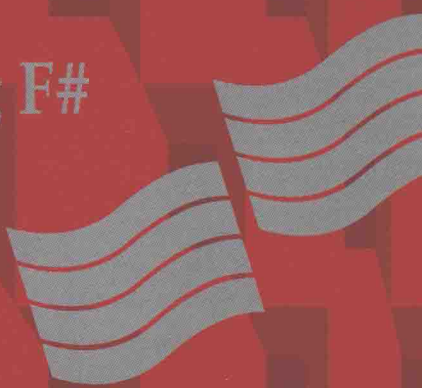
信毅教材大系

函数式F#语言程序设计

• 黎升洪 主编

Functional Programming F#



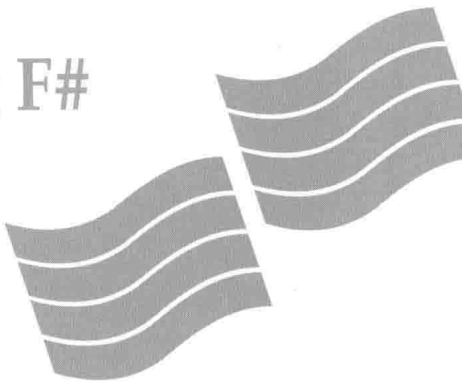
復旦大學出版社

信毅教材大系

函数式F#语言程序设计

● 黎升洪 主编

Functional Programming F#



復旦大學出版社

图书在版编目(CIP)数据

函数式 F#语言程序设计/黎升洪主编. —上海:复旦大学出版社,2014.10
(信毅教材大系)
ISBN 978-7-309-10738-8

Ⅰ. 函… Ⅱ. 黎… Ⅲ. 程序语言-程序设计-高等学校-教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2014)第 119536 号

函数式 F#语言程序设计
黎升洪 主编
责任编辑/张志军

复旦大学出版社有限公司出版发行
上海市国权路 579 号 邮编:200433
网址:fupnet@fudanpress.com http://www.fudanpress.com
门市零售:86-21-65642857 团体订购:86-21-65118853
外埠邮购:86-21-65109143
杭州日报报业集团盛元印务有限公司

开本 787×1092 1/16 印张 23.75 字数 507 千
2014 年 10 月第 1 版第 1 次印刷

ISBN 978-7-309-10738-8/T·517
定价:58.00 元

内容提要

F#是.NET框架下的通用函数式语言，当前IT界用来开发实际应用的函数式语言之一。本书从类型理论出发，通过函数特征阐述类型变换规则需要遵守的约束，着重描述了函数特征、部分应用、惰性赋值、模式匹配、测量单位、尾递归、连续传递风格和单子等函数式语言特有的语言元素工作原理。介绍了F#语言在排序算法和编译器构造方面的应用。

本书适合计算机相关专业本科生或研究生使用，也可供熟悉面向对象强制式编程的工程人员学习函数式编程使用。

总 序

世界高等教育的起源可以追溯到1088年意大利建立的博洛尼亚大学，它运用社会化组织成批量培养社会所需要的人才，改变了知识、技能主要在师徒间、个体间传授的教育方式，满足了大家获取知识的需要，史称“博洛尼亚传统”。

19世纪初期，德国的教育家洪堡提出“教学与研究相统一”和“学术自由”的原则，并指出大学的主要职能是追求真理，学术研究在大学应当具有第一位的重要性，即“洪堡理念”，强调大学对学术研究人才的培养。

在洪堡理念广为传播和接受之际，德国都柏林天主教大学校长纽曼发表了“大学的理想”的著名演说，旗帜鲜明地指出“从本质上讲，大学是教育的场所”，“我们不能借口履行大学的使命职责，而把它引向不属于它本身的目标。”强调培养人才是大学的唯一职能。纽曼关于“大学的理想”的演说让人们重新审视和思考大学为何而设、为谁而设的问题。

