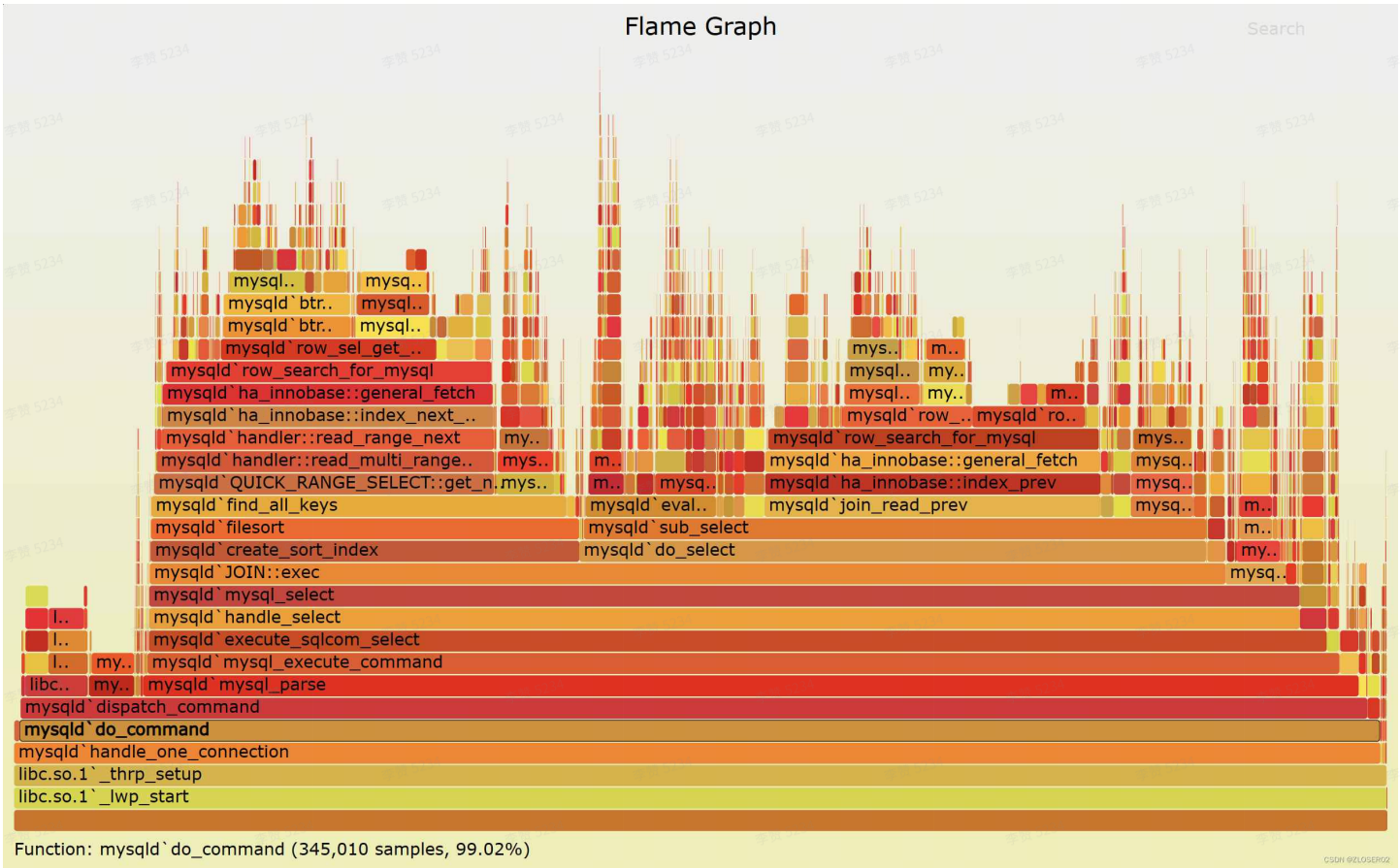
火焰图的生成及原理

如图所示



火焰图有以下特征(on-cpu)

