CENGAGE Learning

• 游戏开发经典丛书 •

(英) Daniel Schuller 著
张磊 李苏军 译

# 精通C#游戏编程

C# Game Programming For Serious Game Creation





清华大学出版社

·游戏开发经典丛书·

# 精通 C#游戏编程

(英) Daniel Schuller 著

张 磊 李苏军 译



清华大学出版社

北 京

北京市版权局著作权合同登记号　图字：01-2011-2582
C# Game Programming For Serious Game Creation
Daniel Schuller 著，张磊 李苏军 译



ISBN 978-7-302-27114-7

Cengage Learning Asia Pte. Ltd.
5 Shenton Way, # 01-01 UIC Building, Singapore 068808



**图书在版编目(CIP)数据**

精通 C#游戏编程/(英)斯库勒(Schuller，D.) 著；张磊，李苏军 译. —北京：清华大学出版社，2012.1
(游戏开发经典丛书)
书名原文：C# Game Programming For Serious Game Creation
ISBN 978-7-302-27114-7

I. 精…　II. ①斯…　②张…　③李…　III. 游戏—C 语言—程序设计　IV. TP312
中国版本图书馆 CIP 数据核字(2011)第 211675 号

**责任编辑**：王　军　徐燕萍
**装帧设计**：孔祥丰
**责任校对**：成凤进
**责任印制**：李红英
**出版发行**：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
**社　总　机**：010-62770175　**邮　购**：010-62786544
**投稿与读者服务**：010-62776969，c-service@tup.tsinghua.edu.cn
**质　量　反　馈**：010-62772015，zhiliang@tup.tsinghua.edu.cn
**印 刷 者**：北京鑫丰华彩印有限公司
**装 订 者**：三河市新茂装订有限公司
**经　　销**：全国新华书店
**开　　本**：185×260　**印　张**：22.5　**字　数**：548 千字
**版　　次**：2012 年 1 月第 1 版　　**印　　次**：2012 年 1 月第1 次印刷
**印　　数**：1～4000
**定　　价**：49.00 元

产品编号：041892-01

# 作者简介

Daniel Schuller 生于英国，是一名计算机游戏开发人员，曾在美国、新加坡和日本工作和生活，目前在英国工作。他在 PC、Xbox 360 和 PlayStation 3 上发布过游戏。Schuller 为 Sony、Ubisoft、Naughty Dog、RedBull 和 Wizards of the Coast 开发过游戏，并运营着一个游戏开发网站，网址为 http://www.godpatterns.com。除了开发计算机游戏，Schuller 还在学习日文，并对人工智能、认知科学和游戏在教育中的作用非常感兴趣。

# 致　　谢

非常感谢每个对本书出版做出过贡献的人，以及书中所用库和工具的开发者们。感谢策划编辑 Jenny Davidson，她使本书的表达语言更加简洁，并确保我可以及时地完成本书。感谢技术编辑 Jim Perry，他帮助我找出了一些代码错误，并给我提供了很有帮助的建议和意见。最后，感谢我的家人和同事对我的支持和鼓励。

# 前　言

希望本书能帮助你开发游戏。

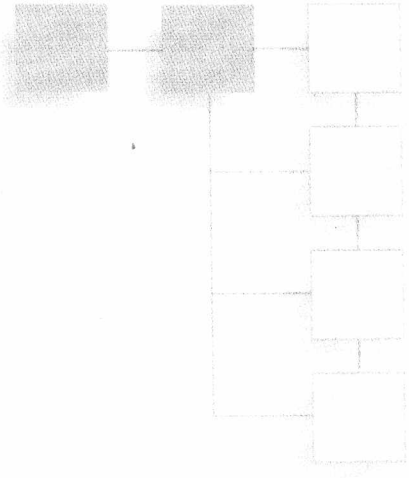
每个人都有出色的游戏构思，但是从最初的构思到完成一个项目所应采取的方法并不明朗。众多的编程语言、库和生产方法让人望而生畏。即使经验丰富的游戏开发人员有时候也无法实现自己最初的设想。如果没有优秀可靠的架构，游戏代码可能变得非常复杂，把开发人员淹没其中。代码越复杂，修改和继续开发游戏的难度就越大。