19世纪后期到20世纪初，美国威斯康辛大学查尔斯·范海斯校长提出“大学必须为社会发展服务”的办学理念，更加关注大学与社会需求的结合，从而使大学走出了象牙塔。

2011年4月24日，胡锦涛总书记在清华大学百年校庆庆典上，指出高等教育是优秀文化传承的重要载体和思想文化创新的重要源泉，强调要充分发挥大学文化育人和文化传承创新的职能。

总而言之，随着社会的进步与变革，高等教育不断发展，大学的功能不断扩展，但始终都在围绕着人才培养这一大学的根本使命，致力于不断提高人才培养的质量和水平。

对大学而言，优秀人才的培养，离不开一些必要的物质条件保障，但更重要的是高效的执行体系。高效的执行体系应该体现在三个方面：一是科学合理的学科专业结构，二是能洞悉学科前沿的优秀的师资队伍，三是作为知识载体和传播媒介的优秀教材。教材是体现教学内容与教学方法的知识载体，是进行教学的基本工具，也

是深化教育教学改革，提高人才培养质量的重要保证。

一本好的教材，要能反映该学科领域的学术水平和科研成就，能引导学生沿着正确的学术方向步入所向往的科学殿堂。因此，加强高校教材建设，对于提高教育质量、稳定教学秩序、实现高等教育人才培养目标起着重要的作用。正是基于这样的考虑，江西财经大学与复旦大学出版社达成共识，准备通过编写出版一套高质量的教材系列，以期进一步锻炼学校教师队伍，提高教师素质和教学水平，最终将学校的学科、师资等优势转化为人才培养优势，提升人才培养质量。为凸显江财特色，我们取校训“信敏廉毅”中一前一尾两个字，将这个系列的教材命名为“信毅教材大系”。

“信毅教材大系”将分期分批出版问世，江西财经大学教师将积极参与这一具有重大意义的学术事业，精益求精地不断提高写作质量，力争将“信毅教材大系”打造成业内有影响力的高端品牌。“信毅教材大系”的出版，得到了复旦大学出版社的大力支持，没有他们的卓越视野和精心组织，就不可能有这套系列教材的问世。作为“信毅教材大系”的合作方和复旦大学出版社的一位多年的合作者，对他们的敬业精神和远见卓识，我感到由衷的钦佩。

王　乔

2012 年 9 月 19 日

前言

程序设计语言的核心功能是实现计算。程序设计语言主要依附两种计算理论模型，一种是以图灵机为理论基础的强制式语言，如C#、C++、Java等。另一种是以λ演算为理论基础的函数式语言，如ML、Ocaml等。λ演算由组合逻辑发展而来，它是函数的形式化语言。强制式语言通过状态迁移实现计算，函数式语言则通过数据定义及在数据定义上的变换实现计算。与面向对象的强制式编程语言相比，函数式编程语言具有以下特点：(1)不仅仅是类的定义，而是函数与类型；(2)不仅仅是状态迁移，而是纯函数式；(3)不仅仅是继承，而是复合；(4)不仅仅是方法的快速处理，而是高阶函数。

函数式编程语言编程使用最简洁的函数形式，与复杂的状态变量相比，其程序更易理解。早期的函数式语言主要在学术研究领域。随着根植于.NET的函数式语言F#与根植于Java SDK的函数式语言Scala推出，函数式语言在业界获得广泛应用。

与强制式语言C#相比，函数式语言F#具有简明、经济、灵活、高效、描述能力强等特点。它同时具有函数式编程模式、强制式编程模式和面向对象编程模式。结合.NET框架提供的强大功能，F#语言在云计算、移动计算、领域语言处理(DSL)、财务计算、Web开发等方面得到广泛应用。

F#的函数式编程模式主要是定义和应用函数，即首先定义函数，再通过因变量值代入函数过程实现求值。这种编程模式完全不同于强制式编程模式，它引入了如函数特征、部分应用、模式匹配、可区分联合、测量单位、延迟赋值、高阶函数值和计算表达式等概念。

