Atitit.java c# python 防止反编译的一个思路---汇编化

[1. 反编译 为源码 vs 反编译为汇编语言 1](#_Toc27726)

[1.1. 为什么java c#.net能顺利反编译为高级语言源码 为什么c++不能？？ 1](#_Toc21201)

[1.2. Java c# c++都可以被反编译为汇编语言 2](#_Toc9508)

[2. 如何防止java　ｃ#　ｐｙｔｈｏｎ 反编译？？ 2](#_Toc28618)

[2.1. 非常简单：：二次编译为汇编代码即可。 2](#_Toc4458)

[2.2. 大力使用magic num 2](#_Toc5249)

[2.3. 使用自己的私有时间标准 2](#_Toc19787)

[2.4. 调用jna api。比如c++的。这样对方必须要会c++才可以读懂 2](#_Toc12015)

[2.5. 使用汇编风格书写源码，这样对付必须对汇编懂才可以看懂 2](#_Toc12137)

[3. 防止反编译java c#代码范例（使用汇编风格书写） 2](#_Toc8572)

[3.1. 对应的高级语言源码 4](#_Toc1455)

[3.2. ----全部源码 4](#_Toc11003)

# 反编译 为源码 vs 反编译为汇编语言

Java c#.net可以直接反编译为源码，，c++只可以反编译为汇编代码。。反编译出来的只是逻辑和源码是一样的，并不是高级语言的源码。。

因为汇编语言比较麻烦，所以一般不把汇编称为源码。。

## 为什么java c#.net能顺利反编译为高级语言源码 为什么c++不能？？

由于java的面向对象特性和java的反射技术，可以很容易的通过对.class文件进行反射来得到java的源代码，也就是反编译！

此外，为了方便调试，加入了大量调试符号，是的开发者容易调试，也容易被反编译。。

对于java的反编译，并不是所有内容都会反编译出来，java源文件中的注释、泛型等内容在经过编译和反编译之后会丢失

因为c++支持不了这么多特性，所以反编译效果不好。。但是如果是debug级别的c++模块，则加入了很多调试符号，比较容易反编译为c++源码

## Java c# c++都可以被反编译为汇编语言

# 如何防止java　ｃ#　ｐｙｔｈｏｎ 反编译？？

## 非常简单：：二次编译为汇编代码即可。

。当然，为了跨平台特性，可以建立一个java语言上的虚拟机，然后编译为此虚拟机代码即可。。。

## 大力使用magic num

## 使用自己的私有时间标准

## 调用jna api。比如c++的。这样对方必须要会c++才可以读懂

## 使用汇编风格书写源码，这样对付必须对汇编懂才可以看懂

# 防止反编译java c#代码范例（使用汇编风格书写）

主要参考jvm虚拟机汇编书写，为了加强安全性，可以自定义虚拟机指令即可。。更加可以使用基于寄存器的虚拟机。。此处为了简单方便，使用了基于stack的虚拟机。。

**import** com.google.common.collect.Maps;

**public** **class** AtiVM {

Object PCRegister;

Cpu cpu;

Stack stack = **new** Stack();

Map Constant\_pool = Maps.*newConcurrentMap*();

**public** **static** **void** main(String[] args) {

AtiVM vm = **new** AtiVM();

vm.Constant\_pool.put("#16", "Fieldref,#17.#19");

vm.Constant\_pool.put("#17", "Class,#18");

vm.Constant\_pool.put("#18", "Utf8,com/attilax/time/TimeJna$CLibrary");

vm.getstatic(CRuntime.*INSTANCE*);// 获取指定类或者接口的静态域，并将其值压入栈顶

vm.push(**null**); // 将null推送至栈顶

vm.invokeinterface(CLibraryInterface.**class**, "time", 2);// 调用接口方法

// ，从栈顶取出俩个数作为参数，把返回值放入栈顶

vm.push(631123200);

vm.sub();// sub 将栈顶两int型数值相减并将结果压入栈顶

vm.push(850348800l);//

vm.cmp(); // cmp 比较栈顶两long型数值大小，并将结果（1，0，-1）压入栈顶

**if** (vm.ifle()) // ifle 当栈顶int型数值小于等于0时跳转

{

vm.getstatic(System.***out***);

vm.push("ok");

vm.invokevirtual(PrintStream.**class**, "println"); //调用实例方法

vm.ret(); // return

} **else** {

// vm.new0(RuntimeException.class);

vm.push("ex1");

vm.invokespecial(RuntimeException.**class**); // 调用超类构造方法，实例初始化方法，私有方法

vm.athrow(); // 将栈顶的异常抛出

}

System.***out***.println("---finish");

}

## 对应的高级语言源码

**public** **class** LissCheckor {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** i= CRuntime.*INSTANCE*.time(**null**);

//ati tmstmp

**long** now=i-631123200;

**if**(now>850348800)

**throw** **new** RuntimeException("ex");

**else**

System.***out***.println("ok");

}

}

## ----全部源码

**package** com.attilax.vm;

**import** java.io.PrintStream;

**import** java.lang.reflect.InvocationTargetException;

**import** java.lang.reflect.Method;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Stack;

**import** org.apache.commons.lang3.reflect.ConstructorUtils;

**import** com.attilax.c.CLibraryInterface;

**import** com.attilax.c.CRuntime;

**import** com.attilax.collection.AArrays;

**import** com.attilax.collection.ACollections;

**import** com.attilax.lang.AInt;

**import** com.attilax.math.ANum;

**import** com.attilax.ref.refx;

**import** com.attilax.reflect.MethodUtils;

**import** com.google.common.collect.Lists;

**import** com.google.common.collect.Maps;

**public** **class** AtiVM {

Object PCRegister;

