xLua 튜토리얼

Lua 스크립트 로드

1. **스트링으로 로드**

가장 기본적인 방법은 LuaEnv.DoString을 통해 string형식의 Lua Script를 로드하는 것이다. 당연히 스트링은 Lua의 문법에 적법해야 한다.

예：luaenv.DoString("print('hello world')")

완전한 코드는 XLua\Tutorial\LoadLuaScript\ByString에서 확인할 수 있습니다.

그러나 이런 방식은 추천하지 않는다. 다음의 방식을 추천합니다.

**2. Lua 파일 로드**

**lua의 require함수를 사용합니다.**

예：DoString("require 'byfile'")

완전한 코드는 XLua\Tutorial\LoadLuaScript\ByFile에서 확인할 수 있습니다.

require은 사실상 매 loader를 선택하여 로드를 한다. 하나의 loader가 성공하면 다른 loader는 사용하지 않으며, 모두 실패시 에러를 보고하게 됩니다.

현재 xLua는 기본 Loader외에 Resource loader가 포함되어 있습니다. 주의해할 것은 Resource는 제한된 suffix만을 지원하므로, Resources 디렉토리에 lua파일을 둘때는 .txt suffix를 붙여야 합니다. (첨부예제 참조)

추천하는 Lua스크리트 로드 방식은 전체 어플리케이션내에 하나의 DoString(“require ‘main'")을 두고, main.lua 에서 다른 스크립트를 로드 하는 것입니다.

만일 “내 lua파일은 다운로드 받거나 어떤 특정한 형식으로 압축된 파일에서 꺼낸 파일이고, 암호화를 해야하고 등등등인데 어떻게 해요?” 라는 질문을 받는다면, xLua의 Custom Loader가 해당 요구를 만족할수 있을것입니다.

**3. Custom Loader**

**xLua에 Custom Loader를 추가하는것은 매우 간단하며, 다음의 방식을 사용하면 됩니다.**

public delegate byte[] CustomLoader(ref string filepath);

public void LuaEnv.AddLoader(CustomLoader loader)

AddLoader를 통해 callback를 추가할 수 있습니다. callback시의 인자는 스트링으로 lua코드에서 require 에 사용된 값이 전달되며, callback시 이 인자를 통해 어떤 파일을 로드할 지 결정하게 된다. 만일 디버그를 지원하려 한다면, filepath는 실제 경로로 할 필요가 있다. 리턴값은 바이트코드이며, 빈 테이블일 경우 loader를 찾을 수 없거나 적법하지 않은 Lua 스크립트임을 나타낸다.

이게 있다면 간단하다?? IIPS의 IFS의 사용?? ???

有了这个就简单了，用IIPS的IFS？没问题。写个loader调用IIPS的接口读文件内容即可。文件已经加密？没问题，自己写loader读取文件解密后返回即可。。。

완전한 예제는 XLua\Tutorial\LoadLuaScript\Loader에서 볼수 있다.

C#에서 Lua로 액세스

C#에서 Lua의 데이터구조에 직접 접근하는것

이 장에서 예제는 XLua\Tutorial\CSharpCallLua에서 볼수 있다.

1. Global Field에 접근

LuaEnv.Global을 참조하면 된다. Get메소드 템플릿이 있어 리턴값의 유형을 지정할 수 있다.

luaenv.Global.Get<int>("a")

luaenv.Global.Get<string>("b")

luaenv.Global.Get<bool>("c")

**2. Global Table에 접근**

**역시** Get 메소드를 사용한다. 그렇다면 어떤 타입으로 해야하는가?

1. class나 struct로 매핑  
   대응되는 table의 필드를 public property로 선언한 class를 선언한다. 인자가 없는 함수형식이어도 된다. 예를들어 {f1 = 100, f2 = 100} 테이블의 경우 public int f1;public int f2; 를 포함한 class면 된다.  
   이 방식에서 xLua는 당신을 위해 new 함수를 만들어줄 것이며, 대응되는 필드를 할당해줄겁니다.

table의 property는 class의 property보다 많거나 적어도 됩니다. 복잡한 유형도 매핑할 수 있습니다.

