



LDOP 工具优化方案字段说明

麒麟软件有限公司

2025 年 02 月 24 日

版本变更说明

日期	版本号	编写人	审核人	发布说明
2024/10/25	V1.0	郭升升		初版报告
2025/02/20	V1.1	郭升升		添加进程绑核功能
2025/02/24	V1.2	郭升升		添加多场景切换功能

目 录

LDOP 工具优化方案字段说明	1
1. 优化方案概述	4
2. 场景优化文件	4
2.1 全局字段	4
2.2 场景字段	5
2.3 方案字段	5
2.4 文件编写	7
3. 非场景优化文件	9
3.1 全局字段	9
3.2 方案字段	10
3.3 文件编写	11

1. 优化方案概述

LDOP 工具优化方案文件是基于 `yaml` 文件格式进行编写，这种编写风格可以提升文件的可读性和可维护性，**整个优化方案文件的数值均保存为字符串形式。**

优化方案文件的字段可分为三大类：全局字段、方案字段和场景字段。全局字段用于描述整个优化方案的相关信息，方案字段则详细列出优化方案中包含的具体优化策略。场景字段与进程监控紧密相关，用于描述监控到的具体进程场景。

根据是否涉及进程监控，优化方案文件分为场景优化文件和非场景优化文件。场景优化文件必须包含场景字段，而非场景优化文件则专注于全局优化。无论是场景优化文件还是非场景优化文件，都包含全局字段和方案字段。

场景优化文件支持多场景进程监控，但在同一时间只能有一个场景优化策略生效。非场景优化文件则在同一时间只能应用一个优化策略，并且只能与一个生效的场景优化策略叠加使用。

2. 场景优化文件

场景优化文件是针对进程监控场景设计，旨在对特定进程及其子进程进行优化。当系统中某个进程及其派生子进程需要特定的优化措施时，可以编写并应用场景优化文件。该文件在应用后，会首先注册进程监控事件，只有当检测到符合预设条件的进程场景时，才会执行其定义的优化策略。此外，如果对子进程也有额外的优化需求，还可以针对子进程单独制定优化策略，从而实现更精细化的优化管理。

2.1 全局字段

全局字段目前共涉及 6 个名称，均为一级节点。目前全局变量所有字段均为预留字段，仅保留在 `yaml` 文件中，无实际意义。仅方便后续扩展使用，具体描述如表 2-1 所示：

字段名称	字段类型	节点级别	字段说明
identifier	字符串	一级节点	方案的唯一标识 id，不同优化方案使用不同的进程标识 id，可以通过 <code>uuidgen</code> 命令生成。(预留字段)

name	字符串	一级节点	优化方案的名称，基于此名称进行方案相关操作（预留字段）
arch	字符串	一级节点	优化方案可以在哪些架构上生效，例如 x86_64, aarch64, loongarch64。(预留字段)
os	字符串	一级节点	系统版本，例如 V10 V11(预留字段)
priority	字符串	一级节点	当前优化方案的优先级，0-100，数值越大，优先级越高(预留字段)
exclusive	字符串	一级节点	设置方案是否为独占运行，若设置独占运行，则 ldop 工具只能运行这一个方案。true:独占 false:非独占(预留字段)

表 2-1

2.2 场景字段

场景字段根据节点级别共涉及 4 个名称，其中 1 个一级节点，3 个二级节点。此部分内容为场景优化文件的核心，必须存在，如表 2-2 所示：

字段名称	字段类型	节点级别	字段说明
scene	字典类型	一级节点	场景字段标识，此字段下面的内容属于场景设置的具体方案
name	字符串	二级节点	场景优化文件的场景名称，ldop 工具根据场景名进行场景识别
type	字符串	二级节点	场景类型，ldop 工具识别到场景后，根据场景类型进行相应操作，目前场景类型为 process
preset	字符串	二级节点	预设进程名，ldop 工具识别到场景和类型后，根据预设进程名进行进程监控，判断是否进入场景。

