**数据中心服务器能耗管理系统设计**

# 阶段目标

## 第一期功能

监控中心

自动发现和初始化接口

（第二期）

监控接口NodeManager1.5

监控数据库

Web 服务器

Web 前端

系统日志

监控接口NodeManager2.0

Web 层

监控层

web数据库

监控接口DNM

(第二期)

| **功能列表 (第一期)** | | | | |
| --- | --- | --- | --- | --- |
| **项目** | **说明** | **服务器节点级** | **群组级 (物理/逻辑)** | **服务器属性筛选(IP, IP Range, 主机名)** |
| 实时监测数据 | 实时监测目标服务器如下监测信息   * 即时（当前）功耗 * 平均功耗 * 最大功耗 * 最小功耗 * 即时（当前）进风口温度 * 平均进风口温度 * 最大进风口温度 * 最小进风口温度   实时监测时间段可设置（30 min – 1 month）  监测数据处理：某些目标服务器在关机状态无功耗读值时，按黑名单方式处理 | Y | Y | Y |
| 监测数据分析 | 在web界面上提供群组或节点   * 在当前实时显示时间段内的的最大，最小以及平均功耗和温度数据 * 在指定时间段内（非实时显示）的最大，最小以及平均功耗和温度数据 |  |  |  |
| 服务器监测数据回溯查询 | 可选择指定时间段内的监测目标服务器(s)如下监测信息   * 即时（当前）功耗 * 平均功耗 * 最大功耗 * 最小功耗 * 即时（当前）进风口温度 * 平均进风口温度 * 最大进风口温度 * 最小进风口温度   最长回溯事件为过去1年 | Y | Y | Y |
| 监测数据采集处理周期 | * 采集周期可设定 （P采集） * 采集到的原始数据直接入库，数据处理周期（P处理）可设定，数据处理周期等于或大于采集周期（整数倍） * P采集=Pbase\*m * P处理=P采集\*n * m,n 整数 | Y | Y | N/A |
| 功耗控制策略注1 | 可按节点，群组或其它属性值批量设定如下策略， 设定策略时可选择设置时即Enable或Disable   * 系统整机功耗控制策略 * CPU功耗控制策略 * 内存功耗控制策略 * 基于进风口温度控制策略 * 最低功耗控制策略（整机控制，目标值为power draw range最小值）   可按节点，群组或按其它属性值批量调整功耗策略   * Disable * Enable * Remove | Y | Y注2 | Y |
| 服务器电源控制 | * 显示目标服务器当前开关机状态（ACPI Power status）   + 查询周期另行设定 * 目标服务器远程开关机（ACPI Power On/Off） | Y | Y注3 | Y |
| 群组管理 | * 物理组：数据中心(Data Center)->机房(Server Room)->机架列(Row)->机架(Rack)->机箱(Chassis)。 物理组依上述顺序有上下层级关系   每台服务器仅能属于一个物理组。  用户不可自定义物理组。   * 逻辑组：服务器与逻辑组为多对多关系 | Y | Y | N/A |
| 插件支持 | 通过不同插件实现跨平台目标服务器功耗和温度监测与控制  一期支持插件如下：   * Node Manager 1.5 (支持/不支持进风口温度Trigger) * Node Manager 2.0 (支持/不支持进风口温度Trigger) | Y | N/A | N/A |
| 目标服务器筛选 | 支持按照目标服务器信息值筛选目标服务器   * IP （管理，内网，外网), * IP Range * 物理组 * 逻辑组 * 主机名 * 备注项内容 * 开关机状态 | Y | Y | Y |
| 服务器功耗管理功能查询 | 显示目标服务器节点是否支持以下功能：   * 功耗监测（整机、内存、CPU） * 整机功耗控制 * CPU功耗控制 * 内存功耗控制 * 进风口温度监测 * 远程开关机 | Y | Y注4 | Y |
| 服务器功耗策略查询 | 可查询目标服务器当前部署功耗控制策略及如下信息   * 策略类型 * 策略作用Domain注5 * 策略ID注6 * 策略触发阀值（如进风口温度阀值） * 策略控制目标(目标功耗值) * 策略状态(enable/disable) | Y | Y注7 | Y |
| 目标服务器系统日志（System Event Log） | * 定期从目标服务器采集SEL * 采集周期可设置 | Y | N/A | N/A |
| 报警处理 | 管理系统应对如下事件进行报警处理：   * 目标服务器功耗和进风口温度监测值超过用户设定阀值，触发阀值采样周期可设定。 * 目标服务器不能满足策略目标值。 * 报警以 web 界面提示或 email 方式送达用户。 |  |  |  |
| 黑名单 | 当采集目标服务器监测数据在给定采集周期内持续超时，放入黑名单。  目标服务器放入黑名单时，在管理系统上应用相应明显提示  放入黑名单时，同时记录该目标服务器开关机状态(Power on / Power Off/ BMC No response) |  |  |  |
| 重检测功能 (Rediscovery) | 重新检测服务器的能力，若在黑名单内的服务器经检测正常，从黑名单移除。  可在黑名单里单选或多选目标服务器进行重检测 | Y | N/A | N/A |
| 查询/编辑服务器信息 | * BMC IP地址 * BMC 用户名 * **注：BMC密码不显示** * 目标服务器内网IP * 目标服务器外网IP * 目标服务器主机名 * **目标服务器资产标示(Asset Tag)** * 目标服务器在机架中位置 * 目标服务器所属组（物理组，逻辑组） * 目标服务器倍增数(multiplier) * 目标服务器备注内容 | Y | N/A |  |
| 目标服务器添加 | * 在UI上逐个添加。 * 通过配置文件批量导入。导入时不覆盖相同节点（以 bmc ip 标识）。 * 配置文件为 excel 格式，内容依数据库目标服务器表而定。 | Y | N/A | N/A |
| 管理系统日志 | 可按时间、类型查看系统事件。  管理系统日志记录信息要求待定 | N/A | N/A | N/A |
| 监控接口 | * IPMITool | N/A | N/A | N/A |
| 开发环境 | * RHEL 6.2 X86-64 * 整个系统分为 web 层与监控层 * 监控层：Python 2.7 / 多线程方式 * 监控层数据库：MySQL XXXX   + Web数据库： * Web层方案   + Web Server: Python Flask xxxx w/ url interface   + Web前端：JS   + Web数据库：MySQ，用户权限等以及Web前端自定义 | N/A | N/A | N/A |
| 管理系统服务器分级 | 一期仅支持单点管理服务器 |  |  |  |
| 管理系统安装 | 尽量简单 |  |  |  |
| 使用者权限 | Web层两种用户：   * 管理员超级用户，具备所有权限 * 普通用户：用户权限由管理员定义权限列表待商讨 |  |  |  |

