



自己动手写 Java虚拟机

张秀宏 著



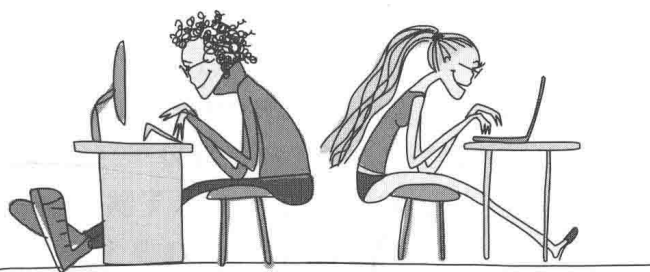
Write Your Own Java Virtual Machine



机械工业出版社
China Machine Press

自己动手写 Java虚拟机

张秀宏 著



Write Your Own Java Virtual Machine



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

自己动手写 Java 虚拟机 / 张秀宏著. —北京: 机械工业出版社, 2016.4
(Java 核心技术系列)

ISBN 978-7-111-53413-6

I. 自… II. 张… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2016) 第 066610 号

自己动手写 Java 虚拟机

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 李 艺

责任校对: 殷 虹

印 刷: 三河市宏图印务有限公司

版 次: 2016 年 5 月第 1 版第 1 次印刷

开 本: 186mm × 240mm 1/16

印 张: 17.5

书 号: ISBN 978-7-111-53413-6

定 价: 69.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

HZBOOKS | 华章科技 | Science & Technology



为什么编写本书

Java 语言于 1995 年首次公开发布，很快便取得了巨大的成功，成为使用最为广泛的编程语言之一。到现在，Java 已经经历了 20 多个年头。在这期间，无论是 Java 语言本身还是 Java 虚拟机技术，都取得了长足的进步。现如今，Java 依然长期占据 TIOBE^①网站的编程语言排行榜首。最近更是被 TIOBE 选为 2015 年度编程语言^②，风采可谓不减当年。

众所周知，Java 早已不仅仅是一个单纯的语言，而是一个开放的平台。活跃在这个平台之上的编程语言除了 Java 之外，还有 Groovy^③、Scala^④、Clojure^⑤、Jython^⑥和 JRuby^⑦等。Java 虚拟机则是支持这个平台的基石。

市面上教授 Java 语言的书籍种类繁多，相比之下，介绍 Java 虚拟机的书籍却是凤毛麟角。这足以说明 Java 作为一门高级语言是多么成功（让程序员远离底层），但并不代表 Java 虚拟机技术不重要。恰恰相反，当 Java 语言掌握到一定程度时，Java 虚拟机原理自然就会成为必须越过的一道鸿沟。

近几年，国内涌现出了一些讨论 Java 虚拟机技术的优秀书籍，这些书籍主要以分析 OpenJDK 或 Oracle JDK 为主。本书另辟蹊径，带领读者自己动手从零开始用 Go 语言编写 Java 虚拟机。这样做好处颇多，弥补了 OpenJDK 等虚拟机的不足。

首先，OpenJDK 等虚拟机实现非常复杂。对于初学者而言，很容易陷入代码的海洋

① <http://www.tiobe.com/>。

② Java 曾被 TIOBE 选为 2005 年度编程语言。

③ <http://www.groovy-lang.org/>。

④ <http://www.scala-lang.org/>。

⑤ <http://clojure.org/>。

⑥ <http://www.jython.org/>。

⑦ <http://jruby.org/>。

和不必要的细节之中。其次，OpenJDK 等虚拟机大多用 C++ 语言编写。C++ 语言非常复杂，理解起来难度很大。最后，单纯阅读代码比较乏味，缺少乐趣，而脱离代码又很难透彻讨论技术。通过自己动手编写代码，很好地避免了上述问题。看着自己实现的 Java 虚拟机功能逐渐增强，看到可以运行的 Java 程序越来越复杂，成就感非常强。总之，通过实践的方式，相信读者可以更深刻地领悟 Java 虚拟机的工作原理。

