Lua on Unity

《龙门镖局》手游项目简介

《啪啪三国》

- 全3D战争卡牌策略网游
- 核心玩法还是“任务—战斗—扭蛋（抽卡）—强化”模式，卡牌的属性为兵种，技能，特性和战力
- 特点：战场，全3D构建的战场上可容纳最多千人进行对战。


神雷天罚 - 张角
对敌方所有部队造成大量伤害


啪啪三國


啪啪三國


魏
武帝·曹操
蜀
昭烈·刘备
群
魔君·董卓
吴
若虎·孙权

《龙门镖局》

- 国内首款Q版3D实时劫镖制，卡牌收集类
- 在2014CJ期间获得360星耀盛典“年度最受期待手游”大奖
- 金翎奖的角逐中又夺得“玩家最期待移动网络游戏”大奖
- 爆笑、幽默风格，夸张的战斗特效和风趣的技能设计


4:05
自动战斗
PCGAMES
太平洋游戏网


龙门镖局
手游来了
Longmen Express


车库
788
12712
1638098
科技 Lv.5
1116/2000
预览
初阶
中阶
高阶
阿拉灯神 高阶
牛逼值 1320
生命 24200
法攻 462
攻速 157
技能
需要 镖局 55 级 或 VIP 8 级


镖队牛逼值 4715
局錄門龍
镖车
阵容
小皮 xiaopi.com

《啪啪三国》

- 初心
  - 没有当年的《啪啪三国》，就没有现在的《龙门镖局》
- 热更新
  - 表格数据、活动图、音效等资源，打包成assetbubdle 参考 http://www.xuanyusong.com/archives/2373
  - 好方便！
- 但代码……
  - C#写业务很享受~ ☺☺
  - 上线前提心吊胆！ ☹
  - 上线后一定有bug！ ☹
  - 整包更新哭到死！ ☹
  - 整包终于更新好了，结果又出新bug，还得重新整包更新！ ☹☹☹

对比

啪啪三国

- 全3D战争卡牌策略网游
- 纯C#框架
- 每次更新都要各种测试，就怕有更新上线有bug
- 每逢出包必通宵

龙门镖局

- C# + lua
- unity测试一遍，应用端测试一遍，几乎不担心bug
- 几乎不通宵

解决方案？

- Unity C# reflection
  - iOS不能用　(System.Reflection.Assembly.Load 无法使用,System.Reflection.Emit 无法使用,System.CodeDom.Compiler 无法使用)　IOS 下不能动态载入dll或者cs 文件
  - 效率一般
- Lua
  - 够轻量级,脚本语言里最快的
  - 游戏客户端常用
- Python
  - 太重量级了
  - 游戏客户端较少使用
- CSLight
  - 书写略丑，略不方便
  - 现在看性能依然不突出

