.NET 面试题解析(04)-类型、方法与继承

系列文章目录地址：

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

做技术是清苦的。一个人，一台机器，相对无言，代码纷飞，bug 无情。须梦里挑灯，冥思苦想，肝血暗耗，板凳坐穿。世界繁华竞逐，而你独钓寒江，看尽千山暮雪，听彻寒更雨歇。——来自《技术人的慰藉》

常见面试题目：

1. 所有类型都继承 System.Object 吗？
2. 解释 virtual、sealed、override 和 abstract 的区别
3. 接口和类有什么异同？
4. 抽象类和接口有什么区别？使用时有什么需要注意的吗？
5. 重载与覆盖的区别？
6. 在继承中 new 和 override 相同点和区别？看下面的代码，有一个基类 A，B1 和 B2 都继承自 A，并且使用不同的方式改变了父类方法 Print（）的行为。测试代码输出什么？为什么？

```
public void DoTest()
{
    B1 b1 = new B1(); B2 b2 = new B2();
    b1.Print(); b2.Print();      //按预期应该输出 B1、B2

    A ab1 = new B1(); A ab2 = new B2();
    ab1.Print(); ab2.Print();   //这里应该输出什么呢？
}
public class A
{
    public virtual void Print() { Console.WriteLine("A"); }
}
public class B1 : A
{
    public override void Print() { Console.WriteLine("B1"); }
}
public class B2 : A
{
    public new void Print() { Console.WriteLine("B2"); }
}
```

7. 下面代码中，变量 a、b 都是 int 类型，代码输出结果是什么？

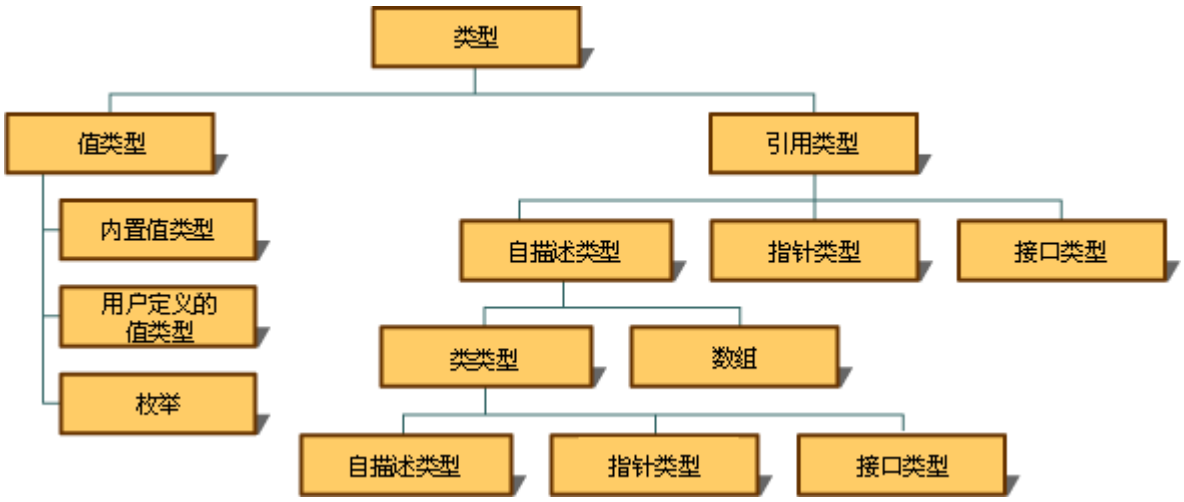
```
int a = 123;
int b = 20;
var atype = a.GetType();
var btype = b.GetType();
Console.WriteLine(System.Object.Equals(atype,btype));
Console.WriteLine(System.Object.ReferenceEquals(atype,btype));
```

