定制特性的概念：将一些附加信息与某个目标元素关联起来的方式，编译器在托管模块的元数据中生成（嵌入）这些额外的信息。

Ps：但编译器只是检测源代码中的特性并生成对应的元数据，不会进行额外的操作。

定制特性的作用：利用定制特性可以“宣告式地”为自己的代码构造添加注解来实现特殊功能。（这种可扩展的元数据信息能在运行时查询，从而动态改变代码的执行方式）

特性Attribute的使用方法：将特性放在目标元素前，放置在方括号中。

下面是书上的例子：

文本

描述已自动生成

本例中可以见到，特性可以用于类，可以用于字段，还可以用于方法。

Ps：使用方括号装载特性仅仅是C#的语法，其他面向CLR的语言不一定使用方括号，比如Basic使用尖括号装载特性

CLR允许将特性应用于可在文件的元数据中表示的几乎任何东西，但常用的特性都是用于元数据定义表中的记录项。C#只允许将特性应用于定义以下任何目标元素的源代码：程序集、模块、类型（类、结构体、枚举、接口、委托）、字段、方法（含构造器）、方法参数、方法返回值、属性、事件和泛型类型参数。

应用特性时，C#允许用一个前缀明确指定特性要应用于的目标元素，下面是书上的例子：

图片包含 文本

描述已自动生成

特性如何用于泛型参数——使用泛型约束，例子如下：

图形用户界面, 文本, 应用程序

描述已自动生成

在上面的例子中定义了一个名为 MyAttribute 的特性类，它继承自 System.Attribute 类。然后，我们在 MyGenericClass 类的定义中使用了 where T : MyAttribute 来指定泛型类型参数 T 必须具有 MyAttribute 特性。

Ps：上例中的特性使用了前缀，标明了特性要应用于的目标元素。

大部分时候特性并不需要指定前缀，除了以下两种情况，下面两种情况需要在使用特性时指定完整的类名称，包括 Attribute 后缀：

1.当程序中存在同名但不同命名空间的两个特性类时，需要使用完整的类名称（包括命名空间和 Attribute 后缀）来消除歧义。

2.当您使用反射来获取特性信息时，需要使用完整的类名称（包括 Attribute 后缀）来获取特定类型的特性。

Ps：特性类的名称通常以 Attribute 结尾。当使用特性时，可以省略 Attribute 后缀。

定制特性类是如何工作的：

当特性被用于目标对象，特性类会调用自己的某个实例构造器。

有的特性要指定参数，下面有个例子：



括号中的“Kernel32”是定位参数，应用该特性时必须要指定，因为这个参数是传给特性的实例构造器的；另外两个赋值的是特性类之中的公共字段或属性，也就是“命名参数”。

当对同一个目标指定多个特性时，不需要考虑添加的先后顺序，比如：

文本

描述已自动生成

甚至可以把多个特性都放入一个方括号中，方括号中的特性以逗号分隔

如何定义一个自己的特性类（特性是类的实例）：

定制特性类必须直接或者间接从System.Attribute类派生

书上的例子是定义一个为枚举类型添加位标志的特性

步骤：

1.定义FlagsAttribute类

图片包含 文本

描述已自动生成

这个类继承System.Attribute，定义一个public无参构造器

Ps：继承Attribute类，使得FlagsAttribute成为符合CLS规范的定制特性，类名带有Attribute后缀是为了保持与标准的相容性，但不是一定要加上Attribute后缀

Ps：特性类只提供一个公共构造器接收特性强制性状态信息，除了公共构造器只能提供公共非静态字段和公共非静态属性（尽量避免使用公共字段）

一些文字和图片的手机截图

描述已自动生成

什么是逻辑状态容器

逻辑状态容器（Logical State Container）是一种用于管理应用程序状态的模式。它通常用于在复杂的应用程序中管理状态，以确保应用程序的行为是可预测和可控的。逻辑状态容器通常包含一个或多个状态对象，这些对象表示应用程序的不同状态。每个状态对象都包含一组相关的数据和操作，用于管理该状态下的应用程序行为。