1.每一列代表一个调用栈，每一个格子代表一个函数。

2.纵轴展示了栈的深度，按照调用关系从下到上排列。最顶上格子代表采样时，正在占用 cpu 的函数。

3.横轴的意义是指：火焰图将采集的多个调用栈信息，通过按字母横向排序的方式将众多信息聚合在一起。需要注意的是它并不代表时间。

4.横轴格子的宽度代表其在采样中出现频率，所以一个格子的宽度越大，说明它是瓶颈原因的可能性就越大。

5.火焰图格子的颜色是随机的暖色调，方便区分各个调用信息。

其他的采样方式也可以使用火焰图， on-cpu 火焰图横轴是指 cpu 占用时间，off-cpu 火焰图横轴则代表阻塞时间。

6.采样可以是单线程、多线程、多进程甚至是多 host，进阶用法可以参考附录进阶阅读。

火焰图能做什么

1.可以分析函数执行的频繁程度(占用cpu时间，或者阻塞的时间)。

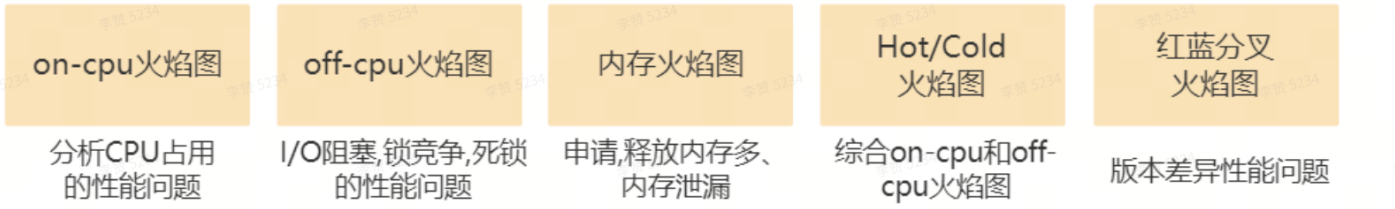
2.可以分析哪些函数经常阻塞。

3.可以分析哪些函数频繁分配内存。

火焰图类型

常见的火焰图类型有 On-CPU，Off-CPU，还有 Memory，Hot/Cold，Differential 等等。它们有各自适合处

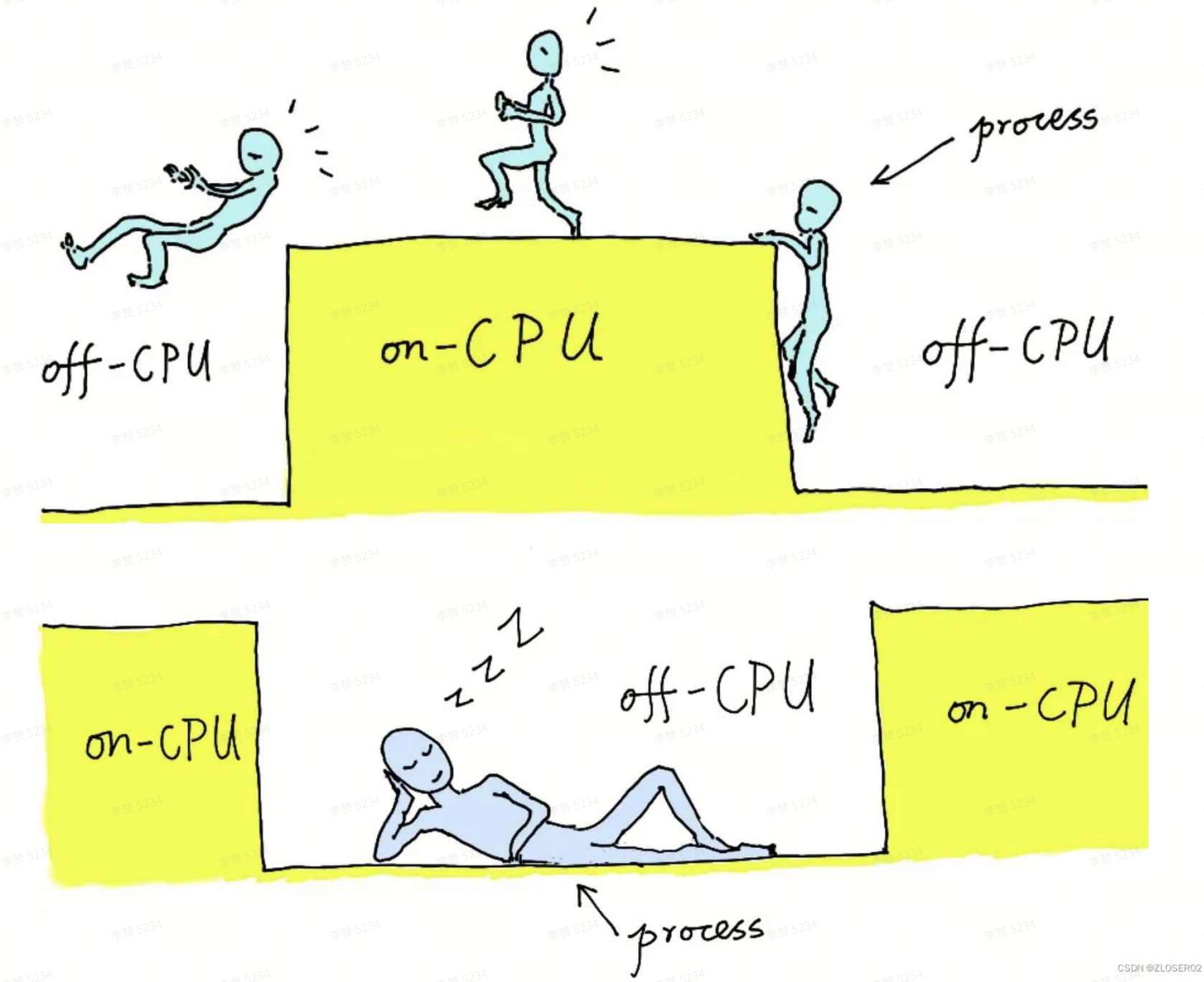
理的场景。



火焰图类型	横轴含义	纵轴含义	解决问题	采样方式
on-cpu火焰图	cpu占用时间	调用栈	找出cpu占用高的问题函数；分析代码热路径	固定频率**采样cpu调用栈**
