配置文件使用指南

配置文件定义了一系列配置项，指导了CPUBench的编译、运行、日志和发布等行为，通过一种可读的方式定义了CPUBench与测试环境的交互方式。配置文件将所有的配置项集中在一起，为测试结果的移植提供了可能，也提高了CPUBench的易用性。客户想要复现某个测试结果，仅需要满足三个条件： 1、 相同版本的CPUBench工具，并通过相同的方式调用它； 2、 相同的软硬件环境； 3、 相同的配置文件；

配置文件是ini格式的文件，遵循业界通用的ini文件语法。您可以定制您的配置文件，并通过以下两种方式来指定它： 1、 通过`--config`或`-c`来指定配置文件的路径； 2、 将配置文件放在config目录中，并通过`--config`或`-c`来指定文件名，文件后缀可以省略；

若未指定配置文件，CPUBench将使用默认配置文件`config-template.ini`。

如果您还不知道如何编写配置文件，可以根据config目录中的默认配置文件`config-template.ini`做修改，下面是您可能感兴趣的跟配置文件相关的说明。

配置选项类型

配置文件提供的配置项有三种类型。 1、 通用配置项 配置文件提供了与命令行相似的通用配置项，可以有效降低命令行的复杂度。如在配置文件中定义：

```
action = standard
benchmarks = IntConcurrent
```

```
tag = test
tune = typical
```

那么以下两条命令行指令完全一致：

```
python3 cpubench.py --config=test.ini
python3 cpubench.py --config=test.ini --action=standard --benchmarks=IntConcurrent -
-tag=test --tune=typical
```

若相同的配置项既出现在命令行又出现在配置文件，以命令行为主。

2、 编译选项 配置文件提供配置项控制workload的编译行为。详情参阅 “配置项” 章节。

3、 测试和环境信息 配置文件支持描述配置项，这类配置项方便客户理解测试信息。

4、 JAVA运行参数 配置文件提供配置项以方便用户进行JVM调优。

配置文件结构

配置文件分为两个部分： 1、 common区 在命名区之前，通常用于定义本次运行的一些通用配置。如：

```
[common]
#=========================Common Settings============================
action = standard
benchmarks = IntConcurrent
tag = test
tune = typical
jobs = 1
iterations = 3
verbose = 0
work_dir = /opt/test_dir
```

2、 命名区 命名区以一个区域标识行打头，区域标识行由一到二个字符串通过&相连，如：

```
[IntConcurrent&typical]
CFLAGS = -O3
```

需要注意的是，多个命名区可能有相同的配置项，此时需要应用优先级规则来决定配置项的取值。规则在后文具体阐述。

注释

配置文件提供#注释功能，注释不会被视为配置项。

命名区

命名区以区域标识行打头，延续到下一个标识行。配置文件支持多个命名区，它们的顺序不会影响配置项的最终值。如果区域标识行有重复，它们标识的命名区内容会自动合并。 区域标识行的格式如下，分为两个标识符：

```
[benchmark&tuning]
```

两个标识符的取值如下所示： benchmark包含了suite以及特定的benchmark名称。tuning包含了调优选项包括typical/extreme。

如果区域标识行的尾部标识是default，可以省略。如：

```
[int_x264&default]
[int_x264]
```

命名区优先级规则：

配置文件通过优先级规则来组织命名区的区域合并、冲突。优先级规则作用于标识行内的两个标识符。

对benchmark标识符，优先级如下：

```
高优先级    specific benchmarks > benchmark suite > default    低优先级
```

对tuning标识符，优先级如下：

```
高优先级    typical = extreme > default    低优先级
```

若按照以上进行各个标识符的优先级排序后，区域内的配置项之间没有冲突，则直接合并，若有冲突，按照以下标识符间优先级来判断，优先级一致，则以最后设置的值为主：

```
高优先级    benchmark > tuning    低优先级
```

例如以下两条命令： 第一条指定了benchmark，会匹配前两个命名空间，由于优先级冲突，选择优先级大的-O3。 第二条指定了tuning和benchmark，按照各个标识符优先级排序，会匹配第一和第三个命名空间，但是由于配置项冲突，则按照标识符间优先级排序，tuning优先级更高，选择-O2。

```
[default]
CC = ${lang_dir}/gcc
CC_VERSION = -v
CXXFLAGS = -O0

[IntConcurrent]
CXXFLAGS = -O3

[default&extreme]
CXXFLAGS = -O2

python3 cpubench.py --benchmarks=int_x264 --jobs=3
python3 cpubench.py --benchmarks=int_x264 --jobs=1 --tune=extreme
```

配置项

通用配置项

Option	Default	Area	Meaning
action	standard	common	指定本次工具的活动。
benchmarks	[]	common	指定本次运行的benchmarks或suite。
tune	typical	common	调优级别，可以选择经典(typical)或最优(extreme)。

tag	""	common	用于标记可执行文件，构建目录和运行目录。
jobs	1	common	支持在多个核上跑多个复制。
verbose	0	common	值为1的时候，开启debug模式。
work_dir	work_dir	common	设置工作目录，所有运行数据都会存放到该路径下。
iterations	3	common	要运行的迭代次数。
taskset	[]	common	绑核列表，由逗号分隔的一系列cpu核号构成，同时支持通过-分隔符来标识一个编号范围，如1-5等价于1,2,3,4,5。 当该列表长度大于jobs时，只会选前jobs的cpu核进行绑定，若小于jobs，则会在超出列表长度时，从列表头重新绑定。
arch	auto	common	指定CPU架构，auto表示自动识别，否则使用用户指定参数。该选项用于不规范CPU Architecture命名情况。
rebuild	0	common	是否在运行前重新编译负载。