## 第二期目标：

| **项目** | **服务器属性筛选(IP, IP Range, 主机名)** |
| --- | --- |
| 开源 |  |
| Web层采用url接口 | * Web层方案   + Web Server: Python Flask xxxx w/ url interface   + Web前端：用户自定义   + Web数据库：MySQL |
| 目标服务器接口基于UDP | 基于一期实际部署多线程模式性能 |
| 发现最优线程数 | 如果不是直接从udp通信，需要用多线程方式提高效率。第二期计划通过测试自动获得性能最优的线程数量。以独立工具模式提供，使用者依据测试工具测试结果调整管理系统线程设定配置文件 |
| 多层级(multi-layer) 监控系统 | 以支持巨量目标服务器 (中心监控服务器+次级监控服务器) |
| 新目标服务器接口 | * DNM (Node Manager Extended Edition) * 其它接口 |
| 目标服务器自动发现和初始化 | 自动发现接口：从指定IP段自动发现目标服务器，确定其类型和性能。 |
| 目标服务器手动添加复核 | 复核手动添加的目服务器中由用户输入的属性，避免出错。 |
| 监测数据统计，筛选，分析功能 | * Web UI上提供统计筛选功能，如：   + 功耗或温度，低于/高于/某一范围内的目标服务器统计   + 其它等 |
| 数据库归档和备份： | 数据库备份与归档，如监测数据，目标服务器属性，群组设定等 |
| 多操作系统支持 | 管理系统支持更多操作系统，如Debian or Ubuntu |
| 便捷安装 |  |