Go 是 Google 公司于 2012 年推出的系统编程语言。从到硬件的距离来看，Go 语言介于 C 和 Java 之间。Go 的语法和 C 类似，但更加简洁，因此很容易学习。Go 语言内置了丰富的基本数据类型，并且支持结构体，所以很适合用来实现 Java 虚拟机。Go 支持指针，但并不支持指针运算，因此用 Go 编写的代码要比 C 代码更加安全。此外，Go 还支持垃圾回收和接口等 Java 类语言中才有的功能，大大降低了实现 Java 虚拟机的难度。

以上是本书采用 Go 语言编写 Java 虚拟机的原因，希望读者在学习本书的过程中，可以喜欢上 Go 这门还很年轻的语言。

本书主要内容

全书一共分为 11 章，各章内容安排如下：

第 1 章：安装开发环境，讨论 java 命令，并编写一个类似 Java 的命令程序。

第 2 章：讨论 Java 虚拟机如何搜索 class 文件，实现类路径。

第 3 章：讨论 class 文件结构，实现 class 文件解析。

第 4 章：讨论运行时数据区，实现线程私有的运行时数据区，包括线程、Java 虚拟机栈、栈帧、操作数栈和局部变量表等。

第 5 章：讨论 Java 虚拟机指令集和解释器，实现解释器和 150 余条指令。

第 6 章：讨论类、对象以及线程共享的运行时数据区，实现类加载器、方法区以及部分引用类指令。

第 7 章：讨论方法调用和返回，实现方法调用和返回指令。

第 8 章：讨论数组和字符串，实现数组相关指令和字符串池。

第 9 章：讨论本地方法调用，实现 Object.hashCode() 等本地方法。

第 10 章：讨论异常处理机制，实现 athrow 指令。

第 11 章：讨论 System 类的初始化过程和 System.out.println() 的工作原理等，并对全书进行总结。

本书面向读者

本书主要面向有一定经验的 Java 程序员，但任何对 Java 虚拟机工作原理感兴趣的读者都可以从本书获益。如前所述，本书将使用 Go 语言实现 Java 虚拟机。书中会简要介绍 Go 语言的部分语法以及与 Java 语言的区别，但不会深入讨论。由于 Go 语言相对比较简单，相信任何有 C 系列语言（如 C、C++、C#、Objective-C、Java 等）经验的读者都可以轻松读懂书中的源代码。

如何阅读本书

本书代码经过精心调整，每一章（第 1 章除外）都建立在前一章的基础上，但每一章又都可以单独编译和运行。本书内容主要围绕代码对 Java 虚拟机展开讨论。读者可以从第 1 章开始，按顺序阅读本书并运行每一章的源代码，也可以直接跳到感兴趣的章节阅读，必要时再阅读其他章节。

参考资料

本书主要参考了下面这些资料：

- 《Java 虚拟机规范》第 8 版
- 《Java 语言规范》第 8 版
- 《深入 Java 虚拟机》(原书第 2 版)[⊖]

其中《Java 虚拟机规范》主要参考了第 8 版，但同时也参考了第 7 版和更老的版本。《Java 语言规范》则主要参考了第 8 版。读者可以从 <http://docs.oracle.com/javase/specs/index.html> 获取各个版本的《Java 虚拟机规范》和《Java 语言规范》。

笔者早在十年前还在上学时就读过由 Bill Venners 著，曹晓钢等翻译的《深入 Java 虚拟机（原书第 2 版）》。但是由于当时水平有限，理解得并不是很深入。时隔十年，重读此书还是颇有收获。较之《Java 虚拟机规范》的严谨和刻板，该书更加通俗易懂。原书作者已经将部分章节放于网上，网址是 <http://www.artima.com/insidejvm/ed2/>，读者可以免费阅读。

以上是 Java 方面的资料。Go 语言方面主要参考了 Go 官网上的各种资料，包括《如

[⊖] 原书名为《Inside the Java Virtual Machine, Second Edition》。

何编写 Go 程序》^①《Effective Go》^②《Go 语言规范》^③以及 Go 标准库文档^④等。另外，在本书的写作过程中，笔者还通过搜索引擎查阅了遍布于网络上（特别是 StackOverflow^⑤和 Wikipedia^⑥）的各种资料，这里就不一一罗列了。