结果

- 以上分析：lua是适合做热更新

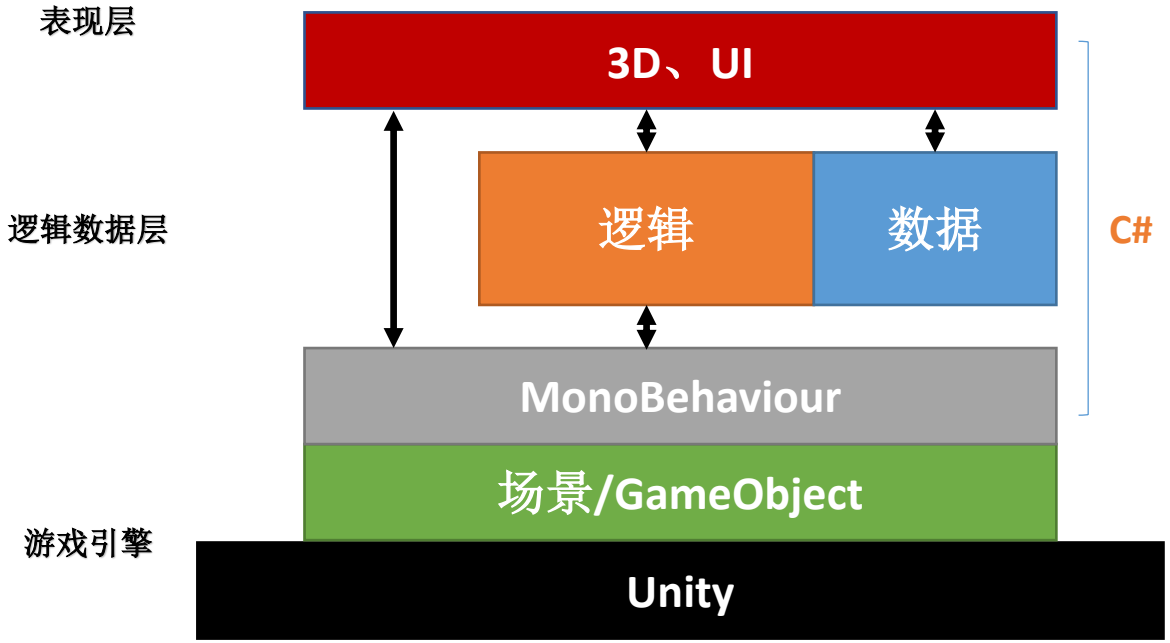
用Lua！

- Pros
  - 支持热更新！
  - 编译后天然防反编译
  - 无类型
    - 方便书写
- Cons
  - 没有IDE和编译器
    - 调试不方便
  - 无类型
    - 调试不方便