本书依据程序设计语言的类型理论编排。主要包括：(1)F#基元类型，(2)F#内置的具有可枚举值的复合类型，(3)F#内置的将数据与方法封装的类类型，(4)用户自定义的复合类型。

本书有下列特点：

(1) 编写角度独特。本书主要依据类型理论编写。同时将度量单位等也视为类型的约束。类型理论引入到程序语言的好处表现为:更少错误,更精简代码,更易于理解和调用,更易于代码重构。

(2) 内容涵盖广。除包含函数式语言的函数值内容、排序算法应用外,还包括委托、反射、泛型等.NET 框架高级应用内容。此外还包括计算表达式、连续传递风格和单子等内容。

(3) 例子丰富。不仅注重描述 F# 的语法说明,更使用大量的例子介绍了 F# 库和.NET 库函数的使用。

本书所使用的软件环境包括 F#、FSharpPowerPack 和 Visual Studio。所有例子均在 Visual Studio 2012(Update 4)中调试通过。

本书适合计算机相关专业本科生或研究生使用,也可供熟悉面向对象强制式编程的工程人员学习函数式语言编程使用。

本书编写过程中得到许多人的无私帮助。特别感谢 Tomas Petricek 对 F# 语言高级特性给予的指导和帮助;感谢江西财经大学信息管理学院徐升华教授、万常选教授、方志军教授的宝贵意见和建议;感谢上海大学缪淮扣教授的帮助。没有他们的无私帮助本书不可能完成,作者在此表示衷心的感谢!

由于作者水平有限,存在许多不足之处,恳请同行专家和广大读者批评指正。

作者邮箱:lee.shenghong@gmail.com

黎升洪

2013 年 12 月

目 录

第1章　F#语言特性与.NET框架基础知识

学习目标

了解F#的特性
了解引用透明性和高阶函数的概念
熟悉.NET框架工作原理
了解.NET框架类型的功能
熟悉值类型和引用类型
了解装箱和拆箱的原理
掌握类型成员概念，熟悉类型成员的常见构成
了解命名空间的点标记引用方式

1.1　F#语言特性与发展历史

1. F#语言特性

计算核心的功能是依据问题的输入求解问题的输出。任何一门程序设计语言的任务就是完成这种计算。程序设计语言设计需要涉及的知识包括数学、软件工程和语言学的语法、语义等许多基础知识。目前为业界广为使用的编程语言有两种。一种是以C、Java和C#等为代表的强制式语言(也称为命令式语言)；另一种是以ML、Haskell为代表的纯函数式编程语言。

函数式编程语言有良好的理论基础，其理论建立在λ演算和指称语义基础上。函数式编程将计算视为数学函数的求值过程，即函数的定义、函数应用与函数复合等操作。这些操作避免了强制式编程语言中通常会出现的状态及可修改数据问题。因为可修改数据有可能带来副作用。强制式编程语言强调状态改变，而函数式编程语言则强调函数定义与函数应用。函数式编程的方法就是数据定义和在数据定义上的变换。

函数式编程语言引入了许多新的特性，例如，部分应用(也称为柯里化)、延迟赋值、连续传递风格和高阶函数(first-class function)等概念。在纯数学领域，高阶函数的概念是指可以将一个函数作为另一个函数的参数使用，函数返回的结果也可以是函数。在计算机科学领域，高阶函数是指编程语言不区别对待值和函数，即凡可以出现值的地方均可出现高阶函数。高阶函数的引入直接导致术语“函数值”的出现，它既可以是值，也可以是函数。