* UDP接口：第一期的监控接口通过ipmitool与目标服务器通信，由于ipmitool发出命令之后会阻塞直到远端回复，对效率影响很大。若需求超出 ipmitool 的负载能力，第二期计划直接从UDP层收发数据，简化层级，提高通信效率及监控能力。

UDP

ipmitool

第一期

第二期

监控接口

RMCP

UDP接口

* URL接口：有些客户原本已有web层的监控工具，web后端不是用Python实现的。第二期计划从我们的web后端向客户的web后端提供URL接口，将我们的监控层与客户原有的web层整合在一起。

我们的Web后端（Python）

我们的Web前端

客户方原有的

Web前端

客户方原有的

Web后端

URL接口

第一期

第二期

* 自动发现接口：从指定IP段自动发现目标服务器，确定其类型和性能。
* 手动添加复核：复核手动添加的目服务器中由用户输入的属性，避免出错。
* 发现最优线程数：如果不是直接从udp通信，需要用多线程方式提高效率。第二期计划通过测试自动获得性能最优的线程数量。以独立工具模式提供，使用者依据测试工具测试结果调整管理系统线程设定配置文件
* 新的接口及现有接口的功能扩展。
* 在各种操作系统上的便捷安装。
* 多层级(multi-layer) 监控系统以支持巨量目标服务器 (中心监控服务器+次级监控服务器)
* 数据归档和备份：监测数据，目标服务器属性，群组设定
* Open Soure

# 开发环境：



Web 层提供管理界面，通过监控层执行管理员的各项指令。管理员可由此读取各项监控数据，设置策略（Policy），管理被监控服务器节点。

监控层执行日常监控任务。此外，也向 web 层提供一组 Python API 接口，用于执行管理员的下述指令：

* 读取监控数据：按照时间起止、周期、显示单位（单台服务器、各级物理组、逻辑组）、显示类型（最大值、最小值、平均值）读取被监控服务器的温度与功耗的实时数据。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_monitor\_data(  start\_time = sysdefault,  end\_time = now,  cycle = sysdefault,  obj\_server = None,  obj\_group = None,  columns = ()  ) | start\_time: ‘yyyy-MM-dd hh:mm:ss’  读取数据的起始时间。缺省值由监控层设定。  end\_time: ‘yyyy-MM-dd hh:mm:ss’  读取数据的截止时间。缺省值为现在。  cycle: int  读取数据的周期的编号。缺省值可在编辑周期时设定。  obj\_server: int or None  目标服务器编号。用于读取单台服务器的数据。  obj\_group: int or None  目标组编号。用于按组读取服务器数据。  obj\_server 与 obj\_group 不能同时为空。若二者皆有值，前者优先。  columns: tuple(int,)  读取的数据项类型。最多可读取八种数据项，计有：  MAX\_TEMP－最高温度  MIN\_TEMP－最低温度  AVR\_TEMP－平均温度  CUR\_TEMP－即时温度  MAX\_POWER－最大功耗  MIN\_POWER－最小功耗  AVR\_POWER－平均功耗  CUR\_POWER－即时功耗 | dict {  ‘result’: result,  ‘start\_time’ : start\_time,  ‘end\_time’ : end\_time,  ‘cycle’ : cycle,  ‘obj\_server’ : obj\_server,  ‘obj\_group’ : obj\_group,  ‘data’ : {  ‘clm\_num’ : clm\_num,  ‘data\_len’: data\_len,  ‘data\_list’:[  [val0, val1……], #周期项  #[数据项，数据项……]。  ……  ]  }  }  result: int  执行结果。非负值表示成功。不成功时没有以下各项。  start\_time: ‘yyyy-MM-dd hh:mm:ss’  数据的实际起始时间。  end\_time: ‘yyyy-MM-dd hh:mm:ss’  数据的实际截止时间。  cycle: 同入口参数。  obj\_server或obj\_group: 同入口参数。这两个值只有生效的一个存在于DICT中，另一个不存在。  clm\_num: int  实际返回数据中每一周期的数据项数。即输入参数columns的长度。  data\_len: int  实际返回数据的周期项数。  ‘data\_list’: list[list[int or None]]  读出来的所有数据。按时间从新到老排列。每一周期项含有一个周期中的几个数据项，数据项的内容和次序与输入参数columns相同。若某一数据项不存在，返回None。 |

