

.NET探秘



[.NET探秘_下载链接1](#)

著者:利丁

出版者:人民邮电出版社

出版时间:2009-9

装帧:

isbn:9787115201768

《.NET探秘MSIL权威指南》是IL汇编语言的权威之作，深入地讲解了IL的全部内容，是.NET底层开发人员必备的参考书，对于从概念设计到实现和维护等软件开发所有阶段都很有价值。《.NET探秘MSIL权威指南》内容基于.NET 2.0版本，同时，考虑到.NET 3.0/3.5在CLR/IL上没有作任何改动，且即将推出的.NET

4.0也只是在CLR上稍作修改而并没有涉及IL语言，所以，《.NET探秘MSIL权威指南》对于目前各个版本的.NET Framework而言都是适用的。

《.NET探秘MSIL权威指南》适合所有.NET开发人员、讲师、研究人员阅读。

作者介绍:

Serge Lidin，曾是.NET CLR小组核心开发人员之一，是MSIL语言的设计者以及该语言规范的作者之一，负责设计并开发了IL汇编器、IL反汇编器等核心组件。目前任职于微软Phoenix小组。开发新一代编译技术底层框架。

包建强，复旦人。微软2008年MVP。现就职于HP某航空项目，从事Silverlight开发和Prism框架设计工作。

目录: 第一部分 快速入门

第1章 简单示例 2

1.1 CLR基础 2

1.2 简单示例：The Code 5

1.2.1 程序头 7

1.2.2 类声明 8

1.2.3 字段声明 9

1.2.4 方法声明 10

1.2.5 全局项 14

1.2.6 映射字段 16

1.2.7 数据声明 16

1.2.8 作为占位符的值类型 17

1.2.9 调用非托管代码 17

1.3 类的预先声明 18

1.4 小结 20

第2章 代码增强 21

2.1 精简代码 21

2.2 保护代码 23

2.3 小结 28

第3章 使代码更简单 29

3.1 别名 29

3.2 编译控制伪指令 31

3.3 关联当前类及其引用项 34

3.4 小结 35

第二部分 底层结构

第4章 托管可执行文件的结构 38

4.1 PE/COFF头 39

4.1.1 MS-DOS头/Stub和PE签名 40

4.1.2 COFF头 40

4.1.3 PE头 43

4.1.4 节头 47

4.2 CLR头 49

4.2.1 头结构 50

4.2.2 Flags字段 51

4.2.3 EntryPointToken字段 52

4.2.4 VTableFixups字段 52

4.2.5 StrongNameSignature字段 53

4.2.6	重定位节	53
4.2.7	文本节	55
4.2.8	数据节	56
4.2.9	数据常量	56
4.2.10	V表	57
4.2.11	非托管导出表	57
4.2.12	线程局部存储(TLS)	59
4.2.13	资源	60
4.2.14	非托管资源	60
4.2.15	托管资源	62
4.3	小结	63
4.3.1	第一阶段：初始化	63
4.3.2	第二阶段：源代码解析	63
4.3.3	第三阶段：映像生成	63
4.3.4	第四阶段：完成	64
第5章	元数据表的组织	65
5.1	什么是元数据	65
5.2	堆和表	67
5.2.1	堆	67
5.2.2	通用元数据头	68
5.2.3	元数据表流	70
5.3	RID和标记	73
5.3.1	RID	73
5.3.2	标记	73
5.3.3	编码标记	75
5.4	元数据有效性检查	78
5.5	小结	79
第三部分	基本组件	
第6章	模块和程序集	82
6.1	什么是程序集	82
6.2	私有程序集和共享程序集	82
6.3	作为逻辑执行单元的应用程序域	83
6.4	清单	84
6.5	Assembly元数据表和声明	86
6.6	AssemblyRef元数据表和声明	87
6.7	加载程序搜索程序集	89
6.8	Module元数据表和声明	92
6.9	ModuleRef元数据表和声明	92
6.10	File元数据表和声明	93
6.11	托管资源元数据表和声明	94
6.12	ExportedType元数据表和声明	97
6.13	ILAsm中清单声明的次序	98
6.14	单模块程序集和多模块程序集	99
6.15	元数据有效性规则概要	100
6.15.1	Assembly表有效性规则	100
6.15.2	AssemblyRef表有效性规则	100
6.15.3	Module表有效性规则	101
6.15.4	ModuleRef表有效性规则	101
6.15.5	File表有效性规则	101
6.15.6	ManifestResource表有效性规则	101
6.15.7	ExportedType表有效性规则	102
第7章	命名空间和类	103
7.1	类的元数据	104
7.1.1	TypeDef元数据表	105