off-cpu火焰图	阻塞时间	调用栈	i/o、网络等阻塞场景导致的性能下降；锁竞争、死锁导致的性能下降问题	固定频率**采样阻塞事件调用栈**
内存火焰图	内存申请/释放函数调用次数	调用栈	内存泄漏问题；内存占用高的对象/申请内存多的函数；虚拟内存或物理内存泄漏问题	有四种方式：跟踪malloc/free；跟踪brk；跟踪mmap；跟踪页错误

火焰图类型	横轴含义	纵轴含义	解决问题	采样方式
Hot/Cold 火焰图	on-cpu和off-cpu综合展示	调用栈	需要结合cpu占用以及阻塞分析的场景；off-cpu火焰图无法直观判断的场景	on-cpu火焰图和off-cpu火焰图结合
红蓝分叉火焰图	红色表示上升，蓝色表示下降	调用栈	处理不同版本性能回退问题	对比两个on-cpu火焰图

On-CPU 火焰图和Off-CPU火焰图的使用场景



取决于当前的瓶颈到底是什么：

1.如果是 CPU 则使用 On-CPU 火焰图，（先看cpu是不是快到百分百）。

2.如果是 IO 或锁则使用 Off-CPU 火焰图（如果cpu占用率不高，就需要用off-cpu）。

3.如果无法确定, 那么可以通过压测工具来确认：

1.通过压测工具看看能否让 CPU 使用率趋于饱和, 如果能那么使用 On-CPU 火焰图

2.如果不管怎么压, CPU 使用率始终上不来, 那么多半说明程序被 IO 或锁卡住了, 此时适合使用 Off-CPU 火焰图.

4.如果还是确认不了, 那么不妨 On-CPU 火焰图和 Off-CPU 火焰图都搞搞, 正常情况下它们的差异会比较大, 如果两张火焰图长得差不多, 那么通常认为 CPU 被其它进程抢占了.