函数式语言的主要编程方式是数据定义和数据变换，在实现上是函数定义与函数应用。函数式编程的优点表现为：

(1) 和强制式语言相比，同样的功能函数式语言编写更简短、简洁。

(2) 可读性和可维护性好。这是由于函数不依靠外部状态仅靠函数参数即可完成求值。

(3) 易于重复迭代开发。由于函数的封装特性，使得代码易于重构，设计的变化同样易于实现。

(4) 易于测试和调试。由于纯函数更易于隔离测试，可以方便设计有效测试案例和无效测试案例并完成测试。

强制式编程语言与函数式编程语言特性比较见表 1-1。

表 1-1 强制式编程语言与函数式编程语言比较

	强制式编程语言	函数式编程语言
编程人员关注重点	如何执行任务(算法)和跟踪状态改变	期望什么信息和需要什么转换规则
状态改变	重要	不存在。重点是函数值
执行顺序	重要	不太重要
主要控制流	条件、循环和函数(方法)调用	函数调用(含递归)
主要操作单元	结构或类的实例	视为高阶对象的函数和数据集合

F# 是运行在.NET 框架上的语言，它同时具备函数式编程语言模式和强制式编程语言模式。F# 语言具有下列特性：

(1) F# 支持多模式编程。F# 语言同时支持 3 种编程模式：函数式编程模式、强制式编程模式和面向对象编程模式。F# 语言是进入实用领域的函数式编程语言之一。

(2) F# 是静态强类型程序语言。静态强类型意味着 F# 具备类型推演功能，就是说不需要显式写出代码应该具备的类型，系统也会自动给出代码所应该具备的类型。强类型特性可以避免许多类型定义错误。这意味着 F# 代码具有简明、高效的特性。

(3) F# 是运行在.NET 框架上的语言。F# 能够访问所有.NET 库和用户编写的.NET 对象。这种特性使得 F# 语言具有更好的业界应用前景。

(4) 用户可以编写 F# 模块访问.NET 应用程序。

(5) F# 支持多处理器与多线程，有利于计算密集型应用。

(6) F# 的类型推演特性意味着可以编写更小巧与灵活的代码，且代码具有更少的潜在错误。

函数式编程语言编程使用最简洁的函数形式，与复杂的状态变量相比，其程序更易于理解。

函数式编程语言提供功能强大的复合结构。

由于不发生副作用(该特性称为**引用透明性**)，在函数式编程语言中，仅表达式的值

起作用。引用透明性导致存在一组功能强大的程序变换定律。

2. F# 发展历史

F# 语言由微软研究院、英国剑桥大学资助，Don Syme 设计和实现了最初的版本。由于微软研究院使用 Apache2 许可发布了 F# 编译器的源代码，F# 现在由 F# 软件基金会负责 F# 应用和多元化。虽然微软官方仅在 Windows 系统上支持 F#，但在 Mac、Linux、Android、iOS、HTML5 和 GPU 等众多平台上都可以使用 F# 语言。F# 由 OCaml 语言发展而来，OCaml 语言则是基于 ML 函数式程序语言发展的。

F# 之前的函数式语言更多的是用在学术研究领域，借助.NET 的强大类库支持，F# 是首个进入实用领域的函数式语言。微软 Visual Studio 2010 集成了 F# 2.0 语言开发，而微软 Visual Studio 2012 集成了 F# 3.0 语言开发。F# 的开源、强类型、多种编程模式特性使得它即可以在 Windows 的 Visual Studio 集成环境下开发，也可以在 Linux 的 Mono 环境下开发。F# 是具备通用目标的软件开发语言，其目前主要的应用领域包括云计算、移动计算、网络开发、语言处理、并发计算等。

1.2　.NET 框架基础知识

.NET 框架是微软 2002 年 2 月推出的全新软件开发、运行环境。微软的构想是一个“不再关注单个网站、单个设备与因特网相连的互联网环境，而是要让所有的计算机群、相关设备和服务商协同工作”的网络计算环境。它包括公共语言运行时(Common Language Runtime，CLR)[①]和基础类库两个部分。.NET 框架的特性包括：

(1) 多语言支持　基于.NET 应用程序可以使用不同编程语言开发。微软在.NET 平台上提供了如 C#、F#、VB.NET 以及 C++等多种语言以及相应的编译器，用户可以选择熟悉的编程语言完成.NET 应用开发。

