，键入定义在 WSDL 由 XML Schema 定义的值。你可以通过在 vSphere API Reference 中使用 **Show WSDL types definition** 连接来获得 WSDL 定义信息。每一个数据类型有一个相关的连接。

18.2.3.1 简单内容

ManagedObjectReference 是最长用的当作参数传入给服务器的数据对象类型。例如，MOB 为 PerformanceManager.QueryPerfProviderSummary 方法显示了需求一个 single 参数，一个你想要获得 PerfProviderSummary 对象的实体的 managed 对象引用(一个 ManagedObjectReference 实例)。

为 ManagedObjectReference 类型使用 vSphere API Reference，你可也通过文档底部的 **Show WSDL type definition** 连接获得概述信息。

Example B-1. XML Schema Definition of ManagedObjectReference Data Object

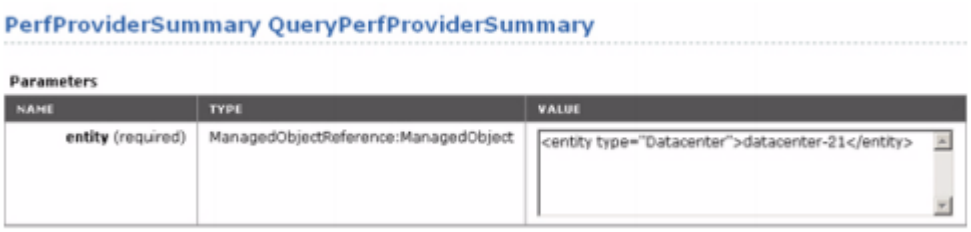
```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
             name="ManagedObjectReference">
    <simpleContent>
        <extension base="xsd:string">
            <attribute name="type" type="xsd:string"/>
        </extension>
    </simpleContent>
</complexType>
```

例子 B-1 展示了一个定义为<SimpleContent>元素的 managed 对象引用，它由一个声明了属性 type 和它关联值的字符串组成。使用这个信息来通过 MOB 中的参数名称替换 typel 来构建合适的结构，设置需要的值，并且提交整个 MOB 字段。(Datecenter 的值显示在 MOB 中)

<entity type=”Datacenter”>datacenter-21</entity>

图 B-2 展示了使用例子 B-1 中的定义为一个目标数据中心的 PerformanceManager.QueryPerfProviderSummary 方法指定 managed 对象引用。

Figure B-2. Using the MOB to Pass Complex Types to a Method



另外一个例子，VirtualMachine.CloneVM_Task 方法需要一个 folder。在这个情况下，参数被定义为一个指定 Folder 对象的 managed 对象引用。使用如例子 B-1 中一样的定义，结果如下：

<folder type=”Folder”>folder-87</folder>

尽管两个例子提交一个 ManagedObjectReference 到 MOB，但每一个方法需要的参数名被特别指定 (PerformanceManager.QueryPerfProviderSummary 方法的实体类型，VirtualMachine.CloneVM_Task 方法的 folder 类型)。

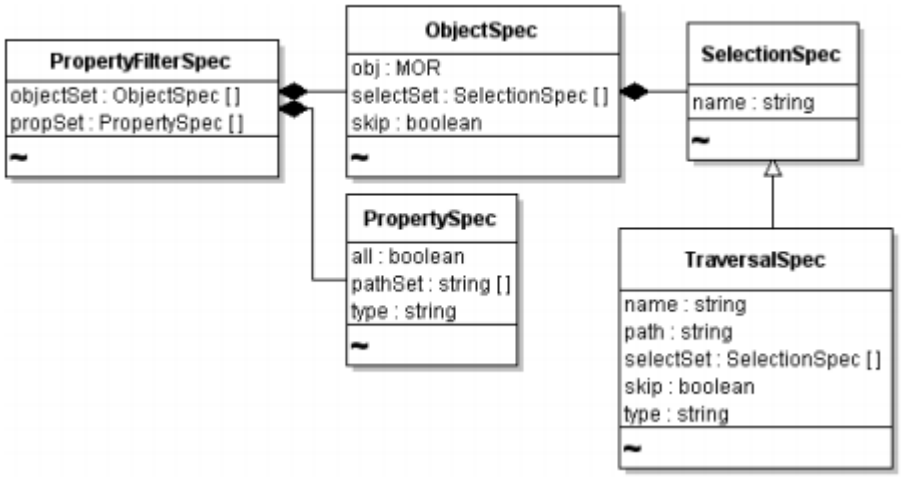
18.2.3.2 复合内容

方法调用需要的很多数据对象有<complexContent> XML 标记组成，可以包含许多其他元素。

例如，PropertyCollector.CreateFilter 有一个 spec 参数其必须在方法调用前被定义好。spec 参数被定义为一个 PropertyFilterSpec 实例。

