# 开发规范-Unity

|  |  |  |  |
| --- | --- | --- | --- |
| 版本序号 | 描述 | 日期 | 作者 |
| V1.0 | 编写初版Unity开发规范 | 2024.05 | 钟樾 |
| V1.1 | 新增：个人电脑层级结构、场景构成  修改：SVN库层级结构 | 2024.09.18 | 钟樾 |

## SVN库层级结构

考虑到项目会有多个子工程文件，且平台也多样性。如果涉及到多个同项目名，建议使用罗马数字对项目进行标识，并在README.txt中进行说明解释

......|--NAS/Project/甲方单位名/项目名/Unity/

| |--U3D\_甲方单位名\_项目名

| |--U3D\_甲方单位名\_项目名\_I

| |--U3D\_甲方单位名\_项目名\_II

| |--工程文件说明.txt // 对工程文件进行说明

## 个人电脑层级结构

|  |  |
| --- | --- |
| 规范 | 示例 |
| |--D盘/E盘  | |--\_Work  | | |--Projects  | | | |--甲方单位名  | | | | |--项目名  | | | | | |--U3D\_单位名\_项目名  | | | | | |--README.txt | |--D盘  | |--\_Work  | | |--Projects  | | | |--Rokid  | | | | |--NJU\_MR  | | | | | |--U3D\_Rokid\_NJU\_MR  | | | | | |--README.txt |

## Unity工程文件结构

|--Assets

| |--#ProjectName#

| | |--Art

| | |--Effects

| | |--Materials

| | |--Objects

| | |--Characters

| | |--#Name#

| | |--Animations

| | |--Materials

| | |--Mesh

| | |--Prefabs

| | |--Textures

| | |--Animals

| | |--Buildings

| | |--Foliage

| | |--Others

| | |--Props

| | |--Vehicles

| | |--Weapons

| | |----PostProcess

| | |----Textures

| | |----UI

| | |--Animations

| | |--Fonts

| | |--Icons

| | |--Sprites

| | |--Codes

| | |--Scripts

| | | |--Editor

| | | |--UI

| | |--Shaders

| | |--Level

| | |--Prefabs

| | |--Scenes

| | |---Media

| | |--Audio

| | |--Video

| | |--RenderTextures

| | |--Settings

| | |--Timeline

| |--\_Dev // 开发者个人测试文件（禁止上传SVN）

| |--Editor

| |--HYFT\_Lib// 公司library

| |--Unity

| |--Art

| |--Code

| |--Editor

| |--Tools // unity编辑器工具

| |--Notes // unity学习笔记等

| |--Plugins

| |--Android

| |--iOS

| |--Special

| |--Resources // 除插件自带的外，全局仅能出现一个该文件夹

| |--StreamingAssets

| |--ScriptTemplates

| |--URP

## 场景结构(待测试)

* 所有空对象位于（0，0，0）处，并具有默认的旋转和比例
* 对于仅作为脚本容器的空对象，请使用“@”作为前缀
* 加载实例化对象时，将其放在“\_Dynamic”下

>[--X.X.X--SceneManager--]

>>--@System--

>>--@Debug--

>>--@Management--

>>--@UI--

>>>--Layouts

--Cameras--

--Lights--

| |--Volumes

--Particles--

--Sound--

--World--

| |--Global

| |--Room1

| |--Architecture

| |--Terrain

| |--Props

--Gameplay--

| |--Actors

| |--Items

| |--Triggers

| |--Quests

.../--\_Dynamic--

## 场景构成

1. 应有主场景（持久关卡）和子场景的概念。方便多人协作时，能够各自进行不同的工作
2. 主场景的主要功能为：存放通用的资源或代码（即很多个子关卡会使用的资源），并控制子场景的加载/卸载
3. 子关卡分为三类，分别为：序列关卡、环境关卡、光照关卡
4. 序列关卡：用于管理Timeline及其所使用资源，使用”\_Cinema”后缀
5. 环境关卡：用于管理环境相关的模型、特效等资源，使用”\_Env”后缀
6. 光照关卡：用于管理光照相关的资源，使用”\_Light”后缀
7. 当有多个相同类型的子关卡时，在后缀中添加说明，如：光照关卡有白天、黑夜等多个关卡，使用”\_Light\_Night，\_Light\_Daytime”进行区分

## C#脚本规范

### Class

* 源文件应仅包含一种公共类型，但允许多个内部类
* 源文件应指定文件中公共类的名称
* 使用明确定义的结构组织命名空间
* 类成员应按顺序排列，并分组为多个部分

Constant Fields// 常量

Static Fields// 静态

Fields// 字段

Constructors// 构造函数

Properties // 属性

Events / Delegates // 事件、委托

LifeCycle Methods (Awake, OnEnable, OnDisable, OnDestroy)// 生命周期函数

Public Methods // public方法

Private Methods // private方法

Nested types// 嵌套类型

* 每个组内的访问顺序为：

Public

Internal

Protected

Private



#### 脚本模板

**如何自定义脚本模板**

* 如果要更改初始脚本，可以修改此处存储的脚本模板

Windows：C：\Program Files\Unity\Editor\Data\Resources\ScriptTemplates

Mac：/Applications/Unity/Editor/Data/Resources/ScriptTemplates

Mac（自 5.2.1f1 起）：/Applications/Unity/Unity.app/Contents/Resources/ScriptTemplates

* 编辑 81-C# Script-NewBehaviourScript.cs.txt文件
* 修改模板文件后，请重新启动 Unity 编辑器以应用这些更改。
* 注意：升级 Unity 版本时，模板文件将被覆盖，您需要再次复制这些文件并将其替换为自定义模板。
* 注意：可以在脚本中更改任何内容，但请记住将 #SCRIPTNAME# 保留在原处。没有它，模板类名将不会更改，并且将生成一个具有不正确类名的新脚本文件。

**如何在unity project上自定义unity脚本模板**

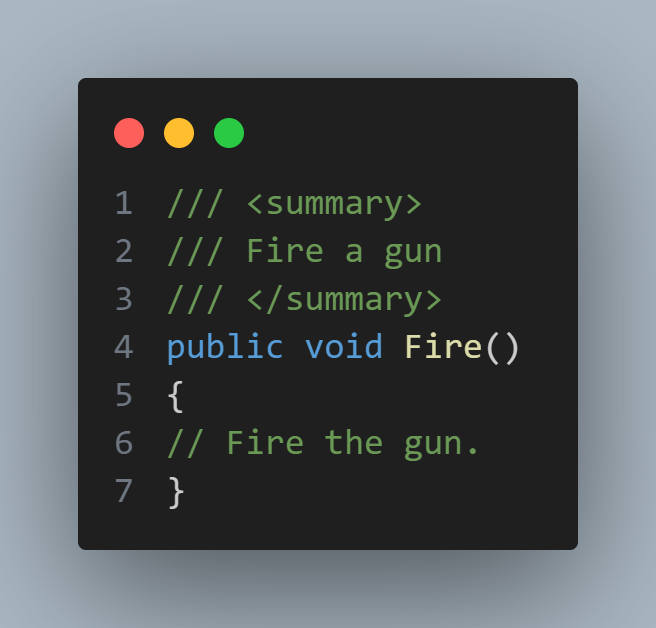
* 在Assets文件夹下，新建文件夹“ScriptTemplates”
* 将编辑好的“81-C# Script-NewBehaviourScript.cs.txt”文件放在“ScriptTemplates”文件夹下
* 修改模板文件后，请重新启动 Unity 编辑器以应用这些更改。

#### 命名空间

* 使用命名空间来确保类/枚举/接口等的作用域不会与其他命名空间或全局命名空间中的现有范围冲突。
* 项目应至少使用命名空间的项目名称，以防止与任何导入的第三方资产发生冲突。

#### 摘要

* 所有的公共功能都应有一个摘要
* 简单地说，任何具有 Public 访问修饰符的函数都应填写其摘要。



#### 折叠组

* 如果类只有少量变量，则不需要折叠组。
* 如果类具有适量的变量 （5-10），则所有可序列化变量都应分配一个非默认的折叠组。一种常见的类别是Config

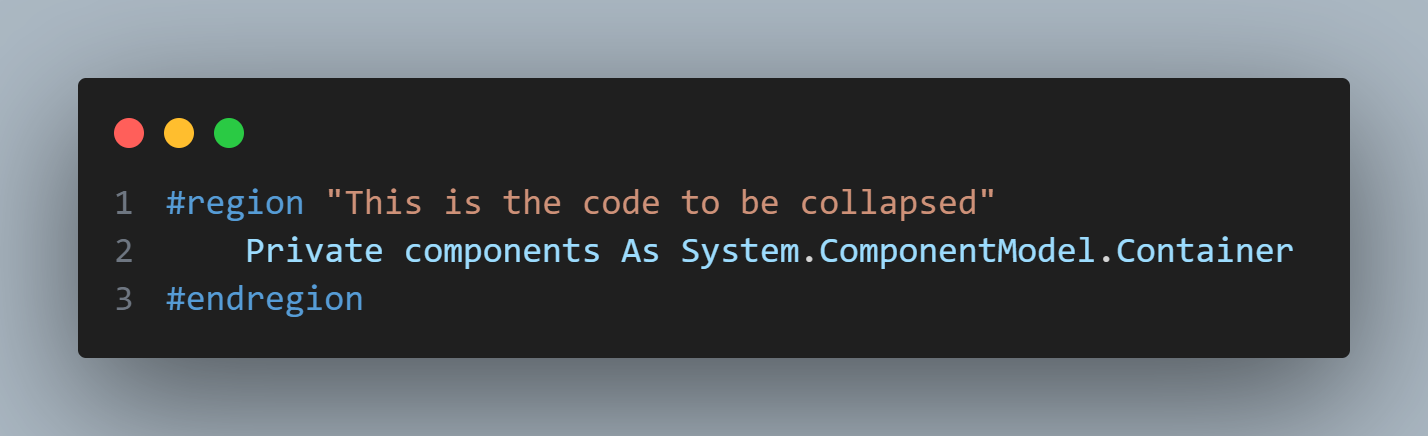
**要创建折叠组，unity有2种方法**

* 第一种是在主类内部定义一个，但这可能会对性能产生影响。这允许共享使用相同的变量名称。[Serializable] public Class
* （付费插件-待议）第二个选项是使用 Odin Inspector([Odin 检查器和序列化器 |改进 Unity 中的工作流程 (odininspector.com)](https://odininspector.com/)) 提供的折叠组属性。



#### Region

* #region指令使您能够折叠和隐藏 C# 文件中的代码部分。有选择地隐藏代码的能力使您的文件更易于管理且更易于阅读。



#### 间距

* 在函数参数之间使用逗号后使用单个空格。例：

Console.In.Read(myChar, 0, 1);

