类型(Class)的各种成员:

1.常量(const)。常量总与类型关联，不与类型的实例关联。常量在逻辑上总是静态成员。

2.字段。静态字段被认为是类型状态，实例字段被认为是对象状态。

3.实例构造器。

4.类型构造器。

5.方法。方法是更改或查询类型或对象状态的函数，作用于类型称为静态方法，作用于对象称为实例方法。

6.操作符重载。操作符重载实际是方法，定义了当操作符作用于对象时，应该如何操作该对象。不是CLS（公共语言规范）的一部分。

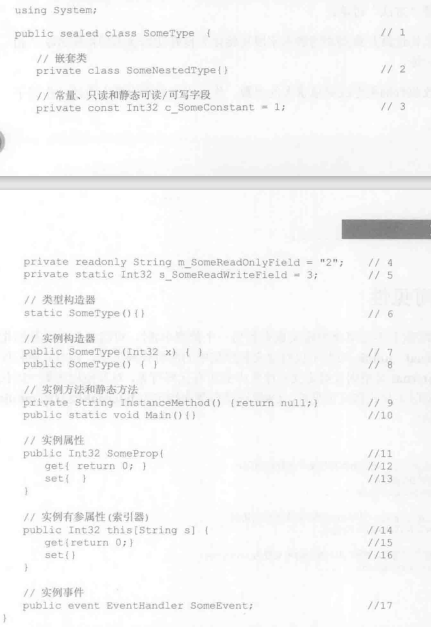
7.转换操作符。转换操作符是定义如何隐式或显式将对象从一种类型转型为另一种类型的方法。不是CLS的一部分。

8.属性。属性允许用简单的字段风格的语法设置或查询类型或对象的逻辑状态，同时保证状态不被破坏。作用于类型称为静态属性，作用于对象称为实例属性，属性可以无参也可以有多个参数。

9.事件。静态事件允许一个类型向一个或多个静态或实例方法发送通知；实例事件允许对象向一个或多个静态或实例方法发送通知。引发事件通常是为了响应提供事件的类型或对象的状态的改变。

10.类型。类型可以定义其他嵌套类型。通常用这个办法将大的、复杂的类型分解成更小的构建单元以简化实现。

这是一个包含以上所有可能成员的类：



图片包含 图示

描述已自动生成

PS：什么是类型状态和对象状态？

"类型状态"和"字段状态"是两个相关的概念，用于描述对象或数据的状态。

类型状态（Type State）：

类型状态是指一个对象或数据所具有的特定状态，这个状态是由对象的类型决定的。

类型状态可以包括对象的属性、方法、行为等，它们定义了对象在不同状态下的行为和可用操作。

类型状态可以通过对象的类型来确定，不同类型的对象可能具有不同的类型状态。

字段状态（Field State）：

字段状态是指对象或数据中特定字段的值所处的状态。

字段状态描述了对象或数据在某一时刻的具体属性或特征。

字段状态可以随着对象的操作和改变而变化，它反映了对象在不同时间点的不同属性值。

举个例子

考虑一个名为"Car"的类，它有一个字段"speed"表示汽车的速度。在这种情况下：

类型状态：Car类定义了一些方法和属性，如"加速"、"刹车"、"获取速度"等。这些方法和属性定义了Car对象的类型状态，即Car对象可以执行的操作和具备的行为。

字段状态：Car类中的字段"speed"表示汽车的速度。该字段的值可以随着汽车的加速或刹车而改变，反映了汽车在不同时间点的速度属性。

例如，假设有一个Car对象car1，它的初始速度为0。在这种情况下：

类型状态：Car类定义了加速和刹车的方法，以及获取速度的属性。这些方法和属性定义了Car对象的类型状态，即car1对象可以调用加速、刹车和获取速度的操作。

字段状态：car1对象的字段"speed"的初始值为0，表示它的速度是0。如果调用car1的加速方法，"speed"字段的值会增加，反映了car1对象的速度属性的变化。