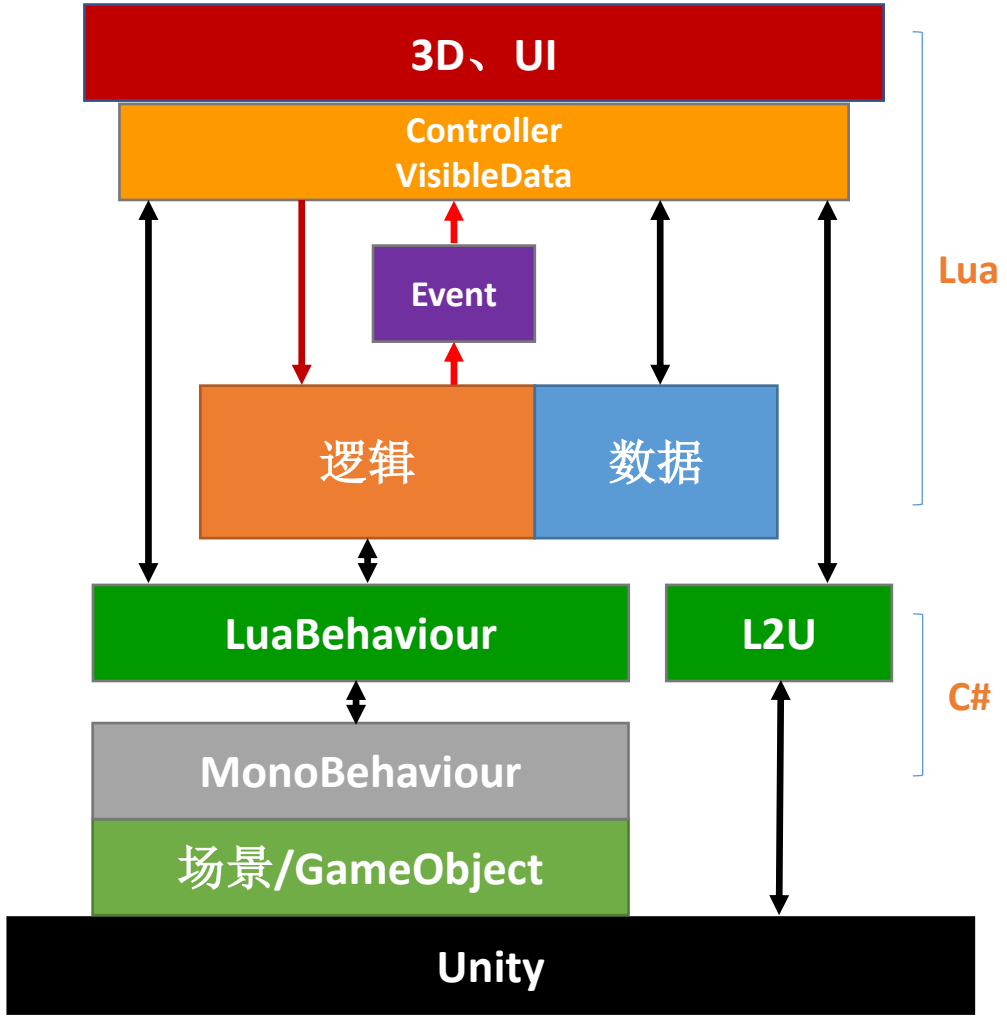
  - 真的调试不方便？

架构设计

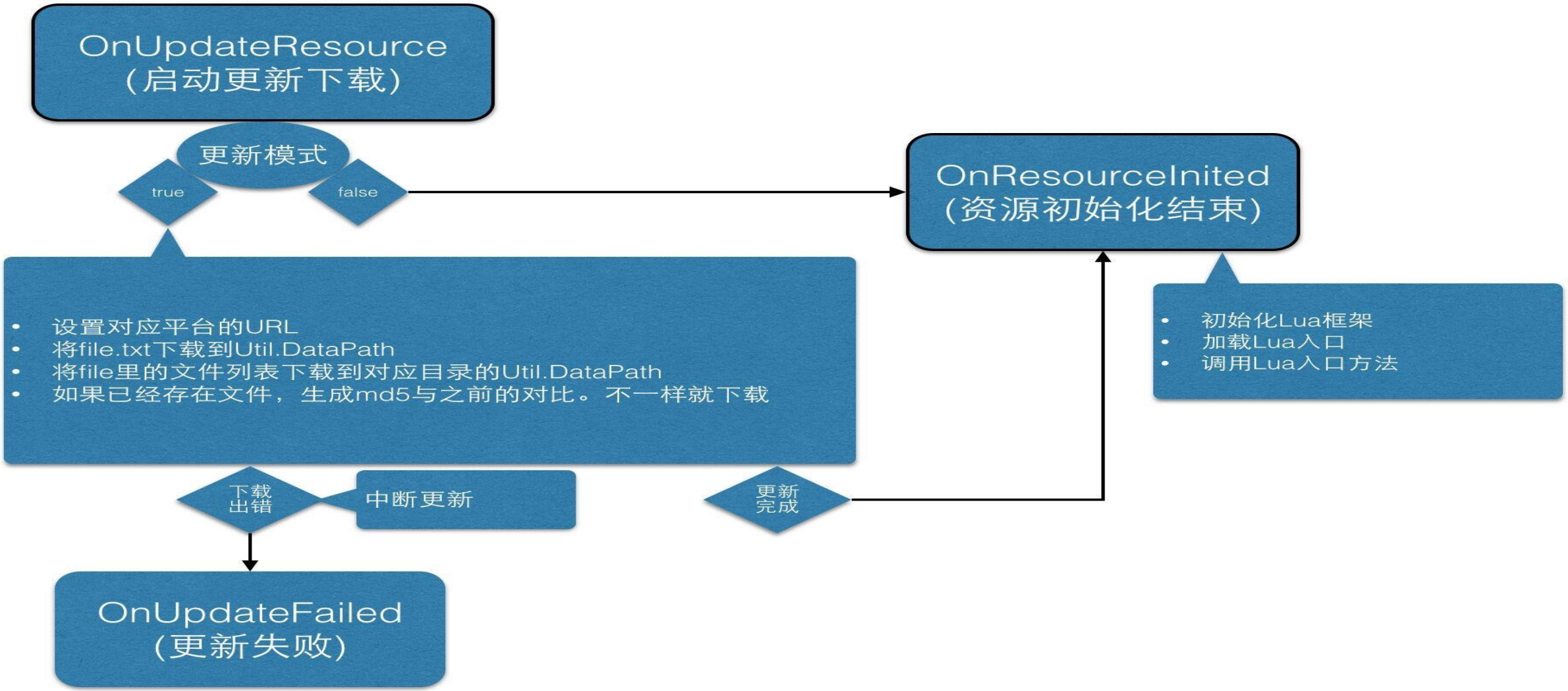
- Unity传统架构图



- Unity+Lua架构图



更新流程


OnUpdateResource
(启动更新下载)
更新模式
true
false
OnResourceInited
(资源初始化结束)
设置对应平台的URL
将file.txt下载到Util.DataPath
将file里的文件列表下载到对应目录的Util.DataPath
如果已经存在文件，生成md5与之前的对比。不一样就下载
初始化Lua框架
加载Lua入口
调用Lua入口方法
下载
出错
中断更新
更新
完成
OnUpdateFailed
(更新失败)

C#调用Lua

- 实例化 lua state对象
- 加载lua代码 LuaState. DoString(string) string为脚本文件
- 调用lua中的方法

```
public class CreateGameObject : MonoBehaviour {

    private string script = @"
            luanet.load_assembly('UnityEngine')
            luanet.load_assembly('Assembly-CSharp')

            Demo = luanet.import_type('Demo')
            Camera = luanet.import_type('UnityEngine.Camera')
            Vector3 = luanet.import_type('UnityEngine.Vector3');
            Resources = luanet.import_type('UnityEngine.Resources')
            GameObject = luanet.import_type('UnityEngine.GameObject')

            local go = GameObject('NewObj')
            go:AddComponent('Demo')

            Demo.echo('echoaaa---------------------------------->>>>'); --pri
            Demo.instance();

            local camera = GameObject.Find('Camera');
            go.transform.parent = camera.transform;