* 不要在括号和函数参数之间使用空格。
* 不要在函数名称和括号之间使用空格。
* 不要在空括号内使用空格

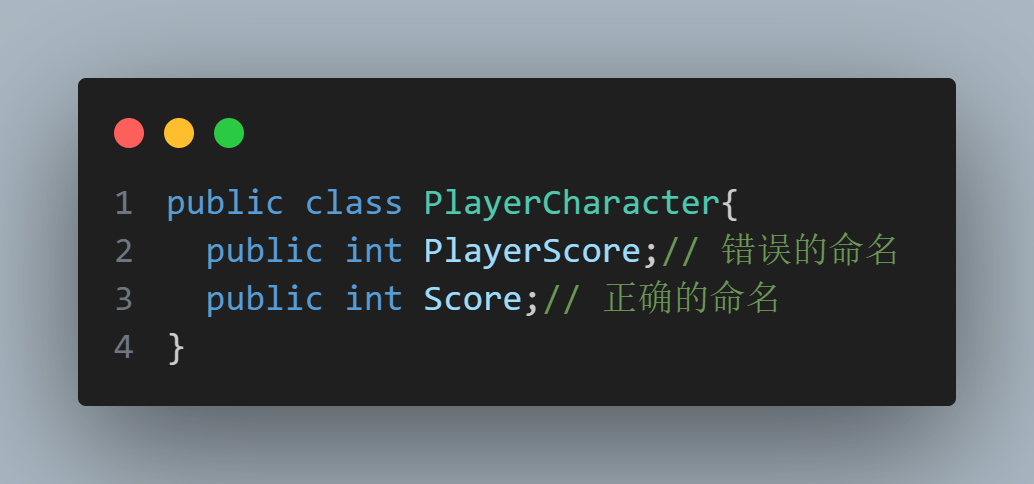
### 编译

* 所有脚本的编译都应零警告和零错误。
* 不要将损坏的脚本提交到源代码管理。如果必须将它们存储在源代码管理中，请将其单独放置在特定文件夹下。

### 变量

#### **变量命名**

* 所有公有变量，使用PascalCase命名法
* 私有变量，使用camelCase命名法
* 所有变量名不与上下文冗余，如

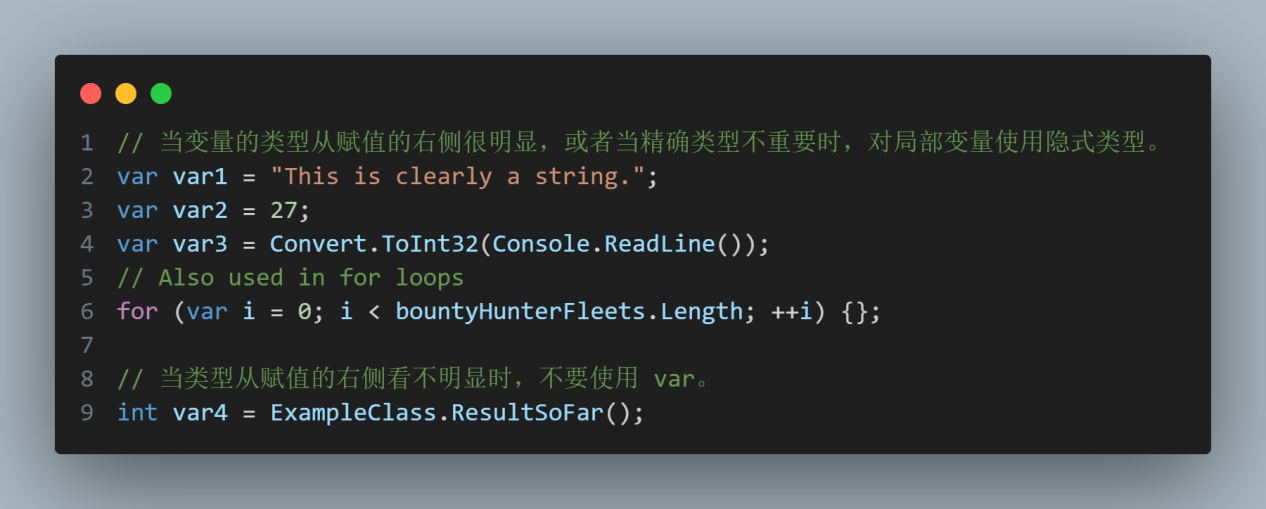


#### **访问级别**

* 首选使用属性**[SerializeField]**，而不是公开变量，只有在必要时才应公开变量。

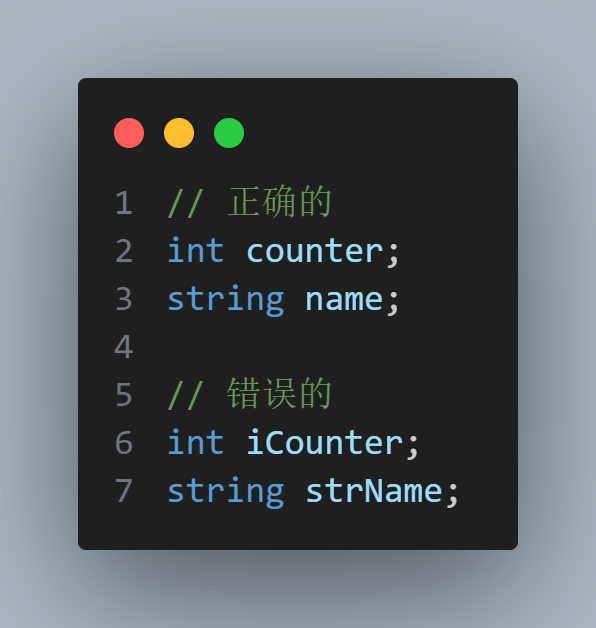
#### **局部变量**

* 局部变量使用camelCase



#### **私有变量**

* 私有变量应具有带下划线的前缀，并使用 camelCase。如：\_myVariable
* 除非已知变量只能在定义它的类中访问，而不应该在子类中访问，否则不要将变量标记为私有变量。
* 请勿在标识符中使用匈牙利符号或任何其他类型标识



#### **可在编辑器访问的变量**

工具提示[Tooltip]

滑条和值的范围[Range(min, max)]

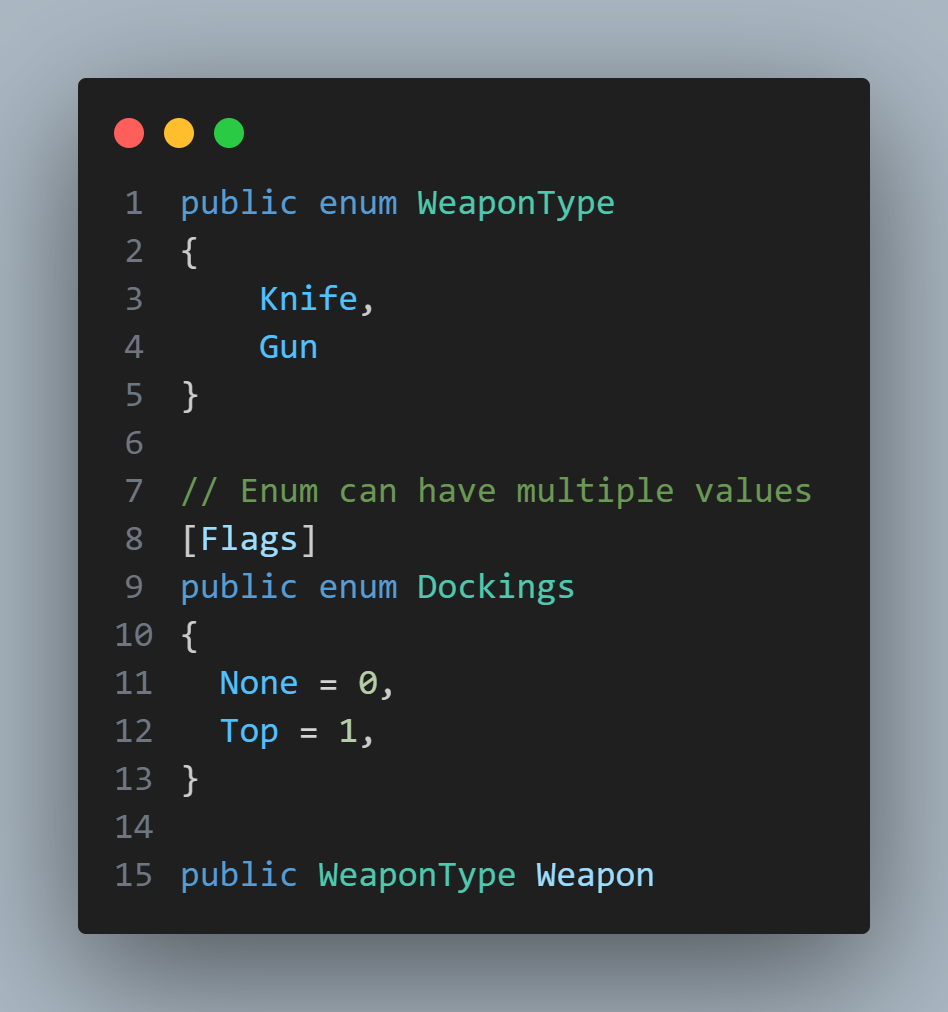
#### **变量类型**

##### 布尔值

* 所有布尔值都应以 PascalCase 命名，但以动词为前缀，如：isDead，isFire
* 如果可能的话，所有布尔值都应该被命名为描述性形容词
* 不要使用布尔值来表示复杂和/或依赖状态。这使得状态添加和删除变得复杂，不再容易阅读。请改用枚举。如：如果武器不能同时装弹和装备，定义一个WeaponState的变量，而不是定义isReloading、isEquipping

##### 枚举

* 枚举使用 PascalCase，并为枚举及其值使用单数名称。
* 例外：“位域”或“位段（Bit field）”枚举应为复数。
* 枚举可以放置在类空间之外，以提供全局访问。



##### 数组

* 数组遵循与上述相同的命名规则，但应以复数名词命名
* 使用 Targets、 Hats、EnemyPlayers
* 而不是TargetList 、HatArray、EnemyPlayerArray 。

##### 接口

* 接口以大写字母I开头，然后以 PascalCase 开头。如：

public interface ICanEat { }

### 函数、事件

* 所有函数都应该是动词

所有函数和事件都执行某种形式的操作，无论是获取信息、计算数据还是导致某些东西爆炸。因此，所有函数都应以动词开头。它们应尽可能用现在时态措辞。他们还应该了解自己在做什么。