(2) 应用平台的独立性　所有用.NET 语言写成的程序都将被编译成一种中间二进制代码**通用中间语言 CIL**(Common Intermediate Language，其前身称为 Microsoft Intermediate Language，MSIL，微软中间语言)，CIL 独立于硬件和操作系统，是平台无关最低级的面向人的语言。CIL 代码在 CLR 环境中运行。CLR 的角色基本上和 Java 平台中的 Java 虚拟机(Java Virtual Machine，JVM)相似。CLR 将 CIL 代码通过即时(Just in Time，JIT)编译器转换成实际运行的机器代码，然后运行。CLR 不关心使用何种开发语言，只要相应的开发语言的编译器面向 CLR 即可。

(3) 不同语言互操作性　为提高软件开发效率，使用一种.NET 编程语言编写的组件能够为另一种.NET 编程语言使用，从而实现不同语言互操作性。

(4) 应用的可移植性　编译成 CIL 中间代码的应用以可移植执行文件(Portable Executable，PE)的形式出现。换句话说，只要在支持.NET 的系统上，就可以将.NET

① 在微软 MSDN 中，CLR 称为公共语言运行时。作者认为，“通用语言运行时”更准确，因为 CLR 可以运行所有基于.NET 的语言，本书依然采用术语“公共语言运行时”。

应用程序部署运行。目前.NET 主要支持的平台包括 Windows 系统和安装有 Mono 的 Linux 系统。

.NET 语言与 CLR 之间的关系如图 1-1 所示。

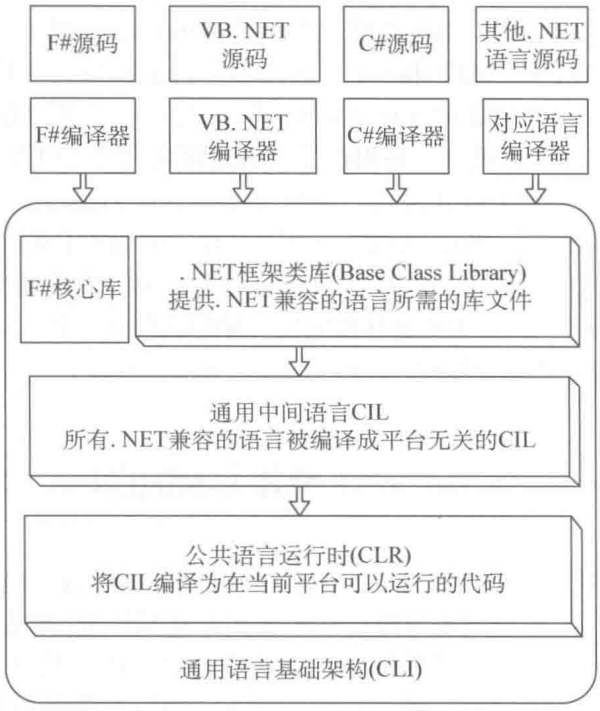


图 1-1 .NET 框架结构示意图

1. 公共语言运行时

公共语言运行时(CLR)定义了基于栈方式运行的虚拟机核心语义。程序设计语言通常需要同时定义语法和语义。程序设计语言的语法确定了程序语言书写的规范,而程序设计语言的语义则用来说明程序执行方式。CLR 仅定义了语义,没有定义语法。就是说,CLR 定义一个可以被多种编程语言使用的通用语义集。所有基于 CLR 的 .NET 语言(如 C#、VB.NET、F#等)均有其对应的语法定义规范,通过对应语言编译器将其编译为 CLR 定义的语义。

