使用对象初始值设定项，你可以在创建对象时向对象的任何可访问字段或属性分配值，而无需调用后跟赋值语句行的构造函数。利用对象初始值设定项语法，你可为构造函数指定参数或忽略参数（以及括号语法）。以下示例演示如何使用具有命名类型 Cat 的对象初始值设定项以及如何调用默认构造函数。请注意，自动实现的属性在 Cat 类中的用法。有关详细信息，请参阅[自动实现的属性（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb384054.aspx)。

class Cat

{

// Auto-implemented properties.

public int Age { get; set; }

public string Name { get; set; }

}

Cat cat = new Cat { Age = 10, Name = "Fluffy" };

**具有匿名类型的对象初始值设定项**

尽管对象初始值设定项可用于任何上下文中，但它们在 LINQ 查询表达式中特别有用。查询表达式常使用只能通过使用对象初始值设定项进行初始化的[匿名类型](https://msdn.microsoft.com/zh-cn/library/bb397696.aspx)，如下面的声明所示。

var pet = new { Age = 10, Name = "Fluffy" };

利用匿名类型，**select** 查询表达式中的 LINQ 子句可以将原始序列的对象转换为其值和形状可能不同于原始序列的对象。如果你只想存储某个序列中每个对象的部分信息，则这很有用。在下面的示例中，假定产品对象 (p) 包含很多字段和方法，而你只想创建包含产品名和单价的对象序列。

var productInfos =

from p in products

select new { p.ProductName, p.UnitPrice };

执行此查询时，productInfos 变量将包含一系列对象，这些对象可以在 **foreach** 语句中进行访问，如下面的示例所示：

foreach(var p in productInfos){...}

新的匿名类型中的每个对象都具有两个公共属性，这两个属性接收与原始对象中的属性或字段相同的名称。你还可在创建匿名类型时重命名字段；下面的示例将 UnitPrice 字段重命名为 Price。

select new {p.ProductName, Price = p.UnitPrice};

**具有可以为 null 的类型的对象初始值设定项**

使用具有可以为 null 的结构的对象初始值设定项会导致编译时错误。

**集合初始值设定项**

集合初始值设定项允许在初始化实现 [IEnumerable](https://msdn.microsoft.com/zh-cn/library/system.collections.ienumerable.aspx) 的集合类或初始化具有 **Add** 扩展方法的类时，指定一个或多个元素初始值设定项。元素初始值设定项可以是简单的值、表达式或对象初始值设定项。通过使用集合初始值设定项，你将无需在源代码中指定对该类的 **Add** 方法的多个调用；编译器将添加这些调用。

下面的示例演示了两个简单的集合初始值设定项：

List<int> digits = new List<int> { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<int> digits2 = new List<int> { 0 + 1, 12 % 3, MakeInt() };

下面的集合初始值设定项使用对象初始值设定项来初始化上一个示例中定义的 Cat 类的对象。请注意，各个对象初始值设定项分别括在大括号中且用逗号隔开。

List<Cat> cats = new List<Cat>

{

new Cat(){ Name = "Sylvester", Age=8 },

new Cat(){ Name = "Whiskers", Age=2 },

new Cat(){ Name = "Sasha", Age=14 }

};

如果集合的 [null](https://msdn.microsoft.com/zh-cn/library/edakx9da.aspx) 方法允许，则可以将 **Add** 指定为集合初始值设定项中的一个元素。

List<Cat> moreCats = new List<Cat>

{

new Cat(){ Name = "Furrytail", Age=5 },

new Cat(){ Name = "Peaches", Age=4 },

null

};

如果集合支持索引，可以指定索引元素。

var numbers = new Dictionary<int, string> {

[7] = "seven",

[9] = "nine",

[13] = "thirteen"

};

// The following code consolidates examples from the topic.

class ObjInitializers

{

class Cat

{

// Auto-implemented properties.

public int Age { get; set; }

public string Name { get; set; }

}

static void Main()

{

Cat cat = new Cat { Age = 10, Name = "Fluffy" };

List<Cat> cats = new List<Cat>

{

new Cat(){ Name = "Sylvester", Age=8 },

new Cat(){ Name = "Whiskers", Age=2 },

new Cat(){ Name = "Sasha", Age=14 }

};

List<Cat> moreCats = new List<Cat>

{

new Cat(){ Name = "Furrytail", Age=5 },

new Cat(){ Name = "Peaches", Age=4 },

null

};

// Display results.

System.Console.WriteLine(cat.Name);

foreach (Cat c in cats)

System.Console.WriteLine(c.Name);

foreach (Cat c in moreCats)

if (c != null)

System.Console.WriteLine(c.Name);

else

System.Console.WriteLine("List element has null value.");

}

// Output:

//Fluffy

//Sylvester

//Whiskers

//Sasha

//Furrytail

//Peaches

//List element has null value.

}

## 请参阅

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

[LINQ 查询表达式（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb397676.aspx)

[匿名类型（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb397696.aspx)