8.class 中定义的静态字段是存储在内存中的哪个地方？为什么会说她不会被 GC 回收？

类型基础知识梳理

类型 Type 简述

通过本系列前面几篇文章，基本了解了值类型和引用类型，及其相互关系。如下图，.NET 中主要的类型就是值类型和引用类型，所有类型的基类就是 `System.Object`，也就是说我们使用 FCL 提供的各种类型的、自定义的所有类型都最终派生自 `System.Object`，因此他们也都继承了 `System.Object` 提供的基本方法。



`System.Object` 可以说是.NET 中的万物之源，如果非要较真的话，好像只有接口不继承她了。接口是一个特殊的类型，可以理解为接口是普通类型的约束、规范，她不可以实例化。（实际编码中，接口可以用 object 表示，只是一种语法支持，此看法不知是否准确，欢迎交流）

在.NET 代码中，我们可以很方便的创建各种类型，一个简单的数据模型、复杂的聚合对象类型、或是对客观世界实体的抽象。类 ***(class)*** 是最基础的 C# 类型（注意：本文主要探讨的就是引用类型，文中所述类型如没注明都为引用类型），支持继承与多态。一个 c# 类 Class 主要包含两种基本成员：

- **状态（字段、常量、属性等）**

- **操作（方法、事件、索引器、构造函数等）**

利用创建的类型（或者系统提供的），可以很容易的创建对象的实例。使用 new 运算符创建，该运算符为新的实例分配内存，调用构造函数初始化该实例，并返回对该实例的引用，如下面的语法形式：

<类名> <实例名> = new <类名>([构造函数的参数])

创建后的实例对象，是一个存储在内存上（在线程栈或托管堆上）的一个对象，那可以创造实例的类型在内存中又是一个什么样的存在呢？**她就是类型对象（Type Object）**。

类型对象（Type Object）

看看下面的代码：

```
int a = 123;                                                    // 创建 int 类型实例 a
int b = 20;                                                     // 创建 int 类型实例 b
var atype = a.GetType();                                        // 获取对象实例 a 的类型 Type
var btype = b.GetType();                                        // 获取对象实例 b 的类型 Type
Console.WriteLine(System.Object.Equals(atype,btype));           //输出：True
Console.WriteLine(System.Object.ReferenceEquals(atype, btype)); //输出：True
```

任何对象都有一个 GetType()方法（基类 System.Object 提供的），该方法返回一个对象的类型，类型上面包含了对象内部的详细信息，如字段、属性、方法、基类、事件等等（通过反射可以获取）。在上面的代码中两个不同的 int 变量的类型（int.GetType()）是同一个 Type，说明 int 在内存中有唯一一个（类似静态的）Systen.Int32 类型。

上面获取到的 Type 对象（Systen.Int32）就是一个类型对象，她同其他引用类型一样，也是一个引用对象，这个对象中存储了 int32 类型的所有信息（类型的所有元数据信息）。