表 2-2

场景字段转化成 yaml 文件内容后，如图 2-1 所示：

```
# 场景字段,以unixbench场景为例,按照下列格式编写
scene:
  name: 'unixbench'
  type: 'process'
  preset: 'Run'
```

图 2-1

2.3 方案字段

方案字段根据节点级别共涉及 8 个名称，其中 1 个一级节点，1 个二级节点，2 个三级

节点，1 个四级节点，3 个五级节点。具体描述如表 2-3 所示：

字段名称	字段类型	节点级别	字段说明
rules	列表	一级节点	所有规则整体字段标识，仅有一个。此字段可以包含很多子规则(rule)字段
rule	字典类型	二级节点	单个规则字段标识，用于描述一个具体规则的优化方案集合
preset	字符串	三级节点	rule 规则字段预设进程名。此字段为空，表示在监控到场景字段进程名后生效；需要监控场景字段进程的子进程，此字段为子进程的进程名
methods	集合	三级节点	规则字段方案合集，包含多个具体 method
method	字典类型	四级节点	具体的 method，必须包含 type key value 三个字段内容
type	字符串	五级节点	method 方案类型，目前可选类型如下： sysctl: 使用 sysctl 命令可以操作的内核参数 service: 系统中的服务，比如防火墙 common_path: 操作文件的绝对路径 bind_cpus: 对指定进程进行绑核操作
key	字符串	五级节点	method 优化名称，目前各种类型 key 含义如下： sysctl: 使用 sysctl 命令可以操作的内核参数名称 service: 系统中的服务名称，比如 firewalld common_path: 以/开始的文件的绝对路径 bind_cpus: node 或者 cpus，node 表示以 node 节点为单位进行绑核，cpus 表示以 cpu 为单位进行绑核
value	字符串	五级节点	method 优化数值，目前各种类型 value 含义如下： sysctl: sysctl 内核参数需要被设置的优化值 service: 系统中的服务启停，比如 start,stop common_path: 文件将要被设置的优化值 bind_cpus:"-~"表示连续范围，"~"表示离散或范围（如 1-3、1,2,3、1-2,4）。key 为 node 时，value 值表示绑定到具体的 node 节点上。key 为 cpus 时，value 值表示绑定到具体 cpu 核心上。

表 2-3

方案字段转化成 yaml 文件内容后，如图 2-2 所示：

```
# 方案字段
rules:
- rule:
  preset: '' #此字段为空,则监控到Run进程后实施后续方案
  methods:
  - method:
    type: 'service' #method字段类型可选service sysctl commonpath bind_cpus
    key: 'firewalld'
    value: 'stop'
  - method:
    type: 'bind_cpus' #bind_cpus字段需要依赖进程监控,对指定进程进行绑核操作
    key: 'cpus'
    value: '1'
- rule:
  preset: 'syscall' #此字段为unixbench场景子进程,识别到UB场景后,监控到此子进程才实施后续方案
  methods:
  - method:
    type: 'sysctl'
    key: 'vm.swappiness'
    value: '11'
  - method:
    type: 'common_path'
    key: '/proc/sys/vm/dirty_ratio'
    value: '31'
```

图 2-2

场景优化文件方案字段有下列注意事项：

1. sysctl 类型中，key 必须是通过 sysctl 命令能查到的内核参数名称，value 值必须是 sysctl 命令可以设置的内核参数数值。
2. service 类型中，key 是一个系统已安装的服务名称，value 是 start 或者 stop。
3. common_path 类型是对 sysctl 类型的有效补充，适用于部分无法通过 sysctl 命令直接操作的内核接口。这类接口通常在 /proc 或 /sys 文件系统中存在对应的文件，可通过 echo 进行写入，或使用 cat 进行读取。在 ldop 优化方案中，可以将这些文件的绝对路径作为 key，并预置能够成功 echo 写入的值作为 value。
4. bind_cpus 类型依赖进程监控，只有监控到对应进程后，才能针对进程进行绑核。

