# 动态负载均衡系统设计总览

## 系统总体设计

### 动态负载均衡系统需求

基于Hyper-V的动态负载均衡系统是一款服务于桌面Hyper-V虚拟机的优化插件。

### 动态负载均衡系统总览

基于Hyper-V的动态负载均衡系统为C/S架构，由两大部分构成，分别是运行于Host的资源动态均衡器（Load-Balancer）和运行于VM的域内性能监测器（Perf-Detector），域内性能监测器为客户端，Host资源动态分配器为服务端，二者通过基于共享内存的Hyper-V socket进行通信，系统总览图如下图#所示：

域内性能监测器实时运行在开启状态的虚拟机，实时监测域内的性能指标，包括性能计数器中CPU、内存相关指标，更细粒度监测进程列表以及各进程占用的资源。每个开启的虚拟机中的性能监测器通过基于共享内存的HvSocket与Host通信，将性能信息发送给Host端的资源动态均衡器。资源动态均衡器可以直接监测各虚拟机的CPU占用率，但在静态内存分配的情况下，Host无法获取域内实际内存占用情况，更无法获取域内进程相关信息，因此资源动态分配器需要接收域内性能监测器发来的性能相关信息。资源动态均衡器根据各虚拟机的性能指标，以及Host的剩余资源情况，选择为资源紧张或空闲的虚拟机进行调度，回收空闲虚拟机已分配的内存和CPU资源，为负载高的虚拟机分配内存和CPU资源，可以提高负载高虚拟机的性能表现。

首先要解决的问题是：监测域内哪些指标能较为准确地反映虚拟机的负载情况，域内监测器不仅要监测CPU占用率和内存占用率，还需要监测反映资源需求情况的指标如线程等待队列长度Processor Queue Length、每秒产生的页错误次数Page Faults/secs。进一步地问题是，Host动态均衡器获取了所有性能指标，并进行分析后，应如何进行动态资源调度，Hyper-V提供了内存热插拔机制，在静态内存分配情况下，开机状态下可以在一定范围内调节静态分配给虚拟机的内存大小，如下图#所示：

Host动态均衡器通过设置开启虚拟机的内存大小来调度各虚拟机的内存资源。至于CPU资源，可以通过设置开启后虚拟机的“CPU限制”、“CPU保留”参数设置虚拟机实际可占用的CPU资源，举例如下图#所示：

给某虚拟机分配2个物理CPU，虚拟机中又两个vCPU，如果“CPU限制”为100，则该虚拟机最多占用全部2个物理CPU，将“CPU限制”改为500，则限制该虚拟机最多占用50%的CPU资源，即1个物理CPU。类似地，“CPU保留”指定了虚拟机最低保留的CPU资源比例，该比例需小于“CPU限制”。综上所述，可以有效动态调度CPU和内存资源。

## 系统模块设计

#### 域内性能监测器

域内性能监视器运行于开启的虚拟机，开机自启并定时检测域内内存、CPU性能相关的程序计数器，将收集到的内存、CPU相关性能信息发送到Host负载均衡器，为支持进程粒度的性能分析，以及供负载均衡器进行性能分析。

域内性能检测器按功能分为如下模块，各模块功能如下：

1. 性能计数器接口模块，该模块封装了实时获取部分Windows系统性能计数器指标的API，供其他模块调用；
2. 内存探测模块，通过调用性能计数器接口模块相关API，定时获取实时内存性能相关指标，如：内存占用率，页错误频率等；
3. CPU探测模块，通过调用性能计数器接口模块相关API，定时获取实时CPU性能相关指标，如：CPU占用率，Process Queue Length等；
4. 进程信息探测模块，定时执行“tasklist /v”命令获取进程详细信息，包括占用的内存大小等，并将结果重定向到文件中；
5. 性能信息发送模块，建立与Host负载均衡器基于共享内存的hvSocket，定义性能消息传输的接口，通过hvSocket定时发送性能信息；
6. 进程控制模块，Host负载均衡器读取用户设置的进程白名单和黑名单，若发现黑名单进程出现或占用大量资源，Host传输一个进程控制文件(json格式)到域内。此时，进程控制模块监测到该文件并杀死黑名单进程；
7. 主控模块，调用CPU、内存、进程信息探测模块定时获取综合的性能信息，定时将性能信息通过性能信息发送模块发送到Host负载均衡器。

各模块的输入输出，如下表# 所示：

|  |  |  |
| --- | --- | --- |
| **模块名称** | **输入** | **输出** |
| 性能计数器接口模块 | 性能指标 | 所选性能指标的值 |
| 内存探测模块 | 无 | 虚拟机内存性能信息 |
| CPU探测模块 | 无 | 虚拟机CPU性能信息 |
| 进程信息探测模块 | 无 | 记录进程详细信息的文件 |
| 性能信息发送模块 | 需要传输的CPU、内存性能信息 | 性能消息发送结果 |
| 进程控制模块 | 进程控制文件 | 是否杀死黑名单进程 |
| 主控模块 | 各个性能探测模块获取的信息 | 无 |

#### 负载均衡器

负载均衡器运行在Host物理机，通过两种方式：Host获取虚拟机性能信息和接收各个虚拟机发来的性能信息，综合分析各虚拟机的性能表现，根据调度策略分配、回收虚拟机资源，以达到负载均衡的效果。

负载均衡器分为如下几个模块：

1. 配置文件读写模块，本系统通过一个配置文件记录用户的设置信息，该模块可以读取配置文件，并初始化相应的数据结构，该模块也可以写配置文件某些字段；
2. 系统信息获取模块，该模块获取物理主机的相关信息，包括硬件配置和物理机性能信息，如CPU核心数、物理内存大小、剩余内存大小等；
3. 虚拟机操作模块。该模块定义虚拟机的实例以及管理虚拟机的方法，包括对虚拟机实例进行抽象表示，获取Host可以得到的虚拟机相关信息，并定义修改虚拟机设置的通用接口等；
4. 动态调节接口模块，本模块封装了对CPU、内存资源进行动态调节的接口，如修改内存大小、设置CPU资源限制、CPU资源保留的接口，该模块需调用虚拟机管理模块，使负载均衡器能够调节开启状态的虚拟机资源；
5. 虚拟机状态监控模块，本模块接收虚拟机状态传输工具发来的消息，修改虚拟机的状态如打开、关闭、切入，以及新建虚拟机、删除虚拟机等操作，及时更新各虚拟机的状态信息如：是否安装、是否开机、记录当前切入的虚拟机等；
6. 性能接收模块，该模块接收虚拟机的连接请求，并建立基于共享内存的hvSocket连接，定义接收虚拟机性能信息传输的接口；
7. HyperV性能监测模块，本模块监测Windows中性能计数器中有关Hyper-V性能的部分，如虚拟机的当前内存压力、平均内存压力等；
8. 负载分析均衡模块，该模块分析性能接收模块获取的各虚拟机性能信息，以及综合系统信息获取模块得到的主机信息，根据设定的负载均衡策略，调用动态调节模块，分别对虚拟机的CPU和内存进行动态调节；
9. 单元测试模块，该模块设计了其他子模块的测试用例，根据测试需要对不同模块进行测试；
10. 主控模块，该模块调用性能接收模块，定时刷新虚拟机的性能信息，作为输入调用系统负载分析模块，启动整个系统流程。