주의해야 할점은 이 과정이 value copy 라는것입니다. 만약 class가 비교적 복잡한 구조라면 copy cost가 커질수 있습니다. 게다가 class의 필드를 변경할때는 table로 바로 동기화되지 않을것입니다. 그 반대도 마찬가지입니다.

이런 방식은 class를 GCOptimize에 추가하여 오버헤드를 줄일수 있습니다. 상세한한 내용은 설정 문서를 참조하세요.

그럼 Reference 방식의 매핑은 없는가? 물론 있습니다.

2) interface에 매핑

이 방식은 Generator에 의존합니다. 미리 해 놓지 않았다면 InvalidCastException이 발생하게 됩니다. Code Generator는 이 interface의 예제를 생성합니다. 만약 프로퍼티에 get이 있다면, Code Generator는 대응되는 table의 필드를 얻을것입니다, 만약 set이 있다면 이 또한 대응되는 필드를 설정하게 될 것입니다. 또한 interface의 메소드로 lua함수를 사용할 수 있습니다.

3) 경량화된 by value 방식 : Dictionary<>나 List<>에 매핑

class나 interface를 정의하기 싫을때, 이 방식을 사용할 수 있습니다. 단 table의 key와 value의 타입이 모두 일치해야 합니다.

不想定义class或者interface的话，可以考虑用这个，前提table下key和value的类型都是一致的。

4) 또다른 방식의 by ref방식 : LuaTable 클래스에 매핑

이 방식의 장점은 코드를 생성할 필요가 없다는 것입니다. 그러나 몇가지 문제가 있는데, 2) 방식에 비해 속도가 느리며, 타입 검사가 없다는 것입니다.

**3. Global function에 접근**

**여전히 Get방식을 사용합니다만 다른점은 Class의 매핑입니다.**

1) delegate에 매핑

이것이 매우 추천하는 방식으로 성능이 좋으며, 안전합니다. 단점은 코드 생성을 해야한다는 점입니다. (코드 생성을 해놓지 않으면 InvalidCastException 이 발생합니다.)

delegate를 어떻게 선언해야 하는가?

function의 모든 인자는 input 타입으로 선언해야합니다.

다중 return값은 어떻게 처리하는가?

왼쪽에서 부터 오른쪽으로 c#의 out 또는 ref 인자로 선언합니다.

파라메터와 리턴값으로 어떤 타입을 지원하는가?

모든 타입을 지원합니다. 각종 복잡한 구조체 및 out, ref, 또 다른 delegate를 return하는것도 가능합니다.

delegate의 사용은 더 쉽습니다. 그냥 함수를 사용하듯이 사용하면 됩니다.

2) LuaFunction에 매핑

이 방식의 장점과 단점은 1)번 방식의 반대이다.

사용도 간단하고, LuaFunction에는 가변인자의 Call함수가 있어, 임의의 타입, 임의의 수량의 인자를 사용할수 있으며, return값은 object의 array여서 lua의 다중 return값을 지원한다.

4. **사용 제안**

1. Lua의 global 데이터에 접근시 특이점은 table 및 function의 binding의 대가가 비교적 크다는 것이다. 그러므로 가능하면 초기화시에 lua function을 한번 얻어와서(delegate에 매핑) 저장해두면, 그 다음부터는 delegate나 table을 직접 사용할 수 있다.

2、如果lua测的实现的部分都以delegate和interface的方式提供，使用方可以完全和xLua解耦：由一个专门的模块负责xlua的初始化以及delegate、interface的映射，然后把这些delegate和interface设置到要用到它们的地方。

Lua에서 C# 으로 접근

본장의 실 예제는 모두 XLua\Tutorial\LuaCallCSharp에 있습니다.