* 导出监控数据：将用户指定的服务器或组在指定时段内的监控数据导出为指定格式文件。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| monidata\_export(  file\_name,  file\_type,  obj\_ids = (),  grp\_ids = (),  start\_time = sysdefault,  end\_time = now,  ) | file\_name: unicode  导出文件名及路径。  file\_type:str  导出文件类型。如：’xml’,’csv’。  obj\_ids:tuple( int)  需要导出的目标服务器编号。  grp\_ids:tuple( int)  需要导出的组编号。  实际导出的服务器为以上两个参数的并集。  start\_time: ‘yyyy-MM-dd hh:mm:ss’  导出数据的起始时间。缺省值由监控层设定。  end\_time: ‘yyyy-MM-dd hh:mm:ss’  导出数据的截止时间。缺省值为现在。 | dict {  ‘result’: result,  ‘fail\_infos’: [  fail\_info\_0,……  ],  }  result: int  执行结果。非负值表示成功。成功时没有后续项。  ‘fail\_infos’: 其中每项为一条失败信息，unicode。 |

* 编辑系统global参数：数据采样周期、最大节点数、最小统计周期、当前统计周期。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| edit\_sysparams(  \*\*kwds,  ) | kwds: dict  需要改变的参数，含以下几项：  sel\_cycle: int  数据采样周期（秒）。  min\_cycle: int  最小统计周期（秒）。不小于采样周期。  max\_nodenum: int  最大节点数。超出报警。  cur\_cycle: int  为最小统计周期的倍数。 | dict {  ‘result’: result,  ‘fail\_infos’: [  fail\_info\_0,……  ],  }  result: int  执行结果。非负值表示成功。成功时没有后续项。  ‘fail\_infos’: 其中每项为一条失败信息，unicode。 |

* 读取系统global参数：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_sysparams() |  | dict {  ‘result’: result,  ‘sel\_cycle’: sel\_cycle,  ‘min\_cycle’: min\_cycle,  ‘max\_nodenum’: max\_nodenum,  ‘cur\_cycle’: cur\_cycle,  }  result: int  执行结果。非负值表示成功。失败时没有后面各项。  后四项含义同edit\_sysparams的入口参数。 |

* 添加目标服务器：手动逐个添加。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| new\_objserver(  bmc\_ip,  inner\_ip,  outer\_ip = ‘’,  asset\_tag = ‘’,  host\_name = ‘’,  user\_name = ‘’,  password = ‘’,  u\_pos = 0,  groups = (),  notes = u‘’,  multiplier = 1,  )  添加之后系统自动读取服务器的协议类型和性能参数。  策略在 set\_policy中设置。  服务器的硬件信息可放在备注中。 | bmc\_ip: str  新增目标服务器的bmc\_ip。  inner \_ip: str  新增目标服务器的内网IP。  outer \_ip: str  新增目标服务器的外网IP。  asset\_tag: str  新增目标服务器的资产标识。  host\_name: str  新增目标服务器的主机名。  user\_name: str  新增目标服务器的监控用户名。  password: str  新增目标服务器的监控用户密码。  u\_pos: int  服务器在机架中的位置。  groups: tuple(int)  新增目标服务器所属的组号（物理组及逻辑组）。  notes: Unicode  备注  multiplier: int（自然数）  服务器乘数。大于1时，表示在同一机箱内有多台目标服务器，且从任一节点采得的功耗原始数据是机箱总功耗，应除以multiplier得到每台服务器的平均功耗。 | dict {  ‘result’: result,  ‘obj\_id’: obj\_id,  ‘server\_info’: server\_info  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  obj\_id: int  新增目标服务器的编号。  sever\_info: Unicode(暂定)  自动检测出的服务器特性。 |