各模块的输入输出，如下表# 所示：

|  |  |  |
| --- | --- | --- |
| **模块名称** | **输入** | **输出** |
| 配置文件读写模块 | 配置文件路径 | 与系统配置相关的数据结构 |
| 系统信息获取模块 | 无 | 物理机相关信息 |
| 虚拟机管理模块 | 虚拟机名 | 虚拟机CPU性能信息 |
| 动态调节接口模块 | 待调节的选项名，以及对应的值 | 动态调节的结果 |
| 性能接收模块 | 需要传输的CPU、内存性能信息 | 性能消息发送结果 |
| 负载分析模块 | 进程控制文件 | 是否杀死黑名单进程 |
| 单元测试模块 | 各个性能探测模块获取的信息 | 无 |
| 主控模块 |  |  |

#### 虚拟机状态传输工具

虚拟机状态传输工具

# 域内性能监测器

### 性能计数器接口模块

#### 功能

本模块提供获取Windows性能计数器部分内存、CPU性能相关指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

1. Windows性能计数器接口类 WinPerfCounter

数据类型：class WinPerfCounter

功能：提供Windows性能计数器获取部分指标的方法，并描述性能计数器获取的性能指标。

详细数据结构：

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| CPUProcessorTime | CPU核心数 | float |
| CPUPrivilegedTime | 表示host管理器的授权码 | float |
| CPUInterruptTime | CPU占用率 | float |
| CPUDPCTime |  | float |
| MEMAvailable | 表示时间戳 | float |
| MEMCommited |  | float |
| MEMCommitLimit |  | float |
| MEMCommitedPerc |  | float |
| MEMPoolPaged |  | float |
| MEMPoolNonPaged |  | float |
| MEMCached |  | float |
| PageFile |  | float |
| PageFile |  | float |
| ProcessorQueueLengh |  | float |
| HANDLECountCounter |  | float |
| THREADCount |  | float |
| CONTENTSwitches |  | int |
| SYSTEMCalls |  | int |
| PagesPerSec |  | float |
| PageFaultsPerSec |  | float |
| PageReadsPerSec |  | float |
| PageWritesPerSec |  | float |
| PagesInputPerSec |  | float |
| PagesOutputPerSec |  | float |

#### 函数及流程图

### CPU探测模块

#### 功能

本模块定时获取并刷新CPU性能相关信息，如：CPU占用率，Process Queue Length等。该模块对应代码中的CpuSpoofer.cs文件。

#### 设计

#### 异常和错误处理

#### 相关数据结构

1. CPU探测类 CpuSpoofer

【数据类型】

class

【功能】

描述收集到的CPU性能信息，并提供定时刷新CPU性能信息的方法。

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| CPUCount | CPU核心数 | int |
| ProcessorQueueLength | 表示host管理器的授权码 | float |
| CPUProcessorTime | CPU占用率 | float |
| CPUPrivilegedTime |  | float |
| CPUInterruptTime | 表示时间戳 | float |
| CPUDPCTime |  | float |
| winPerfCounter |  | WinPerfCounter |
|  |  | Timer |

#### 函数及流程图

CpuSpoofer()

【所属类及所在文件】

CpuSpoofer类，CpuSpoofe.cs文件

【功能描述】

CpuSpoofer类构造函数。

【输入参数】

无

【返回值】

无

【流程图】

开始

WinPerfCounter.initAllCounterValue();

初始化性能计数器的初始值

以RefreshStateByTime函数新建一个线程并启动

结束

否

**RefreshStatesByTime()**

【所属类及所在文件】

CpuSpoofer类，CpuSpoofe.cs文件

【功能描述】

启动一个定时器，以一定时间间隔，获取并刷新CPU性能相关参数。

【输入参数】

无

【返回值】

无

【流程图】

开始

RUtimer = new System.Timers.Timer(100);

创建一个100ms定时的定时器

RUtimer.Elapsed += RefreshCPUArg;

定时时间到，处理函数为RefreshCPUArg;

RUtimer.AutoReset = true;

RUtimer.Enabled = true;

RUtimer.Start();

设置定时时间到自动重新计时，使能定时器并启动

结束

否

是

**RefreshCPUArg(object sender, ElapsedEventArgs e)**

【所属类及所在文件】

CpuSpoofer类，CpuSpoofe.cs文件

【功能描述】

获取并刷新CpuSpoofer类中CPU性能相关参数。

【输入参数】

object sender:表示触发事件的控件对象，EventArgs e：表示事件数据的类的基类

【返回值】

无

【流程图】

开始

CPUProcessorTime = winPerfCounter.getProcessorCpuTime();

ProcessorQueueLength = winPerfCounter.getProcessorQueueLengh();

CPUPrivilegedTime = winPerfCounter.getCpuPrivilegedTime();

CPUInterruptTime = winPerfCounter.getThreadCount();

调用性能计数器接口模块获取上述参数

结束

否

是否捕获异常

打印异常信息

是

否

### 内存探测模块

#### 功能

本模块定时获取并刷新内存性能相关信息，如：可用内存大小，页错误频率等。

#### 设计

#### 相关数据结构

1. 内存探测类 MemSpoofer

【数据类型】

class

【功能】

描述收集到的内存性能信息，并提供定时刷新内存性能信息的方法。

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| MEMAvailable | CPU核心数 | float |
| MEMCommited | 表示host管理器的授权码 | float |
| MEMCommitLimit | CPU占用率 | float |
| MEMCommitedPerc |  | float |
| MEMPoolPaged | 表示时间戳 | float |
| MEMPoolNonPaged |  | float |
| MEMCached |  | float |
| PageFile |  | float |
| PagesPerSec |  | float |
| PageFaultsPerSec |  | float |
| PageReadsPerSec |  | float |
| PageWritesPerSec |  | float |
| PagesInputPerSec |  | float |
| PagesOutputPerSec |  | float |
| MEMPerfCounter |  | WinPerfCounter |
| RUtimer |  | Timer |

#### 流程图

* MemSpoofer ()

