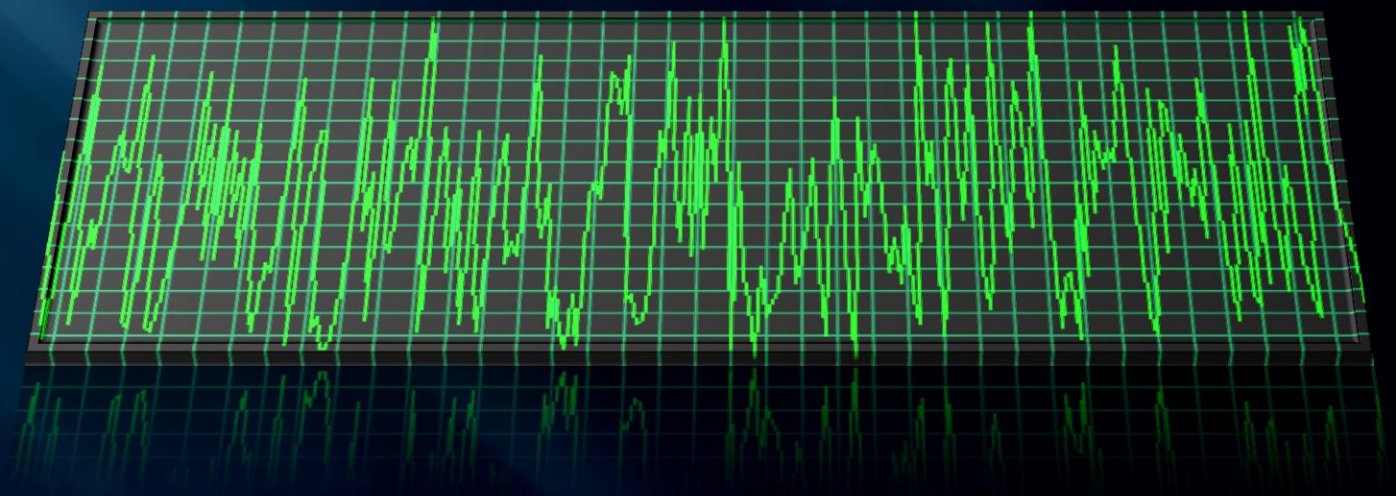
AVM2虚拟机浅析
&
AS3性能优化

By 陈士凯(CSK)
csk@live.com
www.csksoft.net
Ver. 2010.5.29



{

Agenda

- AVM2虚拟机介绍
- 对AS3进行性能分析
- AS3代码优化启发
- 对AVM2进行扩充和改造
- Demos

I

Introduction to AVM2

AVM2 虚拟机

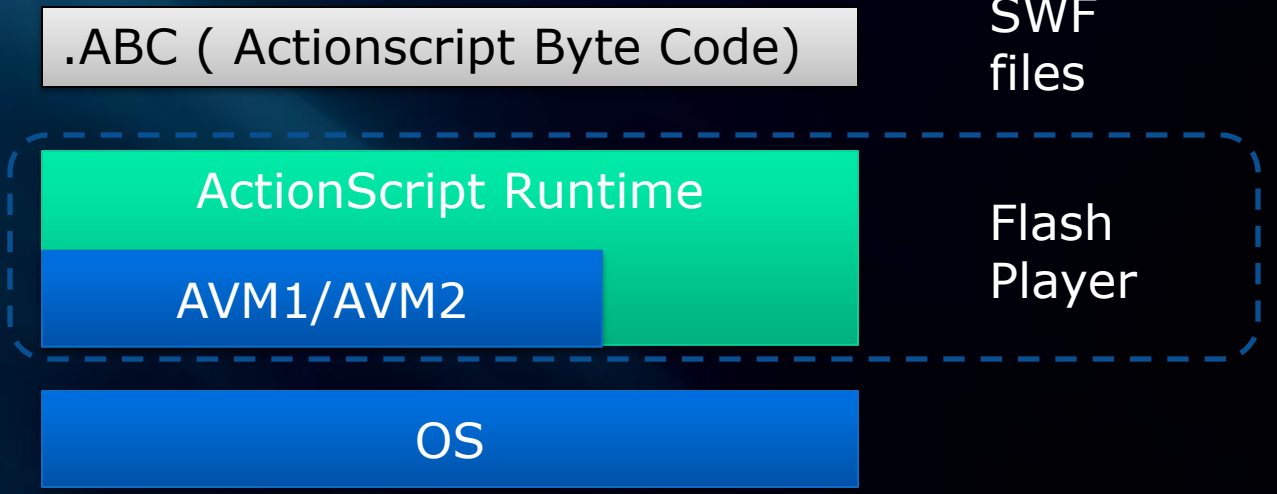
从Flash Player 9.0开始引入，用于AS3.0代码的解释和二进制翻译执行。

采用JIT/Interpret混合执行，大幅提高运行效率
AS3比AS1/2运行速度提升~10x

内建对XML, Array类型的处理支持

已经开源：***Tamarin Project***

Flash Framework

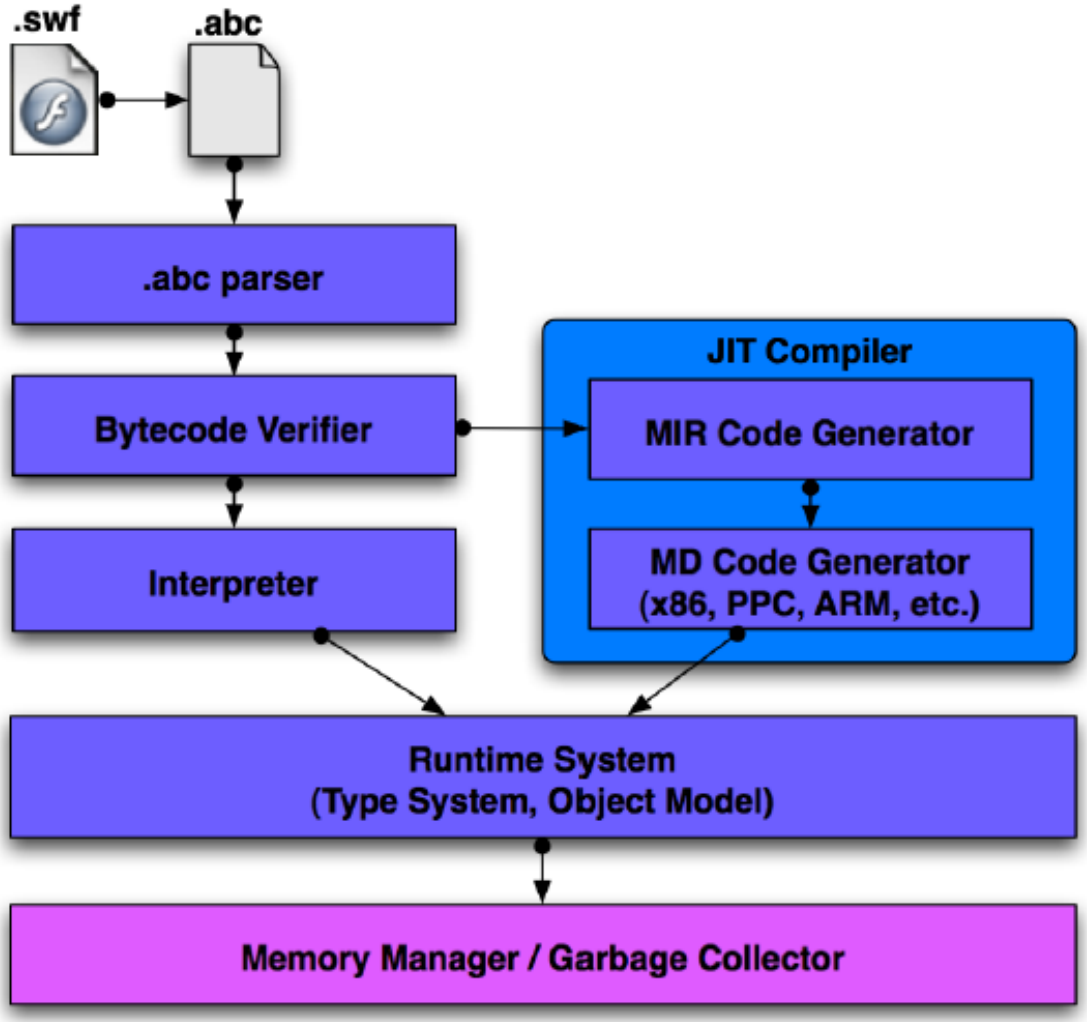


研究AVM2的目的与意义

- 了解AVM2实现和工作特性对AS3执行效率的影响
- AS3代码性能优化
- Flash项目的代码保护和逆向工程
- 增强/扩充AVM2性能/功能，开发第三方AVM2引擎

