C# 混淆工具简介

**obfuscar**

1、源码地址：<https://github.com/lextm/obfuscar.git>

2、使用说明

3、效果

4、实现原理

5、源码分析

混淆特性定义

namespace System{

public sealed class ObsoleteAttribute:Attribute{}

public sealed class AttributeUsageAttribute:Attribute{}

ObfuscationAttribute

http://msdn.microsoft.com/en-us/library/system.reflection.obfuscationattribute(v=vs.110).aspx

}

[Obsolete()]

[AttributeUsage()]

public class ObfuscateAttribute:Attribute{

}

Mono.Options 参数选项

类 AssemblyInfo

class AssemblyInfo

{

List<AssemblyInfo> references; // 依赖的程序集

List<AssemblyInfo> referencedBy; // 被依赖的程序集

List<TypeReference> unrenamedTypeReferences;

// ILSPY\_TEST1.Class1/<Test>d\_\_3

List<MemberReference> unrenamedReferences;

}

RenameParams 重命名参数

方法参数

类参数：泛型参数

event

Action

class ObfuscatedThing

class ObfuscatedClass:ObfuscatedThing

**类**：

方法 MethodKey

成员变量 FieldKey

属性 PropertyKey

事件 EventKey

**方法**

参数

名字

返回值

class ParamSig:IComparable<ParamSig>{

// MethodReference

// ParameterDefinition

// MethodDefinition

}

class NameParamSig:ParamSig,IComparable<NameParamSig>

class MethodKey:NameParamSig,IComparable<MethodKey>

**事件**

class EventKey{

}

依赖的类:

EventReference

EventDefinition

**字段**

class FieldKey{

// FieldDefinition

// FieldReference

}

**类型**

TypeKey 类型

class TypeKey:IComparable<TypeKey>

{

string scope; // ?

string ns;

string name;

string fullname;

}

依赖的类

Mono.Cecil.TypeReference

Mono.Cecil.TypeDefinition 是TypeReference的子类

typeReference as TypeDefinition

**注意：**as 类似强制类型转换，如果转换不成功，返回null

interface IPredicate<T> 谓词

资源：

List<ObfuscatedThing>resources

配置解析

System.Xml.XmlReaderSettings

System.Xml.XmlReader

**ConfuserEx**

1、源码地址：<https://github.com/yck1509/ConfuserEx.git>

2、使用说明

3、效果

4、实现原理

Mono.Cecil 源码行赏

概览

Mono

Mono.Cecil

字段定义

class FieldDefinition:FieldReference,

IMemberDefinition,

IConstantProvider,

IMarshalInfoProvider

从声明看，字段是类的成员，需要实现IMemberDefinition接口，具有Attribute(特性)部分

class FieldReference:MemberReference{

// TypeReference field\_type;

// 一个字段是有类型和名字的，名字存放在MemberReference

}

class MemberReference:IMetadataTokenProvider{

string name; // 名字

TypeReference declaring\_type; // 声明类型 ？

**MetadataToken token**;

}

事件定义

class EventDefinition:EventReference, IMemberDefition

class EventReference:MemberReference{

TypeReference event\_type; // 类型信息

}

属性定义

class PropertyDefinition:PropertyReference,

IMemberDefinition,

IConstantProvider{

// 特性

ushort attributes;

}

abstract class PropertyReference:MemberReference{

TypeReference property\_type;

}

**方法定义**：

class MethodDefinition:MethodReference,

IMemberDefinition,

ISecurityDeclarationProvider

{

MethodBody body;

}

class MethodReference:MemberReference,

IMemthodSignature,

IGenericParameterProvider,

IGenericContext

{

internal ParameterDefinitionCollection parameters;

MethodReturnType return\_type;

MethodCallingConvention calling\_convention;

Collection<GenericParameter> generic\_parameters;

}

类型定义：Class 结构体

模块定义

class ModuleDefinition:ModuleReference,

ICustomAttributeProvider

{

}

class ModuleReference:IMetadataScope

{

string name;

MetadataToken token;

}

Token 令牌

struct MetadataToken{

}

接口

interface IMetadataTokenProvider{

MetadataToken MetadataToken{get;set;}

}

Mono.Cecil.Cil

Mono.Cecil.Metadata

Mono.Cecil.PE

Mono.Collections.Generic

Mono.Security.Cryptography