【所属类及所在文件】

MemSpoofer类，MemSpoofe.cs文件

【功能描述】

MemSpoofer类构造函数。

【输入参数】

无

【返回值】

无

【流程图】

开始

MEMPerfCounter.initAllCounterValue();

初始化性能计数器的初始值

以RefreshStateByTime函数新建一个线程并启动

结束

否

**RefreshMemStatesByTime()**

【所属类及所在文件】

MemSpoofer类，MemSpoofe.cs文件

【功能描述】

启动一个定时器，以一定时间间隔，获取并刷新内存性能相关参数。

【输入参数】

无

【返回值】

无

【流程图】

开始

RUtimer = new System.Timers.Timer(100);

创建一个100ms定时的定时器

RUtimer.Elapsed += RefreshMEMArg;

定时时间到，处理函数为RefreshMEMArg;

RUtimer.AutoReset = true;

RUtimer.Enabled = true;

RUtimer.Start();

设置定时时间到自动重新计时，使能定时器并启动

结束

否

是

**RefreshMEMArg(object sender, ElapsedEventArgs e)**

【所属类及所在文件】

MemSpoofer类，MemSpoofe.cs文件

【功能描述】

获取并刷新MemSpoofer类中内存性能相关参数。

【输入参数】

object sender:表示触发事件的控件对象，EventArgs e：表示事件数据的类的基类

【返回值】

无

【流程图】

开始

MEMAvailable = MEMPerfCounter.getMemAvailable();

MEMCommited = MEMPerfCounter.getMemCommited();

MEMCommitLimit = MEMPerfCounter.getMemCommitLimit();

MEMCommitedPerc = MEMPerfCounter.getMemCommitedPerc();

……

调用性能计数器接口模块获取上述参数

结束

否

是否捕获异常

打印异常信息

否

是

### 进程监测模块

#### 功能

本模块提供获取进程列表及进程详细信息的方法。

#### 设计

1. 该线程启动时间：在用户连接网络后，输入远程服务器IP地址，点击确认连接服务器后；
2. 授权验证线程需要将主机Host的IP地址发送给远程服务器，当服务器需要远程配置还原策略时，服务器需要建立与Host的socket；
3. 授权验证线程发送授权码等授权信息后，如果超时，socket会产生异常，在检测到该异常后要求用户重新输入授权码；
4. 网络监控线程实时监控网络状态，若发生断网则通知授权验证线程（授权验证线程存在），如果授权验证在等待回复，授权验证线程终止等待授权验证，将消息：“断网，授权验证失败，请重新连接网络”提示用户，GUI主线程设置授权状态为为未授权。

#### 异常和错误处理

1. 用户输入远程服务器IP，并点击确认连接后，本线程会首先尝试ping远程服务器，若无法正常连接，则提示用户重新检查服务器IP，并测试网络；
2. 授权未成功时，提示用户授权失败，根据授权失败信息，可以让用户重新输入授权码。
3. 授权验证线程发送验证码给服务器后，需要等待远程服务器应答，若此时网络断开，socket会直接响应异常，授权验证线程捕获该异常后，清空该socket，并提示用户网络断开，需要用户检查服务器IP或检测网络联通性，并重新输入授权码。

#### 相关线程

GUI主线程：本线程需要GUI界面用户输入的远程服务器IP地址，另外，若验证不成功，需要通知GUI主线程提示用户验证不成功，需要重新确认服务器IP和授权码；

远程服务器消息接收线程：运行在远程服务器端，监听Host端发来的授权请求，详情见远程服务器部分。

#### 相关数据结构

授权验证线程发送一次授权验证请求，所依赖的数据结构为一个AuthorityRequest类实例；接收远程服务器的验证信息是一个AuthorityResponse类实例，具体数据结构如下：

1. 授权请求类 AuthorityRequest

数据类型：class AuthorityRequest

功能：描述网络通信模块向远程服务器发送的授权请求

详细数据结构：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 含义 | 类型 | Value |
| Index | 表示一次会话的标识符 | int | 1 |
| AuthorizationCode | 表示host管理器的授权码 | String | 配置文件 |
| PCID | 表示host管理器安装的计算机的唯一识别码 | string | 软件生成 |
| serialNumber | 表示host管理器的唯一识别码 | String | 软件生成/由pcid生成 |
| timestamp | 表示时间戳 | datetime | 读取系统信息 |

1. 授权接收类AuthorityResponse

数据类型：class AuthorityResponse

功能：描述远程服务器返回的授权响应

详细数据结构：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 含义 | 来源 |
| index | int | 表示一次会话的标识符 | 软件生成 |
| isAuthorized | bool | 表示当前是否被授权 | 服务器端消息 |
| isForbidden | bool | 表示host管理器所在的计算机是否被禁用 | 服务器端消息 |
| SerialNumber | String | 表示host管理器的唯一识别码 | 软件生成/由pcid生成 |
| Timestamp | datetime | 表示时间戳 | 读取系统信息 |

#### 流程图

用户连接网络

根据远程服务器IP地址，初始化授权验证线程

用户输入远程服务器IP，点击确认

尝试连接远程服务器，若无法连接，提示用户检查远程服务器IP

弹出对话框，要求用户输入授权码，与服务器建立socket连接

用户输入点击确认后，根据验证码生成相应的数据机构，向服务器发送授权验证请求

设置计时器，等待用户返回授权响应，若超时则提醒用户

授权验证线程接收到授权响应，解析到授权结果，将结果发送到GUI主线程

授权验证线程结束

### 性能信息发送模块

#### 功能

本模块整合CPU探测模块和内存探测模块收集的性能信息，通过基于共享内存的Hyper-V Socket发送给Host负载均衡器。

#### 设计

在C#的WCF模型中，客户端与服务端需要定义ServiceContract服务契约和OperationContract操作契约，ServiceContract作为服务端和客户端交互的接口，OperationContract约定的操作。

#### 相关数据结构

1. PerformanceInfo类

【数据类型】

class

【功能】

收集内存探测模块和CPU探测模块的性能信息

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| perf\_Transfer | 与Host传输的性能信息 | VMPerf class |
| VMName | 虚拟机名 | string |
| memSpoofer | 内存性能探测器实例 | MemSpoofer class |
| cpuSpoofer | CPU性能探测器实例 | CpuSpoofer class |
| processInfo | 进程信息监测类实例 | ProcessInfo class |
| RUtimer | 定时器 | Timers.Timer |

1. 接口IServer

【数据类型】

interface

【功能】

定义服务契约IServer，其中包含传输性能信息的操作契约函数TransferPerfStr(string perf\_Str)

【详细数据结构】：

[ServiceContract]

public interface IServer

{

[OperationContract]

VMPerf TransferPerfStr(string perf\_Str);