火焰图分析技巧

1、纵轴代表调用栈的深度（栈桢数），用于表示函数间调用关系：下面的函数是上面函数的父函数。

2、横轴代表调用频次，一个格子的宽度越大，越说明其可能是瓶颈原因。

3、不同类型火焰图适合优化的场景不同：

1、比如 on-cpu 火焰图适合分析 cpu 占用高的问题函数；

2、off-cpu 火焰图适合解决阻塞和锁抢占问题。

4、无意义的事情：横向先后顺序是为了聚合，跟函数间依赖或调用关系无关；火焰图各种颜色是为方便区

分，本身不具有特殊含义

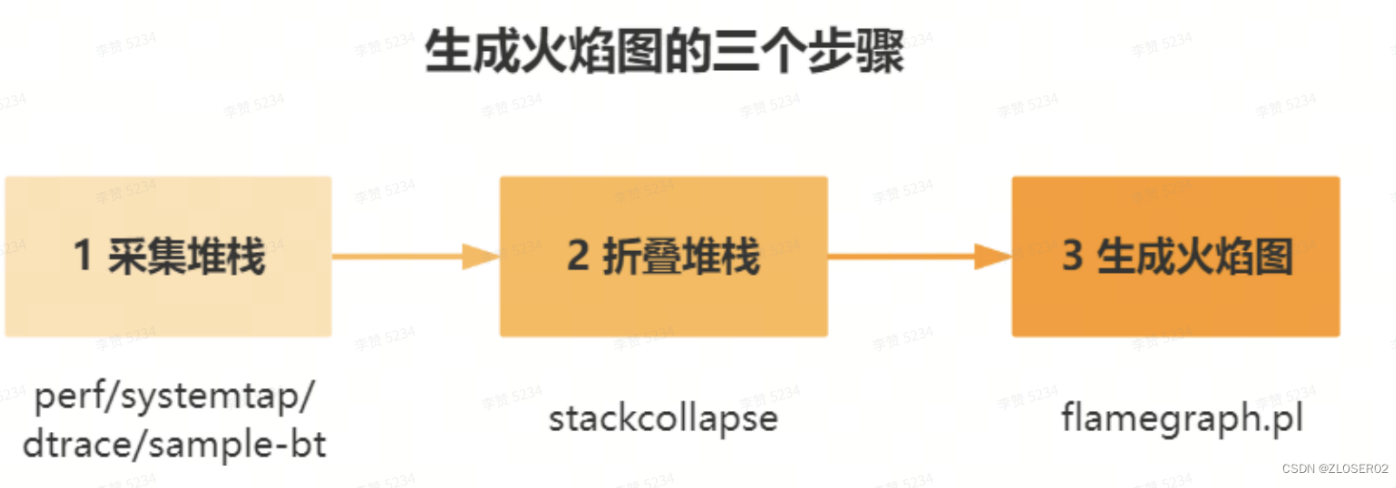
5、多练习：进行性能优化有意识的使用火焰图的方式进行性能调优（如果时间充裕）

如何绘制火焰图

注意：采集需要ROOT权限

生成火焰图的流程

1.生成火焰图的三个步骤



安装火焰图必备工具

1.安装火焰图FlameGraph脚本

1.Brendan D. Gregg 的 Flame Graph 工程实现了一套生成火焰图的脚本。Flame Graph 项目位于GitHub上:git clone https://github.com/brendangregg/FlameGraph.git

2.使用码云gitee的链接：git clone https://gitee.com/mirrors/FlameGraph.git

```
mkdir perf
cd perf
git init
git clone https://github.com/brendangregg/FlameGraph.git
```

2.安装火焰图数据采集工具perf

```
sudo yum install perf
mkdir perf
cd perf
perf record -F 99 -a -g -- sleep 10
```

可以看到在perf文件夹下生成perf.data文件，表示安装成功

如果出现报错

```
WARNING: perf not found for kernel 5.15.0-89

You may need to install the following packages for this specific kernel:
linux-tools-5.15.0-89-generic
linux-cloud-tools-5.15.0-89-generic

You may also want to install one of the following packages to keep up to date:
linux-tools-generic
linux-cloud-tools-generic
```

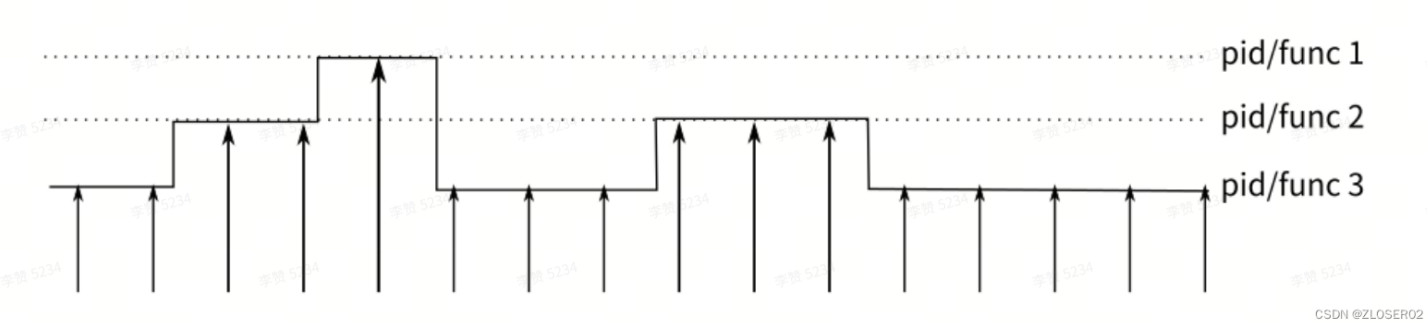
则需要安装linux-tools-generic和linux-cloud-tools-generic，但需选择对应的版本，比如提示的是5.15.0-89，则我们安装5.15.0-89版本:

```
sudo yum install linux-tools-4.15.0-48-generic linux-cloud-tools-4.15.0-48-
generic linux-tools-generic linux-cloud-tools-generic
```

