C++ 后台程序实时性能监控

面对的问组件题：

做后台程序经常会被问一句话，你的程序能撑多少人。一般官方一点的回答是这个得根据实际情况而定。实际上后台程序的性能是可以被量化的。我们开发的每一个服务器程序，对性能都非常有底，以为我们有数据。So，能撑多少人不少随便猜的，让数据报表来说话。

另外一种情况经常发生在开发人员之中，甲乙丙一起讨论接口实现，经常会说这么实现效率太低，那么实现效率才高等。实际上，效率高低都是相对而言的。一个函数1ms执行完毕够快吗？看起来挺快，若某接口需要此函数100次循环，那么情况就不是很乐观了。但是若此接口又是十天半个月才会被触发一次，似乎事情又变的不是很严重。说到这里想起《unix编程艺术》上关于性能优化的总结：

- 最有效的优化往往是优化之外的，如清晰干净的设计
- 最有效的优化就是不优化，摩尔定律会为你优化
- 如果确定要优化，必须找到真正的瓶颈

还有一种跟性能有关的情况是，后台程序经常有很多组件组成。比如在运行期发生接口调用性能下降的情况，必须知道是那些组件性能下降引起的。如果可以实时的知道所有接口的性能数据，以上的问题都可迎刃而解。

总结如下原因，必须开启实时性能监控：

- 我们需要知道系统的吞吐量，以此参数做部署等。
- 实时了解各个系统组件的性能，某组件发生故障，可以及时发现
- 获得程序接口调用热点，调用多且慢的接口才需要优化

解决方案：

后台程序开发一个专门统计性能的组件，其需要有如下功能：

- 可以汇总性能数据，如定时将1小时内说有接口调用开销、次数等数据汇总到文件
- 可以非常方便的与逻辑层接口集成，比如在现有接口增加一行代码即可
- 直观的报表，性能数据写入文件必须按照通用的格式，方便工具分析数据，生成报表

性能监控组件

我实现了一个性能组件performance_daemon_t。接口如下：

```
//! 性能监控
class performance_daemon_t
{
public:
    struct perf_tool_t
    {
        perf_tool_t(const char* mod_):
            mod(mod_)
        {
            gettimeofday(&tm, NULL);
        }
        ~perf_tool_t()
        {
            struct timeval now;
            gettimeofday(&now, NULL);
            long cost = (now.tv_sec - tm.tv_sec)*1000000 + (now.tv_usec
- tm.tv_usec);
            singleton_t<performance_daemon_t>::instance().post(mod,
cost);
        }
        const char*    mod;
        struct timeval tm;
```

```
    };
public:
    performance_daemon_t();
    ~performance_daemon_t();

    //! 启动线程，创建文件
    int start(const string& path_, int seconds_);
    //! 关闭线程
    int stop();

    //! 增加性能监控数据
    void post(const string& mod_, long us);
```

perf_tool_t 是工具类，构造和析构自动调用两次gettimeofday获取函数调用开销，例外有辅助宏定义如下：

```
#define AUTO_PERF() performance_daemon_t::perf_tool_t
__tmp__(__FUNCTION__)
#define PERF(m)     performance_daemon_t::perf_tool_t __tmp__(m)
```

使用示例：

```
void foo()
{
    AUTO_PERF();
    //! TODO -----
}

int main(int argc, char* argv[])
{

    singleton_t<performance_daemon_t>::instance().start("perf.txt", 5);
    foo();
}
```

