## 在 C# 1.0 及更高版本中，可以按以下示例所示声明委托。

// Declare a delegate.

delegate void Del(string str);

// Declare a method with the same signature as the delegate.

static void Notify(string name)

{

Console.WriteLine("Notification received for: {0}", name);

}

// Create an instance of the delegate.

Del del1 = new Del(Notify);

## C# 2.0 提供了更简单的方法来编写上面的声明，如以下示例所示。

// C# 2.0 provides a simpler way to declare an instance of Del.

Del del2 = Notify;

## 在 C# 2.0 及更高版本中，还可以使用匿名方法来声明和初始化[委托](https://msdn.microsoft.com/zh-cn/library/900fyy8e.aspx)，如以下示例所示。

// Instantiate Del by using an anonymous method.

Del del3 = delegate(string name)

{ Console.WriteLine("Notification received for: {0}", name); };

## 在 C# 3.0 及更高版本中，还可以使用 Lambda 表达式来声明和实例化委托，如以下示例所示。

// Instantiate Del by using a lambda expression.

Del del4 = name => { Console.WriteLine("Notification received for: {0}", name); };

有关更多信息，请参见 [Lambda 表达式（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb397687.aspx)。

下面的示例阐释声明、实例化和使用委托。 BookDB 类封装一个书店数据库，它维护一个书籍数据库。它公开 ProcessPaperbackBooks 方法，该方法在数据库中查找所有平装书，并对每本平装书调用一个委托。使用的 **delegate** 类型名为 ProcessBookDelegate。 Test 类使用该类打印平装书的书名和平均价格。

委托的使用促进了书店数据库和客户代码之间功能的良好分隔。客户代码不知道书籍的存储方式和书店代码查找平装书的方式。书店代码也不知道找到平装书后将对平装书执行什么处理。

// A set of classes for handling a bookstore:

namespace Bookstore

{

using System.Collections;

// Describes a book in the book list:

public struct Book

{

public string Title; // Title of the book.

public string Author; // Author of the book.

public decimal Price; // Price of the book.

public bool Paperback; // Is it paperback?

public Book(string title, string author, decimal price, bool paperBack)

{

Title = title;

Author = author;

Price = price;

Paperback = paperBack;

}

}

// Declare a delegate type for processing a book:

public delegate void ProcessBookDelegate(Book book);

// Maintains a book database.

public class BookDB

{

// List of all books in the database:

ArrayList list = new ArrayList();

// Add a book to the database:

public void AddBook(string title, string author, decimal price, bool paperBack)

{

list.Add(new Book(title, author, price, paperBack));

}

// Call a passed-in delegate on each paperback book to process it:

public void ProcessPaperbackBooks(ProcessBookDelegate processBook)

{

foreach (Book b in list)

{

if (b.Paperback)

// Calling the delegate:

processBook(b);

}

}

}

}

// Using the Bookstore classes:

namespace BookTestClient

{

using Bookstore;

// Class to total and average prices of books:

class PriceTotaller

{

int countBooks = 0;

decimal priceBooks = 0.0m;

internal void AddBookToTotal(Book book)

{

countBooks += 1;

priceBooks += book.Price;

}

internal decimal AveragePrice()

{

return priceBooks / countBooks;

}

}

// Class to test the book database:

class TestBookDB

{

// Print the title of the book.

static void PrintTitle(Book b)

{

System.Console.WriteLine(" {0}", b.Title);

}

// Execution starts here.

static void Main()

{

BookDB bookDB = new BookDB();

// Initialize the database with some books:

AddBooks(bookDB);

// Print all the titles of paperbacks:

System.Console.WriteLine("Paperback Book Titles:");

// Create a new delegate object associated with the static

// method Test.PrintTitle:

bookDB.ProcessPaperbackBooks(PrintTitle);

// Get the average price of a paperback by using

// a PriceTotaller object:

PriceTotaller totaller = new PriceTotaller();

// Create a new delegate object associated with the nonstatic

// method AddBookToTotal on the object totaller:

bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

System.Console.WriteLine("Average Paperback Book Price: ${0:#.##}",

totaller.AveragePrice());

}

// Initialize the book database with some test books:

static void AddBooks(BookDB bookDB)

{

bookDB.AddBook("The C Programming Language", "Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);

bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, true);

bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);

bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true);

}

}

}

/\* Output:

Paperback Book Titles:

The C Programming Language

The Unicode Standard 2.0

Dogbert's Clues for the Clueless

Average Paperback Book Price: $23.97

\*/

**可靠编程**

* 声明委托。

下面的语句声明一个新的委托类型。

public delegate void ProcessBookDelegate(Book book);

每个委托类型都描述参数的数目和类型，以及它可以封装的方法的返回值类型。每当需要一组新的参数类型或新的返回值类型时，都必须声明一个新的委托类型。

* 实例化委托。

声明了委托类型后，必须创建委托对象并使之与特定方法关联。在上一个示例中，您通过按下面示例中的方式将 PrintTitle 方法传递到 ProcessPaperbackBooks 方法来实现这一点：

bookDB.ProcessPaperbackBooks(PrintTitle);

这将创建与[静态](https://msdn.microsoft.com/zh-cn/library/98f28cdx.aspx)方法 Test.PrintTitle 关联的新委托对象。类似地，对象 totaller 的非静态方法 AddBookToTotal 是按下面示例中的方式传递的：

bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

在两个示例中，都向 ProcessPaperbackBooks 方法传递了一个新的委托对象。

委托创建后，它的关联方法就不能更改；委托对象是不可变的。

* 调用委托。

创建委托对象后，通常将委托对象传递给将调用该委托的其他代码。通过委托对象的名称（后面跟着要传递给委托的参数，括在括号内）调用委托对象。下面是委托调用的示例：

processBook(b);

与本例一样，可以通过使用 **BeginInvoke** 和 **EndInvoke** 方法同步或异步调用委托。

**请参阅**

[C# 编程指南](https://msdn.microsoft.com/zh-cn/library/67ef8sbd.aspx)

[事件（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/awbftdfh.aspx)

[委托（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/ms173171.aspx)