}

1. DetectorClient类

【数据类型】

class

【继承关系】

继承ClientBase<IServer>, IServer

【功能】

客户端代理类，继承ClientBase<>类，这是一个泛型类，接受服务契约IServer作为泛型参数。DetectorClient类通过基类的信道，调用操作契约函数TransferPerfStr。

【详细数据结构】

public class DetectorClient : ClientBase<IServer>, IServer

{

public DetectorClient(EndpointAddress addy)

: base(new HyperVNetBinding(), addy)

{

}

public VMPerf TransferPerfStr(string perf\_Str) => Channel.TransferPerfStr(perf\_Str);

}

1. Client类

【数据类型】

class

【功能】

根据唯一URI创建HvSocket连接连接Host，提供定时发送性能信息的接口

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| RUtimer | 定时器 | Timers.Timer |
| HvAddr | 客户端连接的地址 | string |
| clientPerfInfo | 客户端收集的性能信息 | PerformanceInfo class |
| detectorClient | 与Host通信客户端代理 | DetectorClient class |
| sendTimeGap | 发送信息的时间间隔，单位ms | int |

#### 流程图

* PerformanceInfo()

【所属类及所在文件】

PerformanceInfo类，PerformanceInfo.cs文件

【功能描述】

PerformanceInfo类构造函数。

【输入参数】

无

【返回值】

无

【流程图】

是

否

开始

memSpoofer = new MemSpoofer();

cpuSpoofer = new CpuSpoofer();

结束

memSpoofer.RefreshMEMArg();

cpuSpoofer.RefreshCPUArg();

刷新memSpoofer和cpuSpoofer中的性能信息

bool ret = SetPerfTransfer(memSpoofer, cpuSpoofer);

调用SetPerfTransfer函数将性能信息写入perf\_Transfer字段中

bool vmIDRet = GetVMName(@"C:\VMID.txt");

读取VMName.txt文件获取虚拟机名

! vmIDRet

VMName = "Unknown";

* GetVMName(string vmIDFilePath)

【所属类及所在文件】

PerformanceInfo类，PerformanceInfo.cs文件

【功能描述】

读取配置文件，获取虚拟机名称。

【输入参数】

vmIDFilePath，配置文件路径

【返回值】

bool值，表示读取是否成功

【流程图】

开始

返回ret

VMName = File.ReadAllText(vmIDFilePath);

bool ret = true;

读取配置文件中vmname字段

是否产生异常

bool ret = false;

!File.Exists(vmIDFilePath)

否

否

是

是

* SetPerfTransfer(MemSpoofer memSpf, CpuSpoofer cpuSpf)

【所属类及所在文件】

PerformanceInfo类，PerformanceInfo.cs文件

【功能描述】

输入内存、CPU监视器实例，从中获取待传输的性能指标，并据此对perf\_Transfer字段进行复制。

【输入参数】

MemSpoofer memSpf, 内存监视器实例，CpuSpoofer cpuSpf，CPU监视器实例

【返回值】

bool值，表示设置是否成功

【流程图】

开始

返回ret

perf\_Transfer.VMName = VMName;

perf\_Transfer.CPUCount = cpuSpf.CPUCount;

perf\_Transfer.CPUProcessorTime = cpuSpf.CPUProcessorTime

……

根据cpuSpf中CPU性能指标设置perf\_Transfer对应指标

是否产生异常

ret = false;

perf\_Transfer.MEMAvailable = memSpf.MEMAvailable;

perf\_Transfer.PagesPerSec = memSpf.PagesPerSec;

……

根据memSpf中内存性能指标设置perf\_Transfer对应指标，bool ret = true

是

否

* Client(string Addr, int TimeGap = 100)

【所属类及所在文件】

Client类，Client.cs文件

【功能描述】

Client类构造函数。

【输入参数】

Addr，客户端连接的Host地址; TimeGap，性能信息发送的时间间隔，默认100ms

【返回值】

无

【流程图】

开始

clientPerfInfo = new PerformanceInfo();

初始化性能信息

HvAddr = Addr;

设置类变量HvAddr

sendTimeGap = TimeGap;

结束

否

* StartUpClient ()

【所属类及所在文件】

Client类，Client.cs文件

【功能描述】

初始化客户端代理detectorClient并启动，调用SendByTime方法定时发送性能信息。

【输入参数】

无

【返回值】

无

【流程图】

开始

detectorClient = new DetectorClient(new EndpointAddress(HvAddr));

初始化客户端代理detectorClient

detectorClient.Open();

启动客户端代理

SendByTime();;

调用SendByTime函数定时发送

结束

否

是

* SendByTime()

【所属类及所在文件】

Client类，Client.cs文件

【功能描述】

以sendTimeGap为定时间隔，定时调用SendPerfTransfer函数发送性能信息。

【输入参数】

无

【返回值】

无

【流程图】

开始

RUtimer = new System.Timers.Timer(sendTimeGap);

创建一个sendTimeGap定时的定时器

RUtimer.Elapsed += SendPerfTransfer;

定时时间到，处理函数为SendPerfTransfer;

RUtimer.AutoReset = true;

RUtimer.Enabled = true;

RUtimer.Start();

设置定时时间到自动重新计时，使能定时器并启动

结束

否

是

* SendPerfTransfer(object sender, ElapsedEventArgs e)

【所属类及所在文件】

Client类，Client.cs文件

【功能描述】

获取内存、CPU性能信息，将类序列化为字符串后，客户端代理detectorClient调用TransferPerfStr函数通知服务端接收。

【输入参数】

object sender:表示触发事件的控件对象，EventArgs e：表示事件数据的类的基类

【返回值】

无

【流程图】

开始

clientPerfInfo.SetPerfTransfer(clientPerfInfo.memSpoofer, clientPerfInfo.cpuSpoofer);

调用SetPerfTransfer函数，将memSpoofer个cpuSpoofer中的性能信息写入clientPerfInfo.perf\_Transfer字段中

返回

否

是否捕获异常

打印异常信息

将VMPerf类型的clientPerfInfo.perf\_Transfer字段序列化为string perfTransferStrSerial，调用

detectorClient.TransferPerfStr(perfTransferStrSerial);

函数通知服务端接收

是

否

### 主控模块

#### 功能

授权验证线程与服务器建立socket连接，向服务器发送授权码，进行授权验证，该线程将该授权结果发送给GUI主线程。若授权验证成功，GUI主线程将执行下一步操作；若授权不成功，GUI主线程提示用户授权码不正确，需要重新输入授权码。

#### 设计

# Host负载均衡器

### 配置文件读取模块

#### 功能

本模块提供获取Windows性能计数器部分内存、CPU性能相关指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