图 B-3 展示了 PropertyFilterSpec 包含的几个数据对象间的关系。

Figure B-3. PropertyFilterSpec and Associated Data Objects



为了提交复合数据结构给 MOB，浏览 vSphere API Reference。找到 PropertyFilterSpec 数据对象。找到 Show WSDL type definition 连接，点击显示 XML 结构定义(参看例子 B-2)。

例子 B-2 展示了 PropertyFilterSpec 数据对象是一个<complexContent>元素其扩展了 DynamicData 类，具有两个额外的属性 propSet(PropertySpec 类型)和 objectSet(ObjectSpec 类型)一个序列。

Example B-2. XML Schema Definition of PropertyFilterSpec Data Object Type

```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
               name="PropertyFilterSpec">
    <complexContent>
        <extension base="vim25:DynamicData">
            <sequence>
                <element name="propSet" type="vim25:PropertySpec" maxOccurs="unbounded"/>
                <element name="objectSet" type="vim25:ObjectSpec" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

因为两个元素被定义为一个序列，他们必须按列出的顺序存在。为了得到 propSet 和 objectSpec 的定义信息，你必须在 vSphere API Reference 中更进一步导航。例子 B-3 仅仅展示了和 PropertySpec 相关的 XML 标记定义。minOccurs=”0”属性意味着该元素不是必须的，maxOccurs=”unbounded”属性意味着该元素可以是任何大小的队列。(当 minOccurs 没有设置时，默认是 1，意味着需要一个实例。)

Example B-3. XML Schema Extract for PropertySpec

```
<sequence>
    <element name="type" type="xsd:string"/>
    <element name="all" type="xsd:boolean" minOccurs="0"/>
    <element name="pathSet" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
```

在 vSphere API Reference 中导航到 ObjectSpec 定义。例子 B-4 展示。

Example B-4. ObjectSpec Definition as XML Schema

```
...
<sequence>
    <element name="obj" type="vim25:ManagedObjectReference"/>
    <element name="skip" type="xsd:boolean" minOccurs="0"/>
    <element name="selectSet" type="vim25:SelectionSpec" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
...
```

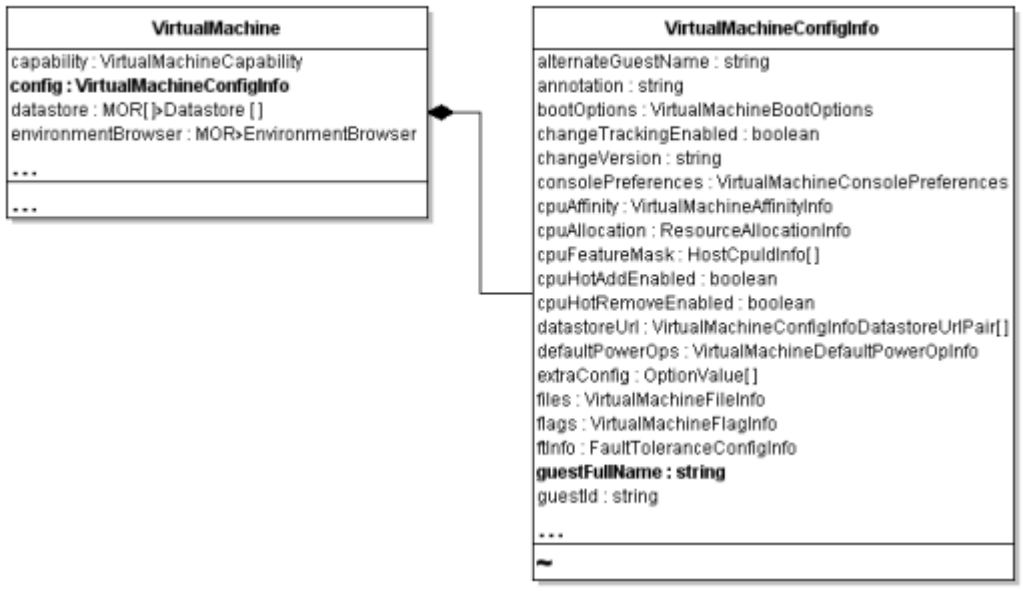
通过例子 B-2，B-3 和 B-4 中展示的 WSDL 定义可以推断出如例子 B-5 展示的结果。

Example B-5. CreateFilter Spec Property Entry