- 7.1.2 TypeRef元数据表 106
- 7.1.3 InterfaceImpl元数据表 106
- 7.1.4 NestedClass元数据表 106
- 7.1.5 ClassLayout元数据表 107
- 7.2 命名空间和类的完整名称 107
 - 7.2.1 ILAsm命名约定 108
 - 7.2.2 命名空间 109
 - 7.2.3 类的完整名称 110
- 7.3 类的特性 111
 - 7.3.1 标志 111
 - 7.3.2 类的可见性和友元程序集 113
 - 7.3.3 类的引用 113
 - 7.3.4 父类型 114
 - 7.3.5 接口实现 114
 - 7.3.6 类的布局信息 115
- 7.4 接口 115
- 7.5 值类型 117
 - 7.5.1 值的装箱和拆箱 117
 - 7.5.2 值类型的实例成员 118
 - 7.5.3 值类型的派生 118
- 7.6 枚举 119
- 7.7 委托 119
- 7.8 嵌套类型 121
- 7.9 类的扩充 123
- 7.10 元数据有效性规则概要 125
 - 7.10.1 TypeDef表有效性规则 125
 - 7.10.2 特定于枚举的有效性规则 126
 - 7.10.3 TypeRef表有效性规则 126
 - 7.10.4 InterfaceImpl表有效性规则 126
 - 7.10.5 NestedClass表有效性规则 127
 - 7.10.6 ClassLayout表有效性规则 127
- 第8章 基本类型和签名 128
 - 8.1 CLR中的基本类型 128
 - 8.1.1 基本数据类型 128
 - 8.1.2 数据指针类型 129
 - 8.1.3 函数指针类型 131
 - 8.1.4 向量和数组 131
 - 8.1.5 修饰符 133
 - 8.1.6 本地类型 134
 - 8.1.7 可变类型 136
 - 8.2 签名中类的表示 138
 - 8.3 签名 139
 - 8.3.1 调用约定 139
 - 8.3.2 字段签名 139
 - 8.3.3 方法和属性签名 140
 - 8.3.4 MemberRef签名 140
 - 8.3.5 间接调用签名 141
 - 8.3.6 局部变量签名 141
 - 8.3.7 类型说明 142
 - 8.4 签名有效性规则概要 143
- 第9章 字段和数据常量 144
 - 9.1 字段元数据 144
 - 9.1.1 定义字段 145
 - 9.1.2 引用字段 146

- 9.2 实例字段和静态字段 147
- 9.3 默认值 147
- 9.4 映射字段 150
- 9.5 数据常量声明 151
- 9.6 显式布局和联合声明 152
- 9.7 全局字段 155
- 9.8 构造函数与数据常量 156
- 9.9 元数据有效性规则概要 158
 - 9.9.1 Field表有效性规则 159
 - 9.9.2 FieldLayout表有效性规则 159
 - 9.9.3 FieldRVA表有效性规则 159
 - 9.9.4 FieldMarshal表有效性规则 160
 - 9.9.5 Constant表有效性规则 160
 - 9.9.6 MemberRef表有效性规则 160
- 第10章 方法 161
 - 10.1 方法元数据 161
 - 10.1.1 Method表的记录字段 162
 - 10.1.2 方法标志 162
 - 10.1.3 方法名称 164
 - 10.1.4 方法实现标志 165
 - 10.1.5 方法参数 166
 - 10.1.6 引用方法 167
 - 10.1.7 方法实现元数据 168
 - 10.2 静态方法、实例方法和虚方法 168
 - 10.3 显式方法重写 172
 - 10.4 方法重写和可访问性 177
 - 10.5 方法头特性 178
 - 10.6 局部变量 180
 - 10.7 类的构造函数 181
 - 10.7.1 类的构造函数和beforefieldinit标志 182
 - 10.7.2 模块构造函数 184
 - 10.8 实例构造函数 184
 - 10.9 实例终结器 186
 - 10.10 可变参数列表 187
 - 10.11 方法重载 189
 - 10.12 全局方法 191
 - 10.13 元数据有效性规则概要 191
 - 10.13.1 Method表有效性规则 192
 - 10.13.2 Param表有效性规则 193
 - 10.13.3 MethodImpl表有效性规则 193
- 第11章 泛型类型 195
 - 11.1 泛型类型元数据 196
 - 11.1.1 GenericParam元数据表 198
 - 11.1.2 GenericParamConstraint元数据表 198
 - 11.1.3 TypeSpec元数据表 199
 - 11.2 约束标志 199
 - 11.3 在ILAsm中定义泛型类型 199
 - 11.4 类型参数寻址 200
 - 11.5 泛型类型实例化 201
 - 11.6 定义泛型类型：继承、实现、约束 202
 - 11.7 定义泛型类型：循环依赖 203
 - 11.8 泛型类型的成员 205
 - 11.9 嵌套泛型类型 210
 - 11.10 元数据有效性规则概要 213

