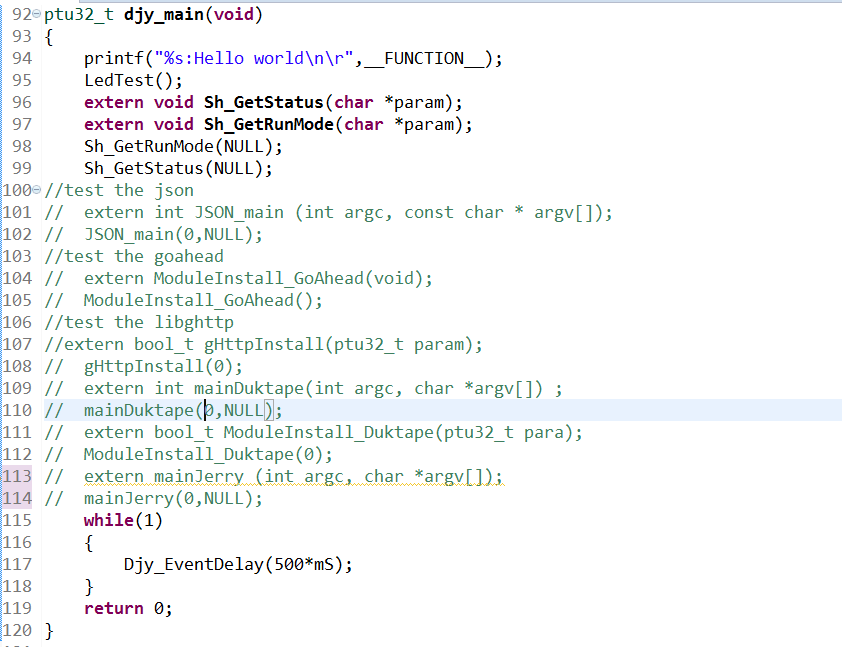
JS引擎资源占比分析

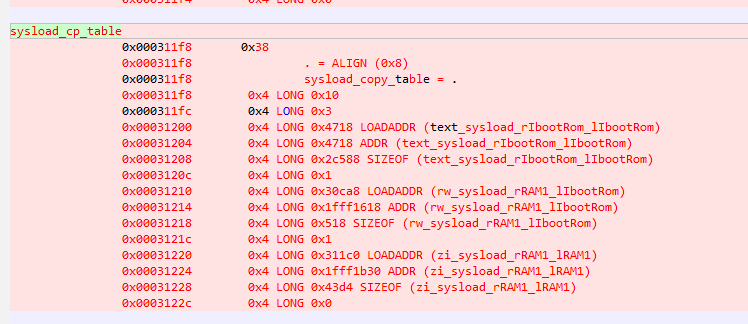
编写：张前福

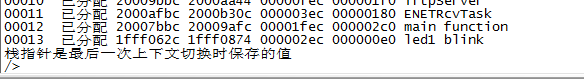
# 静态资源占比分析

## 不带JS引擎



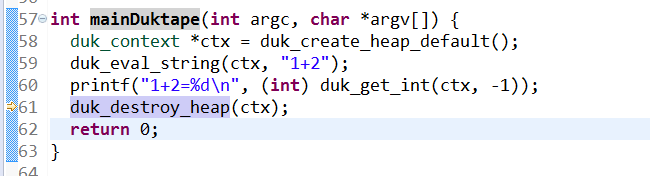
编译后资源堆栈使用情况：



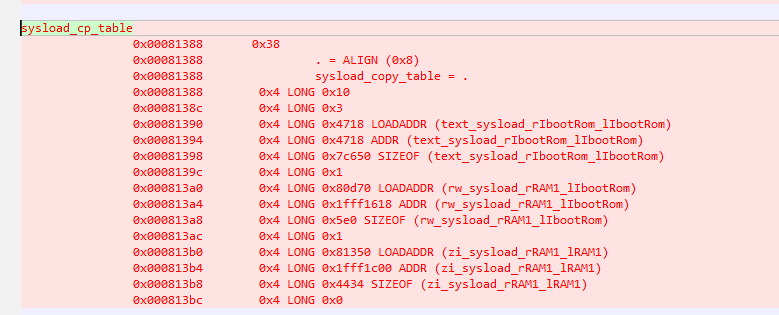


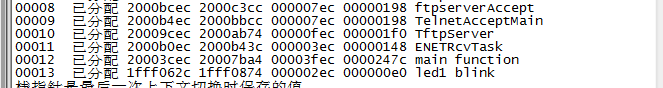
## 使用DUKTAPE

使用形式如下：



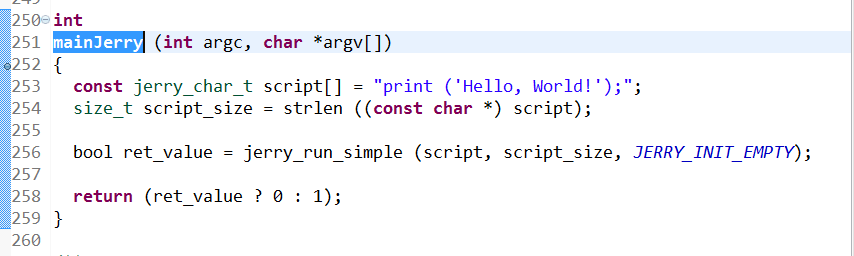
编译后资源堆栈使用情况：



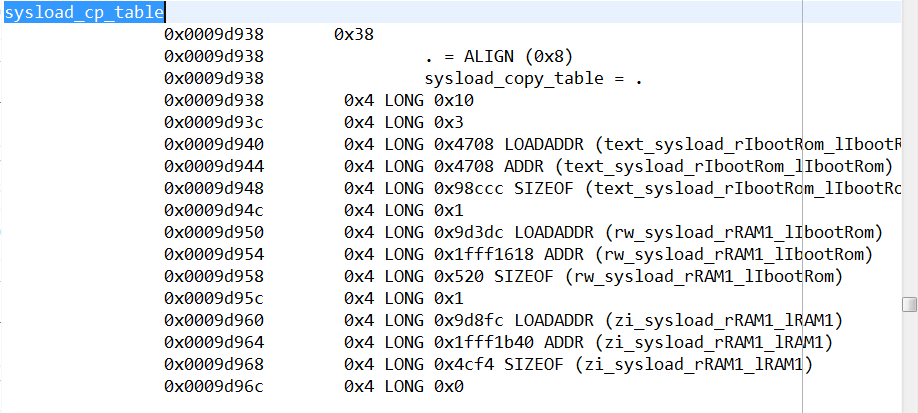


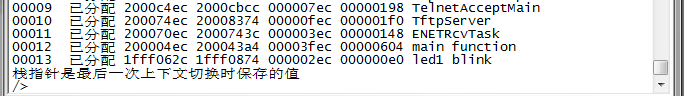
## 使用Jerry（不包含libuv和Iot.Js）

使用形式：



编译后资源堆栈使用情况：





**统计如下：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **资源类型 对比情况** | **text&&rodata** | **data** | **bss** | **stack** |
| **裸main函数** | 0x2c588 | 0x518 | 0x43d4 | 0x2c0 |
| **使用duktape** | 0x7c650 | 0x5e0 | 0x4434 | 0x247c |
| **使用jerry** | 0x98ccc | 0x520 | 0x4cf4 | 0x604 |
| **Duktape资源占用** | **0x500c8（320KB）** | **0xc8(<256B)** | **0x60（<128B）** | **0x21bc(9KB)** |
| **Jerry资源占用** | **0x6c744（440KB）** | **0x08(<16B)** | **0x920（<3KB）** | **0x344(<1KB)** |