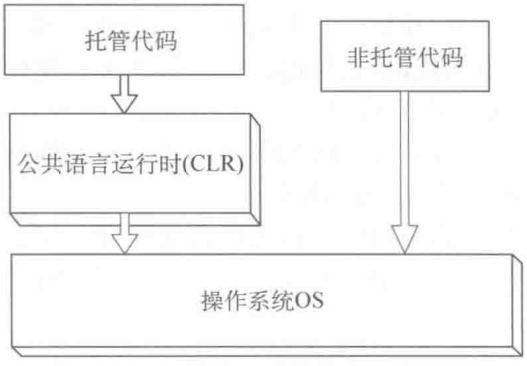


图 1-2 托管代码与非托管代码差异

在 CLR 上面运行的代码称为托管代码(managed code)。不在 CLR 上面运行的代码称为非托管代码(unmanaged code)。托管代码与非托管代码运行环境的差别如图 1-2 所示。

CLR 的核心功能包括内存管理、程序集加载、安全性、异常处理和线程同步等。使用 CLR 的主要优点包括:

(1) 结构化异常处理。

(2) 不同语言互操作性。能够轻松使用由其他语言开发的组件。

(3) 性能和安全性改善。性能得到了改进。使用委托取代函数指针，增强了类型安全和安全性。

(4) 线程管理。

(5) 垃圾回收。应用垃圾回收机制自动释放不再使用托管对象占用空间。

类库提供的可扩展类型。允许创建多线程的可缩放应用程序的显式自由线程处理支持。

2. 通用类型系统与公共语言规范

在程序语言中，类型扮演了非常重要的角色。如果一个值的类型确定，其可能的取值范围便确定，其分配的内存空间大小和其对应的成员属性或方法也确定。然而类型名称通常与语言相关。例如，同样为了表示 −2 147 483 648 到 2 147 483 647 间的数据，VB.NET 语言使用 Integer 数据类型，而 C# 和 F# 语言则使用 int 的数据类型。类型名称不同意味着 C# 和 F# 语言可能无法理解 VB.NET 语言的 Integer。

使用一种编程语言编写的软件组件如果能够为另一种编程语言调用，就可以实现不同语言间的互操作，从而提高代码重用水平和软件开发效率。不同编程语言的互操作性需要解决的一个问题是对类型的理解。为了不同语言中使用不同名称的类型在 CLR 中具有相同的名称，.NET 在 CLR 中定义了**通用类型系统**(Common Type System，CTS)。VB.NET 中的 Integer 和 C# 和 F# 中的 int 都对应与 CTS 中的 System. Int32 类型。CTS 类型也称为. **NET Framework 类型**。

从语言角度看到的类型名称与从 CLR 角度看到的类型名称是不同的。从语言角度看到的类型名称采用语言相关的方式，从 CLR 角度看到的类型名称则是.NET Framework 类型名称。

由于 CTS 还是复杂且较大，为实现不同语言互操作性，微软提出了 CTS 的子集**通用语言规范**(Common Language Specification，CLS)。凡遵守通用语言规范 CLS 的.NET 语言可以实现互操作。即用一种语言编写的类能够直接与另一种语言编写的类通信。通用类型系统(CTS)与通用语言规范(CLS)是.NET 实现不同语言互操作性的基石。

3. 值类型与引用类型

CTS 定义的类型集处于 CLR 核心，其两个核心概念是值类型与引用类型。

1) 值类型与引用类型

值类型(Value Types)与**引用类型**(Reference Types)区别：

(1) 表示的数据不同　值类型通常用来表示基元类型，即简单的类型。例如，所有数值类型(如整型、浮点和双精度)和枚举、字符、布尔、结构是值类型。引用类型用来表示存放复杂类型，例如，类、接口等。

(2) 存放位置不同　通常程序不同类型存放在不同区域。与 C 语言或 C++ 语言中的局部变量和指针变量分配在不同区域类似，值类型实例存放在栈(Stack)中，引用类型实例存放在堆(Heap)中，而指向引用类型真实数据的地址(也成为指针)存放在栈中。栈中即可存放值类型变量和也可存放函数返回的地址。栈存储单元的分配与释放