* 读取目标服务器数据：

|  |  |  |
| --- | --- | --- |
| get\_objserver(  obj\_id  )  含服务器涉及的策略。  策略内容用get\_policy读取。 | obj\_id: int  服务器编号。 | dict {  ‘result’: result,  ‘obj\_id’: obj\_id,  ‘bmc\_ip’: bmc\_ip,  ‘inner\_ip’: inner\_ip,  ‘outer\_ip’: outer\_ip,  ‘asset\_tag’: asset\_tag,  ‘host\_name’: host\_name,  ‘user\_name’: user\_name,  ‘groups’: [grp\_id0, grp\_id1……],  ‘policy’: {  ‘num’: num,  ‘list’: [  {  ‘plc\_id’: plc\_id,  ‘enabled’: enabled,  },  ……  ]  }  ‘capability’: {  ‘power\_monitor’: power\_monitor,  ‘power\_limit’: power\_limit,  ‘inlet\_temp: inlet\_temp,  ‘remote\_control’: remote\_control,  }  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  obj\_id: int  服务器编号。  从bmc\_ip到user\_name的六项同new\_objserver的入口参数。  ‘groups’: [int……]  每项是一个组编号。含逻辑组。  ‘policy’: dict  服务器策略。其中：  ‘num’: int  所有策略数。  ‘list’: list  每项是一个策略。其中：  ‘plc\_id’: int  策略编号。  ‘enabled’: bool  策略是否处于活动状态。  ‘capability’: dict  服务器性能。其中：  power\_monitor: tuple( int )  对功耗查询的支持，最多有三个值，表示整体、CPU、内存三个 domain。  power\_limit: tuple( int )  对功耗控制的支持，最多有三个值，表示整体、CPU、内存三个 domain。  inlet\_temp: int  对读取进风口温度的支持。  remote\_control: int  对远程开关机的支持。 |

* 编辑目标服务器：改变IP及组属关系、备注等。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| edit\_objserver(  obj\_id,  \*\*kwds,  )  服务器的策略用 set\_objpolicy设置。 | obj\_id: int  目标服务器编号。  kwds 中可设定以下三项，在需要改变原有设定时设置：  bmc\_ip,  inner\_ip,  outer\_ip ,  asset\_tag,  host\_name,  user\_name,  password,  u\_pos,  notes,  以上九项同new\_objserver的入口参数。  in\_groups: tuple(int)  每一项为设定目标服务器所属的组号（物理组及逻辑组）。  若新设目标服务器属于某个物理组，则自动删除此服务器与其它物理组的关系。  out\_groups: tuple(int)  每一项为取消目标服务器所属的组号（物理组及逻辑组）。  notes: Unicode  备注。 | dict {  ‘result’: result,  ‘fail\_infos’: [  fail\_info\_0,……  ],  }  result: int  执行结果。非负值表示成功。成功时没有后续项。  ‘fail\_infos’: 其中每项为一条失败信息，unicode。 |

* 删除目标服务器：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| del\_objserver(  obj\_id,  ) | obj\_id: int  服务器编号。 | bool  执行结果。 |

* 选取目标服务器集合：

|  |  |  |
| --- | --- | --- |
| * 接口 | 入口参数 | 返回值 |
| get\_objserver\_set( \*\*kwds ) | kwds: dict  选取的条件。包括：  bmc\_ips: tuple( str )  按 bmc\_ip 集合选取。  inner\_ips: tuple( str )  按内网ip 集合选取。  outer\_ips: tuple( str )  按外网ip 集合选取。  bmcip\_range: tuple( str1, str2 )  按 bmc\_ip 范围选取。  innerip\_range: tuple( str )  按内网ip 范围选取。  outerip\_range: tuple( str )  按外网ip 范围选取。  grps: tuple( int )  按组号集合选取。  hostnames: tuple( str )  按主机名选取。  power\_status: int  按开关机状态选取。  如有多个条件，彼此是与关系。 | tuple( int )  选出的目标服务器编号集合。 |

* 读取组信息：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_groups( \*args ) | args: grp\_id0, grp\_id1……  需要读取组的编号。为空时读取全部组。 | dict {  ‘result’: result,  ‘grp\_num’: grp\_num,  ‘groups’: [  {  ‘grp\_id’: grp\_id,  ‘type’: type,  ‘parent’: parent,  ‘notes’: notes,  }  ]  }  result: int  执行结果。非负值表示成功。不成功时没有以下各项。  grp\_num: int  所有组数。  ‘groups’: list  每项是一个组。其中：  grp\_id: int  组编号。  type: int, PHISICAL与LOGICAL之一。  组类型。物理组或逻辑组。  parent: int  上级组编号。逻辑组无此项。  notes: unicode  备注。 |

