**用gperftools对C/C++程序进行profile**

* [什么是perftools](http://airekans.github.io/cpp/2014/07/04/gperftools-profile#perftools)
* [如何profile](http://airekans.github.io/cpp/2014/07/04/gperftools-profile#profile)
* [动态profile](http://airekans.github.io/cpp/2014/07/04/gperftools-profile#profile-1)
* [对服务器进行profile](http://airekans.github.io/cpp/2014/07/04/gperftools-profile#profile-2)
* [总结](http://airekans.github.io/cpp/2014/07/04/gperftools-profile#section)

**什么是perftools**

在Linux的C/C++编程的世界里，性能调优一直是个让人头疼的事。最出名的**gprof**虽然大家都知道， 其用法比较单一(只支持程序从启动到结束的profile)，而且对程序的运行时间会有比较大的影响， 所以其profile不一定准确。

而**valgrind**功能十分强大，但profile也一般针对整个程序的运行，很难只对程序运行中的某段时间进行profile。 而且也多少会影响程序的运行，且使用的难度也较大，所以我目前还没尝试。

除去上面的两个常见的工具，之前在公司的项目见过使用Google的[gperftools](https://code.google.com/p/gperftools/) 进行profile的，当时就被他简单的使用方法吸引。而最近维护的服务器也有性能问题，需要做性能调优。 在尝试了多种原始的profile方式之后，我选择了**gperftools**。

**如何profile**

在gperftools的文档中，就简单的说了下面的方式来进行profile：

**gcc [...] -o myprogram -lprofiler**

**CPUPROFILE=/tmp/profile ./myprogram**

是的，在编译和安装了**gperftools**之后，只需要上面的步骤就可以进行profile了，非常简单。 而profile的结果就保存在**/tmp/profile**。查看结果只需要用**gperftools**自带的一个**pprof**脚本来看就可以：

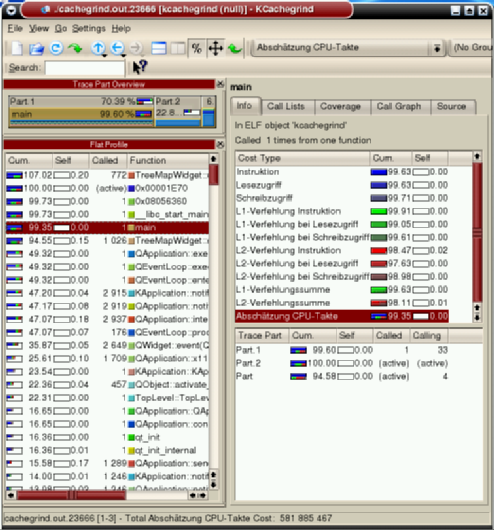
**$ pprof --text ./myprogram /tmp/profile**

**14 2.1% 17.2% 58 8.7% std::\_Rb\_tree::find**

**pprof**的输出也很直观，不过也还不够好，从这个输出中还不好看出调用关系，包括caller和callee。 而pprof也可以输出图示，还可以输出callgrind兼容的格式，这样就可以用**kcachegrind**来看profile结果了。

**$ pprof --callgrind ./myprogram /tmp/profile > callgrind.res**

然后利用**kcachegrind**打开这个callgrind.res文件就可以看到类似下面的画面(图片来自kcachegrind官网)：



这样调优起来就非常直观了。而且这种方式的最大优点是非侵入式，也就是不需要改动一行代码就能够进行profile了。

**动态profile**

上面说到的方式是通过环境变量来触发profile，而跨度也是整个程序的生命周期。 那如果是想要在程序运行的某段时间进行profile呢？如果我想在程序不结束的情况下就拿到profile的结果呢？

这种情况下就需要用到动态profile的方式了。要实现这种方式，就需要改动程序的代码了，不过也比较简单：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include <gperftools/profiler.h>  int main()  {  ProfilerStart("/tmp/profile");  some\_func\_to\_profile();  ProfilerStop();    return 0;  } |

没错，你只需要在你想要profile的函数的开头和结尾加上**ProfilerStart**和**ProfilerStop**调用就可以了。 在**ProfilerStop**结束之后，profile的结果就会保存在**/tmp/profile**里面了。 利用这种方式就可以在指定的时间点对程序进行profile了。

最后需要说的一点是，gperftools的profile过程采用的是采样的方式，**而且对profile中的程序性能影响极小**， 这对于在线或者离线profile都是一个极其重要的特点。

**对服务器进行profile**

对于后端程序员，每天都要和后台服务器打交道。而服务器的特点是长时间运行而不停止， 在这种情况下要对程序进行profile就比较麻烦。

在这我提供一种方式，使得profile服务器可以很方便，也可以按需profile。

首先要注意的一点是，gperftools提供了两种链接方式——动态库和静态库。 其中动态库链接的方式可以用环境变量和改动代码两种方式进行profile，而静态库只能使用改代码的方式。 乍看起来好像是动态库库的方式比较方便，不过在陈硕的[《Linux多线程服务端编程》](http://book.douban.com/subject/20471211/) 中就说过，对于服务器来说，静态编译的方式对于于动态链接有优势，并且部署上也比较方便。 而我自己也是使用的静态链接的方式来使用gperftools的，所以以下假定都是用静态编译。

对于服务器来说，一般的模式是事件循环，而我们也需要在某段时间之内进行profile。 一个很直观的思路是在接受到某种请求的时候开始profile，而接受到另一种请求之后就结束。 那我们就可以用类似下面的代码来实现：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <gperftools/profiler.h>  void on\_request(Request\* req)  {  static bool is\_profile\_started = false;  if (req->type == START\_PROFILE && !is\_profile\_started)  {  ProfilerStart("/tmp/profile");  is\_profile\_started = true;  }  else if (req->type == STOP\_PROFILE && is\_profile\_started)  {  ProfilerStop();  is\_profile\_started = false;  }  else  {  // normal request processing here  }  } |

利用来面的代码，我们可以在想要profile的时间段内分别向服务器发送特殊的请求， 这样就可以在不停止服务器的情况下，对服务器进行profile。

当然这种方式会产生安全问题，在有外网请求的服务器上是不能这么用的。 而且gperftools的文档上也说明了，在线上的服务器最好是不要开启profile，而对测试服务器用就好了。

**总结**

gperftools对于Linux下的服务器profile进行了很大的简化。能够在不改代码或者改极少代码并且 不增加太多的依赖的情况下，对服务器进行在线profile。有了gperftools，Linux程序员的生活可以又轻松一些了！