- 虚拟机实现的学习和研究*

AVM2 Architecture



.ABC 指令集

```
enum AbcOpcode
{
    OP_nop = 0x02,
    OP_throw = 0x03,
    OP_getsuper = 0x04,
    OP_setsuper = 0x05,
    OP_dxns = 0x06,
    OP_dxnslate = 0x07,
    OP_kill = 0x08,
    OP_label = 0x09,
    OP_ifnlt = 0x0C,
    OP_ifnle = 0x0D,
    OP_ifngt = 0x0E,
```

- 基于堆栈机
- 共 172 条指令。
  (数据来源：ActionBlockConstants.h)
- 原生支持类、Closure、异常等特性
- 原生支持Array、XML数据结构

.ABC 指令集一览

指令类别	举例
Load/Store	getlocal, setlocal, …
数值运算	increment, decrement, lessthan, …
位运算	bitnot, bitand, bitor, …
类型转化	coerce, convert_b, coerce_a, convert_i, …
对象创建/操作	newclass, newobject, newarray, …
栈管理	pushnull, pushundefined, pushtrue, dup, …
控制流	iflt, ifle, ifnlt, ifnle, …
函数调用/返回	Call, callproperty, callsuper, …
异常处理	throw
调试支持	Debugfile, debugline
原生数据支持*	Newarray, checkfilter, dxns, …

AS3 -> .ABC

AS3

```
function func(x:int):int {
    var ans:int;
    ans = x + 10;
    return ans;
}
```

.abc

```
0     getlocal0
1     pushscope
2     pushbyte 0
4     setlocal2
5     getlocal1
6     pushbyte 10
8     add
9     convert_i
10    setlocal2
11    getlocal2
12    returnvalue
```

对ABC的解析执行(Interpret)

逐条解析执行，没有优化、低效

```
.abc
0     getlocal0
1     pushscope
2     pushbyte 0
4     setlocal2
5     getlocal1
6     pushbyte 10
8     add
9     convert_i
10    setlocal2
11    getlocal2
12    returnvalue
```

```
for(;;) {
   switch (*pc++){
   …
   case op_add:
      a1 = sp[-1];
      a2 = sp[0];
      sp--;
      …
      dest =
      toplevel->add2(a1, a2);
   }
   …
```

对ABC的即时二进制翻译(JIT)

ABC -> MIR/LIR -> Target Native Code

高效率、允许进行运行时优化，如CSE和死码删除

.abc

```
0     getlocal0
1     pushscope
2     pushbyte 0
4     setlocal2
5     getlocal1
6     pushbyte 10
8     add
9     convert_i
10    setlocal2
11    getlocal2
12    returnvalue
```

MIR/LIR

```
@40 use @17 [0]
@41 use @22 [1]
@42 imm   10
@46 add   @41 @42
@47 def   @46
@49 usea  @47
@50 st 8(@7) <- @49
@54 def @46 spans call
…
@57 use   @54 [4]
@58 ret   @57
```

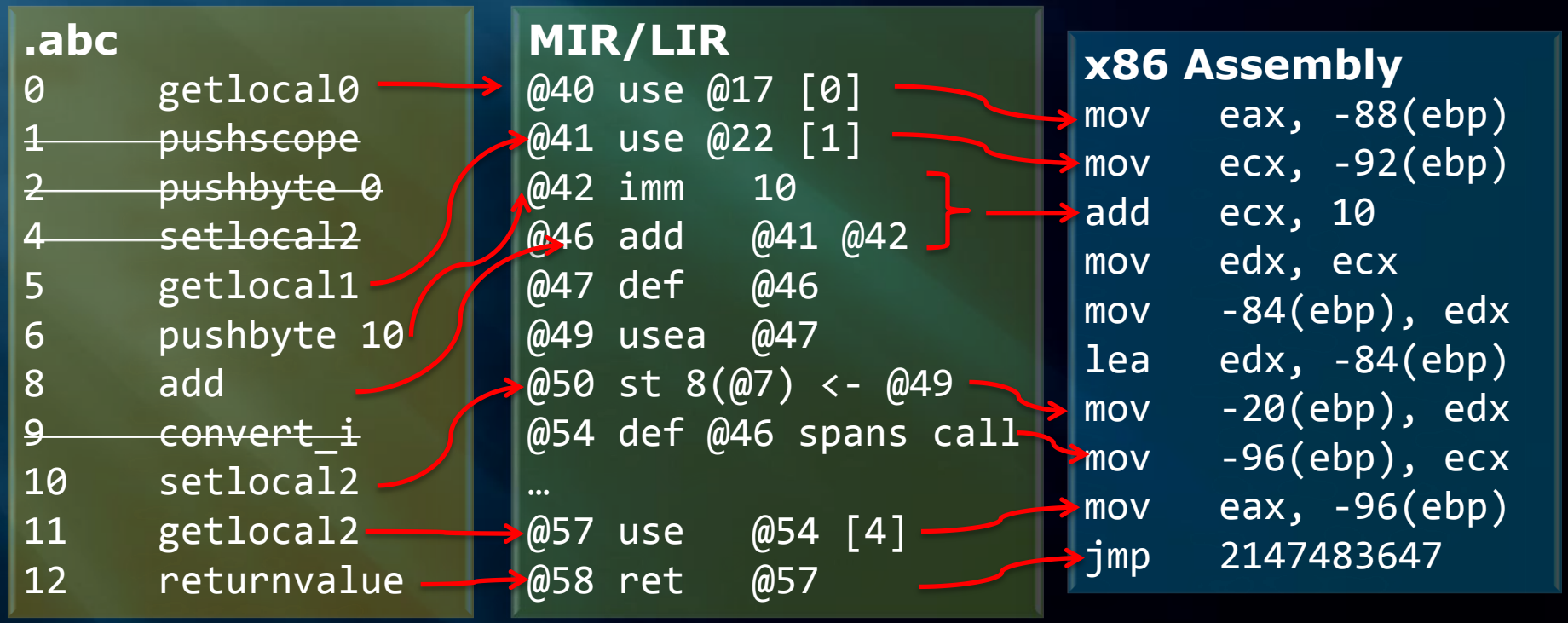
x86 Assembly

```
mov   eax, -88(ebp)
mov   ecx, -92(ebp)
add   ecx, 10
mov   edx, ecx
mov   -84(ebp), edx
lea   edx, -84(ebp)
mov   -20(ebp), edx
mov   -96(ebp), ecx
mov   eax, -96(ebp)
jmp   2147483647
```

对ABC的即时二进制翻译(JIT)