通过开发两个简单的游戏，本书介绍了如何编写简单、整洁、可靠的代码。这些游戏使用 C#编程语言和 OpenGL 创建。C#是一个先进的高级编程语言，所以编写代码更快，需要避免的程序缺陷更少。OpenGL 几乎可以说是游戏业在显示图形时使用的标准方式。读完本书后，你将拥有一个优秀的代码库来进行开发和扩展，以实现自己的想法。

本书的第 I 部分将概述创建优秀游戏的方法和库。第 II 部分将介绍如何使用这些库和如何创建自己的可重用游戏库。你还将学到如何开发简单的卷轴射击游戏，并了解在实现自己的游戏想法时可以采纳的建议和提示。本书的配套资料中包含了开始开发游戏所需的一切。书中的每个代码段在资料中都有完整的源代码和程序。此外，资料中还提供了一些简单的游戏资源，以及指向有价值的游戏开发网站和图形网站的一些链接。本书配套资料可以从 http://www.tupwk.com.cn/downpage 中下载。

# 目　　录

# 第 I 部分



# 背景知识

# 第 1 章

# C#的历史

C#是一种面向对象的现代语言，由 Anders Hejlsberg 带领的 Microsoft 团队开发。公共语言运行时(Common Language Runtime，CLR)是运行 C#的虚拟机。许多语言都运行在 CLR 上，这意味着在 Windows PC、Xbox 360 和 Zune 上都可以编译和使用它们。CLR 由 Microsoft 开发并拥有，但是它也有一个开源版本，叫做 Mono。通过 Mono，C#程序可以运行在 Mac、Linux，或者其他能够编译 Mono 的系统上。

游戏业中一般使用C++，但是这并不代表程序员想这么做。C++是一种庞大的语言，有许多地方会让粗心的程序员掉入陷阱，也有许多根本不曾定义的地方。C#比 C++友好得多，用来编写游戏的话，也会使程序员的工作有趣得多。

## 1.1 C#基础

2000 年 7 月在 Orlando 举行的 Microsoft Professional Developers Conference 会议上第一次公布了 C#。在当时，C#还有点像 Microsoft 版本的 Java，但是很快它就被开发成了一种具有自己显著特点的语言。C#最近的更新颇具创新性，让开发人员倍感兴奋。

C#和 Java 等现代语言编写的程序运行在虚拟机上。而由 C++和 C 等早一些的语言编写的程序要直接运行在特定计算机的硬件上。运行程序的硬件叫做中央处理单元(Central Processing Unit，CPU)，这是计算机的“大脑”。现代 Mac 和 PC 一般都采用 x86 CPU，Xbox 360 采用 Xenon CPU，PS3 则采用 Cell CPU。这些 CPU 彼此之间都存在一些差异，但是它们的基本用途是相同的：运行程序员编写的程序。程序就是一条条的指令。特定 CPU 能够理解的指令叫做机器码。一个 CPU 不太可能理解其他类型的 CPU 的机器码。

编译器把使用 C++、C 或其他语言编写的、人类可读的源代码编译成机器码。例如，

PlayStation 3 的 C++编译器将源代码编译成 Cell CPU 可以运行的机器码。要在 Xbox 360 的 Xenon CPU 上运行相同的 C++代码，必须使用 Xbox 360 的 C++编译器重新进行编译。C# 的工作方式与此不同。C#编译得到的汇编语言叫做公共中间语言(Common Intermediate Language，CIL)。CIL 运行在一个虚拟机上，以生成特定系统的机器码。在 PlayStation 3 中，这意味着要运行 C#代码，需要有一个把 CIL 代码转变为 Cell CPU 可以理解的机器码的虚拟机。图 1-1 显示了使用虚拟机的语言和直接编译为机器码的语言之间的这种区别。