下载本书源代码

本书源代码可以从 <https://github.com/zxh0/jvmgo-book> 获取。代码分为 Go 和 Java 两部分，目录结构如下：

```
https://github.com/zxh0/jvmgo-book/v1/code/
|-go
  |-src
    |-jvmgo
|-java
  |-example
```

Go 语言部分是 Java 虚拟机代码，每章为一个子目录，可以独立编译和运行。Java 语言部分是 Java 示例代码，每章为一个包。Java 代码按照 Gradle^⑦工程标准目录结构组织，可以用 Gradle 编译整个工程，也可以用 javac 分别编译每个文件。

勘误和支持

《Java 虚拟机规范》对 Java 虚拟机的工作机制有十分严谨的描述。但是由于笔者水平和表达能力有限，本书一定存在表述不精确、不准确，甚至不正确的地方。另外，由于时间有限，书中也难免会有一些疏漏之处，还请读者谅解。

本书的勘误将通过 <https://github.com/zxh0/jvmgo-book/blob/master/v1/errata.md> 发布和更新。如果读者发现书中的错误、有改进意见，或者有任何问题需要讨论，都可以在本书的 Github 项目上创建 Issue。此外也可以加入 QQ 群（470333113）与读者交流。

① <https://golang.org/doc/code.html>。

② https://golang.org/doc/effective_go.html。

③ <https://golang.org/ref/spec>。

④ <https://golang.org/pkg/>。

⑤ <http://stackoverflow.com/>。

⑥ <https://en.wikipedia.org/>。

⑦ <http://gradle.org/>。

致谢

首先要感谢我的家人和朋友，没有你们的鼓励、支持和帮助，本书不可能面世。这里特别感谢我的妻子，在我陷入低谷的时候，叮嘱我继续努力不要放弃。还有我的朋友范森，每章开头的可爱鼯鼠就是出自他手，希望这些鼯鼠能给枯燥的文字增添一些色彩。

其次感谢我所在的公司乐元素[⊖]，它为我提供了舒适和愉悦的工作环境，使我在工作之余可以全心投入本书的写作之中。

代码被我放到了 Github 上，地址是 <https://github.com/zxh0/jvm.go>。不过由于能力和时间有限，这个虚拟机离完整实现《Java 虚拟机规范》还相距甚远。2015 年 4 月份，我停止了 jvm.go 的编写，同时开始改造代码，酝酿本书。感谢所有关注过 jvm.go 项目的人，没有你们的帮助就没有 jvm.go，也就没有本书。