ABC -> MIR/LIR -> Target Native Code

高效率、允许运行时优化，如CSE和死码删除



JIT V.S. Interpret

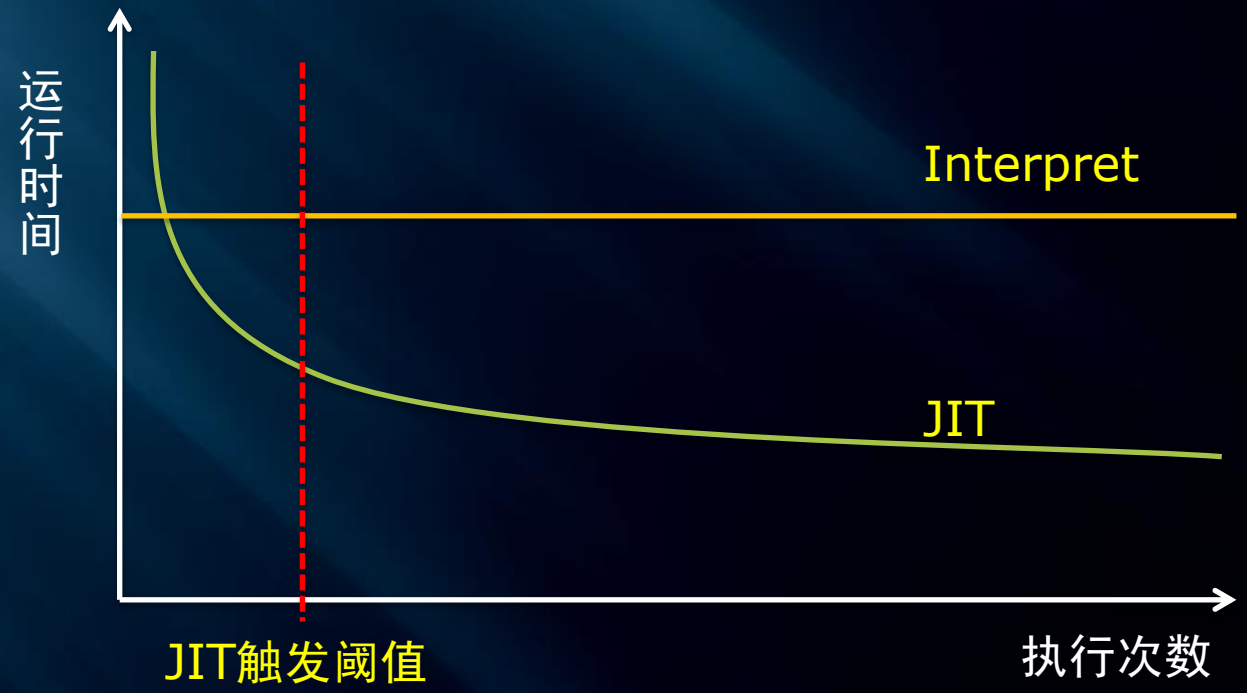
同一段代码在第一次JIT时往往花费的时间较长

代码翻译比较耗时

当再次执行这段代码后，VM将直接调用先前的翻译结果，速度提高。

Interpert 对同一段代码均具有大致相同的执行速度。

JIT并非性能总优于Interpret:



AVM2的JIT策略

对JIT阈值的选择：

- 没有传统的热点(Hotspot)检测机制
- 固定策略：对$init和$cinit区代码进行Interpreting，其他代码均进行JIT

Note: $cinit并非一个类的构造函数。

AVM2 JIT优化

- Concurrent with Verifier
- Early Binding
- Constant Folding
- Copy & Constant Propagation
- Common Subexpression Elimination (CSE)
- Dead Code Elimination (DCE)

* ActionScript 3.0 and AVM2: Performance Tuning

AVM2 的开源进程 -- Tamarin Project

http://www.mozilla.org/projects/tamarin/

Tamarin
[动]绢毛猴(南美洲产)

所有AVM2核心部分：

- AVM2 core
- nanoJIT
- 核心AS3库 (Array/Math/Regexp…)
- ByteArray等部分
- Flash动画渲染部分并未包括

对其他开源项目的影响

- Red-tamarin
- SpiderMonkey(NanoJIT,GC)

了解AVM2的最有利资源

Tamarin Project 的衍生项目

- Tamarin-Tracing Project
  - Adoble贡献于Mozilla, 使用Hotspot Detection进行JIT

- Red-Tamarin Project
  - 注重Shell部分开发，扩充AS3对本地资源设备的控制能力，致力于实现第三方类Flash Player/AIR平台。

- …

II

AS3 Performance Tuning

AS3 代码性能分析与优化

对AVM2自身特性的了解有助于进行AS3代码性能调优

途径:

1. 静态分析 - SWF文件反汇编
2. 动态分析 - Code Tracing 和 Profiling 信息
3. AVM虚拟机实现研究*

ActionScript 3的优化注意点

- 需要手工进行CSE，例如array.length的情况

```
for(var j:int = 0; j < list.length; j++)
{
        copy.push( list[i] );
}
```

~320ms (list.length == 50000)

```
var arrLen:int = list.length;
for(var j:int = 0; j <arrLen; j++)
{
        copy.push( list[i] );
}
```

~93ms (list.length == 50000)

ActionScript 3的优化注意点

- 显式进行类型转换

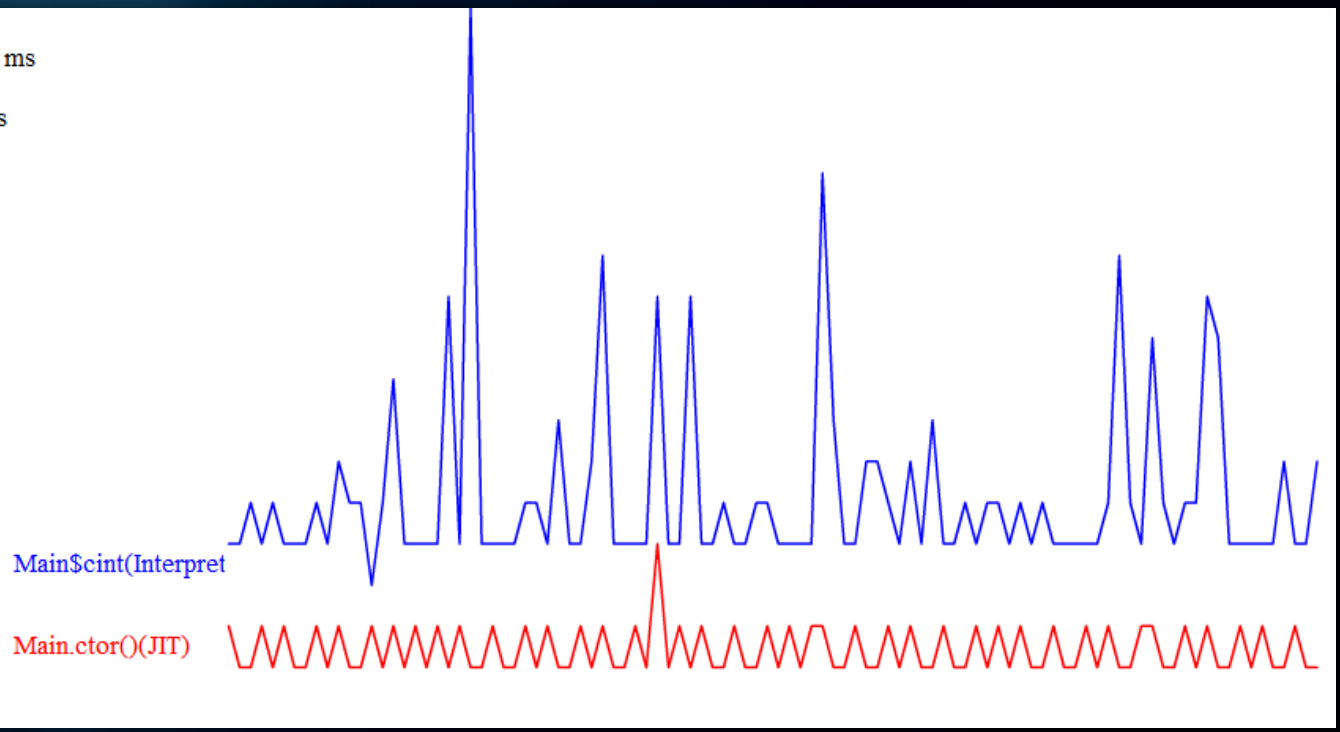
```
var i:int;
// i*2 gets promoted to Number
for (i=0; i<10000; i++) {
        a[i*2] = 0;
}
```

```
for (i=0; i<10000; i++) {
        a[int(i*2+1)] = 1;
}
```