例如Unity中的动画状态机是一个逻辑状态容器

下面是一个管理游戏状态的逻辑状态容器的例子：

文本

描述已自动生成

逻辑状态容器

2.定义这个特性的合法范围

本例中要将FlagsAttribute特性定义为只能够应用于枚举类型

图片包含 文本

描述已自动生成

给FlagsAttribute加上AttributeUsage特性，给AttributeUsageAttribute的实例构造器传入定位参数“AttributeTargets.Enum”指定Flags特性的实例只能用于枚举类型。

Ps：以下是AttributeUsageAttribute特性类的FCL源代码：

文本

低可信度描述已自动生成

AttributeUsageAttribute特性类中的三个属性的作用：

AttributeTargets：指定特性类可以应用于哪些程序元素。它是一个枚举类型，可以使用按位或运算符或（“|”）来指定多个目标。

AllowMultiple：指定特性类是否可以应用多次。它是一个布尔值，如果设置为 true，则允许在同一程序元素上多次应用该特性；否则只能应用一次。

Inherited：属性指定特性类是否可以被派生类继承。它是一个布尔值，如果设置为 true，则派生类会继承基类上应用的该特性；否则不会继承。

关于AttributeTargets这个枚举类型是这样定义的：

表格

描述已自动生成

ps：1.一个特性类可以使用自身来修饰自己，比如AttributeUsageAttribute使用了AttributeUsage特性。2.极少数特性可以允许将它的多个实例应用于同一个目标，比如ConditionalAttribute（要将一个特性的多个实例运用于同一个目标，这个特性类中的AllowMultiple要被设为true）3.可继承特性不会导致在托管模块中为派生类型生成额外的元数据。

定义特征构造器和字段/属性数据类型：

定义特征构造器的实例构造器、字段、属性时，只允许使用以下数据类型：

文本

描述已自动生成

一个不理解的东西，在P378，下图

报纸上的文字和图片的手机截图

描述已自动生成

检测定制特性：

在定义了定制特性类之后，还缺一个步骤，程序代码的行为此时并不会发生变化。因此必须在运行时检查应用了这个特性的类是否关联了定制特性类的元数据，如果应用了这个定制特性则改变方法的行为（比如枚举类在应用了Flags特性后会改变ToString和Format方法的行为）。而在运行时检查的手段是反射。

书上的例子：

文本

描述已自动生成

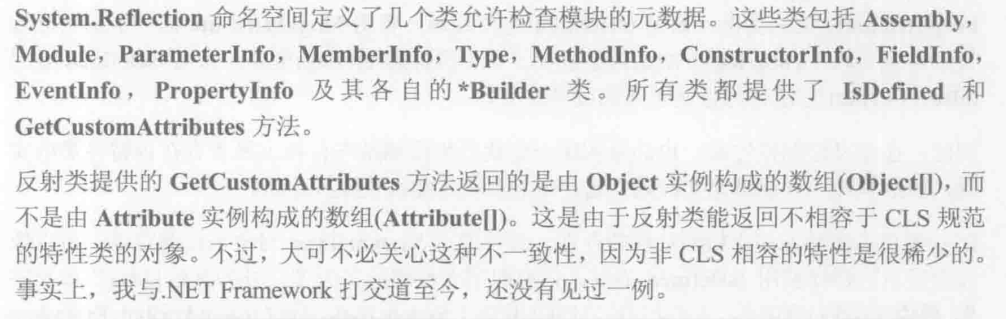
由此可知，在定义定制特性的同时必须实现一些代码来检测某些目标上是否存在该特性类的实例（是否应用该特性），然后执行应用和不应用两种情况下的逻辑分支代码。