最后，感谢机械工业出版社华章分社的编辑，本书能够顺利出版离不开他们的敬业精神和一丝不苟的工作态度。

⊖ <http://www.happyelements.cn/>。

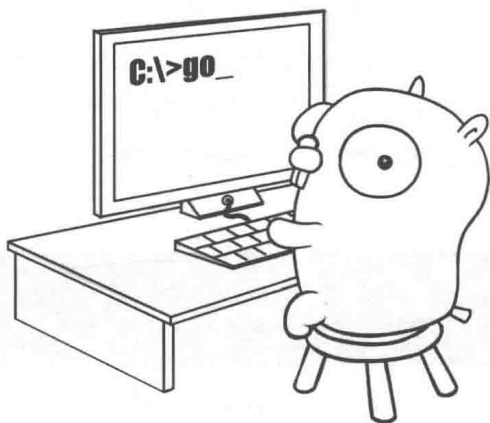
目 录 Contents

前言	2.4 测试本章代码	20
	2.5 本章小结	21
第1章 命令行工具	第3章 解析 class 文件	23
1.1 准备工作	3.1 class 文件	24
1.1.1 安装 JDK	3.2 解析 class 文件	25
1.1.2 安装 Go	3.2.1 读取数据	26
1.1.3 创建目录结构	3.2.2 整体结构	27
1.2 java 命令	3.2.3 魔数	30
1.3 编写命令行工具	3.2.4 版本号	31
1.4 测试本章代码	3.2.5 类访问标志	32
1.5 本章小结	3.2.6 类和超类索引	32
	3.2.7 接口索引表	33
第2章 搜索 class 文件	3.2.8 字段和方法表	33
2.1 类路径	3.3 解析常量池	35
2.2 准备工作	3.3.1 ConstantPool 结构体	35
2.3 实现类路径	3.3.2 ConstantInfo 接口	37
2.3.1 Entry 接口	3.3.3 CONSTANT_Integer_info	39
2.3.2 DirEntry	3.3.4 CONSTANT_Float_info	40
2.3.3 ZipEntry	3.3.5 CONSTANT_Long_info	40
2.3.4 CompositeEntry	3.3.6 CONSTANT_Double_info	41
2.3.5 WildcardEntry	3.3.7 CONSTANT_Utf8_info	42
2.3.6 Classpath	3.3.8 CONSTANT_String_info	43

3.3.9	CONSTANT_Class_info	45	4.4	测试本章代码	81
3.3.10	CONSTANT_NameAnd- Type_info	46	4.5	本章小结	83
3.3.11	CONSTANT_Fieldref_info、 CONSTANT_Methodref_info 和 CONSTANT_Interface- Methodref_info	47	第 5 章 指令集和解释器 85		
3.3.12	常量池小结	49	5.1	字节码和指令集	86
3.4	解析属性表	50	5.2	指令和指令解码	88
3.4.1	AttributeInfo 接口	50	5.2.1	Instruction 接口	89
3.4.2	Deprecated 和 Synthetic 属性	53	5.2.2	BytecodeReader	91
3.4.3	SourceFile 属性	54	5.3	常量指令	92
3.4.4	ConstantValue 属性	55	5.3.1	nop 指令	92
3.4.5	Code 属性	56	5.3.2	const 系列指令	93
3.4.6	Exceptions 属性	58	5.3.3	bipush 和 sipush 指令	94
3.4.7	LineNumberTable 和 LocalVariableTable 属性	59	5.4	加载指令	94
3.5	测试本章代码	61	5.5	存储指令	95
3.6	本章小结	63	5.6	栈指令	96
第 4 章 运行时数据区 65			5.6.1	pop 和 pop2 指令	96
4.1	运行时数据区概述	66	5.6.2	dup 指令	97
4.2	数据类型	67	5.6.3	swap 指令	98
4.3	实现运行时数据区	68	5.7	数学指令	98
4.3.1	线程	68	5.7.1	算术指令	98
4.3.2	Java 虚拟机栈	69	5.7.2	位移指令	99
4.3.3	帧	71	5.7.3	布尔运算指令	101
4.3.4	局部变量表	72	5.7.4	iinc 指令	102
4.3.5	操作数栈	74	5.8	类型转换指令	102
4.3.6	局部变量表和操作数栈 实例分析	76	5.9	比较指令	103
			5.9.1	lcmp 指令	103
			5.9.2	fcmp<op> 和 dcmp<op> 指令	104
			5.9.3	if<cond> 指令	105
			5.9.4	if_icmp<cond> 指令	106
			5.9.5	if_acmp<cond> 指令	107