如：Fire、Jump

* 返回 Bool 的函数应提出问题

当编写一个不改变或修改任何对象的状态，纯粹是为了获取信息、状态或计算是/否值的函数时，它应该提出一个问题。

如：isDead、isOnFire

* 事件处理程序和调度程序应以On

任何处理事件或调度事件的函数都应以动词规则开头并继续遵循。

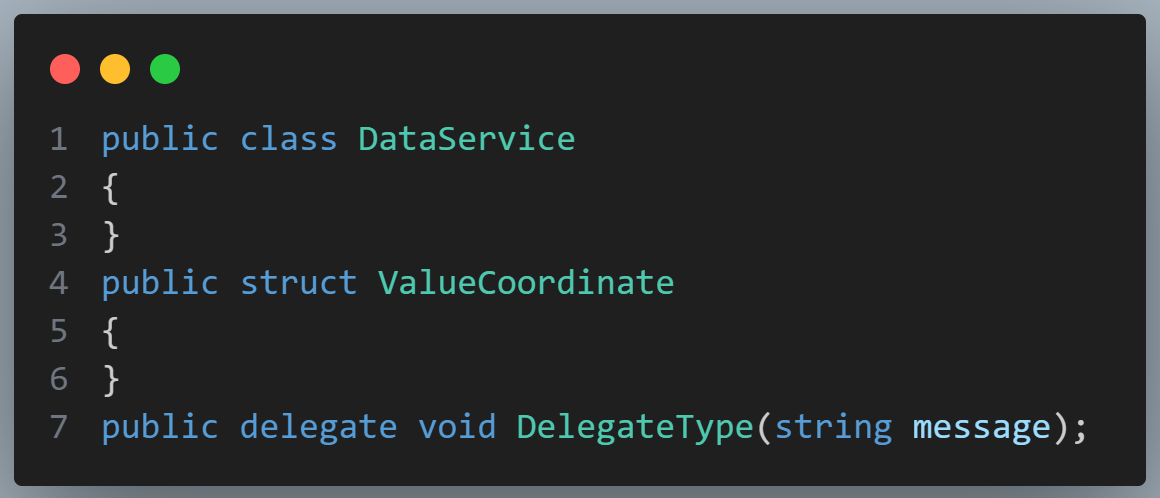
如：OnDeath、OnPickup

### 标识符名称

* 单行语句块可以不使用括号，但该块必须在其自己的行上正确缩进，且不得嵌套在使用括号的其他语句块中。例外：using允许将一个语句嵌套在另一个using语句中，方法是从下一行开始，缩进级别相同；
* 内部和私有字段，使用\_camelCase，并尽可能使用readonly。内部和私有字段，使用前缀\_，静态字段使用s\_，线程静态使用t\_。
* Readonly应位于static之后，如：static readonly；
* 总是指定可见性，可见性应是第一个修饰符：private string \_name；
* 命名空间导入应在文件顶部、声明之外指定， 并namespace应按字母顺序排序，但命名空间除外System.\*，命名空间应放在所有其他命名空间的顶部。
* 任何时候都应避免出现超过一个空行。例如，类型的成员之间不要有两个空行。
* var我们仅当类型在右侧明确命名时才使用，通常是由于new或显式转换
* new()类似地，仅当在变量定义语句或字段定义语句的左侧明确命名类型时，才能使用target-typed 。例如FileStream stream = new(...);，但不能stream = new(...);
* 使用 PascalCasing 命名所有常量局部变量和字段。唯一的例外是互操作代码，其中常量值应与您通过互操作调用的代码的名称和值完全匹配。
* 对所有方法名称（包括本地函数）均使用 PascalCasing
* 公共字段应谨慎使用，使用时应遵循PascalCasing且不带前缀
* 字段应在类型声明的顶部指定，即声明变量应在脚本上部完成
* 使用单语句 if 时，我们遵循以下约定：
  + 切勿使用单行形式（例如if (source == null) throw new ArgumentNullException("source");：）
  + if使用括号始终是可以接受的，并且if/ else if/.../else复合语句的任何使用括号或者单个语句主体跨越多行，则需要使用括号。
  + 仅当与/ /.../复合语句相关的每个块的主体都放在一行上时，才可以省略括号。ifelse ifelse
* 将所有内部(internal )和私有( private)类型设为静态(static)或密封(sealed)，除非需要从它们派生。

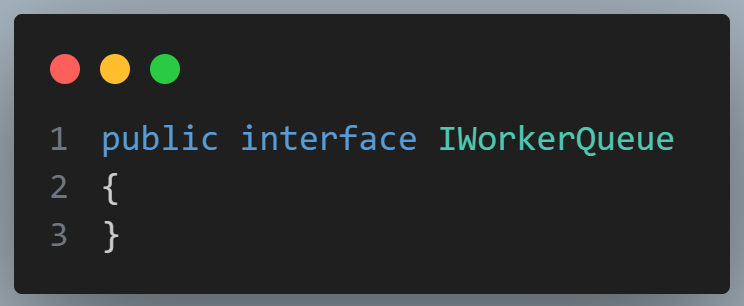
#### Class/struct/delegate

使用 Pascal 大小写(“PascalCasing”)



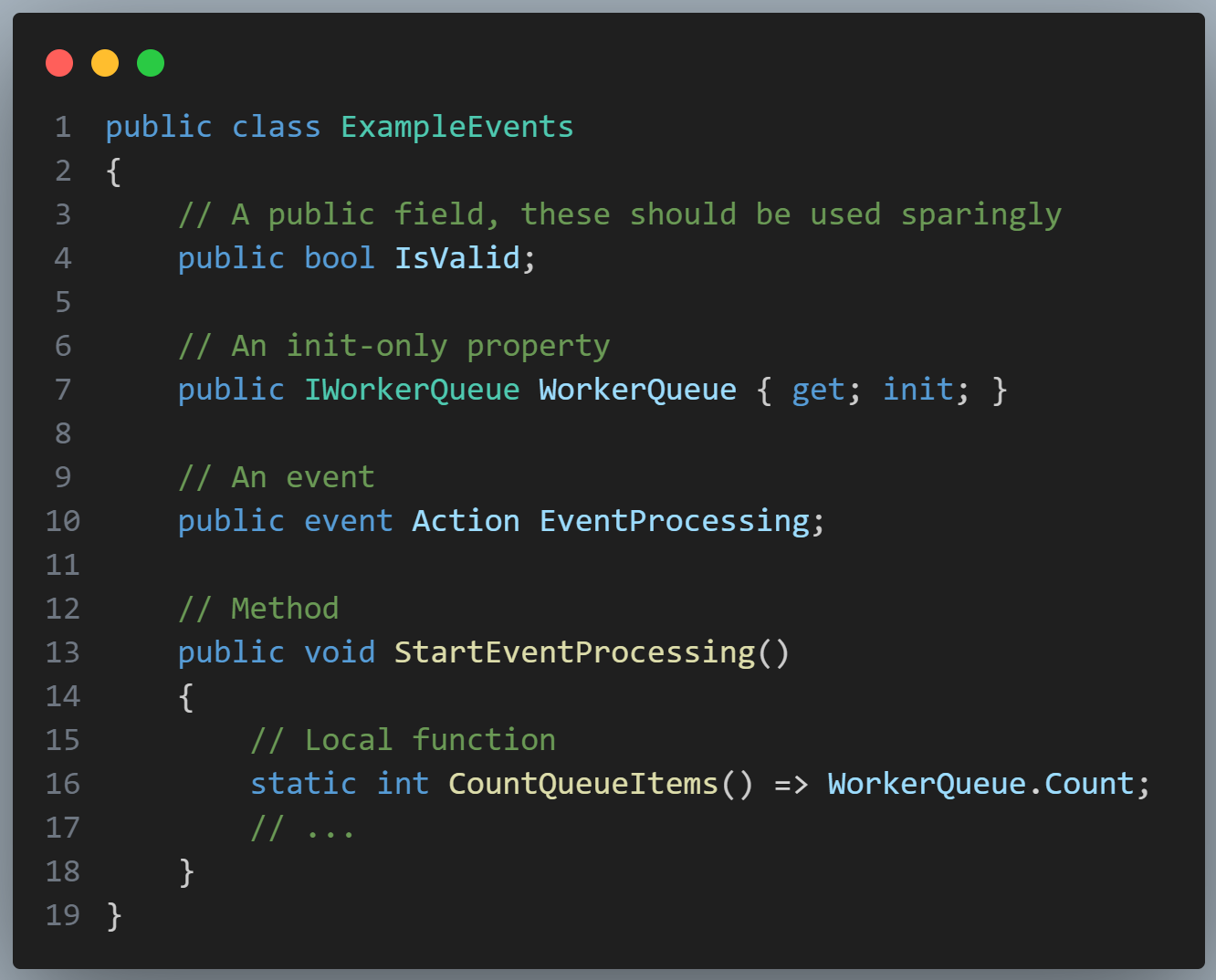
#### Interface

使用 pascal 大小写并在名称前面加上前缀 I



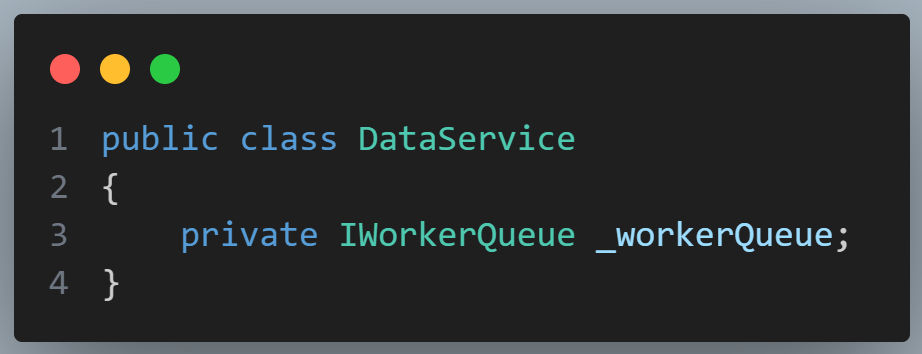
#### Public的命名字段、属性和事件，所有方法和本地函数

使用 Pascal 大小写(“PascalCasing”)