类型的可见性

Internal访问修饰符：当将internal修饰符应用于类型或成员时，它表示它们只能在同一程序集内访问，而对于其他程序集是不可见的。

Internal访问修饰符的用法：

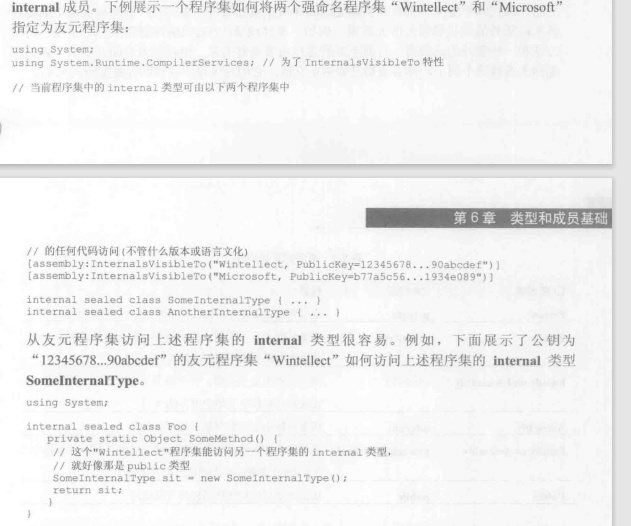
访问级别：internal修饰符指定了类型或成员的访问级别为程序集级别。这意味着只有在同一程序集内的代码才能访问被internal修饰的类型或成员。

默认访问级别：如果不显式指定访问修饰符，类和成员的默认访问级别是internal。这意味着如果没有指定访问修饰符，类和成员将只能在同一程序集内访问。

程序集访问：internal修饰符适用于整个程序集，而不仅仅是单个类或成员。这意味着在同一程序集内的所有类型和成员都可以相互访问。

跨程序集访问（友元程序集）：如果您想从另一个程序集中访问internal类型或成员，可以使用InternalsVisibleTo特性来授权访问。通过在源程序集中添加[assembly: InternalsVisibleTo("OtherAssembly")]，其中"OtherAssembly"是目标程序集的名称，您可以允许目标程序集访问源程序集中的internal类型或成员。

跨程序集访问的例子：



成员的可访问性：

表格

描述已自动生成

CLR允许放宽但不允许收紧成员的可访问性限制。例如，类可重写基类定义的protected方法，将重写方法设为public(放宽限制)，但不能重写基类定义的protected方法，将重写方法设为private方法（收紧限制）。

静态类

C#编译器对静态类的限制：

1.静态类必须直接从Object类派生。继承只适用于类型对象，静态类无法实例化。

2.静态类不能实现任何接口。因为只要使用类的实例才能调用类的接口方法。

3.静态类只能定义静态成员。

4.静态类不能作为字段、方法参数或者局部变量使用，因为这些都代表引用了实例的变量。

例子：

文本, 信件

描述已自动生成

文本

描述已自动生成

分部类、结构和接口

1.Partial关键字。

2.类、结构或接口的定义源代码可以分散到一个或多个源代码文件中。

3.分部类功能完全由C#编译器实现，CLR对该功能一无所知（一个类型的所有源代码文件必须作为一个编译单元编译到一起）

使用分部类的好处：

1.源代码控制。Partial关键字可将类型的代码分散到多个源代码文件中，每个文件都可以单独签出，多个程序员可以同时编辑该类型。

2.在同一个文件中将类或结构分解成不同的逻辑单元。

3.代码拆分。

组件、多态和版本控制：

PS：OOP面向对象编程、CSP组件软件编程和CBP面向组件编程的异同

OOP：

OOP的核心思想是将数据和操作数据的方法封装在一起，形成对象。对象之间通过消息传递进行交互。OOP强调的是系统中对象的种类、职责以及相互关系。

CSP：

CSP是一种并发编程模型，它用于描述两个独立的并发实体通过共享的通讯channel进行通信。CSP模型的关键是关注channel，而不关注发送消息的实体。

CBP：

CBP是一种软件设计方法，它强调将软件系统分解为可重用的组件。每个组件都是一个独立的模块，具有明确定义的接口和行为。组件之间通过接口进行通信和协作，以实现更大的系统功能。