Cpu cpu;

Stack stack = **new** Stack();

Map Constant\_pool = Maps.*newConcurrentMap*();

**public** **static** **void** main(String[] args) {

AtiVM vm = **new** AtiVM();

vm.Constant\_pool.put("#16", "Fieldref,#17.#19");

vm.Constant\_pool.put("#17", "Class,#18");

vm.Constant\_pool.put("#18", "Utf8,com/attilax/time/TimeJna$CLibrary");

vm.getstatic(CRuntime.*INSTANCE*);// 获取指定类或者接口的静态域，并将其值压入栈顶

vm.push(**null**); // 将null推送至栈顶

vm.invokeinterface(CLibraryInterface.**class**, "time", 2);// 调用接口方法

// ，从栈顶取出俩个数作为参数，把返回值放入栈顶

vm.push(631123200);

vm.sub();// sub 将栈顶两int型数值相减并将结果压入栈顶

vm.push(850348800l);//

vm.cmp(); // cmp 比较栈顶两long型数值大小，并将结果（1，0，-1）压入栈顶

**if** (vm.ifle()) // ifle 当栈顶int型数值小于等于0时跳转

{

vm.getstatic(System.***out***);

vm.push("ok");

vm.invokevirtual(PrintStream.**class**, "println"); //调用实例方法

vm.ret(); // return

} **else** {

// vm.new0(RuntimeException.class);

vm.push("ex1");

vm.invokespecial(RuntimeException.**class**); // 调用超类构造方法，实例初始化方法，私有方法

vm.athrow(); // 将栈顶的异常抛出

}

System.***out***.println("---finish");

}

// 调用超类构造方法，实例初始化方法，私有方法

**private** **void** invokespecial(Class<?> class1) {

Object[] params = VmUtil.*pop2objArr*(stack);

**try** {

Object o = ConstructorUtils.*invokeConstructor*(

class1, params);

stack.push(o);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

}

}

// 将栈顶的异常抛出

**private** **void** athrow() {

System.***out***.println("--will throw");

RuntimeException pop = (RuntimeException) stack.pop();

**throw** pop;

}

**private** **void** ret() {

// **TODO** Auto-generated method stub

}

**private** **void** getstatic(Object obj) {

stack.push(obj);

}

//调用实例方法

**private** **void** invokevirtual(Class<?> class1, String meth) {

Object[] arr=VmUtil.*pop2objArr*(stack);

Object obj = arr[arr.length-1];

Object[] params=**new** AArrays(arr).left(arr.length-1);

**try** {

Object ret = MethodUtils.*invokeMethod*(obj, meth, params);

stack.push(ret);

} **catch** (NoSuchMethodException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (IllegalAccessException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

} **catch** (InvocationTargetException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

**private** **void** new0(Class<?> class1) {

}

**private** **boolean** ifle() {

**int** num2 = (**int**) **new** ANum(stack.pop()).toInt();

**if** (num2 <= 0)

**return** **true**;

**else**

**return** **false**;

}

// 比较栈顶两long型数值大小，并将结果（1，0，-1）压入栈顶

**private** **void** cmp() {

Double num2 = **new** ANum(stack.pop()).toDouble();

Double num1 = **new** ANum(stack.pop()).toDouble();

**if** (num1 == num2)

stack.push(0);

**if** (num1 > num2)

stack.push(1);

**if** (num1 < num2)

stack.push(-1);

}

// 将栈顶两int型数值相减并将结果压入栈顶

**private** **void** sub() {

Double num2 = **new** ANum(stack.pop()).toDouble();

Double num1 = **new** ANum(stack.pop()).toDouble();

Double numRet = num1 - num2;

stack.push(numRet);

}

**private** **void** push(Object object) {

stack.push(object);

}

@SuppressWarnings("all")

**private** <ttx> **void** invokeinterface(Class<ttx> class1, String meth,

**int** paramNum) {

Object[] objs\_tmp = VmUtil.*pop2objArr*(stack, paramNum);

Object o = objs\_tmp[paramNum - 1];

Object[] params = AArrays.*left*(objs\_tmp, paramNum - 1);

ttx t = (ttx) o;

// try {

// Object ret = MethodUtils.invokeMethod(t, meth, params);

// stack.push(ret);

// } catch (NoSuchMethodException e1) {

// // **TODO** Auto-generated catch block

// e1.printStackTrace();

// } catch (IllegalAccessException e1) {

// // **TODO** Auto-generated catch block

// e1.printStackTrace();

// } catch (InvocationTargetException e1) {

// // **TODO** Auto-generated catch block

// e1.printStackTrace();

// }

**try** {

Method m1 = refx.*getMeth*(class1, meth, params);

Object ret = m1.invoke(o, params);

stack.push(ret);

} **catch** (SecurityException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

} **catch** (IllegalAccessException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

} **catch** (IllegalArgumentException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

} **catch** (InvocationTargetException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**throw** **new** RuntimeException(e);

}

}

}

---end