上述数据仅仅是静态统计，不包含动态内存分配和使用。

# 动态资源占比

## Duktape

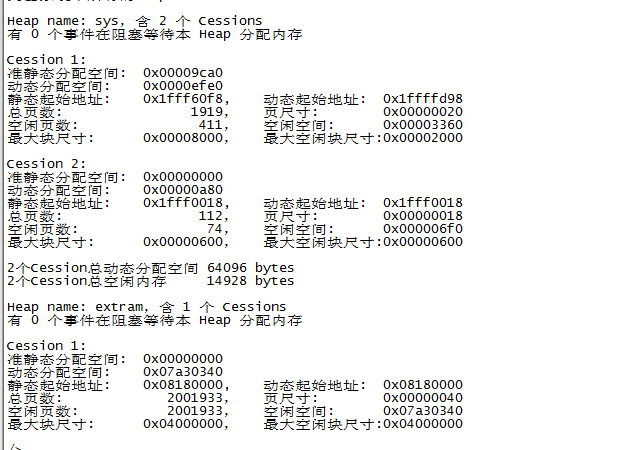
Duktape是使用的系统的malloc函数。测试看看执行一个简单的js语句，消耗的系统ram情况：

测试用例：

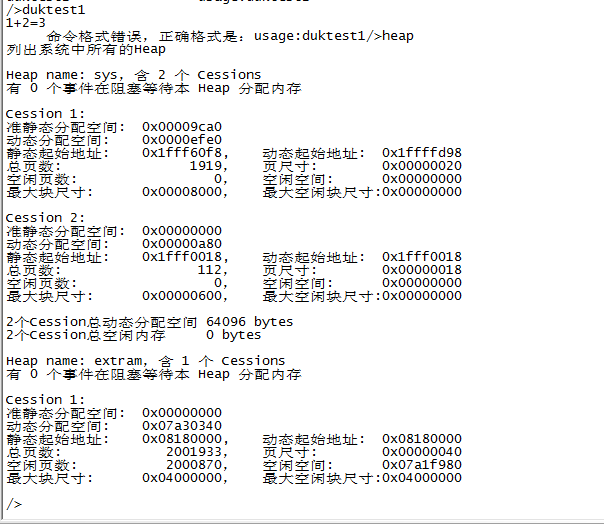


Test1用于执行JS代码，TEST2用于释放JS引擎占用的资源，观察JS引擎启动前、执行中和执行完毕后系统的HEAP情况：如图所示：

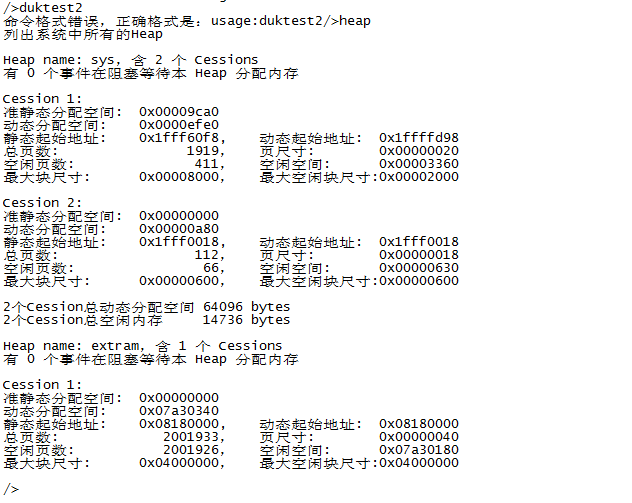
执行前：



执行中：



执行完毕：

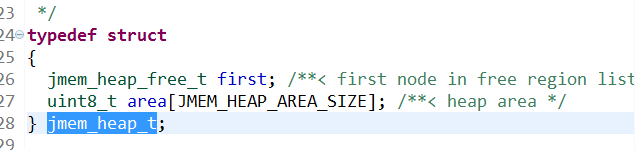


统计结果如下：

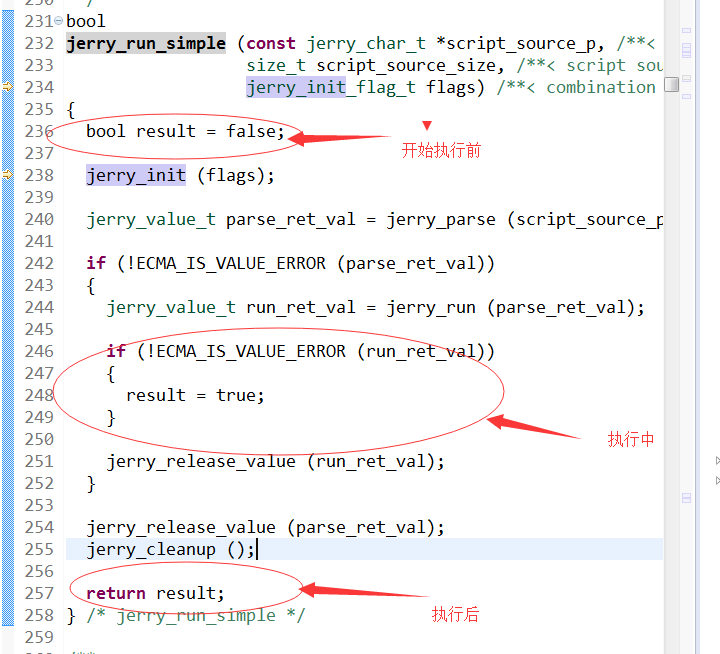
|  |  |  |  |