组件的一些特点：

文本

描述已自动生成

.NET Framework中的版本号含义和分类：

文本

描述已自动生成

版本控制：（需要一个直观的例子）

文本

描述已自动生成

C#提供了5个可以影响组件版本控制的关键字

表格

描述已自动生成

CLR如何调用虚方法、属性和事件：

CLR提供两个方法调用指令：

1.Call指令

Call指令可以调用静态方法、实例方法和虚方法。用call指令调用静态方法必须指定方法的定义类型。用call指令调用实例方法或虚方法必须指定引用了对象的变量。Call指令经常用于以非虚方式调用虚方法。

这是一个直观的例子：

（关于“用call指令调用静态方法必须指定方法的定义类型的含义，调用实例方法必须指定引用了对象的变量”）

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

（关于“用call指令用非虚方式调用虚方法”）

图形用户界面, 文本, 应用程序

描述已自动生成

Ps：以虚方式调用虚方法需要使用类型的实例。当你使用一个对象来调用虚方法时，程序会在运行时根据对象的实际类型来确定调用哪个方法。这意味着，如果对象是派生类类型，那么程序会调用派生类中的方法；如果对象是基类类型，那么程序会调用基类中的方法。

下面是这段话对应的例子：

图片包含 文本

描述已自动生成

2.callvirt指令

Callvirt指令可调用实例方法和虚方法，不能调用静态方法。用callvirt方法调用实例方法或虚方法必须指定引用了对象的变量。用callvirt方法调用非虚实例方法（没有被标记为 virtual、abstract 或 override），变量的类型指明了方法的定义类型。用callvirt方法调用虚实例方法，CLR调查发出调用的对象的实际类型，然后以多态方式调用方法。

Ps：

1.call指令假定对象不为null，而callvirt指令需要检查对象是否为空。因此callvirt指令的执行速度稍慢于call指令。

2.什么是非虚实例方法，和虚实例方法的区别

非虚实例方法和实例方法的区别在于它们的继承和重写行为。实例方法是指在类中定义的方法，它可以访问类的实例成员。非虚实例方法是指在类中定义的实例方法，它没有被标记为 virtual、abstract 或 override。

非虚实例方法不能被派生类重写。这意味着，当你在派生类中定义一个与基类中非虚实例方法同名的方法时，你实际上是在隐藏基类中的方法，而不是重写它。当你使用基类类型的变量来调用这个方法时，程序会调用基类中的方法，而不是派生类中的方法。

相反，虚实例方法可以被派生类重写。这意味着，当你在派生类中定义一个与基类中虚实例方法同名且带有 override 关键字的方法时，你实际上是在重写基类中的方法。当你使用基类类型的变量来调用这个方法时，程序会根据对象的实际类型来确定调用哪个方法。

C#语言生成的IL代码何时调用call指令和callvirt指令

当我们调用一个静态方法或以非虚方式调用虚实例方法时，编译器会生成 call 指令。这种指令会直接调用指定的方法，不会进行任何多态处理。当我们以虚方式调用虚实例方法或者非虚实例方法时，编译器会生成 callvirt 指令。

以下是例子：

图片包含 文本

描述已自动生成

（C#开发团队认为）JIT编译器保证安全性，所以会生成代码验证发出调用的对象是否为空，若为空则抛出异常，因为这个检查的存在，所以调用非虚实例方法时使用callvirt指令而非call指令。

文本

描述已自动生成

这里不能用虚方式调用虚方法，因为虚方式调用虚方法是运行时动态绑定类型对象，而这里是递归的，这样会导致栈溢出。所以使用call指令在编译时就进行绑定（静态绑定）。

将方法定义为非虚方法后，将来永远都不要把它改为虚方法。（P147的“重要提示”）

如果希望方法是多态的，使复杂方法为虚方法，而其他重载为非虚方法（P147、148）

P150-154

1.可以通过在返回类型前加上“new”来防止隐藏基类中同名方法，使得编译器认为这是两个毫无关联的方法。

2.可以通过在返回类型前加上“override”让编译器知道该方法和基类中虚实例方法是相关的。