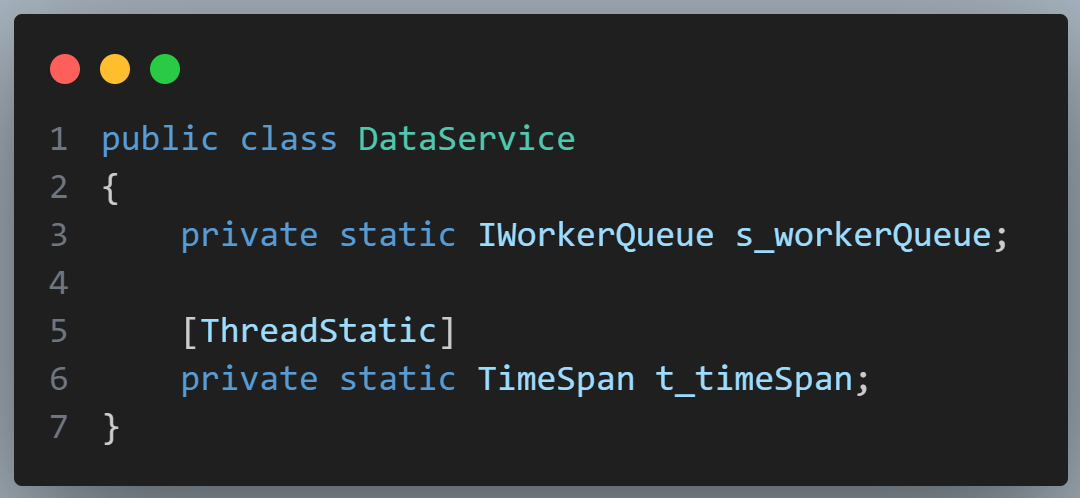
#### Private/internal

使用驼峰式大小写（“camelCasing”），并对它们添加 \_ 作为前缀。 命名局部变量（包括委托类型的实例）时，请使用驼峰式大小写。



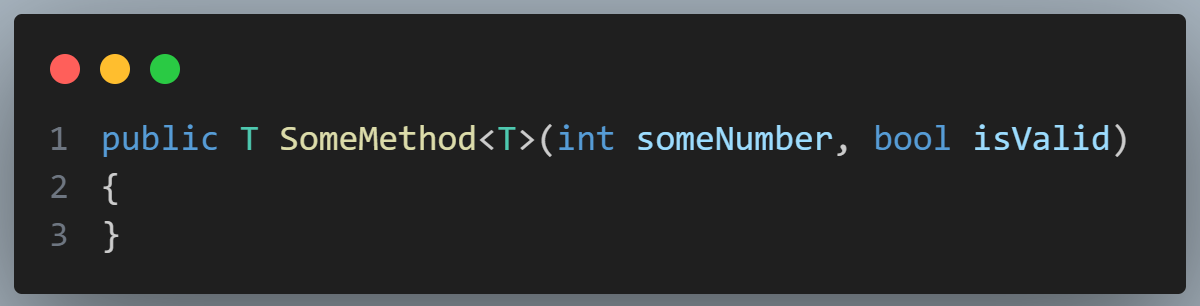
#### private 或 internal 的static 字段

使用 s\_ 前缀，对于线程静态，请使用 t\_



#### 方法参数

使用驼峰式大小写

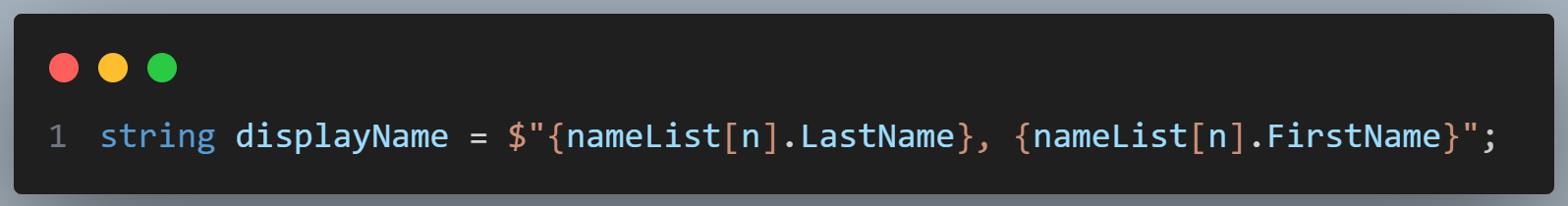


#### 示例文件

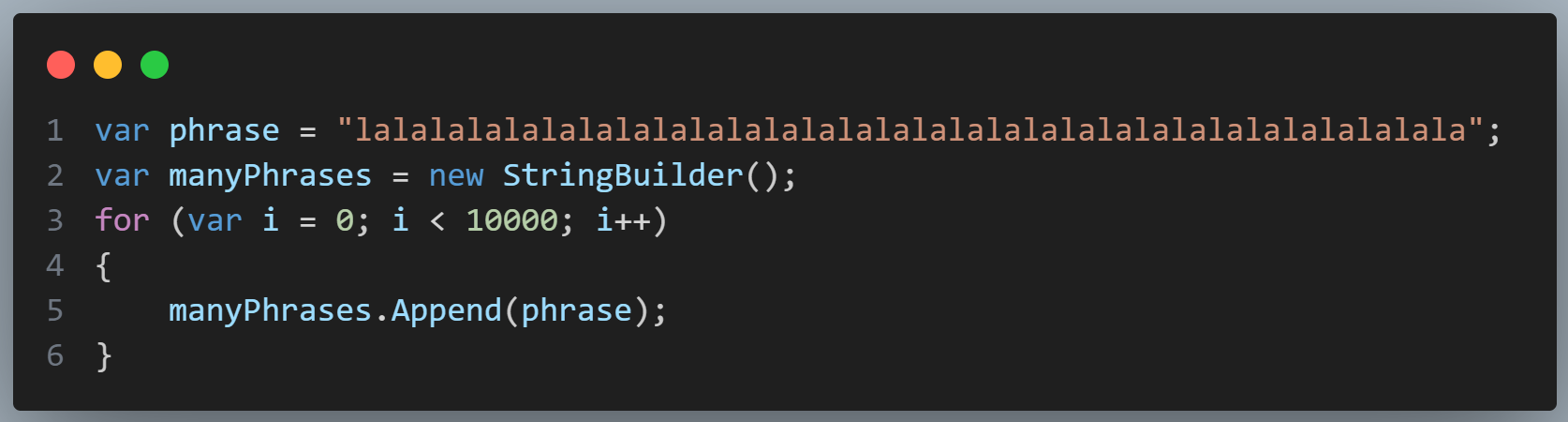
### 编码约定

#### 字符串数据

* 使用字符串内插来连接短字符串

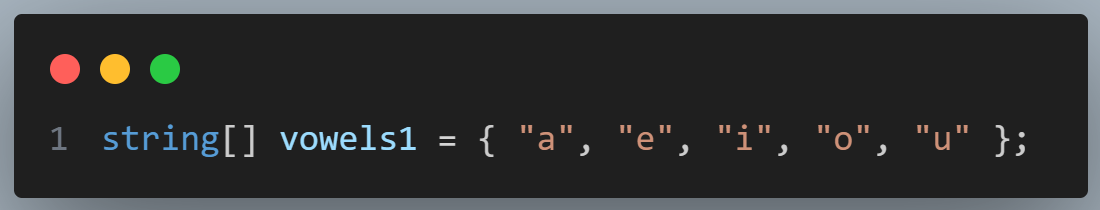


* 若要在循环中追加字符串，尤其是在使用大量文本时，请使用 System.Text.StringBuilder 对象

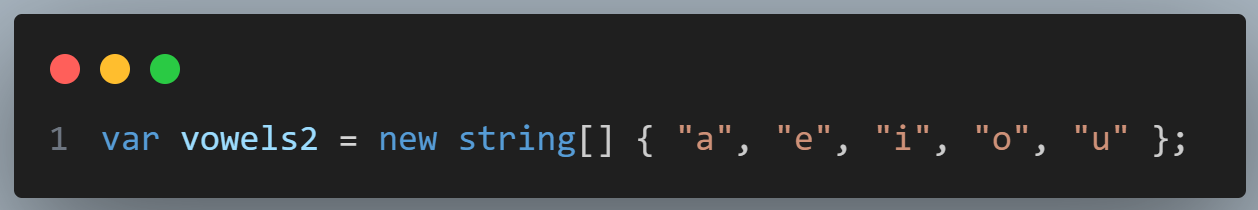


#### 数组

* 当在声明行上初始化数组时，请使用简洁的语法。 在以下示例中，不能使用 var 替代 string[]。

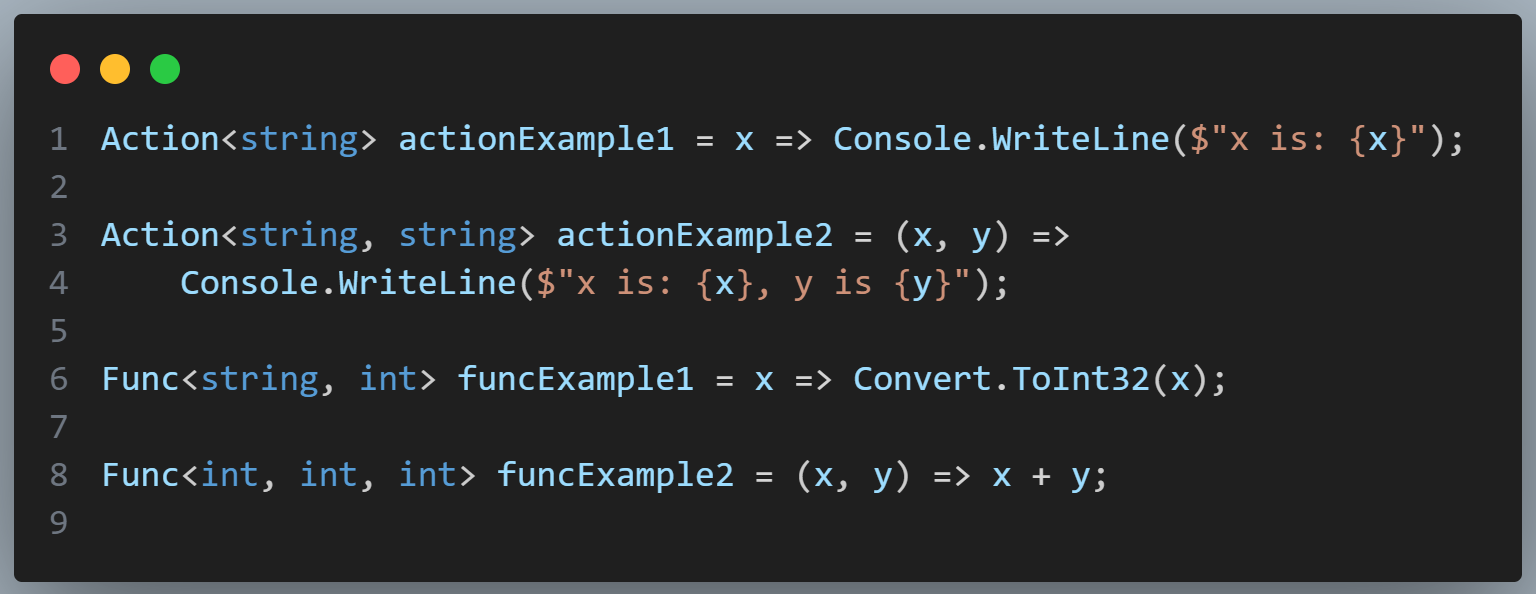


* 如果使用显式实例化，则可以使用 var。

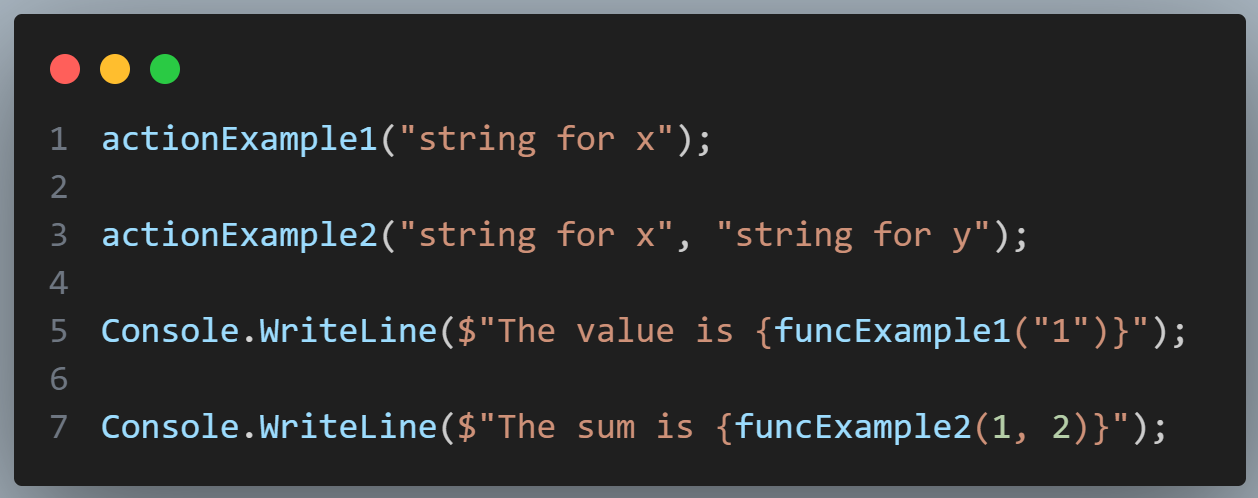


#### 委托

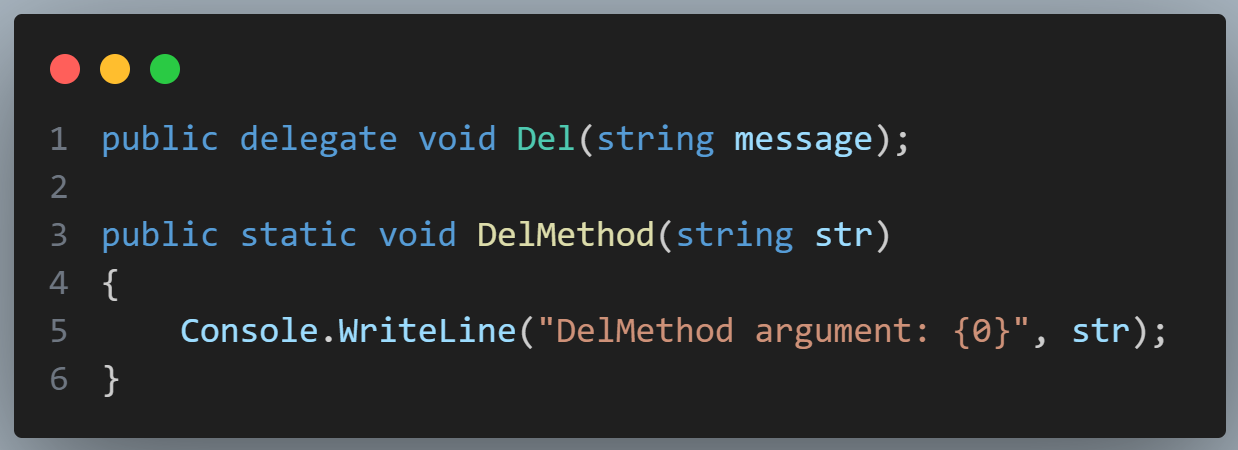
* 使用 Func<> 和 Action<>，而不是定义委托类型。 在类中，定义委托方法。