2.4 文件编写

1. 拷贝表 2-4 文件内容到安装 ldop 工具的环境中，以.yaml 结尾进行命名。

```
# 全局字段
identifier: '59cbb503-09fd-4f37-8ed9-c7bd3945dd33'
name: 'unixbench 优化方案'
arch: 'x86_64,aarch64'
os: 'V11'
priority: 50
exclusive: 'FALSE'

# 场景字段,以 unixbench 场景为例,按照下列格式编写
scene:
```

```

name: 'unixbench'
type: 'process'
preset: 'Run'

# 方案字段
rules:
- rule:
    preset: ""                #此字段为空,则监控到 Run 进程后实施后续方案
    methods:
    - method:
        type: 'service'      #method 字段类型可选 service sysctl commonpath
bind_cpus
    key: 'firewalld'
    value: 'stop'
    - method:
        type: 'bind_cpus'    #bind_cpus 字段需要依赖进程监控,对指定进程进行绑核
操作
    key: 'cpus'
    value: '1'
- rule:
    type: 'process'
    preset: 'syscall'        #此字段为 unixbench 场景子进程,识别到 UB 场景后,监控
到此子进程才实施后续方案
    methods:
    - method:
        type: 'sysctl'
        key: 'vm.swappiness'
        value: '11'
    - method:
        type: 'common_path'
        key: '/proc/sys/vm/dirty_ratio'
        value: '31'

```

表 2-4

2. 修改全局字段的 name 字串,方便用户通过 name 字串名称进行优化方案识别**(非必需)**。
3. 调整场景字段内容。将 scene 字段的 name 和 preset 修改为对应场景名和进程名,以确保场景匹配准确。
4. 调整方案字段内容,根据需要进行增加和删除。表 2-4 文件中存在两个 rule 字段,第一个 rule 字段负责监控到场景进程后实施的优化策略,第二个 rule 字段负责监控到场景进程的子进程后实施的优化策略。
 - (1) 若无需监控并优化场景进程,则删除第一条 rule 字段下的所有内容;若需要,则保留。

- (2) 若无需监控场景进程的子进程，则删除第二条 `rule` 字段下的所有内容；若需要，则保留，并调整 `rule` 字段中的 `preset` 进程名，使其顺利匹配子进程。
- (3) 若多个子进程需要监控优化，则在第二条 `rule` 字段的基础上，新增 `rule` 条目，并针对不同子进程进行设置。
- (4) 根据优化需求，调整 `method` 方案下的优化策略。确保 `method` 方案的 `type`、`key` 和 `value` 符合 2.3 章节要求。

注意：此处 `method` 字段类型不支持 `env` 类型。

3. 非场景优化文件

非场景优化文件主要应用于默认场景下的性能优化。非场景优化方案应用后，方案策略直接在系统上全局生效。

3.1 全局字段

全局字段目前共涉及 6 个名称，均为一级节点。目前全局变量所有字段均为预留字段，仅保留在 `yaml` 文件中，无实际意义。仅方便后续功能扩展使用。具体描述如表 3-1 所示：

字段名称	字段类型	节点级别	字段说明
identifier	字符串	一级节点	方案的唯一标识 <code>id</code> ，不同优化方案使用不同的进程标识 <code>id</code> ，可以通过 <code>uuidgen</code> 命令生成。(预留字段)
name	字符串	一级节点	优化方案的名称，基于此名称进行方案相关操作 (预留字段)
arch	字符串	一级节点	优化方案可以在哪些架构上生效，例如 <code>x86_64</code> , <code>aarch64</code> , <code>loongarch64</code> 。(预留字段)
os	字符串	一级节点	系统版本，例如 <code>V10 V11</code> (预留字段)
priority	字符串	一级节点	当前优化方案的优先级， <code>0-100</code> ，数值越大，优先级越高(预留字段)
exclusive	字符串	一级节点	设置方案是否为独占运行，若设置独占运行，则 <code>ldop</code> 工具只能运行这一个方案。 <code>true</code> :独占 <code>false</code> :非独占(预留字段)