| --- | --- | --- | --- |
| **heap 对比情况** | **Cession1** | **Cession2** | **Cession3** |
| **执行前** | 0x3360 | 0x630 | 0x7a30340 |
| **执行中** | 0x0 | 0x0 | 0x7a1f980 |
| **执行后** | 0x3360 | 0x630 | 0x7a30180 |
| **消耗RAM** | **0x3360** | **0x630** | **0x109c0** |
| **消耗RAM总计** | **0x14410(82KB)** | | |

## Jerry

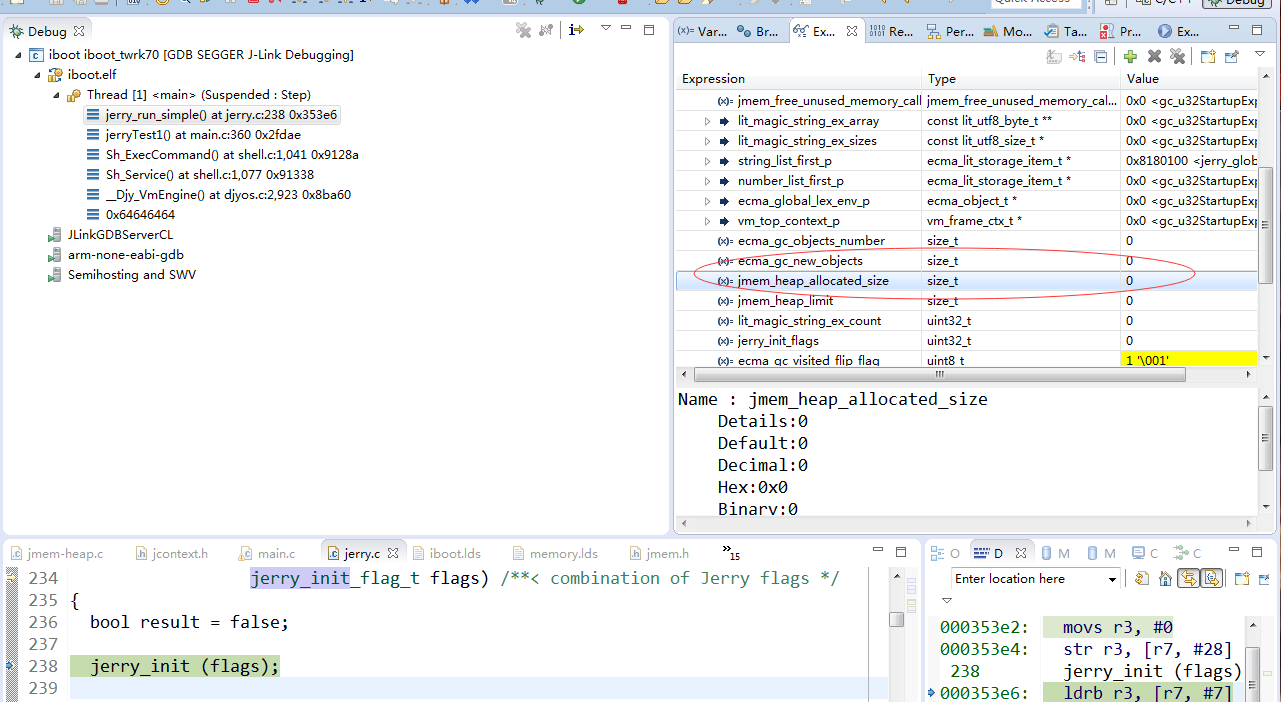
Jerry拥有自己的堆栈分配。最初从整个系统中取一部分静态RAM（很大的一个静态数组，jerry\_global\_heap），定义如下：