- 第12章 泛型方法 214
 - 12.1 泛型方法元数据 214
 - 12.2 泛型方法签名 216
 - 12.3 在ILAsm中定义泛型方法 216
 - 12.4 调用泛型方法 217
 - 12.5 重写虚泛型方法 219
 - 12.6 元数据有效性规则概要 223
- 第四部分 深入执行引擎
- 第13章 IL指令 226
 - 13.1 长参数和短参数指令 227
 - 13.2 标号和流程控制指令 227
 - 13.2.1 无条件转移指令 228
 - 13.2.2 有条件转移指令 228
 - 13.2.3 比较转移指令 228
 - 13.2.4 switch指令 230
 - 13.2.5 break指令 230
 - 13.2.6 托管EH块退出指令 230
 - 13.2.7 托管EH块结束指令 231
 - 13.2.8 ret指令 231
 - 13.3 运算指令 231
 - 13.3.1 栈处理 231
 - 13.3.2 常量加载 232
 - 13.3.3 间接加载 233
 - 13.3.4 间接存储 233
 - 13.3.5 算术操作 234
 - 13.3.6 溢出算术操作 235
 - 13.3.7 位操作 236
 - 13.3.8 移位操作 236
 - 13.3.9 转换操作 237
 - 13.3.10 溢出转换操作 238
 - 13.3.11 逻辑条件检查指令 238
 - 13.3.12 块操作 239
 - 13.4 寻址参数和局部变量 239
 - 13.4.1 方法参数加载 239
 - 13.4.2 方法参数地址加载 240
 - 13.4.3 方法参数存储 240
 - 13.4.4 方法参数列表 240
 - 13.4.5 局部变量加载 240
 - 13.4.6 局部变量引用加载 241
 - 13.4.7 局部变量存储 241
 - 13.4.8 局部块分配 241
 - 13.4.9 前缀指令 241
 - 13.5 寻址字段 242
 - 13.6 调用方法 243
 - 13.6.1 直接调用 243
 - 13.6.2 间接调用 244
 - 13.6.3 尾部调用 245
 - 13.6.4 带约束的虚调用 246
 - 13.7 寻址类和值类型 247
 - 13.8 向量指令 250
 - 13.8.1 向量创建 250
 - 13.8.2 元素地址加载 251
 - 13.8.3 元素加载 251
 - 13.8.4 元素存储 252