关于类型类型对象（Object Type）：

>每一个类型（如 System.Int32）在内存中都会有一个唯一的类型对象，通过（int）a.GetType()可以获取该对象；

>类型对象（Object Type）存储在内存中一个独立的区域，叫加载堆（Load Heap），加载堆是在进程创建的时候创建的，不受 GC 垃圾回

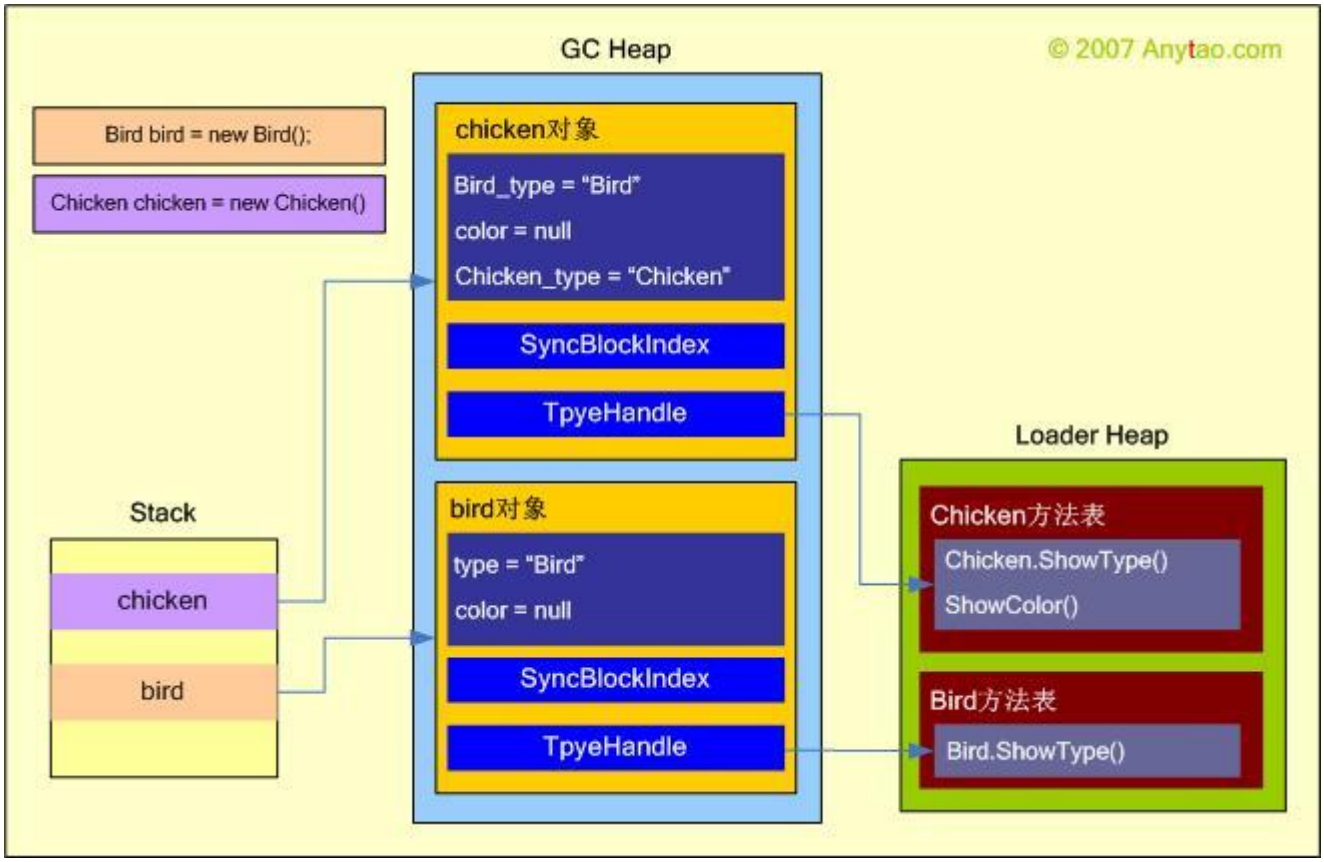
收管制，因此类型对象一经创建就不会被释放的，他的生命周期从 AppDomain 创建到结束；