ActionScript 3的优化注意点

- 在$cinit区避免复杂的代码，构造函数不受此限制。

```
Begin $cinit section
End $cinit section, total time used: 418 ms
Begin Ctor section
End Ctor section, total time used: 43 ms
Begin JITPerf loop
----------------------------------------
List of 500000 elements.
for() unoptimized: 421ms
for() key-key optimized: 78ms
for() Pure operation: 10ms
for() FFT operation: 1197ms
----------------------------------------
List of 500000 elements.
for() unoptimized: 279ms
for() key-key optimized: 180ms
for() Pure operation: 9ms
for() FFT operation: 1261ms
----------------------------------------
List of 500000 elements.
for() unoptimized: 266ms
for() key-key optimized: 83ms
for() Pure operation: 21ms
for() FFT operation: 1063ms
----------------------------------------
```



ActionScript 3的优化注意点

- JIT将对连续整形变量相加失效*

```
var a:int, b:int;
```

a
```
b = a + a + a;
```

b
```
b = int(a+a) + a;
```

c
```
b = a + a;
b += a;
```

用时： a(15ms) > b(5.8ms) > c(5ms)

* http://jpauclair.net/2010/03/15/flash-asm/

ActionScript的静态分析 – SWF的反汇编

- 有诸多工具帮助进行swf->abc的反汇编工作
  - swfdump
    - Flex sdk自带
  - abcdump.as
    - 自身也是用AS3实现，运行于Tamarin的AVM2虚拟机中
    - 比较推荐
  - ASV
    - 收费☺

ActionScript的静态分析

- 对编译器产生的abc字节码进行分析，评估代码的执行效率。

```
private function cse_testing(x:int):int {
    var a:int, b:int;
    a = x + 10;
    b = x + 10;
    return a;
}
```

```
function private::cse_testing(int):int      /* disp_id=0 method_id=4 */
{
  // local_count=4 max_scope=1 max_stack=2 code_len=22
  0     getlocal0
  1     pushscope
  2     pushbyte            0
  4     setlocal2
  5     pushbyte            0
  7     setlocal3
  8     getlocal1
  9     pushbyte            10
  11    add
  12    convert_i
  13    setlocal2
  14    getlocal1
  15    pushbyte            10
  17    add
  18    convert_i
  19    setlocal3
  20    getlocal2
  21    returnvalue
}
```

ActionScript的静态分析 – mxmlc优化性能分析

- 目前的mxmlc(flex sdk 4.0.0 build 14159)似乎**不存在**实质的代码优化

```
private function cse_testing(x:int):int {
    var a:int, b:int;
    a = x + 10;
    b = x + 10;
    return a;
}
```

共用子表达式

无用变量(死码)

```
92  function private::cse_testing(int):int      /* disp_id=0 method_id=4 */
93  {
94    // local_count=4 max_scope=1 max_stack=2 code_len=22
95    0     getlocal0
96    1     pushscope
97    2     pushbyte            0
98    4     setlocal2
99    5     pushbyte            0
00    7     setlocal3
01    8     getlocal1
02    9     pushbyte            10
03    11    add
04    12    convert_i
05    13    setlocal2
06    14    getlocal1
07    15    pushbyte            10
08    17    add
09    18    convert_i
10    19    setlocal3
11    20    getlocal2
12    21    returnvalue
13  }
14
```

a=x+10

b=x+10

* 采用优化模式、非debug版本编译

ActionScript的静态分析 – mxmlc优化性能分析

- 目前的mxmlc(flex sdk 4.0.0 build 14159)似乎**不存在**实质的代码优化
  - JIT会进行Common Sub-expression elimination)和DCE(Dead Code Elimination)，从而弥补编译器的薄弱优化

- 影响和暗示
  - Interpret执行时效果将非常差 – 存在无用功：
    - e.g.

```
var a = // Very Slow Operation (no function call)
var b = // Very Slow Operation (no function call)
return a;  //b的动作完全是无效果的
```

  - JIT的启动阶段将消耗更多时间。代码优化比较耗时

- AS3编写注意点：
  - 尽可能手工进行CSE和DCE的优化

ActionScript的静态分析 – mxmlc优化性能分析2

- 对int->Number类型的静态行为分析

```
var a:int;
a = x + 10;
```

```
AS3
pushbyte        10
add
convert_i
setlocal2
```

```
var a:int;
a = int(x + 10);
```

```
AS3
pushbyte        10
add
callproperty    int (1)
convert_i
```

- add 指令默认操作数均为Number类型，纯整数指令为add_i。 编译器并未采用。

- 进行类型转化在静态编译后只会造成代码膨胀，并且在Interpret阶段效率更低。(但JIT引入后实际影响不大)

ActionScript的静态分析 – mxmlc优化性能分析2

- AVM2中对于callproperty的interpret实现代码：

```
2440            INSTR(callproperty) {
2441                u1 = WORD_CODE_ONLY(WOP_callproperty) ABC_CODE_ONLY(OP_callproperty);
2442            callproperty_impl:
2443                SAVE_EXPC;
2444                GET_MULTINAME_PTR(multiname, U30ARG);
2445                i1 = (intptr_t)U30ARG; /* argc */
2446                a2p = sp - i1; /* atomv */
2447                sp = a2p;
2448                if (multiname->isRuntime())
2449                {
2450                    aux_memory->multiname2 = *multiname;
2451                    sp = initMultiname(env, aux_memory->multiname2, sp);
2452                    multiname = &aux_memory->multiname2;
2453                }
2454                a1 = *sp; /* base */
2455                if (u1 == WORD_CODE_ONLY(WOP_callproplex) ABC_CODE_ONLY(OP_callproplex))
2456                    a2p[0] = nullObjectAtom;
2457                *sp = toplevel->callproperty(a1, multiname, (int32_t)i1, a2p, toplevel->toVTable(a1));
2458                if (u1 == WORD_CODE_ONLY(WOP_callpropvoid) ABC_CODE_ONLY(OP_callpropvoid))
2459                    sp--;
2460                NEXT;
2461            }
```

ActionScript的静态分析 – AS3优化编译器ASC

- Flex中提供了另一个编译器 asc.jar, **其包含比较强大的优化能力**
  - 位于目录 Flexsdk_root/lib/
  - 支持编译产生.swf, .exe, .abc
  - 用于Tamarin项目的内置AS3类编译
- 缺陷
  - 使用不方便，需要手工指定所有依赖的类
  - 不支持mxml
  - 需要playerglobal.abc与global.abc文件 (可取自Adobe Alchemy)
  - 对多AS文件编译存在问题*
  - 采用未公开的-o2编译选项开关，可能不稳定

```
H:\avm2_analysis\demo\samplecode\asc_optimize>H:\avm2_analysis\bin\asc-o.cmd --help
ActionScript 3.0 for AVM+
version 1.0 build 14159
Copyright (c) 2003-2007 Adobe Systems Incorporated
Copyright (c) 1998-2003 Mountain View Compiler Company
All rights reserved

Usage:
  asc {-AS3|-ES|-d|-f|-h|-i|-import <filename>|-in <filename>|-m|-p}* filespec
  -AS3 = use the AS3 class based object model for greater performance and better error
  -ES = use the ECMAScript edition 3 prototype based object model to allow dynamic over
  -d = emit debug info into the bytecode
  -f = print the flow graph to standard out
  -h = print this message
  -i = write intermediate code to the .il file
  -import <filename> = make the packages in the
```