表 3-1

3.2 方案字段

方案字段根据节点级别共涉及 7 个名称，其中 1 个一级节点，1 个二级节点，1 个三级节点，1 个四级节点，3 个五级节点。具体描述如表 3-2 所示：

字段名称	字段类型	节点级别	字段说明
rules	列表	一级节点	所有规则整体字段标识，只能存在一个。此字段可以包含很多子规则(rule)字段
rule	字典类型	二级节点	单个规则字段标识，用于描述一个具体规则的优化方案集合
methods	集合	三级节点	规则字段方案合集，包含多个具体 method
method	字典类型	四级节点	具体的 method，必须包含 type key value 三个字段内容
type	字符串	五级节点	方案类型，目前可选类型如下： env: 环境变量相关参数 sysctl: 使用 sysctl 命令可以操作的内核参数 service: 系统中的服务，比如防火墙 common_path: 操作文件的绝对路径
key	字符串	五级节点	优化名称，目前各种类型 key 含义如下： env: 环境变量名数 sysctl: 使用 sysctl 命令可以操作的内核参数名称 service: 系统中的服务名称，比如 firewalld common_path: 以/开始的文件的绝对路径
value	字符串	五级节点	优化数值，目前各种类型 value 含义如下： env: 环境变量需要被设置的优化值 sysctl: sysctl 内核参数需要被设置的优化值 service: 系统中的服务启停，比如 start,stop common_path: 文件将要被设置的优化值

表 3-2

非场景优化文件方案字段有下列注意事项：

1. sysctl 类型中，key 必须是通过 sysctl 命令能查到的内核参数名称，value 值必须是 sysctl 命令可以设置的内核参数数值。
2. service 类型中，key 是一个系统已安装的服务名称，value 是 start 或者 stop。
3. common_path 类型是对 sysctl 类型的有效补充，适用于部分无法通过 sysctl 命令直接操作的内核接口。这类接口通常在 /proc 或 /sys 文件系统下存在对应的文件，可通过 echo 进行写入，或使用 cat 进行读取。在 ldop 优化方案中，可以将这些文件的绝

对路径作为 key，并预置能够成功 echo 写入的值作为 value。

4. 操作 env 类型的方案，需要重新启动终端环境变量才会生效。

方案字段转化成 yaml 文件内容后，如图 3-2 所示：

```
# 方案字段
rules:
- rule:
  methods:
  - method:
    type: 'service'          #method字段类型可选service sysctl commonpath env
    key: 'firewalld'
    value: 'stop'
  - method:
    type: 'env'              #env类型是非场景文件独有的
    key: 'test_env'
    value: '1234'
  - method:
    type: 'sysctl'
    key: 'vm.swappiness'
    value: '11'
  - method:
    type: 'common_path'
    key: '/proc/sys/vm/dirty_ratio'
    value: '31'
```

图 3-1

3.3 文件编写

1. 拷贝表 3-4 文件内容到安装 ldop 工具的环境中，以.yaml 结尾进行命名。

```
# 全局字段
identifier: '59cbb503-09fd-4f37-8ed9-c7bd3945dd33'
name: 'unixbench 优化方案'
arch: 'x86_64,aarch64'
os: 'V11'
priority: 50
exclusive: 'FALSE'

# 方案字段
rules:
- rule:
  methods:
  - method:
    type: 'service'          #method 字段类型可选 service sysctl commonpath
    key: 'firewalld'
    value: 'stop'
  - method:
    type: 'env'              #env 类型是非场景文件独有的
```

```
key: 'test_env'
value: '1234'
- method:
  type: 'sysctl'
  key: 'vm.swappiness'
  value: '11'
- method:
  type: 'common_path'
  key: '/proc/sys/vm/dirty_ratio'
  value: '31'
```

表 3-3

2. 修改全局字段的 **name** 字符串,方便用户通过 **name** 字符串名称进行优化方案识别(非必需)。
3. 根据优化需求,调整 **method** 方案下的优化策略。。确保 **method** 方案的 **type**、**key** 和 **value** 符合 3.3 章节要求。

注意: 此处 **method** 字段类型不支持 **bind_cpus** 类型。