**C# 객체 생성**

**C#에서는 :**

var newGameObj = new UnityEngine.GameObject();

대응되는 Lua에서의 방식：

local newGameObj = CS.UnityEngine.GameObject()

다음 몇가지를 제외하고 기본적으로 동일하다

1. lua에는 new keyword가 없다.
2. 모든 C# 관련 constructor, static 또는 member property, method등은 cs 파일에 둡니다.  
   만약 constructor가 여러개라면? 걱정마세요, xlua는 overload를 지원합니다. 예를 들어 당신이 GameObject가 string 인자를 가지는 constructor라면 다음과 같이 작성합니다.

local newGameObj2 = CS.UnityEngine.GameObject('helloworld')

C# class의 static property, method에 접근

static property의 get

CS.UnityEngine.Time.deltaTime

static property의 set

CS.UnityEngine.Time.timeScale = 0.5

static method의 사용

CS.UnityEngine.GameObject.Find('helloworld')

작은 팁: 만약 자주 접근을 한다면, local 변수로 옮겨 사용하세요. 접근코드의 시간을 제거해, 성능향상에 도움이 됩니다.

local GameObject = CS.UnityEngine.GameObject

GameObject.Find('helloworld')

C# class 객체의 property, method에 접근

**member property의 get**

testobj.DMF

Member property의 set

testobj.DMF = 1024

member method의 사용

주의: member method를 사용시에는 첫번재 인자에 자신이 들어가야 하므로 콜론(:) 문법을 사용해야한다.

testobj:DMFunc()

상위 클래스의 property， method

**xlua는 base class 의 static property， static method 그리고 base class의 member property 및 method에 접근이 가능하다.**

**인자의 입출력 속성 (out, ref)**

**Lua 에서의 인자 처리 규칙: C#의 일반 패러미터 하나는 하나의 인자로 치며, ref역시 하나의 입력 패러미터로 친다. out은 하나의 인자로 사용하지 않으며 왼쪽부터 순서대로 인자목록에 대응된다.**

**Lua 에서 리턴값 처리 규칙:** C# 함수의 리턴값은 하나의 리턴값이 되며, out 및 ref 패러미터 역시 하나의 리턴값으로 처리된다. 왼쪽부터 순서대로 lua의 다중리턴값에 대응된다.

**method overload**

**그냥 다른 타입의 인자를 넣어 사용하면 각각의 overload된 메소드에 접근하게 된다.**

testobj:TestFunc(100)

testobj:TestFunc('hello')

정수인자를 받는 TestFunc와 스트링인자를 받는 TestFunc를 알아서 구분하게 된다.

주의: xlua는 정해진 한도내에서 함수 오버로딩을 지원한다. 왜냐하면 lua의 타입은 C#만큼 풍부하지 않아 타입의 1대다 대응의 상황이 나올수 있기때문에다. 예를들어 C#의 int, float, double은 모두 lua의 number 타입에 대응된다. 앞의 예제중 TestFunc가 이런 타입으로 오버로딩이 되어있었다면, 이것을 구분해 내지 못하고, 한가지만을 사용하게 될 것이다.(생성된 코드중에 제일 처음 것)

operator

지원하는 operator : +, -, \*, /, ==, 단항-, <, <=, %, []

**default parameter 를 가진 method**

**C#에서 사용하는 방법과 동일하다. 만약 실제 인자보다 적은 수의 인자를 주게되면 default value를 붙여 사용하게 된다.**

**가변인자 method**

C#에 다음과 같은 가변인자를 가진 메소드가 있다면

void VariableParamsFunc(int a, params string[] strs)

lua에서 다음과 같이 사용할 수 있다.

testobj:VariableParamsFunc(5, 'hello', 'john')

Extension methods의 사용

C#에 정의되어 있으면 lua에서 직접 사용할 수 있다.

General (Template) 메소드

직접 지원하지는 않는다. Extension methods의 기능으로 추상화하여 사용함.