ActionScript的静态分析 – AS3优化编译器ASC

```
private function cse_testing(x:int):int {
    var a:int, b:int;
    a = x + 10;
    b = x + 10;
    return a;
}
```

```
function private::cse_testing(int):int        /* disp_id=0 method_id=0 */
{
  // local_count=2 max_scope=1 max_stack=2 code_len=7
  0       getlocal0
  1       pushscope
  2       getlocal1
  3       pushbyte            10
  5       add
  6       returnvalue
}
```

ActionScript的动态分析 – Code Tracing 和 Profiling

- FlashPlayer提供了mm.cfg文件供开发者开启虚拟机的代码跟踪和性能分析：
  - 文件路径(Win32)：
    %USERPROFILE%\mm.cfg
  - 产生的日志文件(Win32)：
    %USERPROFILE%\Application Data\Macromedia\Flash Player\Logs

- 常用的参数

TraceOutputBuffered = 1	开启Trace输出缓冲
AS3Verbose = 1	开启ABC代码执行跟踪
AS3Trace = 1	开启AS3函数级别执行跟踪
AS3StaticProfile = 1	开启静态Profiling信息(代码和尺寸统计)
AS3DynamicProfile = 1	开启动态Profiling信息(各指令用时与统计)
LogGPU = 1	开启GPU使用信息

- 更多的参数信息：
  - 参考文章： http://jpauclair.net/2010/02/10/mmcfg-treasure/

ActionScript的动态分析 – Code Tracing 和 Profiling

AS3Trace = 1开启AS3函数级别执行跟踪

```
1255552 AVMINF: MTHD ProfilerAgent/stopProfiling () @ 0x05DA35A0
1255552 AVMINF: MTHD global/flash.sampler::stopSampling () @ 0x0A8C2B20
1255553 AVMINF: MTHD flash.display::DisplayObject/get root () @ 0x0A8C06B0
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255553 AVMINF: MTHD flash.net::Socket/flush () @ 0x0A8C2AD0
1255553 AVMINF: MTHD flash.net::Socket/close () @ 0x0A8C2B70
1255553 AVMINF: MTHD flash.net::Socket/_init () @ 0x0A8C0DF0
1255553 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
1255554 AVMINF: MTHD flash.utils::Timer/reset () @ 0x0A8C1B20
1255554 AVMINF: MTHD flash.utils::Timer/get running () @ 0x0A8C1C30
1255554 AVMINF: MTHD flash.net::Socket/internalClose () @ 0x0A8C2D00
1255554 AVMINF: MTHD flash.events::EventDispatcher/removeEventListener () @ 0x0A8C2110
1255554 AVMINF: MTHD flash.utils::Timer/stop () @ 0x0A8C2CB0
1255554 AVMINF: MTHD flash.system::System$/resume () @ 0x0A8C2D50
```

ActionScript的动态分析 – Code Tracing 和 Profiling

AS3DynamicProfile = 1

```
total count=278 cycles=35508249 avg CPI=127727
user      1.9%
gc        0%
decoder  97.7%
verifier  1.6%
codegen   0.2%
count       cycles        %count    %time     CPI        opcode
-----       --------      -------   -------   ---        ------
2           34714742       0.7      97.7      17357371   decode
211         602237        75.8       1.6      2854       verifyop
6           89138          2.1       0.2      14856      codegenop
3           78305          1.0       0.2      26101      newclass
7           4813           2.5       0.0      687        findpropstrict
6           3922           2.1       0.0      653        setslot
4           3260           1.4       0.0      815        initproperty
4           2724           1.4       0.0      681        getproperty
9           1859           3.2       0.0      206        getlocal0
```

ActionScript的动态分析 – Code Tracing 和 Profiling

使用ASVerbose=1开关后，AVM2将所有真实的ABC执行过程保存在flashlog.txt

```
34993 exporting ::QName
34994 verify global$init()
34995                          stack:
34996                          scope:
34997                           locals: global *
34998   0:getlocal0
34999                          stack: global
35000                          scope:
35001                           locals: global *
35002   1:pushnamespace http://adobe.com/AS3/2006/builtin
35003                          stack: global Namespace
35004                          scope:
35005                           locals: global *
35006   3:setslot 8
35007                          stack:
35008                          scope:
35009                           locals: global *
35010   5:getlocal0
35011                          stack: global
35012                          scope:
35013                           locals: global *
35014   6:pushdouble Infinity
35015                          stack: global Number
35016                          scope:
35017                           locals: global *
35018   8:setslot 9
35019                          stack:
35020                          scope:
35021                           locals: global *
35022   10:getlocal0
35023                          stack: global
<d Settings\csk\Application Data\Macromedia\Flash Player\Logs
```

谨慎使用该功能！

- 将产生非常大量的tracing log
- AS代码执行速度会有明显减速

ActionScript的动态分析举例 – 分析Flash启动机制和JIT规律

所有代码段均经历类似的三个过程：
Verify(包含MIR产生) -> (目标Native code产生) -> 执行Native Code/解析执行

快速定位到自己感兴趣的内容：
e.g. 搜索关键词　verify Main可定位到Main类的构造函数的验证阶段

```
0858                          stack:
0859                          scope: global@1467101
0860                           locals: global@1467101
0861   34:returnvoid
0862 exit global$init()
0863 verify Main()
0864     define incoming args
0865        @0       arg   0
0866        @1       arg   0
0867        @2       arg   0
0868        @3       arg   0
0869        @4       arg   0
0870        @5       arg   0
0871     alloc local traits
```

ActionScript的动态分析举例 – 分析Flash启动机制和JIT规律

$init $cinit 和构造函数

- $init为全局脚本的初始化函数，用于初始化一个as文件为单位中的类对象（并不是类实例）。$init一般的Qname是global$init或者script$init

- $cinit是一个Class中用于初始化静态变量的静态构造函数。仅在创建类对象时调用，不在创建实例时调用。

```
Class Main
{
   static var myVal:int;

   myVal = someFunction();
   Main():void{
      …
   }

}
```

```
function Main()
{
    //构造函数
}


static function Main$cinit()
{
   myVal = someFunction();
}
```

ActionScript的动态分析举例 – 分析Flash启动机制和JIT规律

$init $cinit 和构造函数 – Tracing的结果

- 执行顺序为：

当前AS脚本为单位：

global$init() -> Main$cinit() -> exit global$init() -> Main()