* 使用 Func<> 或 Action<> 委托定义的签名来调用方法。



* 如果创建委托类型的实例，请使用简洁的语法。 在类中，定义委托类型和具有匹配签名的方法。

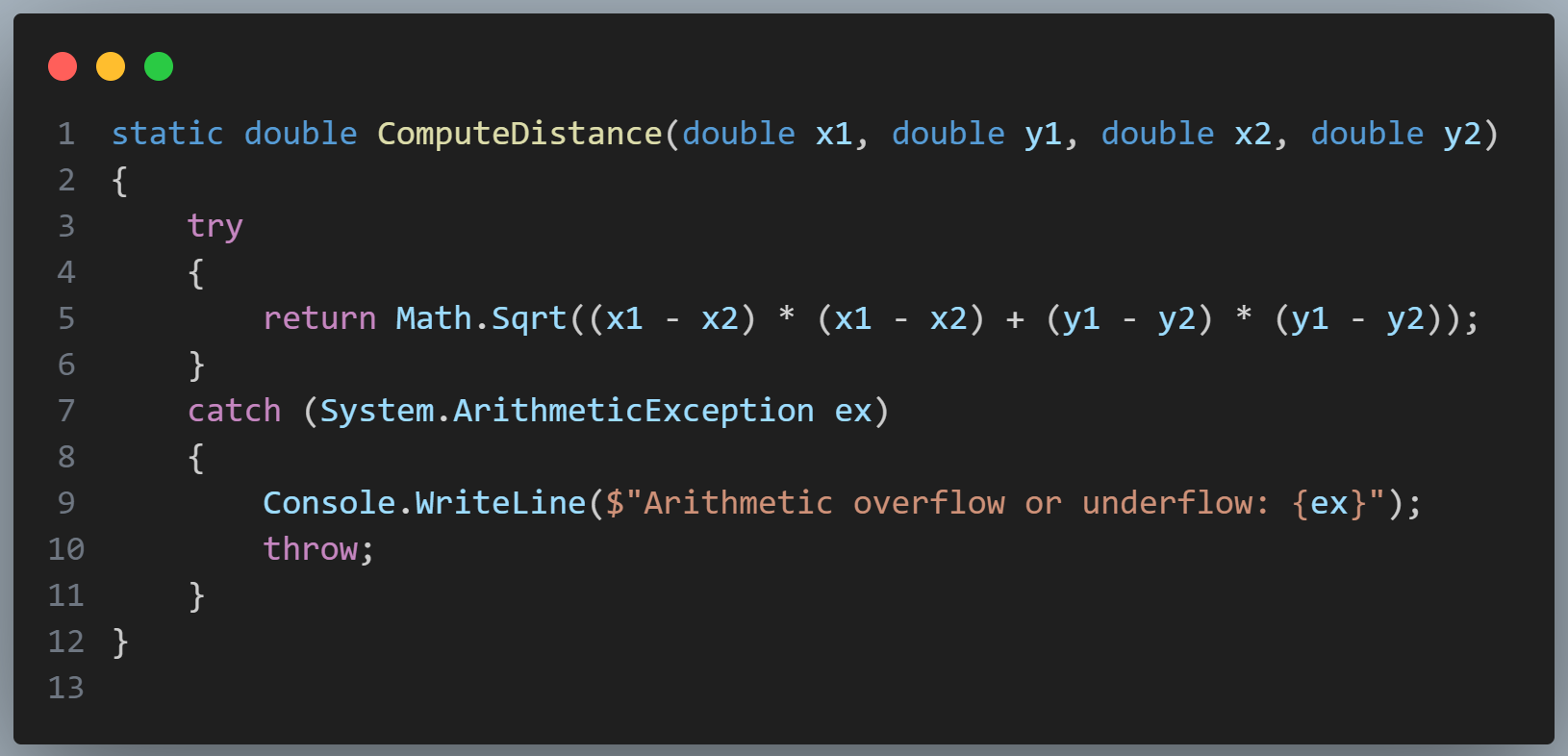


* 创建委托类型的实例，然后调用该实例。

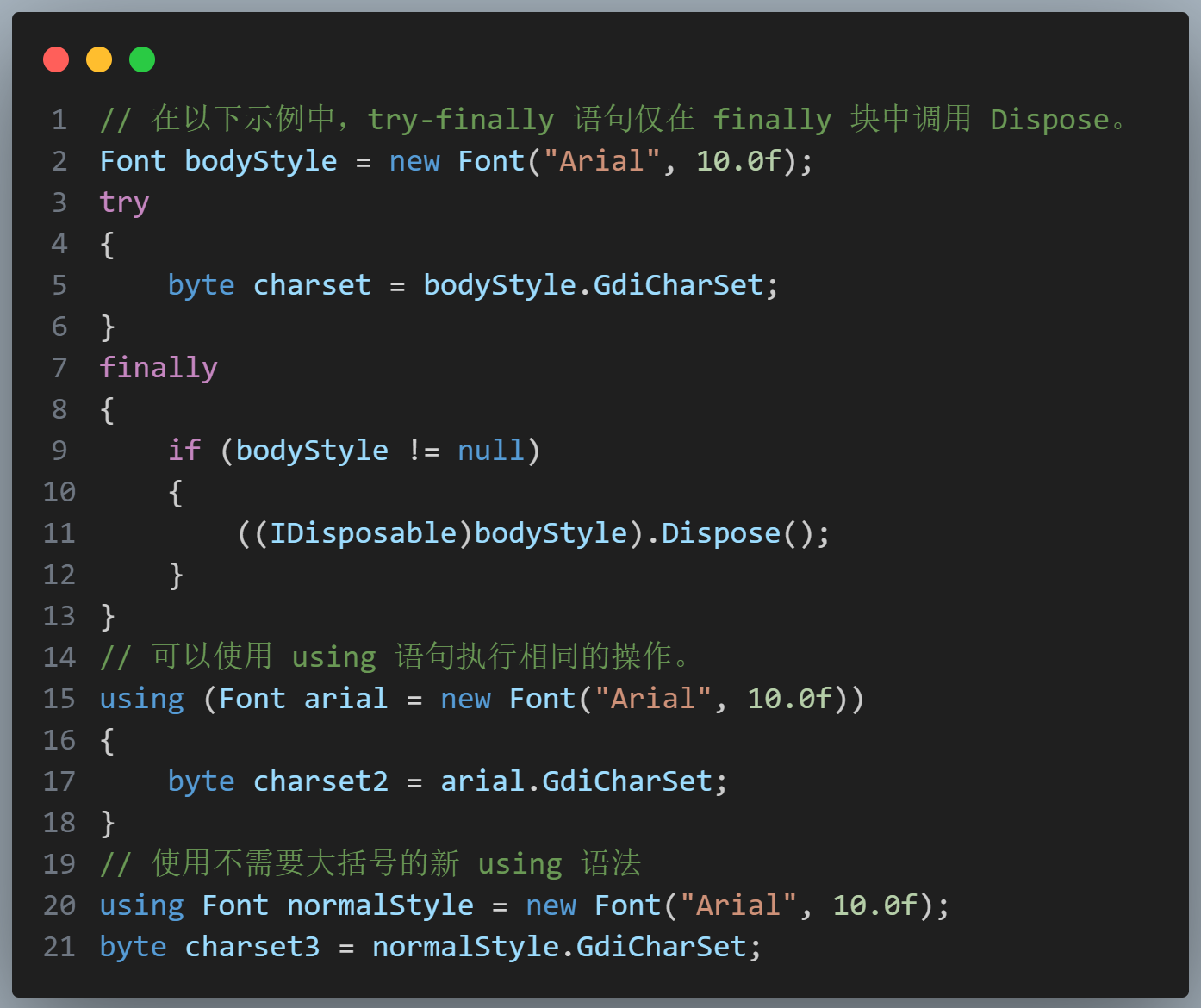


#### Try-catch和using语句处理异常

* 对大多数异常处理使用 try-catch 语句

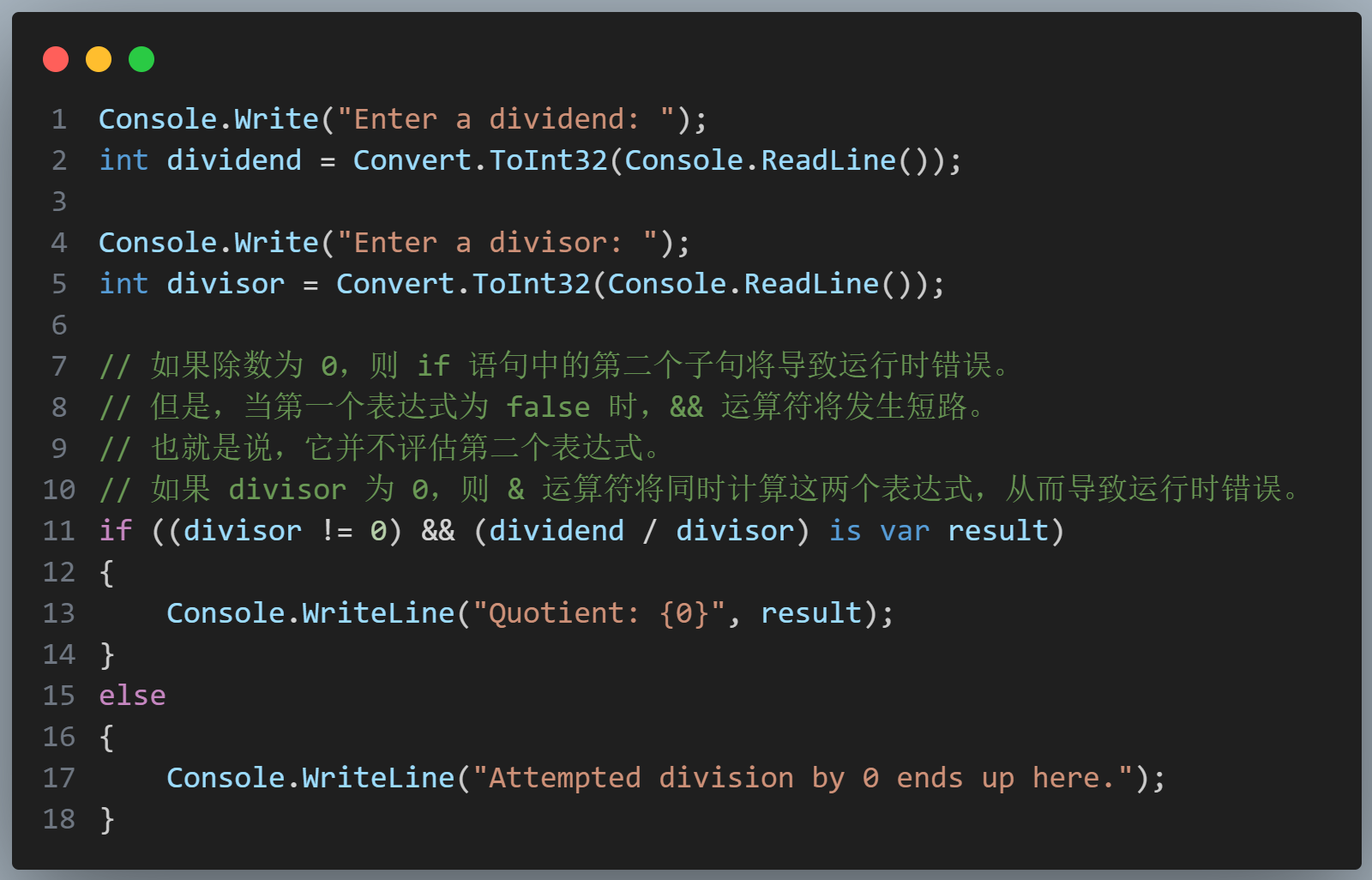


* 通过使用 C# using 语句简化你的代码。 如果具有 try-finally 语句（该语句中 finally 块的唯一代码是对 Dispose 方法的调用），请使用 using 语句代替。