3、perf的原理

每隔一个固定的时间，就在CPU上（每个核上都有）产生一个中断，在中断上看看，当前是哪个pid，哪个函数，然后给对应的pid和函数加一个统计值，这样，我们就知道CPU有百分几的时间在某个pid，或者某个函数上了。

这个原理图示如下：1秒采集99次，1秒采集1000次（采样次数越高容易影响程序性能）



很明显可以看出，这是一种采样的模式，预期：运行时间越多的函数，被时钟中断击中的机会越大，从而推测，那个函数（或者pid等）的CPU占用率就越高。

这种方式可以推广到各种事件，比如ftrace的事件，你可以在这个事件发生的时候上来冒个头，看看击中了谁，然后算出分布，我们就知道谁会引发特别多的那个事件了。

当然，如果某个进程运气特别好，它每次都刚好躲过你发起探测的位置（这时候就需要考虑提高采样频率），你的统计结果可能就完全是错的了。这是所有采样统计都有可能遇到的问题了。

4、常用命令

```
perf list：查看当前软硬件环境支持的性能事件
perf stat：分析指定程序的性能概况
perf top：实时显示系统/进程的性能统计信息
perf record：记录一段时间内系统/进程的性能事件
perf report：读取perf record生成的perf.data文件，并显示分析数据（生成火焰图用的采集命令）
```

perf record -h常用命令

```
-e ：指定性能事件（可以是多个，用,分隔列表）
-p ：指定待分析进程的 pid（可以是多个，用,分隔列表， nginx ， -p 100,101,102,103）
-t ：指定待分析线程的 tid（可以是多个，用,分隔列表）
-u ：指定收集的用户数据，uid为名称或数字
-a：从所有 CPU 收集系统数据
-g：开启 call-graph（stack chain/backtrace）记录（backtrace调试功能的实现原理就是利用函数调用
栈中的信息来追踪程序执行的路径和调用关系。）
-C ：只统计指定 CPU 列表的数据，如：0,1,3或1-2
```

```
-r ：perf 程序以SCHED_FIFO实时优先级RT priority运行这里填入的数值越大，进程优先级越高（即
nice 值越小）
-c ：  事件每发生 count 次采一次样
-F ：每秒采样 n 次
-o <output.data>：指定输出文件output.data，默认输出到perf.data
```

-- sleep 10 表示采样持续10s

生成火焰图实例

1、测试代码

```c
#include <stdio.h>
void func_d()
{
    for (int i = 5 * 10000; i--;);
}
void func_a()
{
    for (int i = 10 * 10000; i--;);
func_d();
}
void func_b()
{
    for (int i = 20 * 10000; i--;);
}
void func_c()
{
    for (int i = 35 * 10000; i--;);
}
int main(void)
{
    printf("main into\n");
    while (1)
    {
        for (int i = 30 * 10000; i--;);
        func_a();
        func_b();
        func_c();
    }
    printf("main end\n");
    return 0;
}
```

```
1 编译：gcc -o test test.c
2 执行：./test
```

```
[root@localhost perf]# ps aux | grep test
root      111484  100  0.0  12552   824 pts/3    R+   09:47   0:35 ./test
root      111511  0.0  0.0 112824   988 pts/0    R+   09:48   0:00 grep --color=auto test
[root@localhost perf]# ps -ef | grep test
```

2、perf采集数据

```
1 perf record -F 99 -p 111484 -g -- sleep 30
```

perf record 表示采集系统事件, 没有使用 -e 指定采集事件, 则默认采集 cycles(即 CPU clock 周期), -F 99 表示每秒 99 次,-p 111484 是进程号, 即对哪个进程进行分析, -g 表示记录调用栈, sleep 30 则是持续 30 秒.

-F 指定采样频率为 99Hz(每秒99次), 如果 99次 都返回同一个函数名, 那就说明 CPU 这一秒钟都在执行同一个函数, 可能存在性能问题.