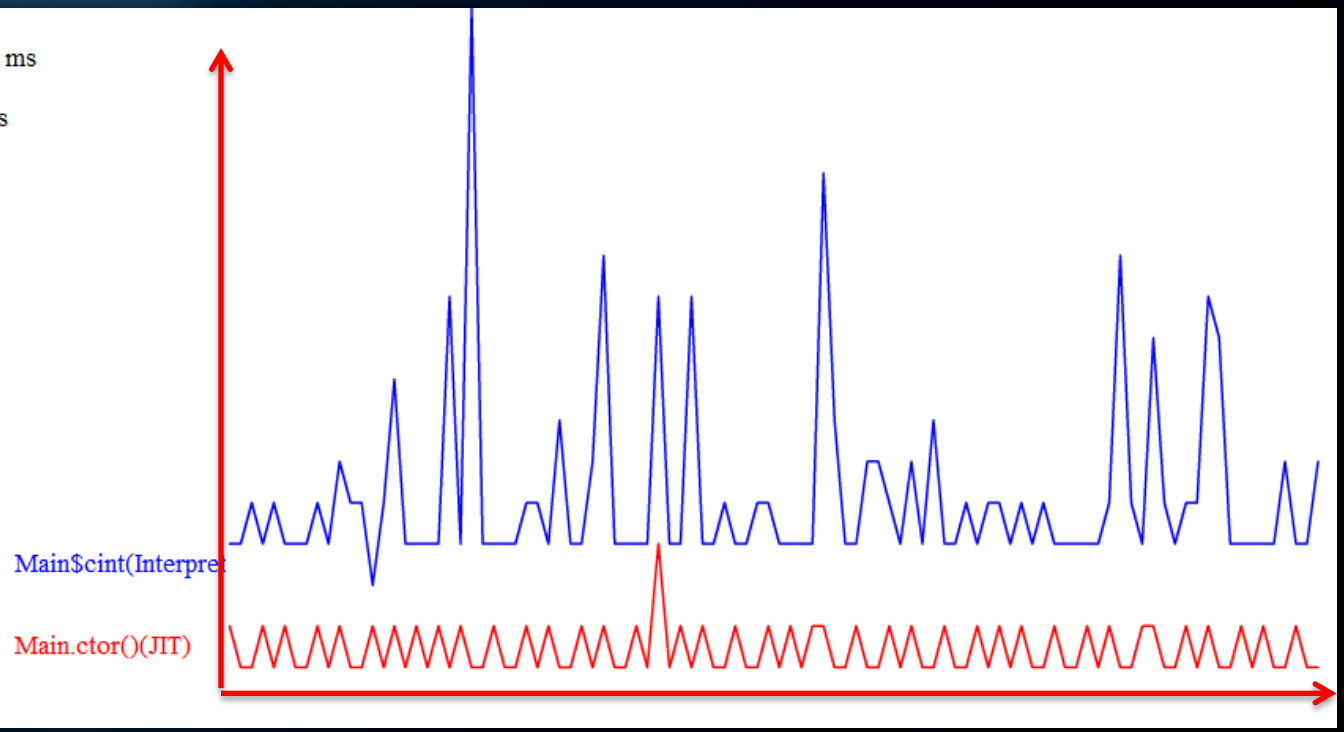
非JIT部分

```
51096 generate Main()
51097     022F0000  cmp   esp, 2147483647
51098     022F0006  jb    50816 interp Main$cinit()
51099     022F000C  push  50817                          stack:
51100     022F000D  mov   50818                          scope:
51101     022F000F  sub   50819                           locals: Main$
                          50820   0:getlocal0
                          50821                          stack: Main$
                          50822                          scope:
                          50823                           locals: Main$
                          50824   1:pushscope
```

ActionScript的动态分析举例 – 分析Flash启动机制和JIT规律

结论：构造函数也会被JIT翻译，仅静态构造函数会被解析执行。

```
Begin $cinit section
End $cinit section, total time used: 418 ms
Begin Ctor section
End Ctor section, total time used: 43 ms
Begin JITPerf loop
----------------------------------------
List of 500000 elements.
for() unoptimized: 421ms
for() key-key optimized: 78ms
for() Pure operation: 10ms
for() FFT operation: 1197ms
----------------------------------------
List of 500000 elements.
for() unoptimized: 279ms
for() key-key optimized: 180ms
for() Pure operation: 9ms
for() FFT operation: 1261ms
----------------------------------------
List of 500000 elements.
for() unoptimized: 266ms
for() key-key optimized: 83ms
for() Pure operation: 21ms
for() FFT operation: 1063ms
----------------------------------------
```



图中演示了$cinit和构造函数执行相同的benchmark的性能表现

ActionScript的动态分析举例 – 分析Flash启动机制和JIT规律

- JIT将对连续整形变量相加失效*

a `b = a + a + a;`

```
getlocal1
getlocal1
add
//不存在convert_i，JIT不做整数优化
getlocal1
add
//convert_i存在，但前一次结果为Number
convert_i
setlocal2
```

b `b = int(a+a) + a;`

```
findpropstrict int //额外函数调用
getlocal1
getlocal1
add
callproperty int (1)
getlocal1
add
convert_i
setlocal2
```

c

```
b = a + a;
b += a;
```

```
getlocal1
getlocal1
add
convert_i
setlocal2
getlocal2
getlocal1
add
convert_i
setlocal2
```

用时： a(15ms) > b(5.8ms) > c(5ms)

分析AVM2实现(Tamarin)进行AS3代码优化

- 举例
  - Array类的AVM2端实现：
    - 对应C++实现ArrayObject和ArrayClass
    - 采用2种数据结构：HashTable和Vector(固定数组)
  - 对性能的影响*：

```
for (i = 0; i < 100000; i++) {
  myArray[i] = i;
}
```

~11ms

```
for (i = 100000; i >=0; i--) {
  myArray[i] = i;
}
```

~44ms

* http://jpauclair.net/2009/12/02/tamarin-part-i-as3-array/

分析AVM2实现(Tamarin)进行AS3代码优化

- 举例
  - Vector类的AVM2端实现：
    - 采用传统C/C++定长数组

```
virtual void grow(uint32 newCapacity, bool exact = false)
{
    if (newCapacity > m_capacity)
    {
        if( !exact )
            newCapacity = newCapacity + (newCapacity >>2);
        //newCapacity = ((newCapacity+kGrowthIncr)/kGrowthIncr)*kGrowthIncr;
        T *newArray = new T[newCapacity];
        if (!newArray)
        {
            toplevel()->throwError(kOutOfMemoryError);
        }
        if (m_array)
        {
            memcpy(newArray, m_array, m_length * sizeof(T));
            delete [] m_array;
        }
        memset(newArray+m_length, 0, (newCapacity-m_capacity) * sizeof(T));
        m_array = newArray;
        m_capacity = newCapacity;
    }
}
```

分析AVM2实现(Tamarin)进行AS3代码优化

- 举例
  - Vector类的AVM2端实现：
    - 与Array性能对比(读取操作)*

Data Types	[1] flash.Vector		[2] Array
	typed	dynamic	Array
Int	19 ms	57 ms	51 ms
UInt	19 ms	57 ms	51 ms
Double[8]	19 ms	64 ms	57 ms
String	65 ms	61 ms	54 ms
Bool	53 ms	52 ms	46 ms
Object	70 ms	64 ms	58 ms
mean	41 ms	59 ms	53 ms

* http://code.google.com/p/polygonal/wiki/FlashPlayerContainerPerformance

ActionScript的增强 – 使用内联汇编或者第三方编译器产生优化ABC

- 内联汇编
  - as3c
- 编写第三方AS3编译器
  - mxmlc开源
  - Tamarin的RTC代码(Run-time compiler for ActionScript 3)
  - Tamarin的ESC代码(ES4 compiler written in ES4)
  - 参考：AS3eval
- 使用Alchemy和Haxe*

ActionScript的增强 – AS3C

- 允许内联ABC汇编或者进行代码替换。

e.g. 简单但高效的整数相加 (目前编译器优化不理想)

```
public function Main()
{
  var src:int;
  src = 1;
  __asm(
        Op.getLocal1();
        Op.pushInt(10);
        Op.addInt();
        Op.setLocal1();
  );
}
```

III

AVM2 Mod

对AVM2的定制

Red-Tamarin

扩充Tamarin对底层资源的访问：Socket，文件、图形、C++类调用

Goal

- the short goal of the project is to provide most of the C standard library (ANSI and POSIX) to an ActionScript environment.
- the medium goal is to provide specialized libraries to use sockets or database like SQLite etc.
- the long term goal is to provide a native API that replicate some part of the Flash Player and Adobe Integrated Runtime (AIR) API