>前问说过，每个引用对象都包含两个附加成员：TypeHandle 和同步索引块，其中 TypeHandle 就指向该对象对应的类型对象；

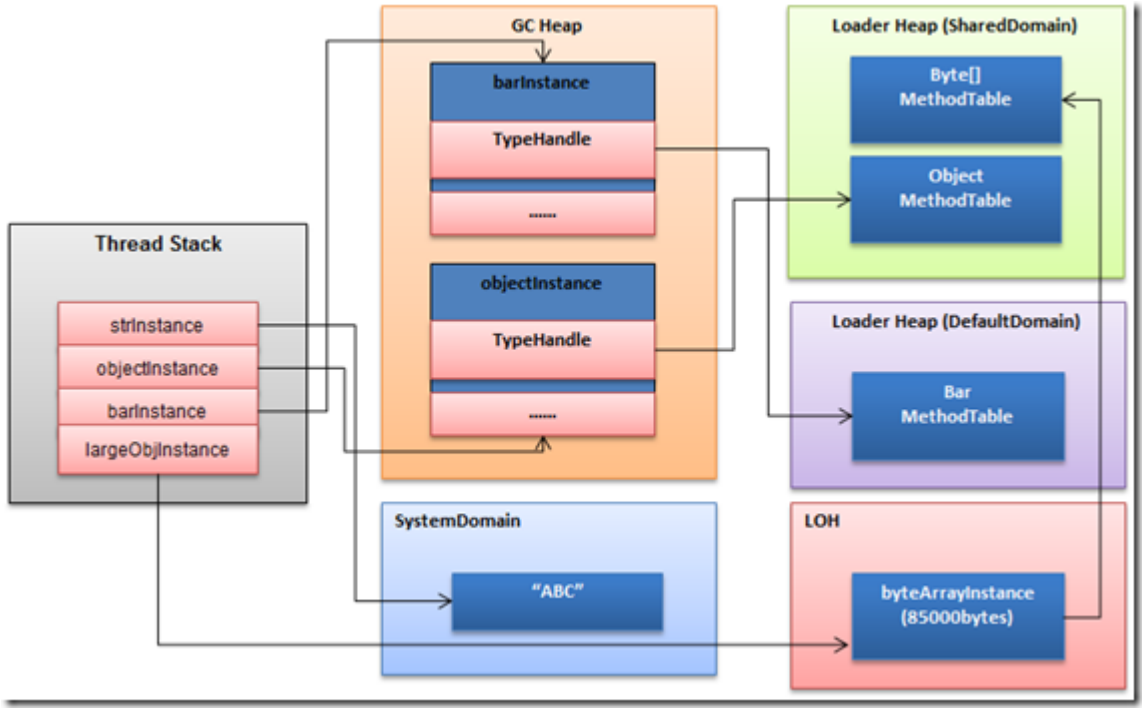
>类型对象的加载由 class loader 负责，在第一次使用前加载；

>类型中的静态字段就是存储在这里的（加载堆上的类型对象），所以说静态字段是全局的，而且不会释放；

可以参考下面的图，第一幅图描述了对象在内存中的一个关系， 第二幅图更复杂，更准确、全面的描述了内存的结构分布。



图片来源



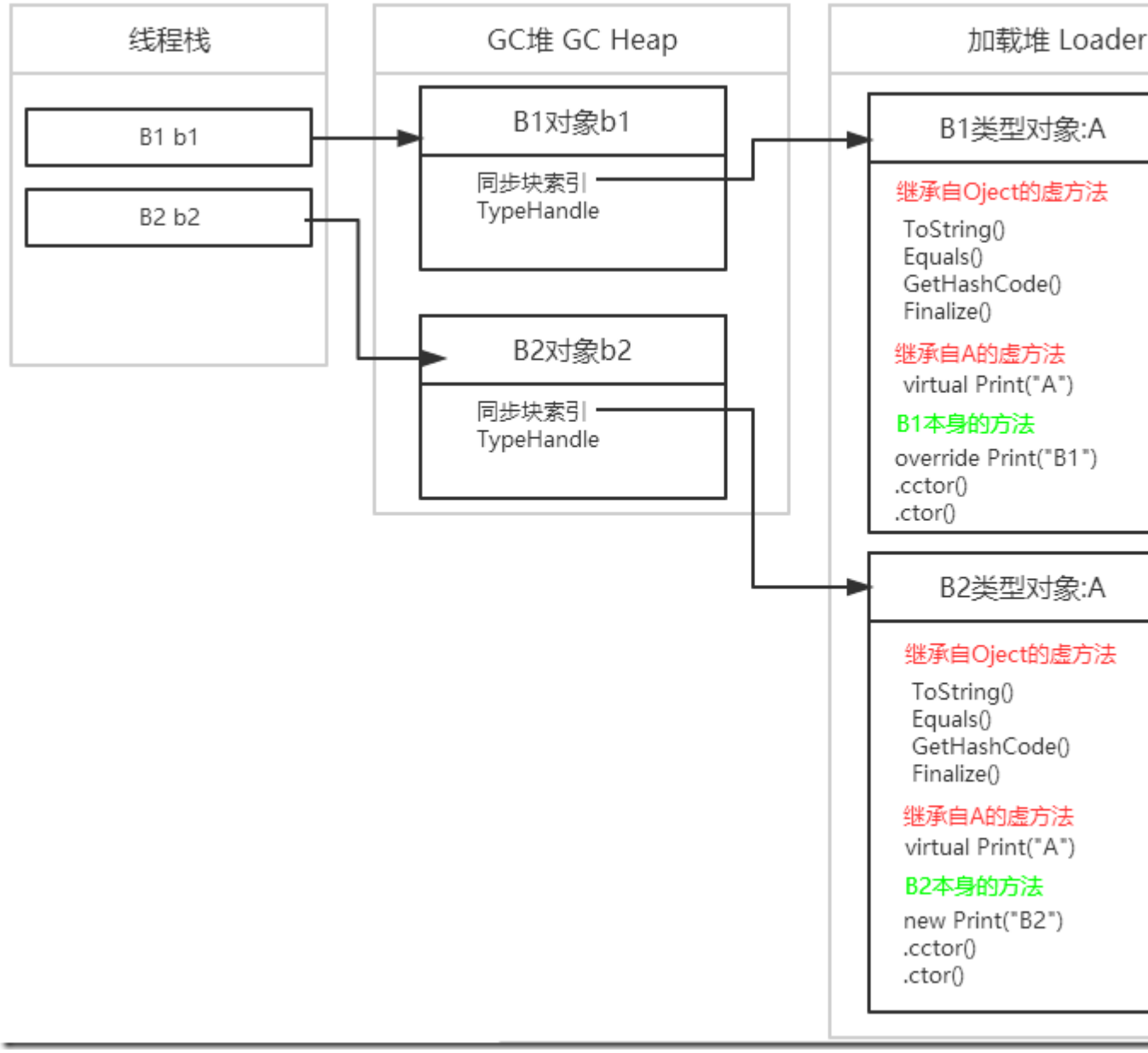
方法表

类型对象内部的主要的结构是怎么样的呢？其中最重要的就是**方法表**，包含了是类型内部的所有方法入口，关于具体的细节和原理这里不多赘述(太多了，可以参考文末给的参考资料)，本文只是初步介绍一下，主要目的是为了解决第 6 题。

```
public class A
{
    public virtual void Print() { Console.WriteLine("A"); }
}
public class B1 : A
{
    public override void Print() { Console.WriteLine("B1"); }
}
public class B2 : A
{
    public new void Print() { Console.WriteLine("B2"); }
}
```

还是以第 6 题的代码为例，上面的代码中，定义两个简单的类，一个基类 A，，B1 和 B2 继承自 A，然后使用不同的方式改变了父类方法的行为。当定义了 b1、b2 两个变量后，内存结构示意图如下：

```
B1 b1 = new B1();
B2 b2 = new B2();
```



方法表的加载：

- 方法表的加载时父类在前子类在后的，首先加载的是固定的 4 个来自 System.Object 的虚方法：ToString, Equals, GetHashCode, and Finalize；
- 然后加载父类 A 的虚方法；
- 加载自己的方法；
- 最后是构造方法：静态构造函数.cctor()，对象构造函数.ctor()；

方法表中的方法入口（方法表槽 ）还有很多其他的信息，比如会关联方法的 IL 代码以及对应的本地机器码等。其实类型对象本身也是一个引用类型对象，其内部同样也包含两个附件成员：同步索引块和类型对象指针 TypeHandel，具体细节、原理有兴趣的可以自己深入了解。

方法的调用：当执行代码 `b1.Print()`时（此处只关注方法调用，忽略方法的继承等因素），通过 b1 的 TypeHandel 找到对应类型对象，然后找到方法