* 添加组：分为物理组与逻辑组。物理组有上下级关系，每台服务器只能设置一个物理组。

物理组由小到大有：机箱、机架、机架列、机房、数据中心。也可自定义其它物理组。

逻辑组之间彼此平行，同一服务器可设置多个逻辑组。服务器也可不属于任何组。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| new\_group(  type,  name,  parent = None,  notes = u‘’  ) | type: int, PHISICAL与LOGICAL之一。  组类型。物理组或逻辑组。  name: Unicode  组名。  parent: int or None  上级组编号。逻辑组无此项。  notes: Unicode  备注。 | dict {  ‘result’: result,  ‘grp\_id’: grp\_id,  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  grp\_id: int  新增组编号。 |

* 编辑组：可编辑组名、上级组和备注，不可编辑组类型。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| edit\_group(  grp\_id,  \*\*kwds,  ) | grp\_id: int  待编辑组的编号。  kwds: dict  改变的属性。包括：  name: Unicode  组名。  parent: int  上级组编号。逻辑组无此项。  notes: Unicode  备注。  组类型type不可编辑。 | bool  执行结果。 |

* 删除组：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| del\_group(  grp\_id,  del\_childs = False,  ) | grp\_id: int  待删除组的编号。  del\_childs: bool  是否同时删除其下级组。对逻辑组无效。  若不删除下级组，则自动将下级组的上级组设定为自身的上级组。 | bool  执行结果。 |

* 设置目标服务器的组属关系：见edit\_objserver。
* 读取组中的目标服务器：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_groupservers(  grp\_id,  ) | grp\_id: int  组编号。 | dict {  ‘result’: result,  ‘server\_num’:server\_num,  ‘server\_ids’: [  Id0, ……  ]  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  server\_num: int  组中的目标服务器数量。  ‘server\_ids’: tuple  其中每项是一个目标服务器编号。 |

* 读取控制策略内容：

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_policy (  plc\_id,  ) | plc\_id: int  策略的编号。 | dict {  ‘result’: result,  ‘plc\_id’: plc\_id,  ‘plc\_content’: {  ‘domain’: domain,  ‘trig\_temp’: trig\_temp,  ‘threshold’: threshold  }  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  plc\_id: int  读取策略编号。  在’plc\_content’中：  domain: int  策略作用域。为三值之一：DM\_SYS, DM\_CPU, DM\_MEM。  trig\_temp: int  触发当前策略的温度。  threshold: int  当前策略的调控目标。 |

* 设置目标服务器控制策略：依次设置增、删、激活、阻止四种操作。可先添加再激活。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| set\_objpolicy(  obj\_ids = (),  add\_plces = (),  del\_plces = (),  enable\_plces = (),  disable\_plces = (),  ) | obj\_ids:tuple( int)  目标服务器的编号。  add\_plces: tuple(int,)  添加的策略编号。  del\_plces: tuple(int,)  删除的策略编号。  enable\_plces: tuple(int,)  激活的策略编号。  disable\_plces: tuple(int,)  阻止的策略编号。 | dict {  ‘result’: result,  ‘obj\_servers’: {  obj\_Id0 : {  ‘add\_plces’: [  plc\_id0……  ],  ‘del\_plces’: [  plc\_id0……  ],  ‘enable\_plces’: [  plc\_id0……  ],  ‘disable\_plces’: [  plc\_id0……  ],  }  …… #每个目标服务器各有一组返回值。  }  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  ‘obj\_servers’: dict  其中每个键obj\_idx为一个目标服务器的编号，值为对此目标服务器设置策略的结果，其中：  ‘add\_plces’中每一项是添加成功的策略编号。其它三项与此相类。 |