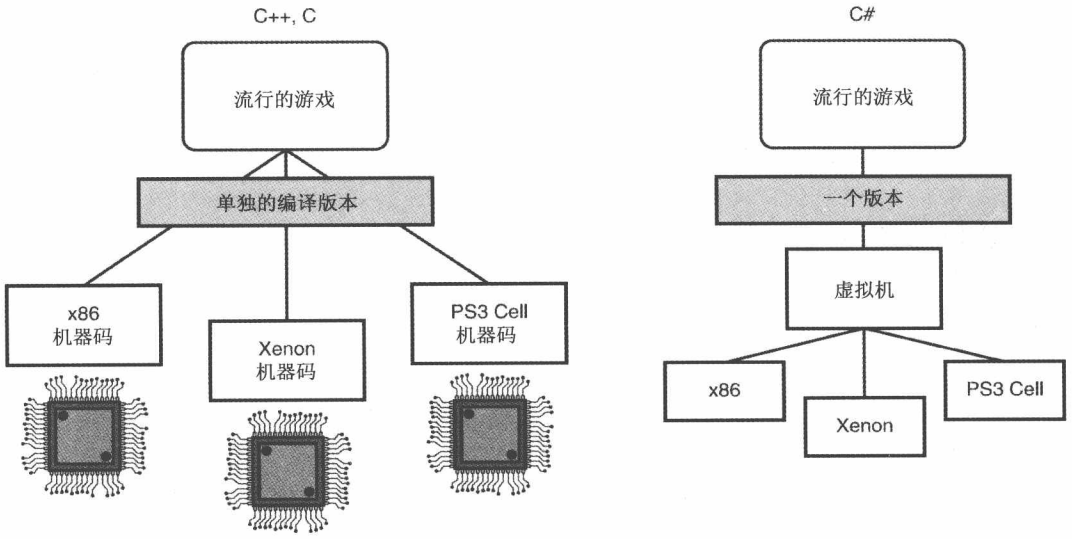



图 1-1 虚拟机和直接编译的代码

C#的虚拟机 CLR 有许多优点。最显著的就是，只需要编写代码一次，而后代码就可以在每个支持 CLR 的 CPU 上运行。使用 C#时，系统不太容易崩溃，因为虚拟机把系统隔离开了。在错误被发送到硬件之前，虚拟机就会捕获它们。内存的分配和释放也会自动完成。只要程序语言会被编译成 CLR，就可以混用它们，诸如 F#、IronLisp、IronRuby、IronPython 和 IronScheme 等语言可以用在一个程序中。游戏程序员可以使用高级的、适合编写 AI 的语言来编写 AI，而使用与 C 类似的过程语言编写图形处理代码。

## C#的版本

C#定期更新功能和特性。即使程序员以前使用 C#编写过程序，也可能会不知道 C#在最近的更新中添加的所有新特性。

### 1. C# 2.0

C# 2.0 添加了泛型和匿名委托。使用示例最容易解释这些新特性。

#### 泛型

泛型是编写代码来使用具有通用属性的通用对象，而不是使用具体对象的一种方法。可以对类、方法、接口和结构使用泛型，以定义它们使用的数据类型。泛型代码使用的数

据类型在编译时指定，这意味着程序员不需要编写代码来检查是否使用了正确的数据类型。这样做的好处是，由于不需要编写测试代码，代码更加简洁，而且由于减少了在运行时发生的类型不匹配错误，代码也更加安全。泛型代码的运行速度一般较快，因为不需要执行很多数据类型的强制转换。

下面的代码没有使用泛型：

```
ArrayList _list = new ArrayList();
_list.Add(1.3); // boxing converts value type to a reference type
_list.Add(1.0);

//Unboxing Converts reference type to a value type.
object objectAtPositionZero = _list[0];
double valueOne = (double) objectAtPositionZero;
double valueTwo = (double) _list[1];
```