* VMConfig类

【数据类型】

class

【功能】

虚拟机的配置信息，包括虚拟机名、预设定内存大小、CPU核心数等

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| VMName | 虚拟机名 | string |
| Installed | 是否安装 | bool |
| MemorySize | 预设内存大小 | UInt64 |
| CPUNum | 预设CPU核心数 | int |
| CPULimt | 预设CPU限制大小 | int |

* SysConfig类

【数据类型】

class

【功能】

负载均衡器的配置选项信息

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| BalanceStrategy | 均衡策略 | string |
| CPUDefaultCount | 系统默认CPU核心数 | int |
| MemoryDefaultSize | 系统默认内存大小 | UInt64 |
| CPUDefaultLimit | 系统默认的CPU核心数 | int |

* MpcVMInfo类

【数据类型】

class

【功能】

多域PC域管理器中，虚拟机配置文件对应的配置信息。

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| VMName | 虚拟机名 | string |
| Installed | 是否安装 | bool |
| VMPath | 虚拟机安装路径 | string |
| Description | 描述信息 | string |
| DefaultUser | 域内用户名 | string |
| Password | 域内密码 | string |
| IPAddress | IP地址 | string |
| StateSnapID | 当前使用快照ID | string |
| RenamePC | 是否重命名域内主机名 | bool |

* GetConfig类

【数据类型】

class

【功能】

解析配置文件，设置配置文件对应的内存数据结构。

【详细数据结构】

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 含义 | 类型 | 初始值 |
| ConfigFileRelPath | 存储配置文件的相对路径 | string | @"..\..\.." |
| VMConfigName | 虚拟机配置文件名 | const string | "VMConfig.json" |
| SysConfigName | 系统配置文件名 | const string | "SystemConfig.json" |
| ProcessConfigName | 进程配置信息文件名 | const string | "ProcessWhiteList.json" |
| VMconfigPath | 虚拟机配置文件相对路径 | string | ConfigFileRelPath |
| SysConfigPath | 系统配置文件相对路径 | string |  |
| ProcessConfigPath | 进程配置信息文件相对路径 | string |  |
| VMStateJsonPath | 当前使用快照ID | string |  |
| currentVMConfig | 当前解析到的虚拟机配置信息 | VMConfig | 无 |
| currentSysConfig | 当前解析到的系统配置信息 | SysConfig | 无 |
| currentMpcVMInfo | 当前解析到的多域PC虚拟机信息 | MpcVMInfo | 无 |

#### 函数及流程图

* GetVMConfig(string VMName)

【所属类及所在文件】

GetConfig类，GetConfig.cs文件

【功能描述】

解析虚拟机配置文件，根据配置文件初始化currentVMConfig变量

【输入参数】

VMName，虚拟机名

【返回值】

VMConfig类实例

【流程图】

否

是

开始

读取路径VMconfigPath下的虚拟机配置文件，初始化JsonTextReader类型的reader

JObject o = (JObject)JToken.ReadFrom(reader)

新建用于操作Json对象的JObject

返回cureentVMConfig

currentVMConfig.VMName = VMName;

JObject vmObj = ((JObject)o[VMName];

currentVMConfig.Installed = Convert.ToBoolean(vmObj["Installed"].ToString()

currentVMConfig.CPUNum = Convert.ToInt32(vmObj["CPUCount"].ToString());

currentVMConfig.MemorySize = Convert.ToUInt64(vmObj["MemorySize"].ToString())；

currentVMConfig.CPULimt = Convert.ToInt32(vmObj["CPULimit"].ToString());

读取Json对象并对currentVMConfig对应字段赋值

o.ContainsKey(VMName)

currentVMConfig=null;

* GetSysConfig ()

【所属类及所在文件】

GetConfig类，GetConfig.cs文件

【功能描述】

解析负载均衡器配置文件，根据配置文件初始化currentSysConfig变量

【输入参数】

无

【返回值】

SysConfig类实例

【流程图】

开始

读取路径SysConfigPath下的虚拟机配置文件，初始化JsonTextReader类型的reader

JObject o = (JObject)JToken.ReadFrom(reader)

新建用于操作Json对象的JObject

返回cureentVMConfig

currentSysConfig.BalanceStrategy = o["BalanceStrategy"].ToString();

currentSysConfig.CPUDefaultCount = Convert.ToInt32(o["CPUDefaultCount"];

…

读取Json对象并对currentSysConfig对应字段赋值

* GetMpcVMInfo(string VMStateJsonPath, string VMName)

【所属类及所在文件】

GetConfig类，GetConfig.cs文件

【功能描述】

解析多域PC虚拟机配置文件，根据配置文件初始化currentMpcConfig变量

【输入参数】

VMStateJsonPath，多域PC虚拟机配置文件路径，VMName，虚拟机名

【返回值】

MpcConfig类实例

【流程图】

开始

返回currentMpcVMInfo

通过StreamReader读取虚拟机配置文件内容

将读取到Host配置文件内容初始化为JObject o

依次将vmObj如下字段读入currentMpcVMInfo对应字段：

VMNane, Installed, VMPath,DefaultUser, Password, IPAddress, StateSnapID, ,Description, RenamePC

JObject vmObj = ((JObject)o["LocalVM"])

VMName == "LocalVM" && o.ContainsKey("LocalVM")

是

否

VMName == "NetVM1" && o.ContainsKey("LocalVM")

依次将vmObj如下字段读入currentMpcVMInfo对应字段：

VMName == "NetVM2" && o.ContainsKey("LocalVM")

依次将vmObj如下字段读入currentMpcVMInfo对应字段：

否

否

是

currentMpcVMInfo=null

是

### 系统信息获取模块

#### 功能

本模块提供获取Windows性能计数器部分内存、CPU性能相关指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

#### 函数及流程图

* SystemInfo类

【数据类型】

class

【功能】

虚拟机的配置信息，包括虚拟机名、预设定内存大小、CPU核心数等

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| VMName | 虚拟机名 | string |
| Installed | 是否安装 | bool |
| MemorySize | 预设内存大小 | UInt64 |
| CPUNum | 预设CPU核心数 | int |
| CPULimt | 预设CPU限制大小 | int |

* SysConfig类

【数据类型】

class

【功能】

负载均衡器的配置选项信息

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| BalanceStrategy | 均衡策略 | string |
| CPUDefaultCount | 系统默认CPU核心数 | int |
| MemoryDefaultSize | 系统默认内存大小 | UInt64 |
| CPUDefaultLimit | 系统默认的CPU核心数 | int |

是

否

开始

memSpoofer = new MemSpoofer();

cpuSpoofer = new CpuSpoofer();

结束

memSpoofer.RefreshMEMArg();

cpuSpoofer.RefreshCPUArg();