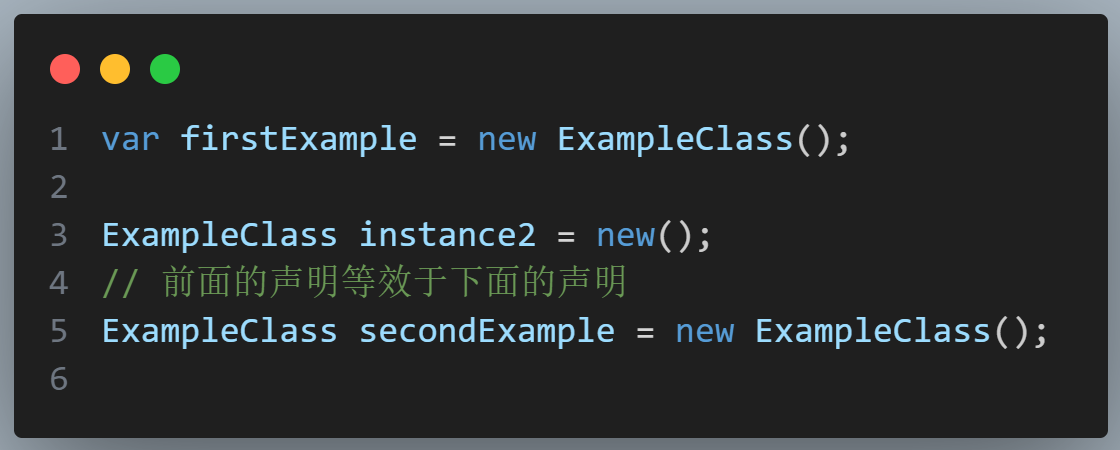
#### && 和 || 运算符

* 在执行比较时，使用 && 而不是 &，使用 || 而不是 |，如以下示例所示。



#### New运算符

* 使用对象实例化的简洁形式之一，如以下声明中所示

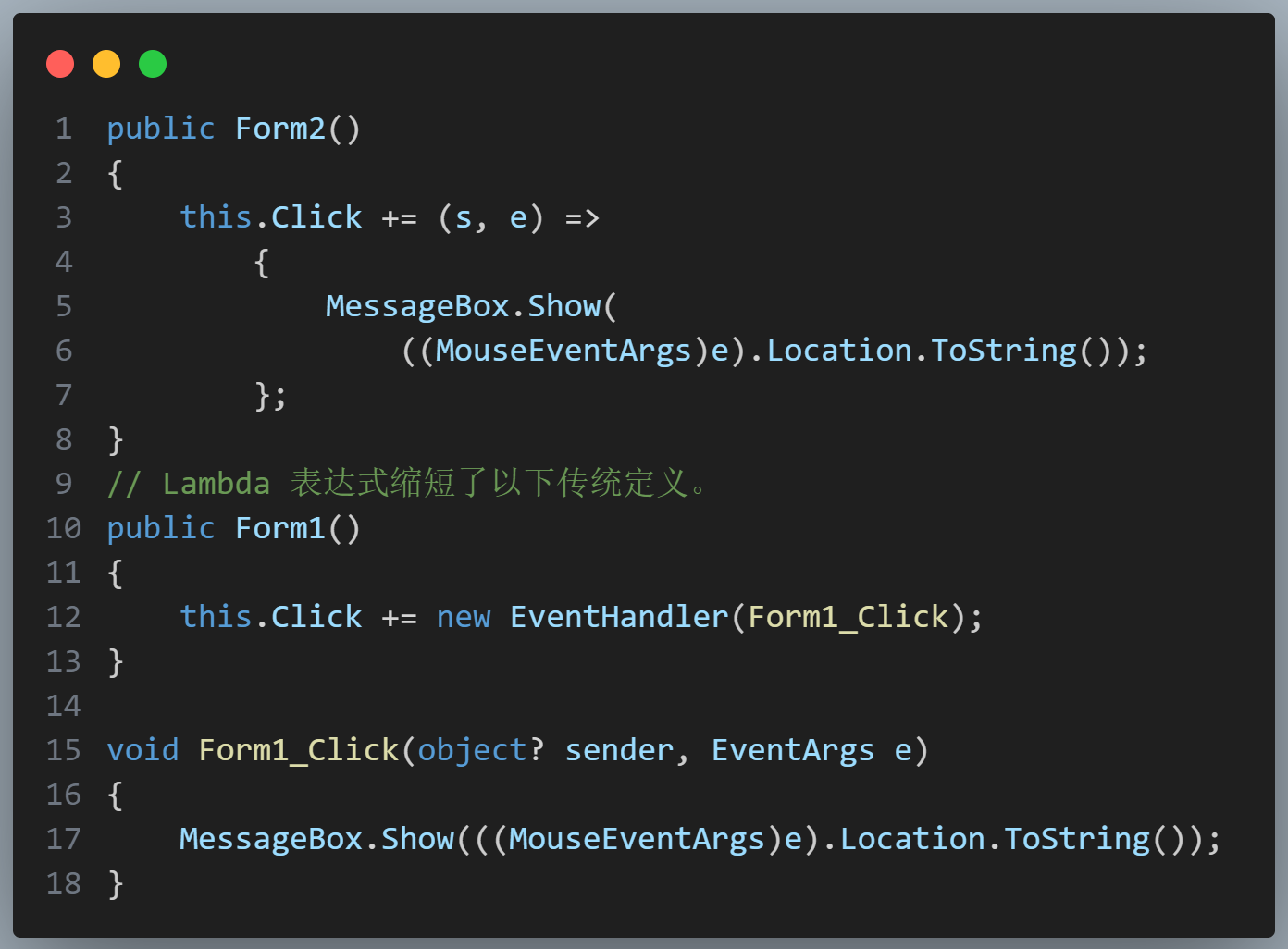


* 使用对象初始值设定项简化对象创建，如以下示例中所示



#### 事件处理

* 使用 lambda 表达式定义稍后无需移除的事件处理程序：

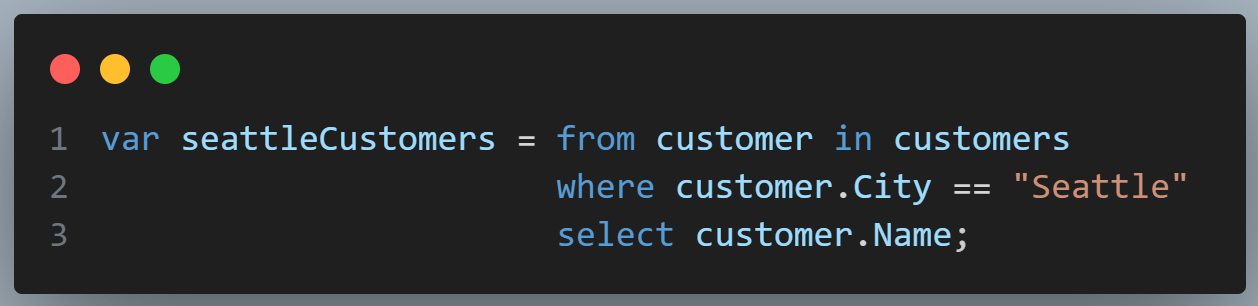


#### 静态成员

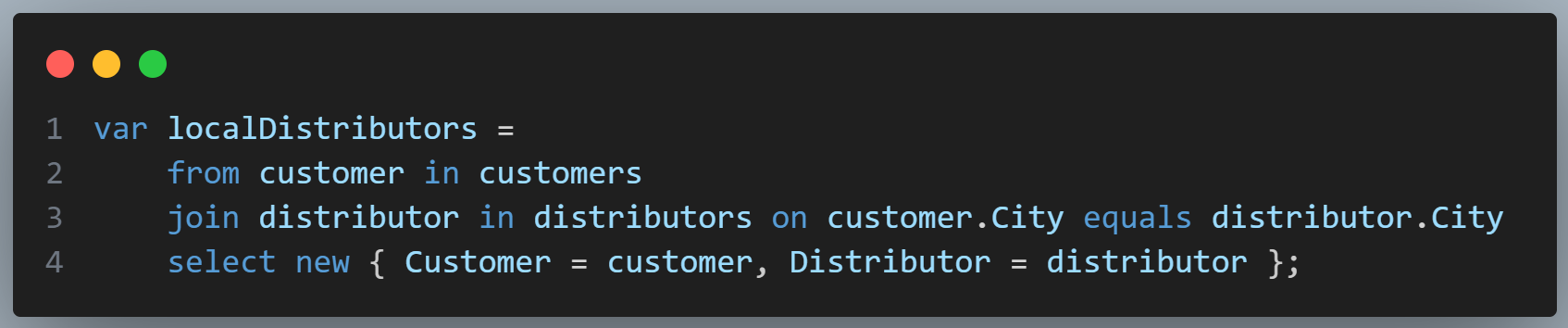
* 使用类名调用 static 成员：ClassName.StaticMember。
* 这种做法通过明确静态访问使代码更易于阅读。
* 请勿使用派生类的名称来限定基类中定义的静态成员。
* 编译该代码时，代码可读性具有误导性，如果向派生类添加具有相同名称的静态成员，代码可能会被破坏。