这段代码创建了一个列表，并添加了两个十进制数字。C#将每个事物都看作对象，数字也不例外。C#中的全部对象都从 object 类继承。对象存储在 ArrayList 中。上面的代码添加了数字，但是 ArrayList 不知道它们是数字。对于 ArrayList 来说，它们就是对象，ArrayList 只能识别对象。把数字传递给 ArrayList.Add 时，它们被转换为 object 类型，然后添加到 ArrayList 的对象集合中。

每次从列表中取出对象时，必须将它从对象重新转换为原来的类型。类型转换的效率很低，而且使代码变得杂乱而难以阅读，所以非泛型代码很难处理。泛型代码可以解决这种问题。为了理解它们如何解决问题，必须理解引用类型和值类型的区别。

C#有两类宽泛的类型：引用类型和值类型。值类型是基本类型，例如 int、float、double、bool、string 等。引用类型是程序员使用关键字 class 定义的大型数据结构，例如 Player、Creature、Spaceship 等。使用关键字 struct 也可以自定义值类型。

```
// An example reference type
public class SpaceShip
{
  string _name;
  int _thrust;
}
// An example value type
public struct Point
{
  int _x;
  int _y;
}
```

值类型直接把数据存储到内存中。引用类型间接地把数据存储到内存中。在内存中，引用类型的地址是另外一个内存地址，数据实际存储在这个地址。初看起来，这可能有点奇怪，但是熟悉了这些概念后就会感到这其实很简单。两种类型在内存中的区别如图 1-2 所示。

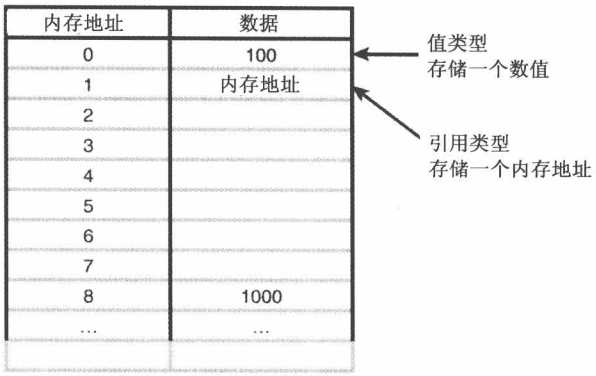



图 1-2 内存中的值类型和引用类型

如果把计算机的内存视为一个巨大的图书馆，其中都是书架，创建一个值类型就是在书架上加入几本书。创建引用类型则更像是在书架上放入一张纸条，指出书存放在图书馆的哪个地方。

下面是值类型的一个例子。

```
int i = 100;
```

如果在内存中查看 *i* 的存储位置，就会得到数字 100。在源代码中我们使用了变量名 *i*，在汇编代码中 *i* 是内存地址。所以这行代码将值 100 放到下一个可用的内存地址，并把该内存地址叫做 *i*。

```
int i = 100;
int j = i; // value type so copied into i
Console.WriteLine(i); // 100
Console.WriteLine(j); // 100
i = 30;
Console.WriteLine(i); // 30
Console.WriteLine(j); // 100
```

在这段代码中，值类型 *i* 被赋给 *j*，意味着它的数据将被复制。内存地址 *j* 中存储的数据将与 *i* 存储的数据完全相同。

下一个示例与此示例类似，但是使用了引用类型。首先，需要通过定义新类来创建一个引用类型。这个类的作用只是存储数字，就像是我们自己创建的 int 类型。这个类型将被命名为 Int，使用大写 I 来与 C#内置的 int 类型进行区分。

```
class Int()
{
  int _value;
  public Int(int value)
  {
    _value = value;
  }
```

```
    public void SetValue(int value)
    {
      _value = value;
    }
    public override string ToString()
    {
      return _value.ToString();
    }
}

Int i = new Int(100);
Int j = i; // reference type so reference copied into i
Console.WriteLine(i); // 100
Console.WriteLine(j); // 100
i.SetValue(30);
Console.WriteLine(i); // 30
Console.WriteLine(j); // 30 <- the shocking difference
```