为了便于阅读, perf record 命令可以统计每个调用栈出现的百分比, 然后从高到低排列.

```
1 perf report -n --stdio
```

3、折叠堆栈

1、首先用perf script 工具对 perf.data 进行解析
生成折叠后的调用栈: `perf script -i perf.data &> perf.unfold`

```
[root@localhost perf]# perf script -i perf.data &> perf.unfold
[root@localhost perf]# ll
总用量 880
-rw-------. 1 root root 254036 7月  25 09:49 perf.data
-rw-r--r--. 1 root root 620493 7月  25 09:53 perf.unfold
-rwxr-xr-x. 2 root root  15960 7月  25 09:47 test
-rw-r--r--. 1 root root    452 7月  25 09:46 test.cpp
[root@localhost perf]# ../FlameGraph/stackcollapse-perf.pl perf.unfold &> perf.folded
```

2、然后将解析出来的信息存下来, 供生成火焰图

用 stackcollapse-perf.pl 将 perf 解析出的内容 perf.unfold 中的符号进行折叠
:./FlameGraph/stackcollapse-perf.pl perf.unfold &> perf.folded

注意: ./FlameGraph/stackcollapse-perf.pl路径为自己的路径,根据自己的情况进行修改

#生成火焰图需要的统计信息

```
[root@localhost perf]# ../FlameGraph/stackcollapse-perf.pl perf.unfold &> perf.folded
```

4、生成火焰图

最后生成 svg 图-绘制火焰图:

注意: ./FlameGraph/flamegraph.pl路径为自己的路径,根据自己的情况进行修改

./FlameGraph/flamegraph.pl perf.folded > test_oncpu.svg

```
[root@localhost perf]# ../FlameGraph/flamegraph.pl perf.folded > test_oncpu.svg
```

我们也可以使用管道将上面的流程简化为一条命令:

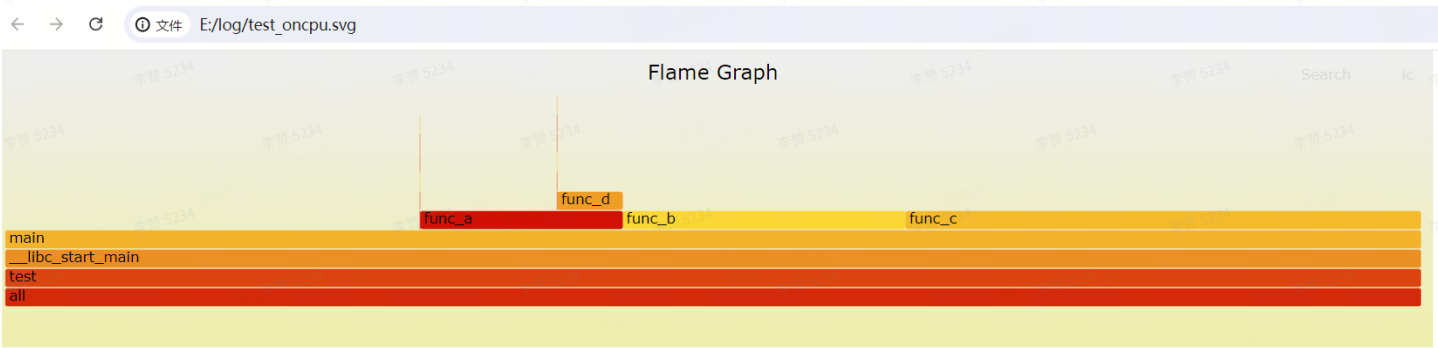
注意: ./FlameGraph/stackcollapse-perf.pl路径为自己的路径,根据自己的情况进行修改

perf script | ./FlameGraph/stackcollapse-perf.pl | ./FlameGraph/flamegraph.pl > test_oncpu.svg

```
perf script默认是输入perf.data，如果需要指明输入数据用perf script -i xxx
```

5、火焰图分析

最后就可以用浏览器打开火焰图进行分析啦.



火焰图是基于 stack 信息生成的 SVG 图片, 用来展示 CPU 的调用栈。

1.y 轴表示调用栈, 每一层都是一个函数. 调用栈越深, 火焰就越高, 顶部就是正在执行的函数, 下方都是它的父函数.

2.x 轴表示抽样数, 如果一个函数在 x 轴占据的宽度越宽, 就表示它被抽到的次数多, 即执行的时间长. 注意,x 轴不代表时间, 而是所有的调用栈合并后, 按字母顺序排列的.

3.火焰图就是看顶层的哪个函数占据的宽度最大. 只要有 “平顶”(plateaus), 就表示该函数可能存在性能问题。

4.颜色没有特殊含义, 因为火焰图表示的是 CPU 的繁忙程度, 所以一般选择暖色调.