AVM2 定制 – 增加自己的Native Class

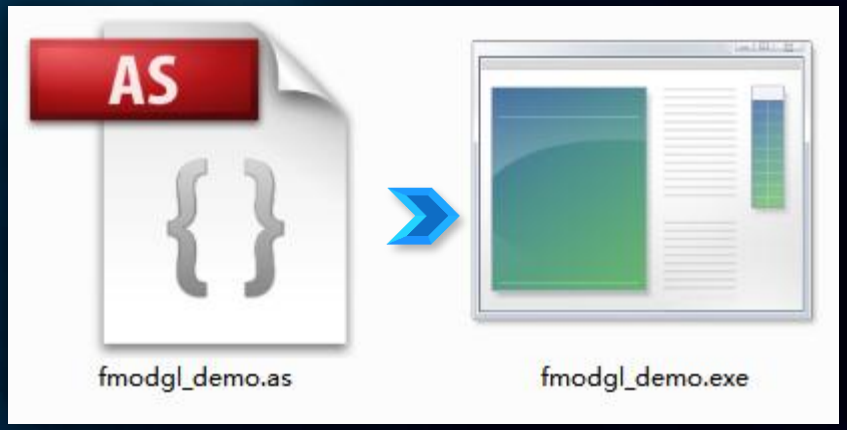
- 利用Tamarin的nativegen.py实现AS代码调用C++类函数
  - 实例：简易的AS3 OpenGL库

```
[native(cls="GlClass", methods="auto")]
internal class __glclass
{
    public native static function _createGLWindow( width:int, height:int, depth:int, isFullWindow:Boolean):int;
    public native static function _disposeGLWindow():void;

    public native static function _glClear( mask:uint):void;
    public native static function _glLoadIdentity():void;

    public native static function _glTranslatef(x:Number, y:Number, z:Number):void;
    public native static function _glRotatef(angle:Number, x:Number, y:Number, z:Number):void;


    public native static function _glBegin(mode:uint):void;
    public native static function _glEnd():void;

    public nati
    public nati
    public nati
}
```

```
/**
 * Simple OpenGL Native Wrapper For AS3
 */
class GlClass : public ClassClosure
{
public:
    GlClass(VTable* cvtable);
    ~GlClass();

    int _createGLWindow( int width, int height, int depth, bool isFullWindow);
    void _disposeGLWindow();

    void _glClear( unsigned int mask);
    void _glLoadIdentity();

    void _glTranslatef(double x, double y, double z);
    void _glRotatef(double angle, double x, double y, double z);
```

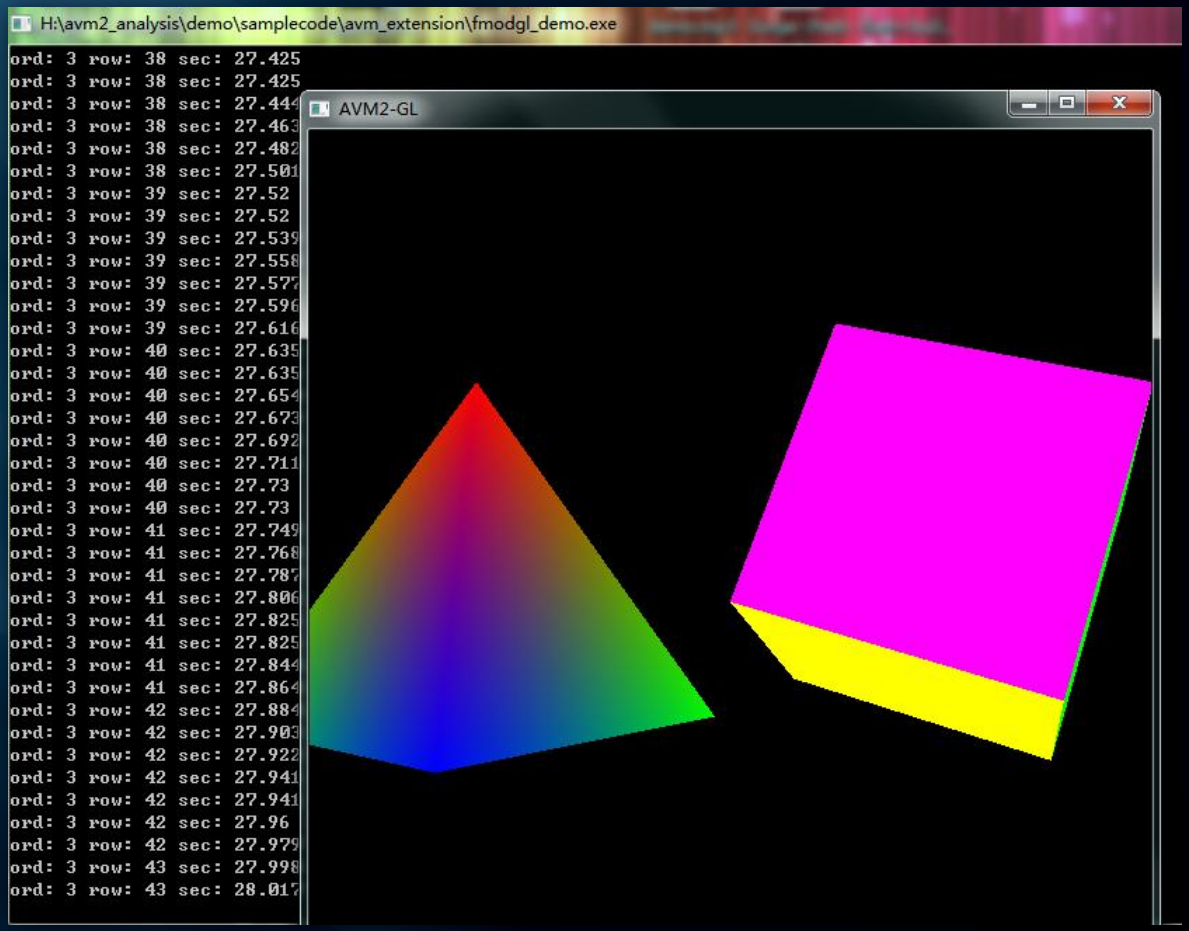
AVM2 定制 – 打包并发布成可执行文件

- 使用asc.jar的-exe参数可以将指定的as脚本编译并和AVM2虚拟机打包成一个可执行文件。



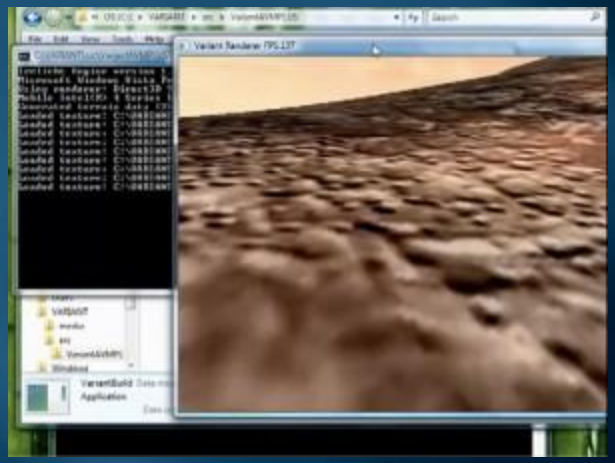
AVM2 定制 – DEMO

- AS3版本的Fmod和Opengl



AVM2 定制 -- RedTamarin

- E.g. Actionscript3 driving that hardware accelerated 3d engine



http://labs.influxis.com/?tag=avm2

200 fps
3D Engine: Irrlicht
C++ Hosting

Questions?