在这里，当 *i* 改变时，*j* 也会改变。这是因为 *j* 本身存储没有数据的副本，它只是具有与 *i* 相同的数据引用。当底层数据改变时，无论使用 *i* 还是使用 *j* 访问数据，总是会返回相同的结果。

再次以图书馆为例。在图书馆的索引系统中查找想要的图书，得到了两个书架位置。第一个位置 *i* 没有存放书，而只有一张纸条。检查位置 *j* 时，发现了类似的一张纸条。这两张纸条都指向了另外一个位置。无论采纳哪张纸条的指示，都可以得到要寻找的图书。书只有一本，但是引用却有两个。

装箱(boxing)是将值类型转换为引用类型的过程。取消装箱(unboxing)是将新的引用类型转换回原来的值类型的过程。在 C#的第 1 版中，即使对象列表中只包含值类型，在把这些对象添加到列表中时也需要将它们转换为引用类型。引入泛型后，情况发生了变化。下面的示例代码使用了泛型功能，比较它们与前面使用 ArrayList 的示例之间的区别。

```
List<float> _list = new List<float>();
_list.Add(1);
_list.Add(5.6f);
float valueOne = _list[0];
float valueTwo = _list[1];
```

这段示例代码创建了一个由 float 类型的成员组成的列表。float 是值类型。在这里不需要像前面那样通过强制转换对 float 执行装箱和取消装箱操作，代码直接就可以完成工作。泛型列表知道自己存储的类型，所以不需要将值类型转换为对象类型。由于不需要总是在类型之间进行转换，所以得到的代码更加高效。

泛型对于创建可重用的数据结构(例如列表)十分有用。

### 匿名委托

在 C#中，委托是一种把一个函数作为另一个函数的参数进行传递的方法。当需要对许

多不同的列表采取重复性的操作时，这种特性就非常实用。下面用一个函数迭代列表，用另一个函数对列表的每个成员执行操作。示例如下：

```
void DoToList(List<Item> list, SomeDelegate action)
{
  foreach(Item item in list)
  {
    action(item);
  }
}
```

匿名委托允许在 DoToList 函数调用中直接写函数，而不需要声明或预定义该函数。由于没有预定义，该函数没有名称，所以叫做匿名函数。

```
// A quick way to destroy a list of items
DoToList(_itemList, delegate(Item item) { item.Destroy(); });
```

这个示例演示了如何迭代列表，并对每个成员应用匿名函数。示例中的匿名函数调用每个成员的 Destroy 方法。这的确很方便。匿名函数还有另外一个技巧。观察下面的代码：

```
int SumArrayOfValues(int[] array)
{
  int sum = 0;
  Array.ForEach(
    array,
    delegate(int value)
    {
      sum + = value;
    }
  );
  return sum;
}
```

这被叫做闭包。它为每次循环迭代创建一个委托，并使用相同的变量 *sum*。变量 *sum* 甚至可以超出作用域，但是匿名委托仍然将保持对它的引用。它将成为一个只能被封闭它的函数访问的私有变量。闭包在函数式程序语言中有着大量应用。

### 2. C# 3.0

C# 3.0 不满足于做少量改进，所以引入了极具创新性的新功能，可以显著减少样板代码的数量。最大的变化就是引入了 LINQ 系统。

#### LINQ

LINQ 是 Language-Integrated Query(语言集成查询)的缩写。它与处理数据结构的 SQL(结构化查询语言，一种用于从数据库中检索并操作数据的语言)有些类似。下面通过示例简单地了解一下 LINQ 的用法。