Enumeration 타입

열거형 값은 열거 타입의 정적 property처럼 사용한다.

testobj:EnumTestFunc(CS.Tutorial.TestEnum.E1)

상기된 EnumTestFunc함수의 인자는 Tutorial.TestEnum형 이다.

그외 만약 열거형이 코드생성에 포함되면, 열거형은 \_\_CastFrom메소드를 지원하여 정수나 스트링으로 부터 변환하여 사용할 수 있다.

CS.Tutorial.TestEnum.\_\_CastFrom(1)

CS.Tutorial.TestEnum.\_\_CastFrom('E1')

delegate의 사용 (사용, +, -)

C#의 delegate의 사용: 보통의 lua함수를 사용하는것과 동일하다.

* operator: 다음의 예시와 같이 사용하며, 동일한 타입의 C# delegate나 lua 함수를 사용할 수 잇다.
* operator: + operator와 동일함.

ps: delegate 프로퍼티는 LuaFunction으로 대입될 수 있다.

예시:

예를 들어 C#의 testobj에 다음과 같이 event가 선언되어있다면

public event Action TestEvent;

Lua에서 다음과 같이 이벤트를 추가하며

testobj:TestEvent('+', lua\_event\_callback)

다음과 같이 이벤트를 제거한다.

testobj:TestEvent('-', lua\_event\_callback)

64bit 정수 지원

Lua 5.3버전은 자체적으로 64bit정수를 지원한다. luajit버전은 lua 5.1의 기반하기 때문에 64bit정수를 지원하지 않는다. xLua는 64비트 정수 지원을 위한 extension을 제공하며, C#의 long과 ulong은 lua의 userdata로 매핑된다.

1. lua에서 64정수의 연산, 비교, 출력을 지원함
2. Lua number와의 연산 비교를 지원함.
3. 주의해야 할것은 64비트 확장에서 실제로는 int64만 존재한다. ulong의 경우도 long으로 컨버트되어 lua에 전달되어 사용된다. 우리는 java와 동일한 방식을 사용하여 관련 API를 제공한다. API문서를 참조바람.

C#의 복잡한 구조체와 table의 자동 전환

C#의 복잡한 구조체와 lua의 table를 자동으로 변환해 준다. 각 필드는 정확하게 일치해야하며, 값은 copy된다. 다음의 예제를 살펴보자.

C#에 다음과 같은 B 구조체(class도 가능)가 있고

public struct A

{

public int a;

}

public struct B

{

public A b;

public double c;

}

이 구조체를 인자로 받는 다음과 같은 함수가 있을때

void Foo(B b)

lua에서 이것을 다음과 같이 사용할 수 있다.

obj:Foo({b = {a = 100}, c = 200})

타입정보 (C#의 typeof)

**예를 들어** UnityEngine.ParticleSystem 클래스의 타입정보를 얻으려면 다음과 같이 사용한다.

typeof(CS.UnityEngine.ParticleSystem)

**type cast**

**lua는 타입이 없기 때문에 강타입 언어의 강제 type cast의 개념이 존재하지 않는다. 그러나 비슷한것이 있다면, xLua에 특정 객체를 사용하기 위해 지정된 생성코드를 사용해야 한다는것이다. 그렇다면 이것은 어디에서 사용이 될까? 서드파티 라이브러리를 사용시 interface나 추상클래스만이 노출되어있고, 실제 클래스가 노출되어 있지 않을때, 이럴때는 코드생성을 할수가 없다. 코드 생성이 되지 않은 클래스를 사용시 xLua에 의해 Reflection을 통해 사용을 하게 되며, 이것이 빈번하게 사용될 경우 성능에 영향을 주게된다. 이때 이 interface나 추상클래스를 이용하여 다음과 같이 지정할 수 있다.**

cast(calc, typeof(CS.Tutorial.Calc))

CS.Tutorial.Calc라는 클래스의 코드를 생성하여 calc 객체를 접근하게 된다.