刷新memSpoofer和cpuSpoofer中的性能信息

bool ret = SetPerfTransfer(memSpoofer, cpuSpoofer);

调用SetPerfTransfer函数将性能信息写入perf\_Transfer字段中

bool vmIDRet = GetVMName(@"C:\VMID.txt");

读取VMName.txt文件获取虚拟机名

! vmIDRet

VMName = "Unknown";

* GetVMName(string vmIDFilePath)

【所属类及所在文件】

PerformanceInfo类，PerformanceInfo.cs文件

【功能描述】

读取配置文件，获取虚拟机名称。

【输入参数】

vmIDFilePath，配置文件路径

【返回值】

bool值，表示读取是否成功

【流程图】

开始

返回ret

VMName = File.ReadAllText(vmIDFilePath);

bool ret = true;

读取配置文件中vmname字段

是否产生异常

bool ret = false;

!File.Exists(vmIDFilePath)

否

否

是

是

* SetPerfTransfer(MemSpoofer memSpf, CpuSpoofer cpuSpf)

【所属类及所在文件】

PerformanceInfo类，PerformanceInfo.cs文件

【功能描述】

输入内存、CPU监视器实例，从中获取待传输的性能指标，并据此对perf\_Transfer字段进行复制。

【输入参数】

MemSpoofer memSpf, 内存监视器实例，CpuSpoofer cpuSpf，CPU监视器实例

【返回值】

bool值，表示设置是否成功

【流程图】

开始

返回ret

perf\_Transfer.VMName = VMName;

perf\_Transfer.CPUCount = cpuSpf.CPUCount;

perf\_Transfer.CPUProcessorTime = cpuSpf.CPUProcessorTime

……

根据cpuSpf中CPU性能指标设置perf\_Transfer对应指标

是否产生异常

ret = false;

perf\_Transfer.MEMAvailable = memSpf.MEMAvailable;

perf\_Transfer.PagesPerSec = memSpf.PagesPerSec;

……

根据memSpf中内存性能指标设置perf\_Transfer对应指标，bool ret = true

是

否

* Client(string Addr, int TimeGap = 100)

### 虚拟机管理模块

#### 功能

本模块提供获取Windows性能计数器部分内存、CPU性能相关指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

* PerformanceSetting结构体

【数据类型】

Struct

【功能】

从Host可以直接获得的性能相关的虚拟机信息；

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| Name | 系统内唯一的实例名 | string |
| ElementName | 虚拟机名 | string |
| InstanceID | 虚拟机ID | string |
| NumberOfProcessors | CPU核心数 | UInt16 |
| EnabledState | 预设CPU限制大小 | UInt16 |
| HealthState | 虚拟机健康状态 | UInt16 |
| ProcessorLoad | 当前虚拟机CPU占总CPU资源的负载率 | UInt16 |
| ProcessorLoadHistory | 记录前100次CPU负载率的数组 | UInt16[] |
| MemoryUsage | 分配的内存大小 | UInt64 |
| CPU\_Reservation | CPU保留 | UInt16[] |
| CPU\_Limit | CPU限制 | UInt64 |
| CPU\_Weight | CPU权重 | UInt32 |
| RAM\_VirtualQuantity | 分配的内存大小 | UInt64 |
| RAM\_Weight | 内存权重 | UInt32 |
| RAM\_DynamicMemoryEnabled | 是否打开动态内存 | bool |

* VirtualMachine类

【数据类型】

class

【功能】

表示虚拟机的属性及操作虚拟机的方法；

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| vmName | 虚拟机名 | UInt16 |
| performanceSetting | 与性能相关的虚拟机设置 | UInt16 |
| VirtualMachineStatus | 枚举虚拟机所有状态 | UInt16 |
| vmStatus | 虚拟机状态 | UInt16[] |
| scope | C# WMI类库中ManagementScope类实例，用于同虚拟机建立连接 | ManagementScope |
| managementService | C# WMI类库中ManagementObject类实例，提供操作虚拟机的服务 | ManagementObject |
| virtualMachine | C# WMI类库中ManagementObject类实例，表示操作的虚拟机 | ManagementObject |

#### 函数及流程图

* VirtualMachine(String \_vmName, ManagementScope scopeM, ManagementObject managementServiceM)

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

VirutualMachine类构造函数。

【输入参数】

\_vmName，虚拟机名；scopeM，虚拟机所在的名空间；managementServiceM，虚拟机管理服务对象

【返回值】

无

【流程图】

否

是

开始

vmName = \_vmName;

scope = scopeM;

managementService = managementServiceM;

将上述输入参数赋值给类变量

结束

调用GetPerformanceSetting();设置performanceSetting属性。

virtualMachine = WmiUtilities.GetVirtualMachine(vmName, scope);

调用GetVitualMchine方法获取虚拟机对象，赋值给类变量virtualMachine

是否产生异常

抛出异常“没有找到虚拟机”

performanceSetting.EnabledState == 2

vmState = PowerOff

设置虚拟机状态为关机

vmState = PowerOn

设置虚拟机状态为开机

是

否

* GetPerformanceSetting()

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

获得虚拟机性能相关设置，并给performanceSetting类变量赋值。

【输入参数】

无

【返回值】

PerformanceSetting class类型

【流程图】

开始

返回performanceSetting

获取虚拟机的summaryInfo集合:

virtualSystemSettings=virtualMachine.GetRelated(“Msvm\_SummaryInformation”)

读取virtualSystemSettings，依次对performanceSetting各个字段进行赋值

* IsPowerOn()

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

判断是否开机。

【输入参数】

无

【返回值】

bool值，表示是否开机

【流程图】

开始

返回ret

调用GetPerformanceSetting ()函数，获取当前信息

是

performanceSetting.EnabledState == 2（开机状态为2）

否

bool ret = false

bool ret = true

* IsPowerOff()

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

判断是否关机。

【输入参数】

无

【返回值】

bool值，表示是否关机

【流程图】

开始

返回ret

调用GetPerformanceSetting ()函数，获取当前信息

是

performanceSetting.EnabledState == 3（关机状态为3）

否

bool ret = false

bool ret = true

* PowerOn()

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

执行虚拟机的开机操作。

【输入参数】

无

【返回值】

bool值，表示是否成功开机

【流程图】

开始

返回result

bool result = false;

ManagementBaseObject inParams = virtualMachine.GetMethodParameters("RequestStateChange")

将inParams参数设定为需要修改状态的格式

inParams["RequestedState"] = StartID; 将需要修改的状态设置为开机

ManagementBaseObject outParams = virtualMachine.InvokeMethod("RequestStateChange", inParams, null)

设置对应outParams参数

result =WmiUtilities.ValidateOutput(outParams, scope)