            go.transform.localScale = Vector3.one;
            local texture = go:AddComponent('UITexture');
            texture.mainTexture = Resources.Load('ipad');
            texture:MakePixelPerfect();
        ";

    // Use this for initialization
    void Start () {
        LuaState l = new LuaState();
        l.DoString(script);
```

lua调用C#

- 最早之前是使用反射调用,如上图中的方法
- 使用wrap进行反射使用，可以提高效率
- 将C#文件生成wrap文件，当lua虚拟机启动的时候回加载wrap文件，lua就可以识别使用
- lua调用wrap，wrap调用C#

无以规矩 不成方圆

- 文件规范
  - 文件夹结构
  - 文件名
- 代码规范
  - 代码结构
  - 变量名/函数名
  - 调用方式

Lua 编程规范

文件组织

- 文件编码统一使用 UTF-8 格式
- 文件名一律小写，用下划线"_"分隔各单词，".lua"结尾，如：
  `state_machine_base.lua`
- 文件名单词尽量不用缩写
- 单元测试文件名统一以模块名开头，"_test.lua" 结尾，如：
  `task_system_test.lua`

- LuaScripts：Lua 语言扩展
  - _depends：第三方依赖库
  - _tests：自动化测试
  - common：游戏系统底层功能
  - 其他各系统模块文件夹

- 各文件夹下的文件名不可重名！

代码书写

- ","后如接其他内容，须空格分隔
- "()"或"{}"内如有其他内容，两端须空格分隔
- "[]"内两端不加空格分隔，如：
```
math.abs( lhs[1] - rhs[1] )
self[1] = x + ( t[1] - x ) * t
self.events = classEvents( { "gameEntered", "sceneSwitching", "sceneSwitched", } )
```

- 除了 bool 类型可使用 if、if not 进行判断以外，其他所有值类型都必须显式判断值是否符合条件，如：
```
if character.isPlayerSide() then
end
if combats ~= nil and combats.enemies ~= nil then
end
```