#### LINQ查询

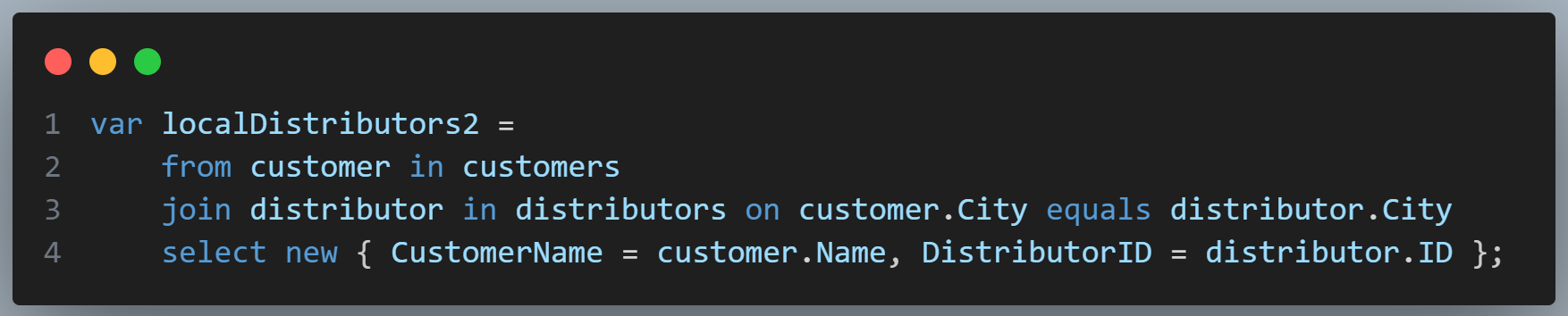
* 对查询变量使用有意义的名称。 下面的示例为位于西雅图的客户使用 seattleCustomers。



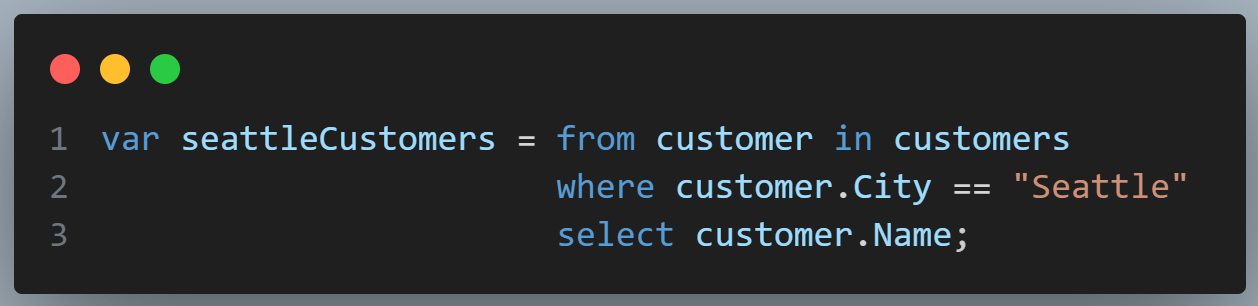
* 使用别名确保匿名类型的属性名称都使用 Pascal 大小写格式正确大写。



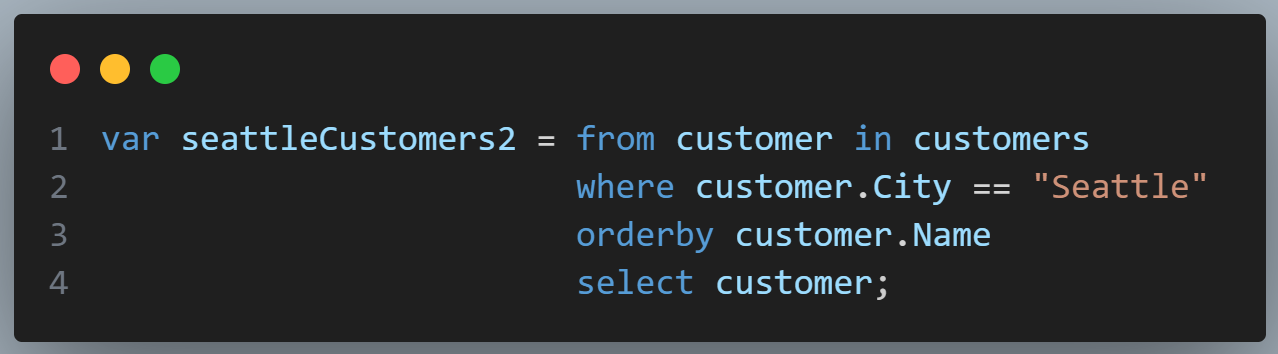
* 如果结果中的属性名称模棱两可，请对属性重命名。 例如，如果你的查询返回客户名称和分销商 ID，而不是在结果中将它们保留为 Name 和 ID，请对它们进行重命名以明确 Name 是客户的名称，ID 是分销商的 ID。



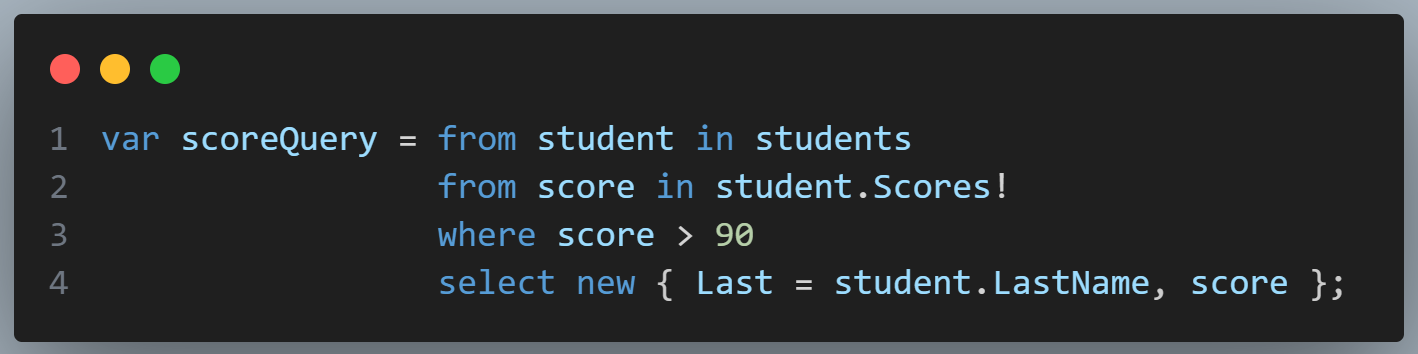
* 在查询变量和范围变量的声明中使用隐式类型化。 有关 LINQ 查询中隐式类型的本指导会替代适用于隐式类型本地变量的一般规则。 LINQ 查询通常使用创建匿名类型的投影。 其他查询表达式使用嵌套泛型类型创建结果。 隐式类型变量通常更具可读性。