* 读取目标服务器控制策略：见 get\_objserver。
* 批量导出目标服务器到XML文件：导出全部或某一组的服务器资料。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| file\_export(  file\_name,  grp\_id = None,  ) | file\_name: Unicode  导出文件名及路径。  grp\_id: int or None  导出的组号。默认导出所有目标服务器。 | bool  执行结果。 |

* 从EXCEL文件批量导入目标服务器：导入上条命令生成的文件，可选择导入到某一目标组。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| file\_import(  file\_name,  grp\_id = None,  ) | file\_name: Unicode  导入文件名及路径。  grp\_id: int or None  导入的组号。默认导入服务器无组。 | bool  执行结果。 |

* 从设定的IP网段中搜索目标服务器：以与前述导出文件相同的格式返回web界面。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| find\_objservers(  ip\_start,  ip\_end,  connector = None  )  这是第二期的功能。 | ip\_start: str  搜索IP起点。  ip\_end: str  搜索IP终点。  connector: int or None  指定连接方式。 | dict {  ‘result’: result,  ‘find\_num’: find\_num,  ‘obj\_servers’: [  {  （待定）  }  ],  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  ‘find\_num’: int  找到的有效服务器数。 |

读取日志：可设定显示日志的起止时间和类型（各有不同的优先级）。

|  |  |  |
| --- | --- | --- |
| 接口 | 入口参数 | 返回值 |
| get\_log(  start\_time,  end\_time,  types = (),  ) | start\_time: ‘yyyy-MM-dd hh:mm:ss’  读取数据的起始时间。缺省值为最小时间。  end\_time: ‘yyyy-MM-dd hh:mm:ss’  读取数据的截止时间。缺省值为现在。  types: tuple(int)  读取的日志类型。 | dict {  ‘result’: result,  ‘start\_time’ : start\_time,  ‘end\_time’ : end\_time,  ‘log\_num’: log\_num,  ‘logs’: [  log\_text,  ……  ],  }  result: int  执行结果。非负值表示成功。不成功时没有其它项。  start\_time: ‘yyyy-MM-dd hh:mm:ss’  数据的实际起始时间。  end\_time: ‘yyyy-MM-dd hh:mm:ss’  数据的实际截止时间。  ‘log\_num’: int  读到的日志条数。  log\_text: Unicode  每条日志的内容。 |

监控中心以多线程方式运行任务。任务有采样、设置、查询三种。在每个线程中调用对应的监控接口执行任务。每个任务可能需要运行多条指令（ipmitool）。

监控数据库表结构：

* 目标服务器表：服务器编号、IP、物理组编号、监控协议类型、实时温度读取算法的参数，实时读取功耗算法的参数(Node Manager 1.5, 2.0;等不需要此参数)。
* 监控数据显示周期表：周期编号、周期时长、保存数据时间（超期数据移到备份库）、保存数据类型（最大、最小、平均、末）、对应的监控数据表名（见下）。
* 监控数据表：服务器编号、监控周期、周期末时间、最大温度、最小温度、平均温度、最大功耗、最小功耗、平均功耗。（如果性能需要，此表可根据不同的监控周期拆成多个表）
* 服务器策略表：服务器编号、策略编号、策略状态。
* 策略表：策略编号、策略内容、备注。
* 组表：组编号、组名、组类型、上级组编号、备注。（含物理组与逻辑组）
* 服务器逻辑组表：服务器编号、逻辑组编号。（服务器与逻辑组是多对多关系，故单设一表。服务器与物理组是多对一关系，故放在服务器表中。）
* 管理员表：用户名、密码、权限。
* 备份表：用于保存监控数据表的超期数据，字段同监控数据表。
* 日志表：时间、行为、类型。（用于记录各种管理员操作。系统故障等事件记录在系统日志（文件）里）

Web 数据库是 web server 的一部分，用于存放无须通过监控中心使用的 web界面相关数据。

Web层设置用户权限：完全由web地层实现，与监控层无关。

* 增加用户：用户有多项可设置的权限。
* 编辑用户：改变权限、邮箱及密码。
* 删除用户。
* 节点添加
* 导入导出
* 组设置
* 组查询
* 策略管理
* 策略查询
* 周期设定
* log导出

监控层

监控中心

任务队列

结果队列

目标服务器接口

（在多线程中）

缓存

数据库

Web 接口