表槽，然后是对应的 IL 代码，第一次执行的时候，JIT 编译器需要把 IL 代码编译为本地机器码，第一次执行完成后机器码会保留，下一次执行就不需要 JIT 编译了。这也是为什么说.NET 程序启动需要预热的原因。

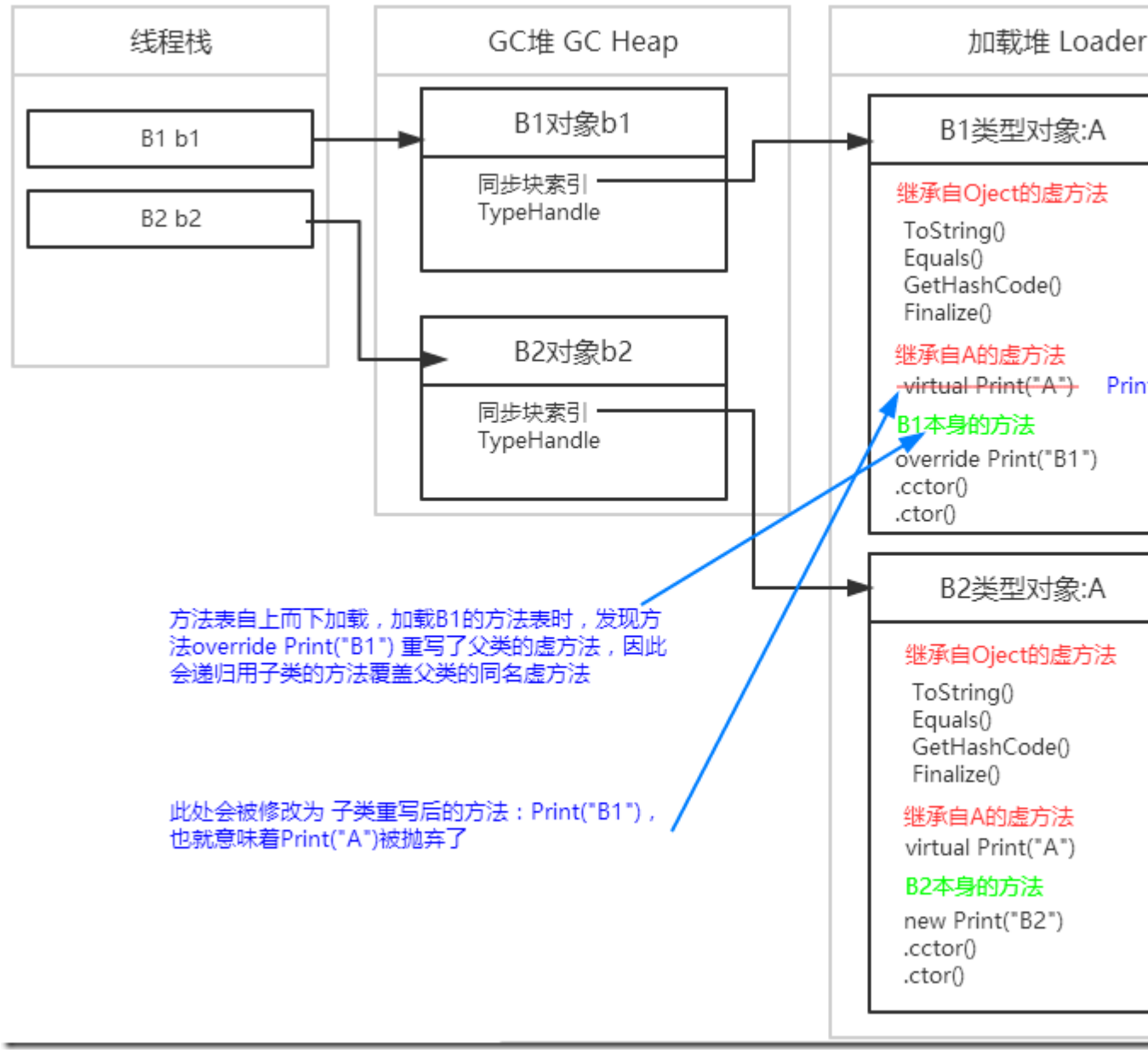
.NET 中的继承本质

方法表的创建过程是从父类到子类自上而下的，这是.NET 中继承的很好体现，**当发现有覆写父类虚方法会覆盖同名的父方法**，所有类型的加载都会递归到 System.Object 类。