Reference

参考资料

- ActionScript Virtual Machine 2 (AVM2) Overview, Adobe
- ActionScript 3.0 and AVM2: Performance Tuning, Adobe
- Tamarin Project, http://www.mozilla.org/projects/tamarin/
- AS3 hidden treasure in the mm.cfg file
- Tamarin part I : AS3 Array
- Tamarin part II – More on Array and Vector
- Tamarin part III – Garbage Collector in Flash (10.0)
  http://jpauclair.net/

Reference

参考资料

- HOWTO work with Native Classes
  - http://code.google.com/p/redtamarin/wiki/NativeClasses
- Flash Player 9+ container performance comparison table
  - http://code.google.com/p/polygonal/wiki/FlashPlayerContainerPerformance
- Tamarin Build Documentation
  - https://developer.mozilla.org/En/Tamarin/Tamarin_Build_Documentation
- Red-Tamarin Project
  - http://code.google.com/p/redtamarin/

// Backups

扩充资料

.ABC 指令集 – 获得完整的指令列表

- Adobe ActionScript Virtual Machine 2 (AVM2) Overview
  - 官方文档
  - 含有指令行为描述
  - 有少量bug
  - 有未公开指令

- Tamarin代码: ActionBlockConstants.h/avmcore.cpp
  - 最详细的指令列表
  - 最详细的指令行为描术 (Interpret行为代码)
  - 可能与FlashPlayer行为存在差别
  - 纯C++代码，不便于查阅

- FlashPlayer AS3 Tracing Log
  - FlashPlayer真实行为的写照
  - Tracing Log体积过大，不容易查找
  - 需要有一定反汇编和调试经验

AVM2 Verify 过程

```
50801                          locals: global@1467101
50802   24:newclass Main$cinit()
50803 verify Main$cinit()
50804                         stack:
50805                         scope: [global Object$ flash.events::EventDispatcher$ flash.display::DisplayObject$ flash.display::
      InteractiveObject$ flash.display::DisplayObjectContainer$ flash.display::Sprite$]
50806                          locals: Main$
50807   0:getlocal0
50808                         stack: Main$
50809                         scope: [global Object$ flash.events::EventDispatcher$ flash.display::DisplayObject$ flash.display::
      InteractiveObject$ flash.display::DisplayObjectContainer$ flash.display::Sprite$]
50810                          locals: Main$
50811   1:pushscope
50812                         stack:
50813                         scope: [global Object$ flash.events::EventDispatcher$ flash.display::DisplayObject$ flash.display::
      InteractiveObject$ flash.display::DisplayObjectContainer$ flash.display::Sprite$] Main$
50814                          locals: Main$
50815   2:returnvoid
50816 interp Main$cinit()
50817                         stack:
50818                         scope:
50819                          locals: Main$
50820   0:getlocal0
50821                         stack: Main$
50822                         scope:
50823                          locals: Main$
50824   1:pushscope
50825                         stack:
50826                         scope: Main$
50827                          locals: Main$
50828   2:returnvoid
50829 exit Main$cinit()
50830                         stack: global@1467101 Main$
50831                         scope: global@1467101 Object$ flash.events::EventDispatcher$ flash.display::DisplayObject$ flash.display
      InteractiveObject$ flash.display::DisplayObjectContainer$ flash.display::Sprite$
50832                          locals: global@1467101
50833   26:popscope
50834                         stack: global@1467101 Main$
50835                         scope: global@1467101 Object$ flash.events::EventDispatcher$ flash.display::DisplayObject$ flash.display
      InteractiveObject$ flash.display::DisplayObjectContainer$
50836                          locals: global@1467101
```

AVM2 JIT优化分析 – CSE 和 DCE

```
0:getlocal0
       @40          use   @17 [0]
1:pushscope
2:pushbyte 0
          cse   @12
4:setlocal2
5:getlocal1
       @41          use   @22 [1]
6:pushbyte 10
       @42          imm   10
8:add
       @43          i2d   @41
       @44          i2d   @42
       @45          fadd  @43 @44
9:convert_i
       @46          add   @41 @42
          dead   @45
          dead   @44
          dead   @43
10:setlocal2
11:getlocal2
12:returnvalue
          cse   @15
@58 ret   @57
```

AS3优化编译器ASC使用简介

- 未公开的优化选项开关 –o2
- 编译一个as文件至swf:
  - Main.as -> 800x600 60fps swf
  - java –jar asc.jar –import global.abc –import playerglobal.abc –swf main,800,600,60 main.as
- 编译as至abc
  - java –jar asc.jar –import otherlibl.abc –in otherfile.as –o2 main.as
- –import playerglobal.abc 与 –o2 无法同时出现 (Bug?)

AS3优化编译器ASC– ASC/Main.java

```
case 'o':
    if ( "-O".equalsIgnoreCase(flag))
    {
        optimize = true;
    }
    else if (flag.substring(0, 3).equalsIgnoreCase("-O2") )
    {
        if ( null == optimizer_configs)
        {
            optimizer_configs = new ObjectList<ConfigVar>();
        }

        if ( flag.length() > 4)
        {
            String option_name = flag.substring(4);
            String option_value = "";

            int eq_pos = option_name.indexOf('=');

            if ( eq_pos != -1 )
            {
                option_value = option_name.substring(eq_pos+1);  //  Skip the "=" char
                option_name  = option_name.substring(0, eq_pos);
            }

            optimizer_configs.add(new ConfigVar("o2", option_name, option_value));
        }
    }
    break;
```

分析AVM2实现(Tamarin)进行AS3代码优化

```
// This routine checks to see if our dense portion is directly next
// to any entries in our HT.  If so, the HT entries are deleted and added
// to the dense portion.  If the HT is completely emptied, it is cleared.
void ArrayObject::checkForSparseToDenseConversion()
{
    // check for lowHTentry being consumed
    if (m_lowHTentry == NO_LOW_HTENTRY)
        return;

    if (getDenseLength() != m_lowHTentry)
        return;

    while (getDenseLength() == m_lowHTentry)
    {
        AvmAssert (ScriptObject::hasUintProperty (m_lowHTentry));

        // Move prop from HT to dense Array. No need to update m_length
        Atom lowHT = ScriptObject::getUintProperty (m_lowHTentry);
        this->m_denseArr.push (lowHT);

        // Delete prop from HT
        ScriptObject::delUintProperty (m_lowHTentry);

        // If our low entry happened to match our length, we're out of HT entries
        // and we can just quit.
        if ((m_lowHTentry + 1) == m_length)
        {
            m_lowHTentry = NO_LOW_HTENTRY;
        }
        else
        {
            // Find the next integer HT prop and update m_lowHTentry
            // This is tricky.  Our HT section could be huge but very sparse
            // Do we want to linearly walk from index+1 to m_length or do
            // we want to walk the entire HT looking for a low integer value?
```

* http://jpauclair.net/2009/12/02/tamarin-part-i-as3-array/

Tamarin 的支持平台

Platform	Status
Windows XP	supported, acceptance and performance tests automated in buildbot
Windows 64 bit XP	supported, acceptance and performance tests automated in buildbot
mac x86 OSX 10.4	supported, acceptance and performance tests automated in buildbot
mac x86 OSX 10.5	supported, acceptance tests automated in buildbot
mac x64 OSX 10.5	supported, acceptance and performance tests automated in buildbot
mac ppc OSX 10.4	supported, acceptance and performance tests automated in buildbot
mac ppc OSX 10.5	supported, acceptance tests automated in buildbot
mac ppc64 OSX 10.5	supported, acceptance tests automated in buildbot
linux x86 Ubuntu 8.04	supported, acceptance and performance tests automated in buildbot
linux x64 Ubuntu 8.10	supported, acceptance tests automated in buildbot
Windows Mobile 5	supported, acceptance and performance tests automated in buildbot
Solaris Sparc 10	supported, acceptance tests automated in buildbot

谢谢各位