* 对齐 from 子句下的查询子句，如上面的示例所示。
* 在其他查询子句前面使用 where 子句，确保后面的查询子句作用于经过缩减和筛选的一组数据。

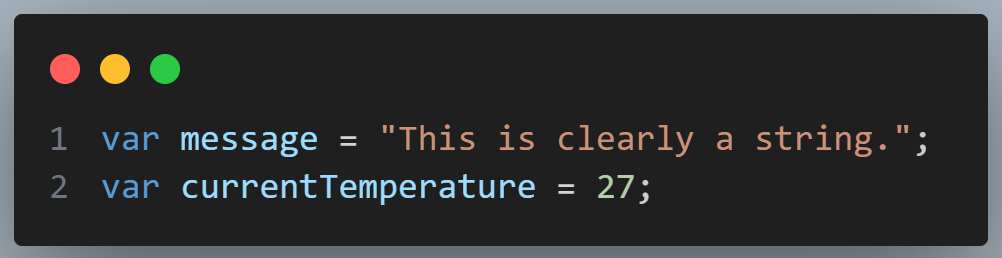


* 使用多行 from 子句代替 join 子句来访问内部集合。 例如，Student 对象的集合可能包含测验分数的集合。 当执行以下查询时，它返回高于 90 的分数，并返回得到该分数的学生的姓氏。

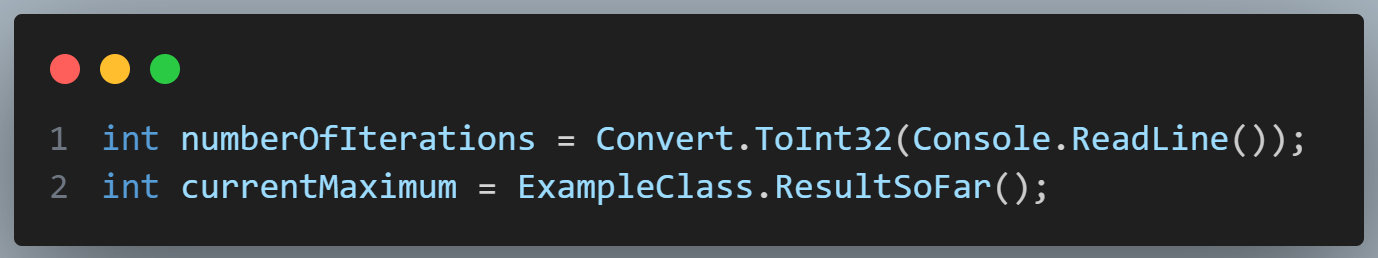


#### 隐式类型本地变量

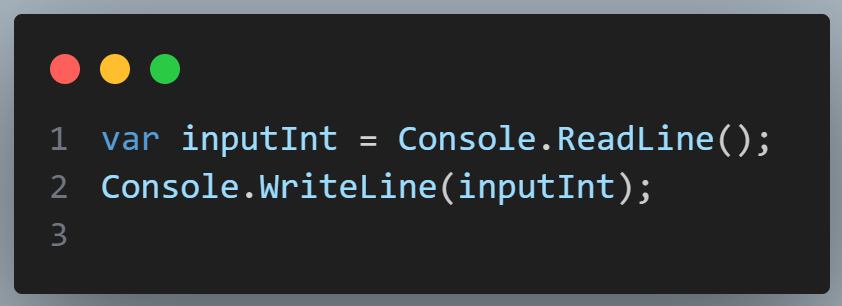
* 当变量的类型在赋值右侧比较明显时，对局部变量使用隐式类型。



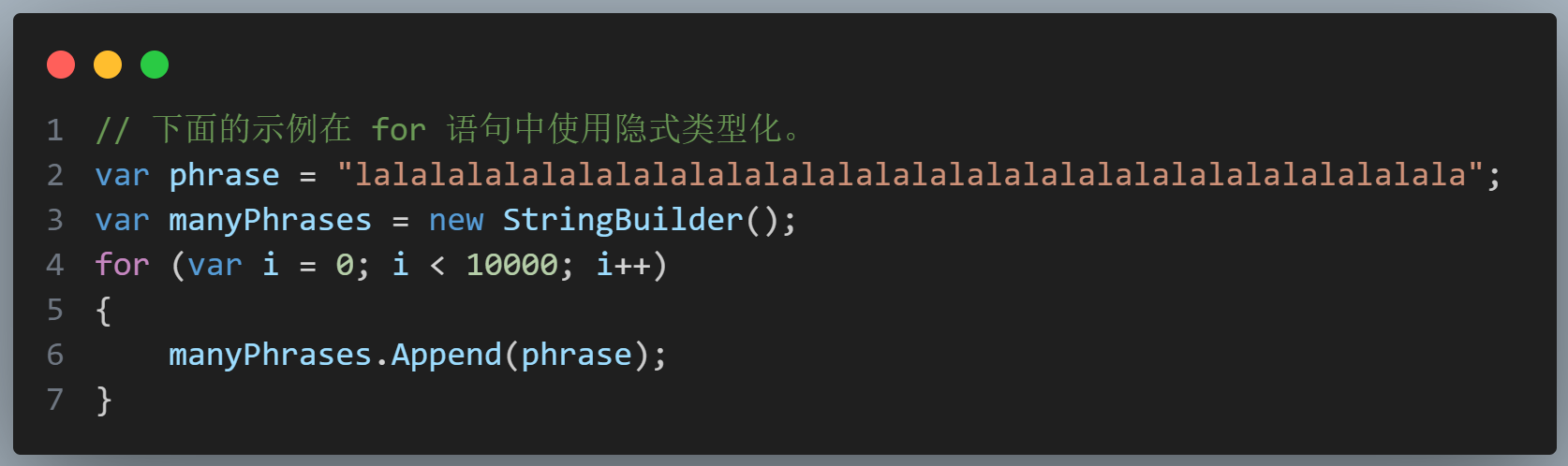
* 当类型在赋值右侧不明显时，请勿使用 var。 请勿假设类型明显来自方法名称。 如果变量类型是 new 运算符、对文本值的显式强制转换或赋值，则将其视为明确的变量类型。



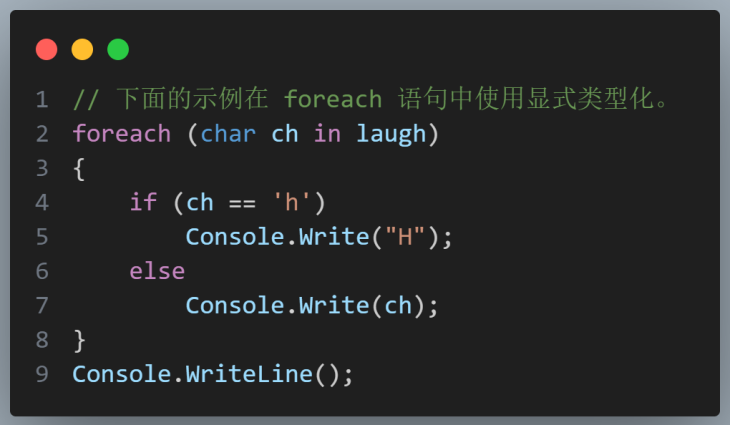
* 不要使用变量名称指定变量的类型。 它可能不正确。 请改用类型来指定类型，并使用变量名称来指示变量的语义信息。 以下示例应对类型使用 string，并使用类似 iterations 的内容指示从控制台读取的信息的含义。



* 避免使用 var 来代替 dynamic。 如果想要进行运行时类型推理，请使用 dynamic
* 在 for 循环中对循环变量使用隐式类型。



* 不要使用隐式类型化来确定 foreach 循环中循环变量的类型。 在大多数情况下，集合中的元素类型并不明显。 不应仅依靠集合的名称来推断其元素的类型。



* 对 LINQ 查询中的结果序列使用隐式类型。

#### 将using指令放在命名空间声明之外

* 当 using 指令位于命名空间声明之外时，该导入的命名空间是其完全限定的名称。 完全限定的名称更加清晰。 如果 using 指令位于命名空间内部，则它可以是相对于该命名空间的，也可以是它的完全限定名称。



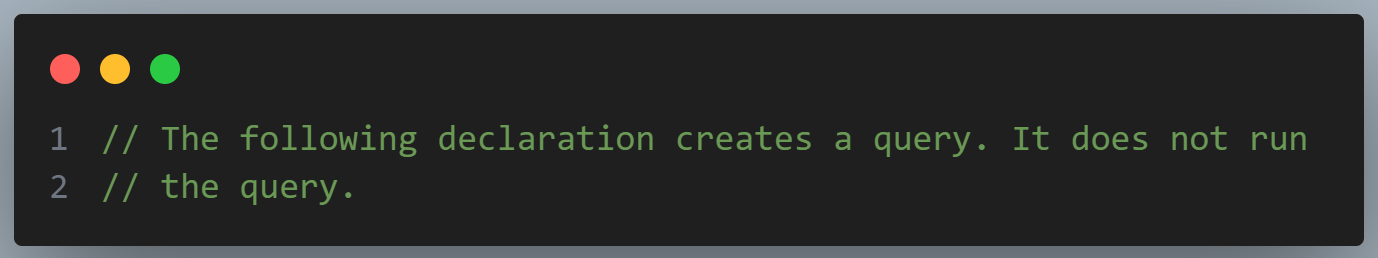
#### 样式指南

一般情况下，对代码示例使用以下格式：

* 使用四个空格缩进。 不要使用选项卡。
* 一致地对齐代码以提高可读性。
* 将行限制为 65 个字符，以增强文档上的代码可读性，尤其是在移动屏幕上。
* 将长语句分解为多行以提高清晰度。
* 对大括号使用“Allman”样式：左和右大括号另起一行。 大括号与当前缩进级别对齐。
* 如有必要，应在二进制运算符之前换行。

#### 注释样式

* 使用单行注释（//）以进行简要说明。
* 避免使用多行注释（/\* \*/）来进行较长的解释。 注释不进行本地化处理。 相反，配套文章中提供了较长的解释。
* 若要描述方法、类、字段和所有公共成员，请使用 XML 注释。
* 将注释放在单独的行上，而非代码行的末尾。
* 以大写字母开始注释文本，在注释分隔符 (//) 与注释文本之间插入一个空格，以句点结束注释文本，如下面的示例所示。



* 多行注释示例：



#### 布局约定

好的布局利用格式设置来强调代码的结构并使代码更便于阅读。

* 使用默认的代码编辑器设置（智能缩进、4 字符缩进、制表符保存为空格）。 有关详细信息，请参阅选项、文本编辑器、C#、格式设置。
* 每行只写一条语句。
* 每行只写一个声明。
* 如果连续行未自动缩进，请将它们缩进一个制表符位（四个空格）。
* 在方法定义与属性定义之间添加至少一个空白行。
* 使用括号突出表达式中的子句，如下面的代码所示。