5.10	控制指令	108	6.6	类和对象相关指令	144
5.10.1	goto 指令	108	6.6.1	new 指令	144
5.10.2	tableswitch 指令	108	6.6.2	putstatic 和 getstatic 指令	146
5.10.3	lookupswitch 指令	110	6.6.3	putfield 和 getfield 指令	148
5.11	扩展指令	111	6.6.4	instanceof 和 checkcast 指令	150
5.11.1	wide 指令	111	6.6.5	ldc 指令	154
5.11.2	ifnull 和 ifnonnull 指令	113	6.7	测试本章代码	156
5.11.3	goto_w 指令	113	6.8	本章小结	160
5.12	解释器	114			
5.13	测试本章代码	118	第 7 章	方法调用和返回	161
5.14	本章小结	120	7.1	方法调用概述	161
第 6 章	类和对象	121	7.2	解析方法符号引用	163
6.1	方法区	122	7.2.1	非接口方法符号引用	163
6.1.1	类信息	122	7.2.2	接口方法符号引用	165
6.1.2	字段信息	124	7.3	方法调用和参数传递	166
6.1.3	方法信息	125	7.4	返回指令	169
6.1.4	其他信息	127	7.5	方法调用指令	170
6.2	运行时常量池	127	7.5.1	invokestatic 指令	170
6.2.1	类符号引用	129	7.5.2	invokespecial 指令	170
6.2.2	字段符号引用	130	7.5.3	invokevirtual 指令	172
6.2.3	方法符号引用	132	7.5.4	invokeinterface 指令	174
6.2.4	接口方法符号引用	132	7.6	改进解释器	176
6.3	类加载器	133	7.7	测试方法调用	178
6.3.1	readClass()	134	7.8	类初始化	181
6.3.2	defineClass()	135	7.9	本章小结	185
6.3.3	link()	136	第 8 章	数组和字符串	187
6.4	对象、实例变量和类变量	136	8.1	数组概述	187
6.5	类和字段符号引用解析	141	8.2	数组实现	188
6.5.1	类符号引用解析	141	8.2.1	数组对象	188
6.5.2	字段符号引用解析	142	8.2.2	数组类	190

8.2.3	加载数组类	191	9.4.2	System.arraycopy() 方法	227
8.3	数组相关指令	191	9.4.3	Float.floatToRawIntBits() 和 Double.doubleToRawLongBits() 方法	229
8.3.1	newarray 指令	192	9.4.4	String.intern() 方法	229
8.3.2	anewarray 指令	194	9.4.5	测试本节代码	230
8.3.3	arraylength 指令	195	9.5	Object.hashCode()、equals() 和 toString()	231
8.3.4	<I>aload 指令	196	9.6	Object.clone()	233
8.3.5	<I>astore 指令	197	9.7	自动装箱和拆箱	235
8.3.6	multianewarray 指令	198	9.8	本章小结	238
8.3.7	完善 instanceof 和 checkcast 指令	201	第 10 章 异常处理		239
8.4	测试数组	203	10.1	异常处理概述	239
8.5	字符串	204	10.2	异常抛出	240
8.5.1	字符串池	205	10.3	异常处理表	241
8.5.2	完善 ldc 指令	206	10.4	实现 athrow 指令	245
8.5.3	完善类加载器	207	10.5	Java 虚拟机栈信息	248
8.6	测试字符串	207	10.6	测试本章代码	251
8.7	本章小结	210	10.7	本章小结	252
第 9 章 本地方法调用		211	第 11 章 结束		253
9.1	注册和查找本地方法	212	11.1	System 类是如何被 初始化的	253
9.2	调用本地方法	213	11.2	初始化 System 类	255
9.3	反射	215	11.3	System.out.println() 是如何 工作的	258
9.3.1	类和对象之间的关系	215	11.4	测试本章代码	260
9.3.2	修改类加载器	217	11.5	总结	260
9.3.3	基本类型的类	219	附录 指令表		263
9.3.4	修改 ldc 指令	220			
9.3.5	通过反射获取类名	221			
9.3.6	测试本节代码	224			
9.4	字符串拼接和 String.intern() 方法	225			
9.4.1	Java 类库	225			



第 1 章 命令行工具

Java 虚拟机非常复杂，要想真正理解它的工作原理，最好的方式就是自己动手写一个。本书的目的就是带领读者按照 Java 虚拟机规范^①，从零开始，一步一步用 Go 语言实现一个功能逐步增强的 Java 虚拟机。第 1 章将编写一个类似 java^② 的命令行工具，用它来启动我们自己的虚拟机。在开始编写代码之前，需要先准备好开发环境。

本书假定读者使用的是 Windows 操作系统，因此书中出现的命令和路径等都是 Windows 形式的。如果读者使用的是其他操作系统（如 Mac OS X、Linux 等），需要根据自己情况做出相应调整。由于 Go 和 Java 都是跨平台语言，所以本书代码在常见的操作系统中都可以正常编译和运行。