测试和环境信息

Option	Default	Area	Meaning
cpu_name	""	common	制造商确定的处理器名称。
machine_name	""	common	机器名。
cpu_max_mhz	""	common	芯片供应商指定的CPU的最大速度，以MHz为单位。
cpu_nominal_mhz	""	common	芯片供应商指定的CPU速度，以MHz为单位。
disk	""	common	运行目录的磁盘子系统。
memory	""	common	主内存的大小。
l1_cache	""	common	1级（主）缓存。
l2_cache	""	common	2级缓存。

l3_cache	""	common	3级缓存。
manufacturer	""	common	硬件制造商。
os	""	common	操作系统名称和版本。
compiler	""	common	编译器的名称和版本。
file_system	""	common	运行目录的文件系统（ntfs，ufs，nfs等）。
run_level	""	common	运行级别。
ptrsize	""	common	基准测试使用的指针位数。32/64
huge_page_size	""	common	大页内存。
transparent_huge_pages	""	common	透明大页内存。
test_date	""	common	测试时间。
test_sponsor	""	common	赞助此测试的实体。
software_available_time	""	common	软件有效期。
hardware_available_time	""	common	硬件有效期。
software_others	""	common	其他软件相关信息。
hardware_others	""	common	其他硬件相关信息。
license_id	""	common	许可证编号。

编译选项

Option	Default	Area	Meaning
LD_LIBRARY_PATH	/usr/bin/gcc	namespace	动态链接库搜索路径设置。
CC	/usr/bin/gcc	namespace	如何调用C编译器。
CC_VERSION	-v	namespace	如何获取C编译器版本。
CFLAGS	""	namespace	既不是优化也不是可移植性的C语言编译标志。
CXX	/usr/bin/g++	namespace	如何调用C++编译器。

CXX_VERSION	-v	namespace	如何获取C++编译器版本。
CXXFLAGS	""	namespace	既不是优化也不是可移植性的C++语言编译标志。
FC	/usr/bin/gfortran	namespace	如何调用gfortran。
FC_VERSION	-v	namespace	如何获取gfortran版本。
FFLAGS	""	namespace	指定fortran语言编译选项。
CLD	""	namespace	在编译C语言的程序时，如何调用链接器。
CXXLD	""	namespace	在编译C++语言的程序时，如何调用链接器。
FLD	""	namespace	在编译fortran语言的程序时，如何调用链接器。
CLD_FLAGS	""	namespace	适用所有C Workload使用的链接标志。
CXXLD_FLAGS	""	namespace	适用所有CXX Workload使用的链接标志。
FLD_FLAGS	""	namespace	适用所有Fortran Workload使用的链接标志。
LD_FLAGS	""	namespace	适用所有模块时使用的链接标志。
LIBS	""	namespace	指定链接库选项。
AR	ar	namespace	应用于创建静态库，但是启用特殊选项(如：LTO)时，需要编译器自带AR工具(如：gcc-ar)。
RANLIB	ranlib	namespace	应用于静态库建立索引，以便编译器加速查找符号表。启用特殊选项(如：LTO)时，需要使用编译器自带RANLIB工具(如：gcc-ranlib)。

JAVA运行参数

Option	Default	Area	Meaning
java_options	""	namespace	传递给JVM的调优参数。注意：禁止通过-Xmx设置最大堆容量。

调试选项

Option	Default	Area	Meaning
perf	0	common	是否执行采集任务，0不采集，1采集。
perf_timeout	60	common	当workload运行结束后，继续等待perf_timeout指定的时间，若采集任务还未结束，则强制终止采集。单位：秒。
perf_info	{}	common	用户自定义的采集命令。

该选项属于可变选项，格式为perf_{target}={command}。其中{target}可由用户自定义，用于解释采集目标，区分采集数据。{command}为字符串，识别为一条采集命令。用户需要在采集命令中通过{pid}来预置传PID的位置。若采集命令不是通过标准输出展示采集信息，通常会通过-o选项来指定数据的保存路径。用户可直接在命令中指定目录，也可通过{output}来使输出文件保存到框架默认的采集数据存储目录。

框架内置采集命令为： CPUBENCH#IO: pidstat 1 -d -p {pid} CPUBENCH#CPU: pidstat 1 -p {pid} CPUBENCH#MEM: pidstat 1 -r -p {pid} CPUBENCH#STAT: perf stat -ddd -p {pid} -o {output}/perf.txt CPUBENCH#HOTSPOT: perf record -a -s -d -p {pid} -o {output}/perf.data

perf_target [] common 当用户只想执行特定几个采集命令时，可以通过该选项来指定这些命令。选项的值由逗号分隔的采集目标构成，例如：perf_target = IO,MEM。