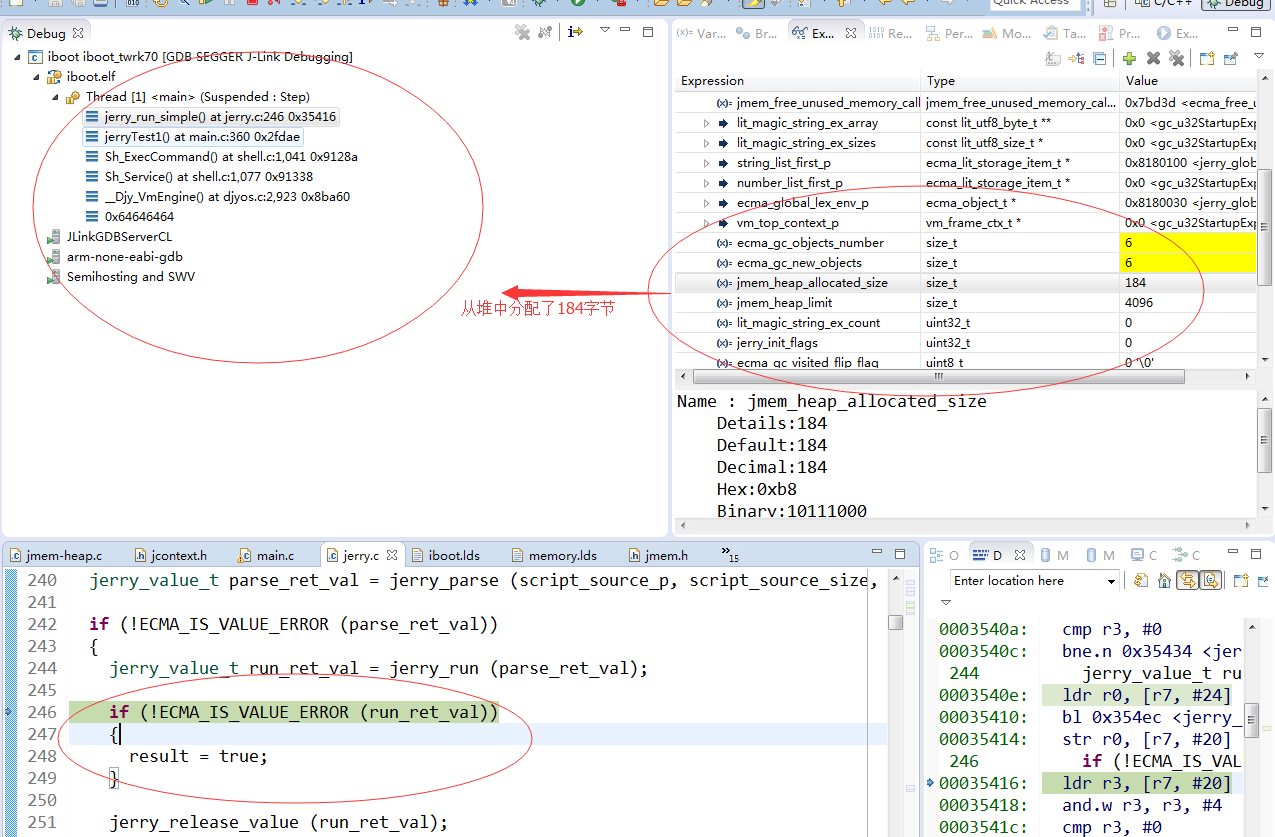


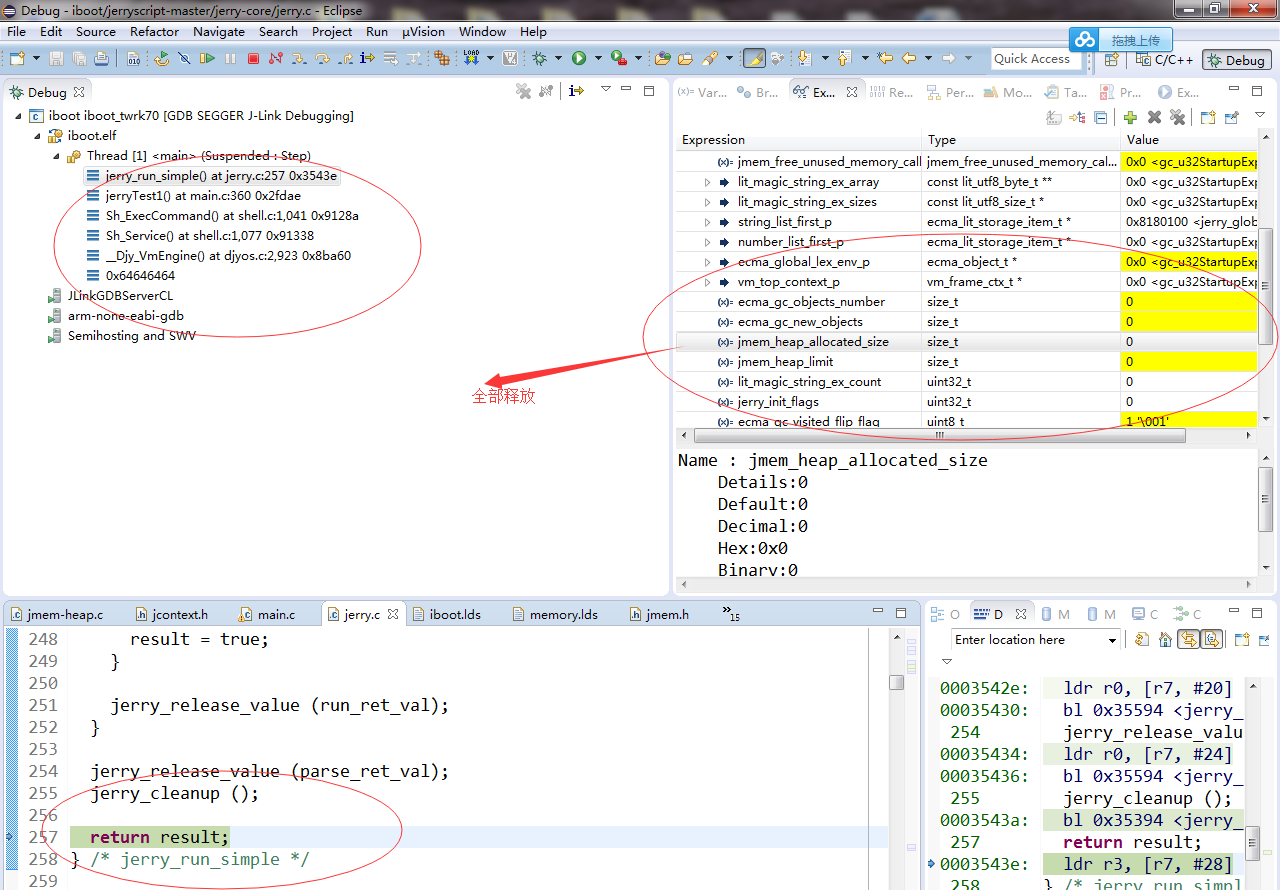
全局量jerry\_global\_context记录了heap的使用情况，其成员jmem\_heap\_allocated\_size记录了已经分配的堆资源，因此通过代码调试关注该变量即可。



观测结果如下所示：







通过统计，jerry执行一个输出语句消耗的内存大概为200字节。

|  |  |
| --- | --- |
| **heap 对比情况** |  |
| **执行前** | 0B |
| **执行中** | 184B |
| **执行后** | 0B |
| **消耗RAM** | **184B** |