1.1 准备工作

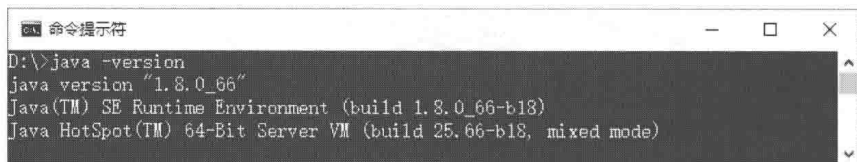
1.1.1 安装 JDK

我们都知道，要想运行 Java 程序，只有 Java 虚拟机是不够的，还需要有 Java 类库。

① 如无特殊说明，本书中出现的“Java 虚拟机规范”均指《Java 虚拟机规范第 8 版》，网址为 <http://docs.oracle.com/javase/specs/jvms/se8/html/index.html>。

② 后文中，首字母小写的 java 特指 java 命令行工具。

Java 虚拟机和 Java 类库一起，构成了 Java 运行时环境。本书编写的 Java 虚拟机依赖于 JDK 类库，另外，编译本书中的 Java 示例代码也需要 JDK。从 Oracle 网站^①上下载最新版本（写作本章时是 8u66）的 JDK 安装文件，双击运行即可。安装完毕之后，打开命令行窗口执行 `java -version` 命令，如果看到类似图 1-1 所示的输出，就证明安装成功了。



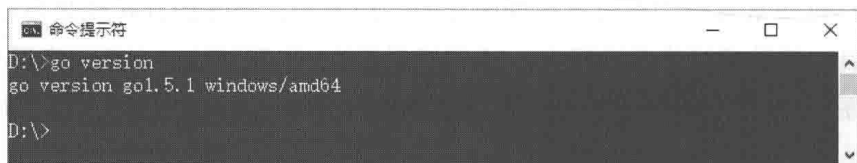
```

命令提示符
D:\>java -version
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)
  
```

图 1-1 java -version 命令输出

1.1.2 安装 Go

从 Go 语言官网^②下载最新版本（写作本章时是 1.5.1）的 Go 安装文件，双击运行即可。安装完毕之后，打开命令行窗口执行 `go version` 命令，如果看到类似图 1-2 所示的输出，就证明安装成功了。



```

命令提示符
D:\>go version
go version go1.5.1 windows/amd64

D:\>
  
```

图 1-2 go version 命令输出

`go`^③命令是 Go 语言提供的命令行工具，用来管理 Go 源代码。`go` 命令就像瑞士军刀，里面包含了各种小工具。用 Go 语言编写程序，基本上只需要 `go` 命令就可以了。`go` 命令里的小工具是各种子命令，`version` 是其中之一。其他常用的子命令包括 `help`、`fmt`、`install` 和 `test` 等。

`go` 命令行工具希望所有的 Go 源代码被都放在一个工作空间中。所谓工作空间，实际上就是一个目录结构，这个目录结构包含三个子目录。

- `src` 目录中是 Go 语言源代码。
- `pkg` 目录中是编译好的包对象文件。

① <http://www.oracle.com/technetwork/java/javase/downloads/index.html>。

② <https://golang.org/dl/>（如果 Go 官网无法访问，可以从 <http://golangtc.com/download>）下载。

③ 后文中，首字母小写的 `go` 特指 `go` 命令行工具。

□ bin 目录中是链接好的可执行文件。

实际上只有 src 目录是必须要有的，go 会自动创建 pkg 和 bin 目录。工作空间可以位于任何地方，本书使用 D:\go\workspace 作为工作空间。那么 go 如何知道工作空间在哪里呢？答案是通过 GOPATH 环境变量。在桌面上右键单击“我的电脑”图标，在弹出的菜单中单击“属性”，然后单击“高级系统设置”；在“系统属性”对话框中单击“环境变量”按钮，然后添加 GOPATH 变量即可，如图 1-3 所示。