```
<spec>
    <propSet>
        <type>VirtualMachine</type>
        <all>false</all>
        <pathSet>config.guestFullName</pathSet>
    </propSet>
    <objectSet>
        <obj type="Folder">group-v4</obj>
        <skip>true</skip>
    </objectSet>
</spec>
```

在这个例子中，<spec>元素确定 CreateFilter 方法的 spec 参数。元素标签的顺序如属性在 XML 结构中定义的一样(例子 B-2)。pathSet 属性定义了到感兴趣的内嵌数据对象的完整路径。在例子 B-5 中，pathSet 属性定义到目标虚拟机的 guestFullName 属性的路径。图 B-4 展示这些内嵌的数据对象的 UML 图。

Figure B-4. Nested Data Objects



所有这些细节都在 vSphere API Reference 中。通过测试 WSDL 定义，你可以构建需要的字符串来通过 MOB 提交参数。表 B-1 提供了同时使用 MOB 和 vSphere API Referrnce 操作步骤总结。

Table B-1. Comparison of Datatypes for MOB Usage

Datatype	How to Input Values for Methods
Primitive	Enter the value as plain text regardless of its data type (`int`, `string`, `boolean`). Do not use quotes or other markup.
Array	Use the name of the parameter as the name of the element, wrap the values in a series of opening and closing tags for each array element.
Complex	Obtain XML Schema format information from the *vSphere API Reference* for the type (from the **Show WSDL type definition** link). Use the schema definition to construct the sequence of tags around the value (or values) you want to pass to the MOB.

19、HTTP 访问 vSphere 服务器文件

在大多数情况下，客户端应用通过使用 vSphere Web Services SDK 与 vSphere 服务器交互。有一些情况下，直接访问 ESX/ESXi 或 vCenter 服务器系统上的配置文件，日志文件，其他数据更有效率。本章解释了直接文件访问。

19.1 HTTP 访问介绍

ESX/ESXi 和 vCenter 服务器系统支持 HTTP 和安全 HTTP 文件访问。你可以使用微下列类型的访问使用 HTTP/HTTPS

- ESX/ESXi 和 vCenter 服务器系统上数据存储访问。
- ESX/ESXi 系统上的 ESX/ESXi 配置和日志文件访问。
- ESX/ESXi 系统上的更新包访问。

你可以使用 HTTP 方法 GET，HEAD，PUT，和 DELETE 来访问文件。HTTP/HTTPS 的 URL 请求必须包含嵌入的关键字来指定访问的类型。表 C-1 展示了服务器访问类型相关的关键字和 HTTP 方法。

Table C-1. HTTP Access to vSphere Servers

Server Access	URL Keyword	HTTP Method or Methods
Datastore	folder	GET, HEAD, PUT, DELETE
ESX/ESXi configuration file	host	GET, HEAD, PUT (See Table C-2 for the specific methods supported for each file type.)
Update bundle	tmp	PUT

使用 PUT 方法来创建一个新文件或覆盖已经存在的文件。你可以通过使用一个支持的最上层目录一致的 URL 来创建一个子目录。你不能创建数据存储或数据中心因为 URL 必须引用一个有效的数据中心或数据存储。

你可以使用一个 Web 浏览器来浏览和下载文件。你不能使用浏览器来上传或删除文件。

19.2 HTTP 访问的 URL 语法

到一个 vSphere 服务器的 HTTP 请求的 URL 声明包括下列之一的关键字，其决定了访问类型。

- “数据存储访问(/folder)”
- “主机文件访问(/host)”
- “更新包访问(/tmp)”

19.2.1 数据存储访问(/folder)

对于数据存储访问的 HTTP 请求使用下列语法：

http-method http[s]://*server*/folder[[*/path*]?dcPath=path[&dsName=name]]

http-method	方法 GET，HEAD，PUT，或 DELETE 之一
http://或 https://	访问协议(标准访问或安全访问)
server	ESX/ESXi 或 vCenter 服务器目标系统。服务器的值可以是 IP 地址或 DNS 名称
/folder	指定访问 ESX/ESXi 或 vCenter 服务器系统上的数据存储。数据存储 URL 可以包括下列可选元素： ■ *path* – 到这个数据存储中的一个文件或目录的路径，相对这个数据存储的根目录。 ■ dcPath – 到一个数据中心的清单路径。在请求中指定数据中心路径为一个 name-value 对。对于一个位于根目录中

命名为MyDatacenter的数据中心，dcPath的值为 MyDatacenter。对于一个位于根目录的 NorthAmerica 子目录中的命名为 YourDatacenter 的数据中心，dcPath 的值为 NorthAmerica/YourDatacenter。

- dsName – 和这个数据中心关联的数据存储。在请求中数据存储名称指定为 name-value 对。

下面的例子展示了语法。假如目标服务器是一个 ESX/ESXi 系统，dcPath 是可选的并且默认为 dcPath=ha-datacenter。

Example	Description
/folder	Directory listing of known datacenters on this server.
/folder?dcPath=*path*	Directory listing of all datastores available at the specified datacenter.