FCL提供了以下方法检验特性的存在：

图片包含 表格

描述已自动生成

Ps：如果只想判断目标是否应用一个特性，使用IsDefine；构造特性对象必须调用GetCustomAttribute和GetCustomAttributes。调用这三种方法内部都必须扫描托管模块的元数据，执行字符串比较来定位指定的定制特性类（这反射会消耗更多的性能，反射操作基于字符串和扫描托管模块元数据）。因此需要大量使用反射操作时可以缓存这些方法的调用结果来减少调用请求相同的信息。



文本

描述已自动生成

两个特性实例的相互匹配：

为什么需要判断两个特性的实例是否相等？

在某些情况下，我们可能需要检查两个特性实例是否具有相同的值，以便在运行时做出相应的决策。例如，我们可能需要检查一个类是否具有与另一个类相同的特性，以确定它们是否可以互换使用。此外，判断两个特性实例是否相等也可以用于测试目的，以确保代码的正确性。

下面是一个例子，用于判断两个特性实例是否相等以测试代码的正确性：

图形用户界面, 文本, 应用程序

描述已自动生成在这个例子中，定义了一个名为MyAttribute的特性类，它具有两个属性：Name和Value。然后在一个名为MyClass的类上应用了这个特性，并为它指定了属性值。接下来定义了一个测试类MyTests，其中包含一个名为TestMyAttribute的测试方法。在这个测试方法中，我们创建了一个期望的特性实例，并使用Attribute.GetCustomAttribute方法获取MyClass上应用的特性实例。然后使用Assert.AreEqual方法来判断这两个特性实例是否相等。如果它们不相等，测试将失败，表明代码存在错误。

System.Attribute重写了Object的Equals方法，会在内部比较两个对象的类型，不一致返回false，若一致用反射比较两个特性对象中的字段值（为每个字段调用Equals），所有字段匹配则返回true否则返回false。可以在自己的定制类中重写这个Equals去掉反射以提升性能。

System.Attribute还有个public可见性的虚方法Match，默认实现是调用Attribute类的Equals方法并返回它的结果，这个方法可以重写。

检测特性时不创建从Attribute派生的对象，这样可以提高效率，因为不需要构造（反序列化）特性类的任何实例。

需要使用到System.Reflection.CustomAttribute类

例子：

在这个例子中定义了一个名为MyAttribute的特性类，它具有两个属性：Name和Value。然后在一个名为MyClass的类上应用了这个特性，并为它指定了属性值。在主方法中使用CustomAttributeData.GetCustomAttributes方法获取MyClass上应用的特性数据，并遍历它们以查找我们感兴趣的特性。当找到一个类型为MyAttribute的特性时，输出一条消息，并遍历该特性的命名参数以显示它们的值。

文本

描述已自动生成

结果：

图形用户界面, 文本, 应用程序

描述已自动生成

条件特性类：

什么是条件特性类？

条件特性类（Conditional attribute）是一种预定义特性，它标记了一个条件方法，其执行依赖于指定的预处理标识符。它会引起方法调用的条件编译，取决于指定的值，比如 Debug 或 Trace。例如，当调试代码时显示变量的值1

为什么需要用条件特性类？

因为可以使用特性简化开发，在设计和调试时使用特性辅助开发是很常见的。在没有使用代码分析工具时，这些调试和辅助开发用的特性会留在元数据中造成元数据膨胀，比如在发布版本中这些特性是不会用到的。因此需要使用条件特性类，在Debug阶段才生成这个特性，而Relese版本不生成这些特性。

书上的例子，SuppressMessage特性：

应用了System.Diagnostics.ConditionalAttribute特性的特性类称为“条件特性类”

文本

中度可信度描述已自动生成

只有预定义了TEST或者VERRIFY（#define TEST和#define VERRIFY）才会生成该特性的实例。另外CondAttribute特性类是一个稀有的可以让多个实例应用于同一个类型的特性类。