- 13.9 代码可验证性 252
- 第14章 托管异常处理 255
 - 14.1 EH子句的内部表示 255
 - 14.2 EH子句的类型 256
 - 14.3 EH子句声明的标号格式 258
 - 14.4 EH子句声明的作用域格式 259
 - 14.5 处理异常 262
 - 14.6 异常类型 264
 - 14.6.1 加载程序异常 264
 - 14.6.2 JIT编译器异常 264
 - 14.6.3 执行引擎异常 265
 - 14.6.4 互操作异常 266
 - 14.6.5 子类异常 266
 - 14.6.6 非托管异常映射 266
 - 14.7 EH子句结构化规则概要 267
- 第五部分 特殊组件
- 第15章 事件和属性 270
 - 15.1 事件和委托 270
 - 15.2 事件元数据 273
 - 15.2.1 Event表 273
 - 15.2.2 EventMap表 274
 - 15.2.3 MethodSemantics表 274
 - 15.3 事件声明 275
 - 15.4 属性元数据 277
 - 15.4.1 Property表 278
 - 15.4.2 PropertyMap表 279
 - 15.5 属性声明 279
 - 15.6 元数据有效性规则概要 280
 - 15.6.1 Event表有效性规则 280
 - 15.6.2 EventMap表有效性规则 281
 - 15.6.3 Property表有效性规则 281
 - 15.6.4 PropertyMap表有效性规则 281
 - 15.6.5 MethodSemantics表有效性规则 281
- 第16章 自定义特性 283
 - 16.1 自定义特性的概念 283
 - 16.2 CustomAttribute元数据表 284
 - 16.3 自定义特性的值编码 285
 - 16.4 自定义特性的逐字描述 287
 - 16.5 自定义特性声明 288
 - 16.6 自定义特性的分类 291
 - 16.6.1 执行引擎和JIT编译器 292
 - 16.6.2 互操作子系统 293
 - 16.6.3 安全 295
 - 16.6.4 Remoting子系统 296
 - 16.6.5 Visual Studio调试器 297
 - 16.6.6 程序集链接器 297
 - 16.6.7 公共语言规范(CLS)兼容性 298
 - 16.6.8 伪自定义特性 298
 - 16.7 元数据有效性规则概要 300
- 第17章 安全特性 301
 - 17.1 声明性安全 301
 - 17.2 声明性操作 302
 - 17.3 安全许可权限 303
 - 17.3.1 访问许可权限 303

17.3.2 身份许可权限	306
17.3.3 自定义许可权限	308
17.3.4 许可权限集	309
17.4 声明性安全元数据	310
17.5 许可权限集的Blob编码	311
17.6 安全特性声明	311
17.7 元数据有效性规则概要	312
第18章 托管代码和非托管代码的互操作	314
18.1 Thunk and 包装器	315
18.1.1 P/Invoke Thunk	315
18.1.2 实现映射元数据	317
18.1.3 IJW Thunk	317
18.1.4 COM可调用包装器	318
18.1.5 运行时可调用包装器	319
18.2 数据封送	320
18.2.1 blittable类型	320
18.2.2 in/out参数	321
18.2.3 字符串封送	322
18.2.4 对象封送	323
18.2.5 更多对象封送	324
18.2.6 数组封送	325
18.2.7 委托封送	325
18.3 为非托管代码提供托管方法作为回调	326
18.4 作为非托管导出的托管方法	328
18.5 小结	334
第19章 多语言工程	336
19.1 IL反汇编器	336
19.2 双向解析的原则	340
19.3 创造性的双向解析	341
19.4 使用类的扩充	342
19.5 通过双向解析进行模块链接	342
19.6 ASMMETA: 解决循环依赖	344
19.7 内嵌在高级语言中的IL	346
19.8 在调试模式下编译	347
19.9 小结	352
第六部分 附录	
附录A ILAsm语法	356
附录B 元数据表	376
附录C IL指令集	387
附录D IL汇编器和IL反汇编器的命令行选项	394
附录E 离线验证工具	399
部分术语翻译说明	416
• • • • •	(收起)

[.NET探秘_下载链接1](#)

标签

.NET

CLR

编程

C

#NET探秘

MSIL权威指南

ILasm

计算机

评论

翻译的有些生硬

支持“包子”是本人应尽的义务。哪怕骗也要骗得像样！
大家没事儿去买本正版书吧。就算练练英语，看看翻译得如何“晕乎乎”吧。哇哈哈。
。。。 （某人还不快请我吃饭？）

可以看出作者在译作方面经验不足，有些地方意思搞错了，某些术语采用了一些偏门的译法、语句有些生涩。

[.NET探秘_下载链接1](#)

书评

Expert .NET 2.0 IL Assembler drills down into the internal structures and operations of the .NET common language runtime and teaches you how to master the IL assembly language, ILAsm. (The rule of thumb is that if the runtime can do it, ILAsm must be able to...

翻译的不好，有的地方言语不达，有的地方句子还保持英文的句式。就不举例了。感觉有点赶任务的做法，比较匆忙，没有精雕细琢。

经常博客园上 看译者的文章。从不怀疑译者的技术水平。但是翻译的确实错误不少。别的错误地方就记不得了。正在看第八章签名： 下面据三个明显的错误地方
141页，IMAGE_CEE_UNMANAGED_CALLCONV_C (0x1). C/C++-style calling convention.
The call stack is cleaned up by the ...

[.NET探秘 下载链接1](#)