- 代码中用空行分隔不同的操作段
- 不要加入无用的多个空行或空格
- 文件最后空一行（且仅空一行）

注释

- 所有注释全部用中文
- 文件头部写上统一格式的注释，描述下文件的用途，如：
```
--
--〈代码文件解释说明〉
--
```
- 变量注释一般跟在变量申明后，如：
```
local v --〈注释〉
```

- 不要写代码已经自解释了的白痴注释！
- 大段复杂代码必须按段写！注！释！

Log 输出

- 所有 log 输出都用英文（句首字母大写）

- print：调试信息
- log：一般日志信息
- error：错误日志

变量

- 函数名首字母小写，后续单词首字母大写，中间无空格
- 变量名单词可以使用缩写，但必须保持变量名可读性
- 所有变量必须先声明后使用，声明时必须显式赋初值，如：
```
self.team = nil
local _camera = nil
```
- 尽量避免定义全局变量！

函数

- 函数名首字母小写，后续单词首字母大写，中间无空格
- 函数名单词尽量不用缩写
- 非全局函数以"_"开头
- 严格限制定义全局函数！

类

- 类定义函数一律非全局，如：
```
local function classEscortController()
```
- 在类定义前引入需要的全局系统，如：
```
require( "escort/escort_system" )
```
- 在类定义前引入需要的其他类定义，一律非全局，如：
```
local class60ControllerBase = require( "gameobject_controller_base" )
```

- 公有成员以"self."开头，保护成员以"self._"开头，私有成员以"local _"开头，如：

```
local function classXxxx()
    local self = {}

    self.state = 0
    local _taskId = 0

    function self._registerOperation( time, operationFunc )
    end

    return self
end
```

- 私有成员函数必须在前置声明中声明，放在其被使用的函数之后实现（或类定义最后）
- 以"_init"命名的非全局成员为构造器，在"return self"之前调用完成初始化
- 函数实现尽量按照其被调用顺序或相关性排列，避免跳跃阅读

- 类定义格式参考：
  - _TEMPLATE_Normal.lua
  - _TEMPLATE_GameObject_Controller.lua
  - _TEMPLATE_System.lua

il2cpp相关

- 由于ulua在il2cpp方式下编译的缺陷，需要避免一些形式的写法
- 不要在lua代码里tail call C#函数，即不要用如下形式书写：

```
function foo( ... )
    xxxx
    return some_cs_function( ... )
end
```

而写为

```
function foo( ... )
    xxxx
    local some_var = some_cs_function( ... )
    return some_var
end
```

- 供lua调用的C#函数不要用默认参数，即不要用如下形式书写：

```
int FunctionForLua( int i, float f = 0 )
{
    xxxx
}
```

而是从lua那边传完整的参数列表
如果认为确实需要默认参数比较好，则使用重载形式，即：

```
int FunctionForLua( int i )
{
    return FunctionForLua( int i, 0 );
}

int FunctionForLua( int i, float f )
{
    xxxx
}
```

事实上，建议多数纯C#代码里也使用重载形式，Unity，NGUI代码里基本也都是这么干的。

symbol被strip相关

- 如果在c#代码里有仅给lua调用而c#那边不会调用的非静态成员或成员函数 ，则在编译c#代码时编译器发
现该symbol没有被使用而可能会strip掉，这样的成员或函数应当在
PrecompilationSymbolsForULua.cs调用
一下避免被strip。

提交代码

- 在 SVN 的提交界面，必须双击打开每个文件查看 diff，确认所有改动都是自己有！意！为！之！

其他

- 要注意全局变量的使用，尽量多使用local变量，局部变量手误写错变成全局变量，成员变量手误写错取值为nil
- 尽量少在update里少调用lua
- lua不支持C#的泛型，不能用GetComponnet<类名>()的形式，只能用GetComponnet (“类名”)的形式去获取组件
- 之前使用sumbline编辑lua（弊端没有任何提示，不可以调试）
- 现在使用babelua编辑（可调试，并且有代码提示，效率高）

Q&A