/folder?dcPath=*path*&dsName=*name*	Top-level directory listing of the datastore.
/folder/*path*?dcPath=*path*&dsName=*name*	Directory listing of all files in a datastore directory.
/folder/*path*/disk-flat.vmdk?dcPath=*path*&dsName=*name*	Access individual files.

19.2.2 主机文件访问(/host)

对于访问 ESX/ESXi 配置文件的 HTTP 请求使用下列语法：

GEG http[s]://*my_system*/host

http-method http[s]://*my_system*/*file*

Syntax Element	Description
http-method	One of GET, HEAD, or PUT, depending on the type of configuration file (see Table C-2).
http:// or https://	Access protocol (standard access or secure access).
esx-server	IP address or a DNS name.
/host	List of configuration files that you can access. (Use /host to retrieve the list.)
/host/*file*	A specific ESX/ESXi configuration file.

表 C-2 展示了 ESX 主机配置文件和符合访问的 HTTP/HTTPS 的方法。文件集可能会因为版本而改变。

Table C-2. HTTP Host Access (/host URL)

Configuration File	HTTP Access Method(s)	Configuration File	HTTP Access Method(s)
hostAgentConfig.xml	GET, HEAD, PUT	ipmi0_sel.raw	GET, HEAD
sfcb.cfg	GET, HEAD, PUT	ipmi0_sel	GET, HEAD
openwsman.conf	GET, HEAD, PUT	ipmi0_sdr_content.raw	GET, HEAD
license.cfg	GET, HEAD, PUT	ipmi0_sdr_header.raw	GET, HEAD
vmware.lic	GET, HEAD, PUT	ipmi0_sensor_readings.raw	GET, HEAD
vmware_config	GET, HEAD, PUT	ipmi1_sel.raw	GET, HEAD
vmware_configrules	GET, HEAD, PUT	ipmi1_sel	GET, HEAD
proxy.xml	GET, HEAD, PUT	ipmi1_sdr_content.raw	GET, HEAD
snmp.xml	GET, HEAD, PUT	ipmi1_sdr_header.raw	GET, HEAD
syslog.conf	GET, HEAD, PUT	ipmi1_sensor_readings.raw	GET, HEAD
ssl_cert	GET, HEAD, PUT	ipmi2_sel.raw	GET, HEAD
ssl_key	PUT	ipmi2_sel	GET, HEAD
hosts	GET, HEAD, PUT	ipmi2_sdr_content.raw	GET, HEAD
motd	GET, HEAD, PUT	ipmi2_sdr_header.raw	GET, HEAD
vpxa.cfg	GET, HEAD, PUT	ipmi2_sensor_readings.raw	GET, HEAD
esx.conf	GET, HEAD, PUT	ipmi3_sel.raw	GET, HEAD
config.log	GET, HEAD	ipmi3_sel	GET, HEAD
messages	GET, HEAD	ipmi3_sdr_content.raw	GET, HEAD
hostd.log	GET, HEAD	ipmi3_sdr_header.raw	GET, HEAD
vpxa.log	GET, HEAD	ipmi3_sensor_readings.raw	GET, HEAD

19.2.3 更新包访问(/tmp)

更新包访问的 HTTP 请求使用下列语法：

PUT http[s]://*esx-server*/tmp/*file-path*

http://或 https://	访问协议(标准访问或安全访问)
esx-server	IP 地址或 DNS 名称
/tmp/*file-path*	ESX/ESXi 系统上的目标文件

19.2.4 HTTP 访问权限需求

HTTP 访问一个 vSphere 文件是到一个关联 vSphere 清单 folder 结构的数据存储对象的访问。HTTP 访问需要和其他获得这些文件机制一样的权限，例如 vSphere 客户端。表 C-3 展示了需要的权限。

Table C-3. Privileges Required for HTTP Access Datastore Objects

Object Associated with File	Portion of URL	Required Privileges
Root folder	/folder	System.View
Datacenter	?dcPath	Datastore.Browse
		Datastore.FileManagement
Datastore	&dsName	Datastore.Browse
		Datastore.FileManagement
Host	/host	Host.Config.AdvancedConfig
	/tmp/	Host.Config.SystemManagement

20、权限参考

VMware vSphere 组件通过权限，角色，和许可系统来保证安全。本章列出了执行不同操作所需的权限和读取属性需要的权限。

20.1 调用操作所需的权限

默认，所有在 vCenter 服务器系统上的 Windows Administrators 组的成员拥有相同的赋予所有对象上 Administrator 角色所拥有的权限。当直接连接 ESX/ESXi 主机时，root 和 vpxuser 用户账户有相同的如 Administrator 角色所拥有的权限。

所有其他用户初始在任何对象上没有任何许可，这意味着你不能查看这些对象或在他们上执行操作。一个具有 Administrator 权限的用户必须赋予这些用户允许他们执行任务的许可。