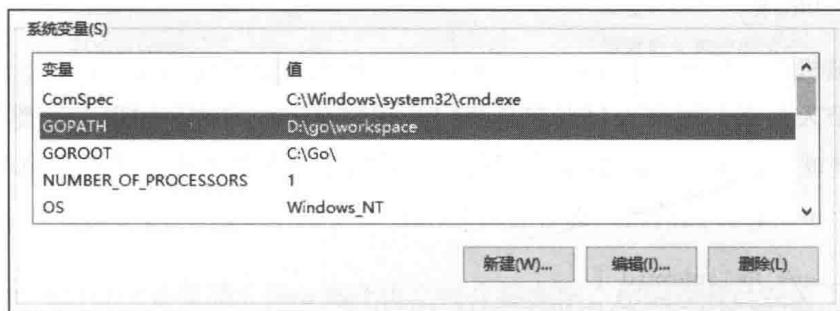


图 1-3 设置 GOPATH 环境变量

打开命令行窗口，执行 go env 命令，如果看到类似图 1-4 所示的输出，GOPATH 环境变量就设置成功了。

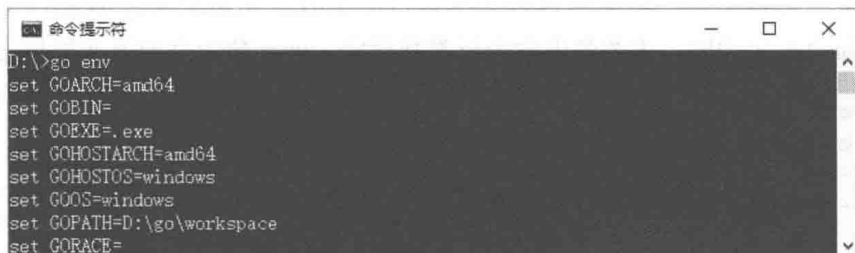


图 1-4 使用 go env 命令查看 GOPATH 环境变量

1.1.3 创建目录结构

Go 语言以包为单位组织源代码，包可以嵌套，形成层次关系。本书编写的 Go 源文件全部放在 jvmgo 包中，其中每一章的源文件又分别放在自己的子包中。包层次和目录结构有一个简单的对应关系，比如，第 1 章的代码在 jvmgo\ch01 目录下。除第 1 章以外，每一章都是先复制前一章代码，然后进行修改和完善。每一章的代码都是独立的，可以单独编译为一个可执行文件。下面创建第 1 章的目录结构。

在 D:\go\workspace\src (也就是 %GOPATH%\src) 目录下创建 jvmgo 目录, 在 jvmgo 目录下创建 ch01 目录。现在, 工作空间的目录结构如下:

```
D:\go\workspace\src
  |-jvmgo
    |-ch01
```

1.2 java 命令

Java 虚拟机的工作是运行 Java 应用程序。和其他类型的应用程序一样, Java 应用程序也需要一个入口点, 这个入口点就是我们熟知的 main() 方法。如果一个类包含 main() 方法, 这个类就可以用来启动 Java 应用程序, 我们把这个类叫作主类。最简单的 Java 程序是只有一个 main() 方法的类, 如著名的 HelloWorld 程序。

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

那么 Java 虚拟机如何知道我们要从哪个类启动应用程序呢? 对此, Java 虚拟机规范没有明确规定。也就是说, 是由虚拟机实现自行决定的。比如 Oracle 的 Java 虚拟机实现是通过 java 命令来启动的, 主类名由命令行参数指定。java 命令有如下 4 种形式:

```
java [-options] class [args]
java [-options] -jar jarfile [args]
javaw [-options] class [args]
javaw [-options] -jar jarfile [args]
```

可以向 java 命令传递三组参数: 选项、主类名 (或者 JAR 文件名) 和 main() 方法参数。选项由减号 (-) 开头。通常, 第一个非选项参数给出主类的完全限定名 (fully qualified class name)。但是如果用户提供了 -jar 选项, 则第一个非选项参数表示 JAR 文件名, java 命令必须从这个 JAR 文件中寻找主类。javaw 命令和 java 命令几乎一样, 唯一的差别在于, javaw 命令不显示命令行窗口, 因此特别适合用于启动 GUI (图形用户界面) 应用程序。

选项可以分为两类: 标准选项和非标准选项。标准选项比较稳定, 不会轻易变动。非标准选项以 -X 开头, 很有可能会在未来的版本中变化。非标准选项中有一部分是高级选