# 基于Cocos引擎的Lua函数效率分析器

### 基本思路：

Lua的lua\_hook功能能在每个函数被调用时和返回时触发一个特定回调，通过计算调用和返回的时间差即可知道一个函数的耗时

Lua的lua\_getinfo函数能取得当前Lua的完整调用栈信息

在ios和android平台可以使用std::chrono::high\_resolution\_clock获得高精度时间，windows平台上该接口精度不足，需要更换为QueryPerformanceCounter和QueryPerformanceFrequency两个接口

通过在hook函数开头和返回时分别计时来尽量排除计时器本身开销带来的误差

### 实现功能：

能统计Lua虚拟机中每个函数的独占开销占比、总开销占比、调用次数、平均调用时间，最长调用时间和时间加权的平均调用时间

在调用结束后，生成一个csv格式的表格文件，方便进行分析

### 代码实现：

C++部分代码：

// lua profiler

static inline QWORD profileTimer()

{

#ifdef \_MSC\_VER

LARGE\_INTEGER time;

QueryPerformanceCounter(&time);

return time.QuadPart;

#else

return std::chrono::high\_resolution\_clock::now().time\_since\_epoch().count();

#endif

}

static inline QWORD profileTicker()

{

#ifdef \_MSC\_VER

LARGE\_INTEGER time;

QueryPerformanceFrequency(&time);

return time.QuadPart;

#else

return std::chrono::high\_resolution\_clock::duration(std::chrono::seconds(1)).count();

#endif

}

struct ProfileStack

{

std::string name;

QWORD startTime;

QWORD startOffset;

QWORD startCost;

};

struct ProfileStatistics

{

std::string name;

QWORD selfCost = 0;

QWORD totalCost = 0;

QWORD maxCost = 0;

QWORD callCount = 0;

double sqrSum = 0;

};

static std::vector<ProfileStack> profileStack;

static QWORD profileOffset = 0;

static QWORD profileCost = 0;

static std::unordered\_map<std::string, ProfileStatistics> profileStatistics;

static void profileHook(lua\_State\* L, lua\_Debug\* d)

{

auto p1 = profileTimer();

if (d->event) // return

{

if (profileStack.empty()) return;

auto& ci = profileStack.back();

auto offset = profileOffset - ci.startOffset;

auto cost = p1 - ci.startTime - offset;

auto selfCost = cost - (profileCost - ci.startCost);

profileCost += selfCost;

auto& s = profileStatistics[ci.name];

s.selfCost += selfCost;

s.totalCost += cost;

s.callCount += 1;

s.sqrSum += (double)cost \* (double)cost;

s.maxCost = std::max(s.maxCost, cost);

profileStack.pop\_back();

auto p2 = profileTimer();

profileOffset += p2 - p1;

}

else // call

{

profileStack.push\_back(ProfileStack());

auto& s = profileStack.back();

s.startCost = profileCost;

{

lua\_getinfo(L, "Sn", d);

std::ostringstream oss;

oss << d->short\_src;

if (d->linedefined > 0) oss << "(" << d->linedefined << ")";

if (d->name) oss << ":[" << d->namewhat << "]" << d->name;

s.name = oss.str();

}

auto p2 = profileTimer();

auto offset = p2 - p1;

profileOffset += offset;

s.startTime = p2;

s.startOffset = profileOffset;

}

}

static void dumpProfile()

{

std::vector<ProfileStatistics> list;

for (auto& i : profileStatistics)

{

list.push\_back(i.second);

list.back().name = i.first;

}

auto fileUtils = cocos2d::FileUtils::getInstance();

std::ofstream fout(fileUtils->getSuitableFOpen(fileUtils->getWritablePath() + "profile.csv"));

auto ticker = profileTicker() / 1000.0;

fout << "function name,total cost,self cost,call count,avg cost,max cost,avg^2 cost" << std::endl;

for (auto& i : list)

{

fout << i.name << ",";

fout << i.totalCost \* 100.0 / profileCost << "%,";

fout << i.selfCost \* 100.0 / profileCost << "%,";

fout << i.callCount << ",";

fout << (double)i.totalCost / i.callCount / ticker << ",";

fout << i.maxCost / ticker << ",";

fout << i.sqrSum / i.totalCost / ticker << std::endl;

}

}

static void beginProfile()

{

sealp::callLuaFunction("showNotify", "beginProfile");

profileOffset = 0;

profileCost = 0;

profileStack.clear();

profileStatistics.clear();

auto L = cocos2d::LuaEngine::getInstance()->getLuaStack()->getLuaState();

lua\_sethook(L, profileHook, LUA\_MASKCALL | LUA\_MASKRET, 0);

}

void SealUtilToLua::beginProfile()

{

auto L = cocos2d::LuaEngine::getInstance()->getLuaStack()->getLuaState();

lua\_Debug ar;

if (lua\_getstack(L, 0, &ar))

{

auto scheduler = cocos2d::Director::getInstance()->getScheduler();

scheduler->performFunctionInCocosThread([]() { ::beginProfile(); });

}

else ::beginProfile();

}

static void endProfile()

{

auto L = cocos2d::LuaEngine::getInstance()->getLuaStack()->getLuaState();

lua\_sethook(L, profileHook, 0, 0);

dumpProfile();

sealp::callLuaFunction("showNotify", "endProfile");

}

void SealUtilToLua::endProfile()

{

auto L = cocos2d::LuaEngine::getInstance()->getLuaStack()->getLuaState();

lua\_Debug ar;

if (lua\_getstack(L, 0, &ar))

{

auto scheduler = cocos2d::Director::getInstance()->getScheduler();

scheduler->performFunctionInCocosThread([]() { ::endProfile(); });

}

else ::endProfile();

}

将SealUtilToLua::beginProfile()与SealUtilToLua::beginProfile()两个接口导出到Lua，在Lua中通过控制台指令调用即可

### 样例报表：

见profile.csv，配合excel的筛选和排序功能能非常方便的定位开销过高的函数，通过数值间的关联性还能分析出函数是否被重复调用等其他问题