调用函数执行开机操作

是否出现异常

输出“无法开机”及异常原因

是

否

* PowerOff()

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

执行虚拟机的关机操作。

【输入参数】

无

【返回值】

bool值，表示是否成功关机

【流程图】

开始

返回result

bool result = false;

ManagementBaseObject inParams = virtualMachine.GetMethodParameters("RequestStateChange")

将inParams参数设定为需要修改状态的格式

inParams["RequestedState"] = SthutdownID; 将需要修改的状态设置为关机

ManagementBaseObject outParams = virtualMachine.InvokeMethod("RequestStateChange", inParams, null)

设置对应outParams参数

result =WmiUtilities.ValidateOutput(outParams, scope)

调用函数执行开机操作

是否出现异常

输出“无法开机”及异常原因

是

否

* ModifySettingData(String settingName, String settingData)

【所属类及所在文件】

VirutualMachine类，VMManagement.cs文件

【功能描述】

修改虚拟机的内存或CPU配置。

【输入参数】

settingName，要修改的配置项；settingData，修改后的数据

【返回值】

bool值，表示是否修改成功

【流程图】

①

判断是否修改CPU配置:

settingName.Substring(0,3) == “CPU”

获取要修改的字段名: settingName = settingName.Substring(4)

获取处理器配置对象集合:processorSettingDatas=

virtualSystemSetting.GetRelated(“Msvm\_ProcessorSettingData”)

获取集合processorSettingDatas中的第一个元素:

processorSettingData = processorSettingDatas.FirstItem()

获取processorSettingData[settingName]的类型type, 并将settingData转换为该类型,然后赋值:

processorSettingData[settingName] = (Type)SettingData

获取方法ModifyResourceSettings的输入参数

inParams=managementService.GetMethodParameters(“ModifyResourceSettings”)

设置输入参数inParams[“ResourceSettings”]=processorSettingData.GetText(...)

调用方法ModifyResourceSettings修改系统设置outParams=

managementService.InvokeMethod(“ModifyResourceSettings”, inParams, null)

验证调用ModifyResourceSettings的输出参数:

result=WmiUtilities.ValidateOutput(outParams, scope)

是否出现异常?

否

是

否

是

②

result = false; 获取虚拟机的配置对象:

virtualSystemSettings=WmiUtilities.GetVirtualMachineSettings(virtualMachine)

开始

输出“无法修改CPU参数”及异常原因

返回result

①

判断是否修改内存配置:

settingName.Substring(0,3) == “RAM”

获取要修改的字段名: settingName = settingName.Substring(4)

获取内存配置对象集合:memorySettingDatas=

virtualSystemSetting.GetRelated(“Msvm\_MemorySettingData”)

获取集合memorySettingDatas中的第一个元素:

memorySettingData= memorySettingDatas.FirstItem()

获取memorySettingData[settingName]的类型type, 并将settingData转换为该类型,然后赋值:

memorySettingData[settingName] = (Type)SettingData

获取方法ModifyResourceSettings的输入参数

inParams=managementService.GetMethodParameters(“ModifyResourceSettings”)

设置输入参数inParams[“ResourceSettings”]

调用方法ModifyResourceSettings修改系统设置outParams=

managementService.InvokeMethod(“ModifyResourceSettings”, inParams, null)

验证调用ModifyResourceSettings的输出参数:

result=WmiUtilities.ValidateOutput(outParams, scope)

是否出现异常?

否

是

是

②

输出“无法修改内存参数”及异常原因

返回result

### 动态调节接口模块

#### 功能

本模块提供了动态调节CPU、内存指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

#### 函数及流程图

* isDynamicMem(VirtualMachine VM)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

判断是否开启了动态内存。

【输入参数】

VM，VirtualMachine类实例

【返回值】

bool值，表示是否开启动态内存

【流程图】

开始

返回ret

bool ret = (VM.performanceSetting.RAM\_DynamicMemoryEnabled == true)

* AdjustMemorySize(VirtualMachine VM, UInt64 MemorySize)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

调整虚拟机的内存大小。

【输入参数】

VM，VirtualMachine类实例；MemorySize，调整后的内存大小

【返回值】

bool值，表示内存大小调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("RAM\_VirtualQuantity", Convert.ToString(MemorySize));

* AppendVMMemory(VirtualMachine VM, UInt64 MemorySizeBaseLine, UInt64 MemorySize, int rank)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

按比例扩展虚拟机内存大小。

【输入参数】

VM，VirtualMachine类实例；MemorySizeBaseLine，用户设置的基准内存；MemorySize，当前的内存大小；rank，扩展的等级

【返回值】

bool值，表示内存扩展大小是否成功

【流程图】

开始

返回ret

UInt64 appendSize = MemorySizeBaseLine / 8;内存扩展大小为基准内存的1/8

bool ret = VM.ModifySettingData("RAM\_VirtualQuantity", Convert.ToString(MemorySize + appendSize));

rank == 1

ret = false

是

否

UInt64 appendSize = MemorySizeBaseLine / 4;内存扩展大小为基准内存的1/4

bool ret = VM.ModifySettingData("RAM\_VirtualQuantity", Convert.ToString(MemorySize + appendSize));

rank == 2

是

是

否

否

* RecycleVMMemory(VirtualMachine VM, UInt64 MemorySizeBaseLine, UInt64 MemorySize, int rank)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

按比例回收虚拟机内存大小。

【输入参数】

VM，VirtualMachine类实例；MemorySizeBaseLine，用户设置的基准内存；MemorySize，当前的内存大小；rank，内存回收的等级

【返回值】

bool值，表示内存回收是否成功

【流程图】

开始

返回ret

UInt64 recycleSize= MemorySizeBaseLine / 8;内存回收大小为基准内存的1/8

bool ret = VM.ModifySettingData("RAM\_VirtualQuantity", Convert.ToString(MemorySize-recycleSize));

rank == 1

ret = false

是

否

UInt64 recycleSize= MemorySizeBaseLine / 4;内存回收大小为基准内存的1/4

bool ret = VM.ModifySettingData("RAM\_VirtualQuantity", Convert.ToString(MemorySize -recycleSize));

rank == 2

是

是

否

否

* AdjustMemoryWeight(VirtualMachine VM, int MemoryWeight)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

调整虚拟机的内存大小。

【输入参数】

VM，VirtualMachine类实例；MemoryWeight，调整后的内存权重

【返回值】

bool值，表示内存权重调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("RAM\_Weight", Convert.ToString(MemoryWeight));

* AdjustCPUCount(VirtualMachine VM, int Count)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

关机状态下，调整虚拟机分配的CPU核心数。

【输入参数】

VM，VirtualMachine类实例；Count，调整后的CPU核心数

【返回值】

bool值，表示CPU核心数调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("CPU\_VirtualQuantity", Convert.ToString(Count));