使用的是 LIFO (Last in first out,后入先出)算法,堆则使用动态分配方式,不用的对象需要 CLR 的垃圾回收机制释放。在访问性能上,访问存放在栈中的对象实例比访问存放在堆中的对象实例更快。栈与堆的区别如图 1-3 所示。

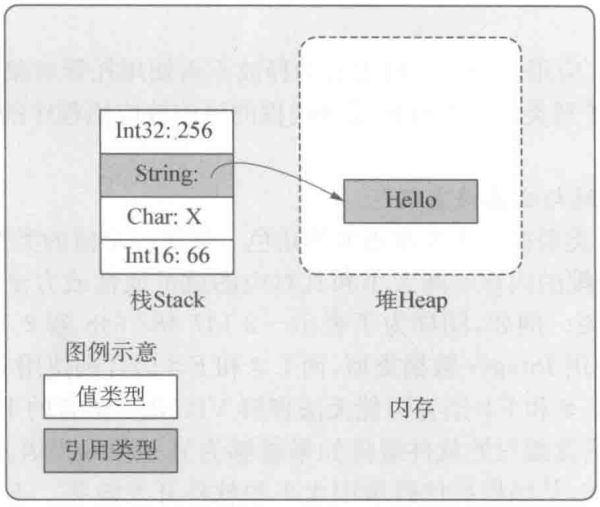


图 1-3 栈与堆的区别

(3) 派生的父类不同 所有值类型的父类为 System.Value Type,而所有引用类型的父类为 System.Object。CTS 部分类型分类如图 1-4 所示。

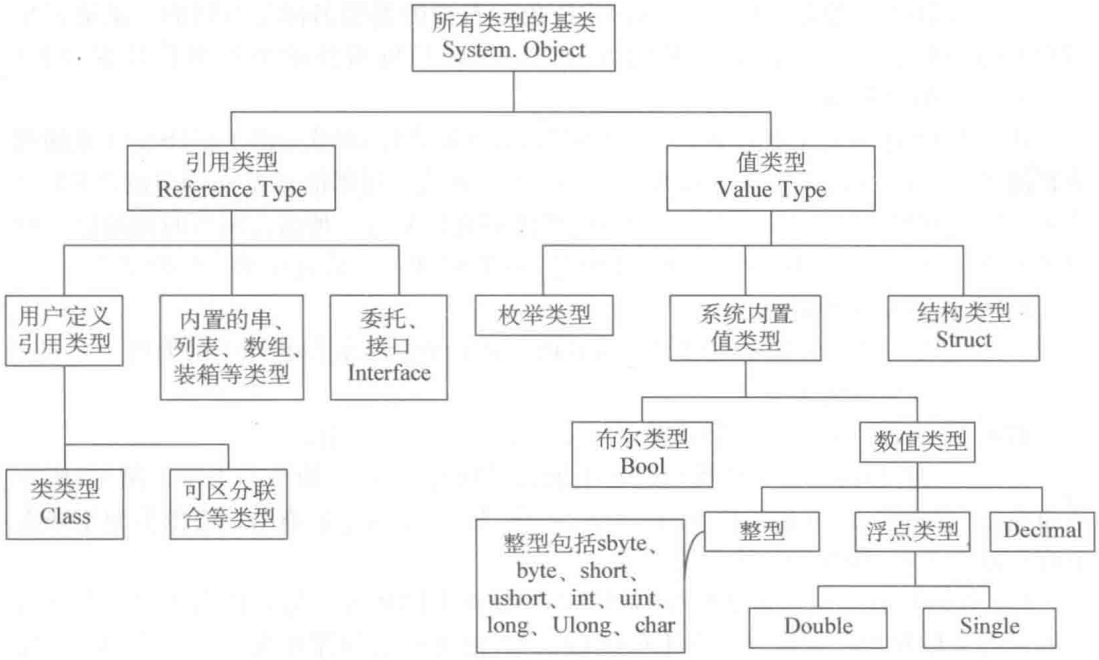


图 1-4 CTS 部分类型分类示意图