performance_daemon_t 每隔5秒将性能统计数据输出到perf.txt， perf.txt的内容是CVS文件格式。

报表工具：

perf.txt 文件内容还不够直观，示例内容如下：

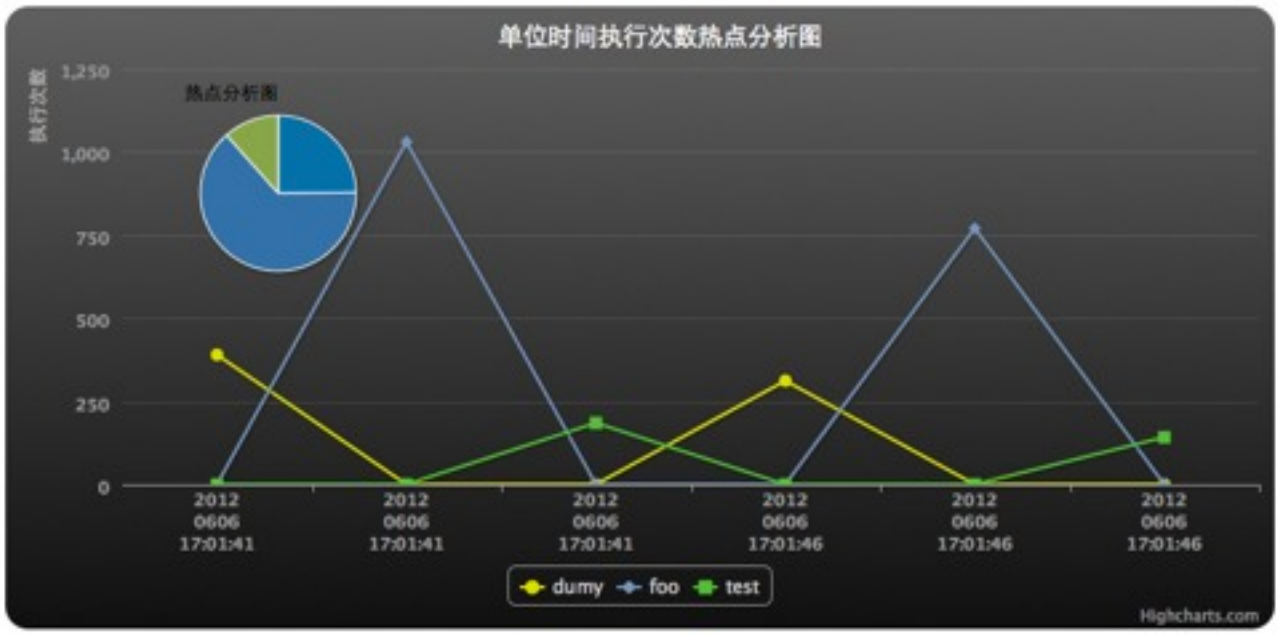
```
time,mod,max_cost[us],min_cost[us],per_cost[us],request_per_second,exe_times
20120606-17:01:41,dumy,515,174,254,3937,390
20120606-17:01:41,foo,5924,4,506,1976,1030
20120606-17:01:41,test,304,8,243,4115,185
time,mod,max_cost[us],min_cost[us],per_cost[us],request_per_second,exe_times
20120606-17:01:46,dumy,1086,222,280,3571,312
20120606-17:01:46,foo,5707,194,503,1988,770
20120606-17:01:46,test,807,8,265,3773,142
```

为生成足够友好、直观的报表，我实现了一个WEB报表页面，http://ffown.sinaapp.com/perf/， 将perf.txt 内容直接粘贴到web 页面，点击转换输出如下报表：

各个接口性能监控-折线图：

此图显示了三个接口随时间顺序的走势，可以非常清楚foo、test、dumy三个接口那个时间性能高，哪个时间性能低，一目了然。

接口热点分布图：



显示三个接口随时间调用次数走势，可以很清楚显示哪个时间段是高峰期。大饼图显示了哪个接口是热点接口，很明显，foo 接口调用次数最多，优化当优先优化foo。

组件实现浅析：

post 接口：

程序把接口调用开销投递到性能组件任务队列中，保证了对接口性能影响最小。

timer定时回调：

timer_service_t 是我用epoll 实现的定时器，主要实现如下：

```
void timer_service_t::run()
{
    struct epoll_event ev_set[64];
```

```
    //! interupt();

    struct timeval tv;

    do
    {
        ::epoll_wait(m_efd, ev_set, 64, m_min_timeout);

        if (false == m_runing)//! cancel
        {
            break;
        }

        gettimeofday(&tv, NULL);
        long cur_ms = tv.tv_sec*1000 + tv.tv_usec / 1000;

        process_timer_callback(cur_ms);

    }while (true) ;
}
```

process_timer_callback 中检测链表内所有的定时任务，若超时，触发回调函数。

备注：

有人可能当心AUTO_PERF(); 会影响接口性能，其实其平均开销大约为1us

代码实现：

https://ffown.googlecode.com/svn/trunk/example/ff_performance

WEB 报表生成工具：

http://ffown.sinaapp.com/perf/