- 继承是可传递的，子类是对父类的扩展，必须继承父类方法，同时可以添加新方法。
- 子类可以调用父类方法和字段，而父类不能调用子类方法和字段。
- 子类不光继承父类的公有成员，也继承了私有成员，只是不可直接访问。
- new 关键字在虚方法继承中的阻断作用，中断某一虚方法的继承传递。

因此类型 B1、B2 的类型对象进一步的结构示意图如下：

- 在加载 B1 类型对象时，当加载 override B1.Print(“B1”)时，发现有覆写 override 的方法，会覆盖父类的同名虚方法 Print(“A”)，就是下面的示意图，简单来说就是在 B1 中 Print 只有一个实现版本；
- 加载 B2 类型对象时，new 关键字表示要隐藏基类的虚方法，此时 B2 中的 Print(“B2”)就不是虚方法了，她是 B2 中的新方法了，简单来说就是在 B2 类型对象中 Print 有 2 个实现版本；



B1 b1 = new B1();

```
 B2 b2 = new B2();
b1.Print(); b2.Print();      //按预期应该输出 B1、B2

A ab1
```

= new

```
 B1();
A ab2 = new B2();
ab1.Print(); ab2.Print();   //这里应该输出什么呢？
```

上面代码中红色高亮的两行代码，用基类（A）和用本身 B1 声明到底有什么区别呢？类似这种代码在实际编码中是很常见的，简单的概括一下：

- 无论用什么做引用声明，哪怕是 object，等号右边的[= new 类型()]都是没有区别的，也就说说对象的创建不受影响的，b1 和 ab1 对象在内存结构上是一致的；
- 他们的的差别就在引用指针的类型不同，这种不同在编码中智能提示就直观的反应出来了，在实际方法调用上也与引用指针类型有直接关系；
- 综合来说，不同引用指针类型对于对象的创建（new 操作）不影响；但对于对象的使用（如方法调用）有影响，这一点在上面代码的执行结果中体现出来了！

上面调用的 IL 代码：

```
IL_001b:  newobj     instance void CLRTest.ConsoleTest.BlogTest/B1::.ctor() A ab1 =
IL_0020:  stloc.2
IL_0021:  newobj     instance void CLRTest.ConsoleTest.BlogTest/B2::.ctor() A ab2 =
IL_0026:  stloc.3
IL_0027:  ldloc.2
IL_0028:  callvirt   instance void CLRTest.ConsoleTest.BlogTest/A::Print()  ab1.Print
IL_002d:  nop
IL_002e:  ldloc.3
IL_002f:  callvirt   instance void CLRTest.ConsoleTest.BlogTest/A::Print()  ab2.Print
```

对于虚方法的调用，在 IL 中都是使用指令 callvirt，该指令主要意思就是具体的方法在运行时动态确定的：

callvirt 使用虚拟调度，也就是根据引用类型的动态类型来调度方法，callvirt 指令根据引用变量指向的对象类型来调用方法，在运行时动态绑定，主要用于调用虚方法。

