bpftrace 速查表

smallnest

1 基本语法

概念	描述
探针	格式：provider:probe
Action	代码在 { ... } 中
内建变量	$1, $2 (参数), pid, tid, uid, gid, comm, cpu, curtask, rand
	args, arg0, arg1, ..., argN (函数参数)
Maps	@name[key] = value;
过滤探针	/filter/ { actions }

2 常见探针

探针	描述
kprobe:function	内核函数入口
kretprobe:function	内核函数返回
uprobe:/path/to/binary:function	用户空间函数入口
uretprobe:/path/to/binary:function	用户空间函数返回
tracepoint:subsystem:event	静态跟踪点
profile:hz:rate	周期采样
interval:ms:rate	定时器
software:event:count	软件事件（例如，cpu-clock, task-clock）
hardware:event:count	硬件事件（例如，cache-misses, cpu-cycles）

3 列出探针

命令	描述
`bpftrace -l`	列出所有探针
`bpftrace -l "tracepoint:*"`	列出所有 tracepoint 探针
`bpftrace -l "kprobe:vfs_*"`	列出所有 vfs 相关的 kprobe 探针

`bpftrace -lv "tracepoint:syscalls:sys_enter_*"`列出所有 sys_enter 相关的 tracepoint 探针及其参数

4 内置函数

函数	描述
printf(format, args...)	打印格式化字符串
time(format)	当前时间，格式化输出
count()	计数器
sum(int n)	求和
avg(int n)	平均值
min(int n)	最小值
max(int n)	最大值
hist(int n)	直方图
lhist(int n, int min, int max, int step)	线性直方图
kstack(), ustack()	内核和用户栈跟踪
ntop(int n)	转换 IP 地址为字符串
reg(char *name)	读取 CPU 寄存器值

5 高级特性

特性	描述
Wildcards	在探针定义中使用 *（例如，kprobe:vfs_*）
Frequency Counting	count() 可以与映射结合进行频率分析
Associative Arrays	@map[key1, key2, ...] = value;
Strings	使用 str() 转换为字符串，strcmp() 进行比较
Aggregations	clear(@map), print(@map), zero(@map)
Join	join(char *arr[]) 连接数组元素
Timestamps	nsecs, elapsed

6 示例

6.1 系统调用计数

```
#!/usr/bin/env bpftrace

tracepoint:raw_syscalls:sys_enter
{
```

```
    @syscalls[comm] = count();
}

interval:s:5
{
    print(@syscalls);
    clear(@syscalls);
}
```

6.2 函数延迟测量

```
#!/usr/bin/env bpftrace

kprobe:vfs_read
{
    @start[tid] = nsecs;
}

kretprobe:vfs_read
/@start[tid]/
{
    @duration = hist(nsecs - @start[tid]);
    delete(@start[tid]);
}

interval:s:10
{
    print(@duration);
    clear(@duration);
}
```

6.3 内存分配跟踪

```
#!/usr/bin/env bpftrace

uprobe:/lib/x86_64-linux-gnu/libc.so.6:malloc
/comm == "target_process"/
{
    @bytes[ustack] = sum(arg0);
}

interval:s:30
{
    print(@bytes);
    clear(@bytes);
}
```

7 一行代码示例

bpftrace 一行代码	解释
`bpftrace -e 'tracepoint:syscalls:sys_enter_* { @[probe] = count(); }'`	统计所有系统调用的次数
`bpftrace -e 'kprobe:vfs_read { @bytes = sum(arg2); }'`	累计读取的字节数
`bpftrace -e 'kprobe:vfs_write { @bytes[comm] = sum(arg2); }'`	按进程名统计写入的字节数
`bpftrace -e 'tracepoint:block:block_rq_issue { @[args->dev] = count(); }'`	统计每个块设备的 I/O 请求数
`bpftrace -e 'software:major-faults:1 { @[comm] = count(); }'`	统计每个进程的主缺页次数
`bpftrace -e 'kprobe:do_nanosleep { @[comm] = count(); }'`	统计每个进程的睡眠次数
`bpftrace -e 'tracepoint:sched:sched_process_exec { @[args->filename] = count(); }'`	统计每个可执行文件的执行次数
`bpftrace -e 'kprobe:tcp_retransmit_skb { @[sock_ip(args->sk)] = count(); }'`	统计每个 IP 地址的 TCP 重传次数
`bpftrace -e 'tracepoint:irq:irq_handler_entry { @[args->name] = count(); }'`	统计每个中断处理程序的调用次数
`bpftrace -e 'kprobe:kmalloc { @bytes = hist(arg1); }'`	创建内核内存分配大小的直方图
`bpftrace -e 'kprobe:do_sys_open { @[comm] = count(); }'`	统计每个进程的文件打开次数
`bpftrace -e 'uprobe:/lib/x86_64-linux-gnu/libc.so.6:malloc { @bytes = hist(arg0); }'`	创建用户空间内存分配大小的直方图
`bpftrace -e 'profile:hz:99  @[comm] = count(); '`	每秒采样 99 次，并统计每个进程名的调用次数
`bpftrace -e 'interval:s:5  printf("Every 5 seconds"); '`	每 5 秒打印一次”Every 5 seconds”

8 最佳实践

- 使用过滤器减少开销并关注相关事件
- 结合多个探针以关联事件
- 使用映射和聚合进行高效数据收集
- 小心使用栈跟踪和字符串操作（高开销）
- 首先在非生产系统上测试脚本
- 使用 interval 探针定期输出和清除数据