VM.IsPowerOn()

否

是

ret = false

* AdjustCPUWeight (VirtualMachine VM, int CPUWeight)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

调整虚拟机的CPU权重。

【输入参数】

VM，VirtualMachine类实例；CPUWeight，调整后的CPU权重

【返回值】

bool值，表示CPU权重调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("CPU\_Weight", Convert.ToString(CPUWeight));

* AdjustCPULimit(VirtualMachine VM, UInt64 CPULimit)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

调整虚拟机的CPU限制参数。

【输入参数】

VM，VirtualMachine类实例；CPULimit，调整后的CPU限制参数

【返回值】

bool值，表示CPU限制参数调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("CPU\_Limit", Convert.ToString(CPULimit));

* AdjustCPUReservation(VirtualMachine VM, UInt64 CPUReservation)

【所属类及所在文件】

DynamicAdjustment类，DynamicAdjustment.cs文件

【功能描述】

调整虚拟机的CPU保留参数。

【输入参数】

VM，VirtualMachine类实例；CPUReservation，调整后的CPU保留参数

【返回值】

bool值，表示CPU保留参数调节是否成功

【流程图】

开始

返回ret

bool ret = VM.ModifySettingData("CPU\_Reservation", Convert.ToString(CPUReservation));

### 性能接收模块

#### 功能

本模块接收所有域內性能监测器发送来的性能信息。

#### 设计

#### 异常和错误处理

#### 相关数据结构

1. 接口IServer

【数据类型】

interface

【功能】

定义服务契约IServer，其中包含传输性能信息的操作契约函数TransferPerfStr(string perf\_Str)

【详细数据结构】：

[ServiceContract]

public interface IServer

{

[OperationContract]

VMPerf TransferPerfStr(string perf\_Str);

}

1. DetectorServer类

【数据类型】

class

【继承关系】

继承IServer接口

#### 函数及流程图

### 负载分析模块

#### 功能

本模块提供获取Windows性能计数器部分内存、CPU性能相关指标的API。

#### 设计

#### 异常和错误处理

#### 相关数据结构

* MemoryBalancer类

【数据类型】

class

【功能】

内存负载均衡类，包含监测内存所需要的的属性，以及进行定时内存负载均衡调节的方法；

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| currentVirtualMachine | 系统内唯一的实例名 | string |
| percentageThredHold | 虚拟机名 | string |
| currentVMPerf | 虚拟机ID | string |
| NumberOfProcessors | CPU核心数 | UInt16 |
| EnabledState | 预设CPU限制大小 | UInt16 |
| HealthState | 虚拟机健康状态 | UInt16 |
| ProcessorLoad | 当前虚拟机CPU占总CPU资源的负载率 | UInt16 |
| ProcessorLoadHistory | 记录前100次CPU负载率的数组 | UInt16[] |
| MemoryUsage | 分配的内存大小 | UInt64 |
| CPU\_Reservation | CPU保留 | UInt16[] |
| CPU\_Limit | CPU限制 | UInt64 |
| CPU\_Weight | CPU权重 | UInt32 |
| RAM\_VirtualQuantity | 分配的内存大小 | UInt64 |
| RAM\_Weight | 内存权重 | UInt32 |
| RAM\_DynamicMemoryEnabled | 是否打开动态内存 | bool |

* VirtualMachine类

【数据类型】

class

【功能】

表示虚拟机的属性及操作虚拟机的方法；

【详细数据结构】

|  |  |  |
| --- | --- | --- |
| 属性 | 含义 | 类型 |
| vmName | 虚拟机名 | UInt16 |
| performanceSetting | 与性能相关的虚拟机设置 | UInt16 |
| VirtualMachineStatus | 枚举虚拟机所有状态 | UInt16 |
| vmStatus | 虚拟机状态 | UInt16[] |
| scope | C# WMI类库中ManagementScope类实例，用于同虚拟机建立连接 | ManagementScope |
| managementService | C# WMI类库中ManagementObject类实例，提供操作虚拟机的服务 | ManagementObject |
| virtualMachine | C# WMI类库中ManagementObject类实例，表示操作的虚拟机 | ManagementObject |

#### 函数及流程图

### 主控模块

#### 功能

本模块启动性能接收模块接收域内的性能信息，启动虚拟机状态监控模块监控虚拟机开、关机等状态信息，初始化负载分析模块，对各虚拟机进行负载均衡调节。

#### 设计

#### 异常和错误处理

#### 相关数据结构

无

#### 函数及流程图

* Main(string[] args)

【所属类及所在文件】

Program类，Program.cs文件

【功能描述】

程序main函数。

【输入参数】

string[] args，输入的命令行参数

【返回值】

无

【流程图】

(!File.Exists(MpcVmConfigpath

开始

返回

VMState vmState = new VMState(MpcVmConfigpath);

初始化虚拟机状态监控类实例vmState

vmState.StartMessageReceiver();

调用StartMessageReceiver()函数接收多域PC发送来的虚拟机状态改变信息

!File.Exists(MpcVmConfigpath)

输出：控制台输出配置文件不存在

string HvAddr = "hypervnb://00000000-0000-0000-0000-000000000000/C7240163-6E2B-4466-9E41-FF74E7F0DE47";前面都是0表示接收所有虚拟机发来的连接请求，后面为在注册中注册的可信程序ID。

Server detectorServer = new Server(HvAddr);

输入负载均衡器的URI，初始化性能接收模块的服务端

是

否

var task = new Task(() =>{detectorServer.StartUpServer();})

task.Strart();

将服务器的StartUpServe()函数作为一个线程启动，接收域内发送来的性能信息。

输入参数，初始化负载均衡模块，LoadBalancer testLoadBalancer；

testLoadBalancer.setDetectorServer(detectorServer);

设置负载均衡模块的消息接收服务器

testLoadBalancer.BalanceByTime();

调用BalanceByTime()函数定时分析性能信息，必要时进行资源调节

Console.ReadLine();

是否捕获异常

输出异常消息

否

是

## 初步测试实验数据

开机时间

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CPU核心数 | 内存大小 | 保留 | 限制 | 是否开机负载均衡器 | 开机时间 |
| 2 | 2046MB | 100 | 100 | 是 | 29s |
|  |  |  |  |  | 21.21s/20.15s |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

重要参考文献：

<https://docs.microsoft.com/en-us/azure/monitoring/infrastructure-health/vmhealth-windows/winserver-memory-pagespersec>

<https://docs.microsoft.com/zh-cn/virtualization/hyper-v-on-windows/user-guide/make-integration-service#register-a-new-application>).