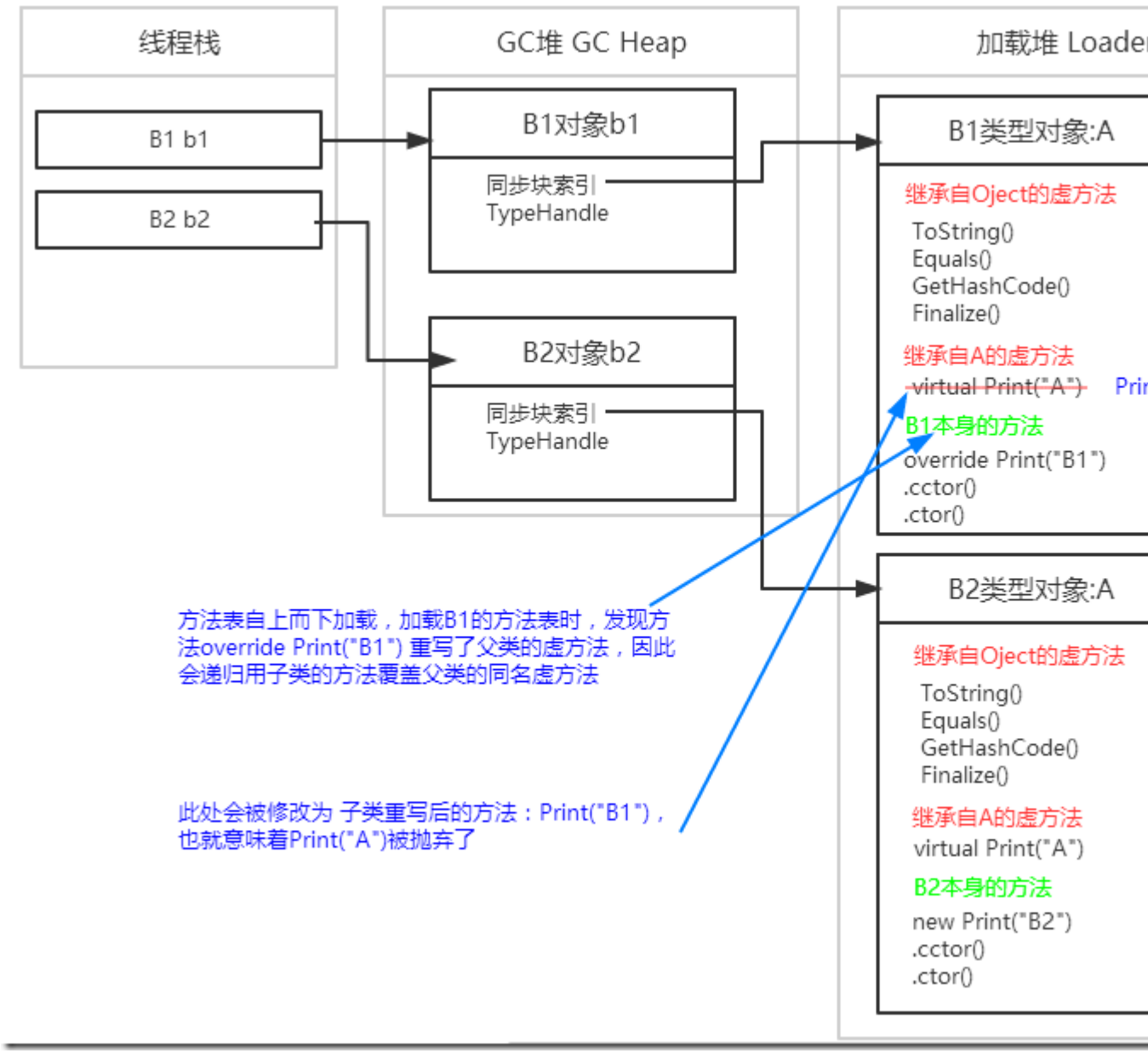
不同的类型指针在虚拟方法表中有不同的附加信息作为标志来区别其访问的地址区域，称为 **offset**。不同类型的指针只能在其特定地址区域内进行执行。编译器在方法调用时还有一个原则：

执行就近原则：对于同名字段或者方法，编译器是按照其顺序查找来引用的，也就是首先访问离它创建最近的字段或者方法。

因此执行以下代码时，引用指针类型的 offset 指向子类，如下图，，按照就近查找执行原则，正常输出 B1、B2

B1 b1 = new B1();

```
 B2 b2 = new B2();
b1.Print(); b2.Print();      //按预期应该输出 B1、B2
```

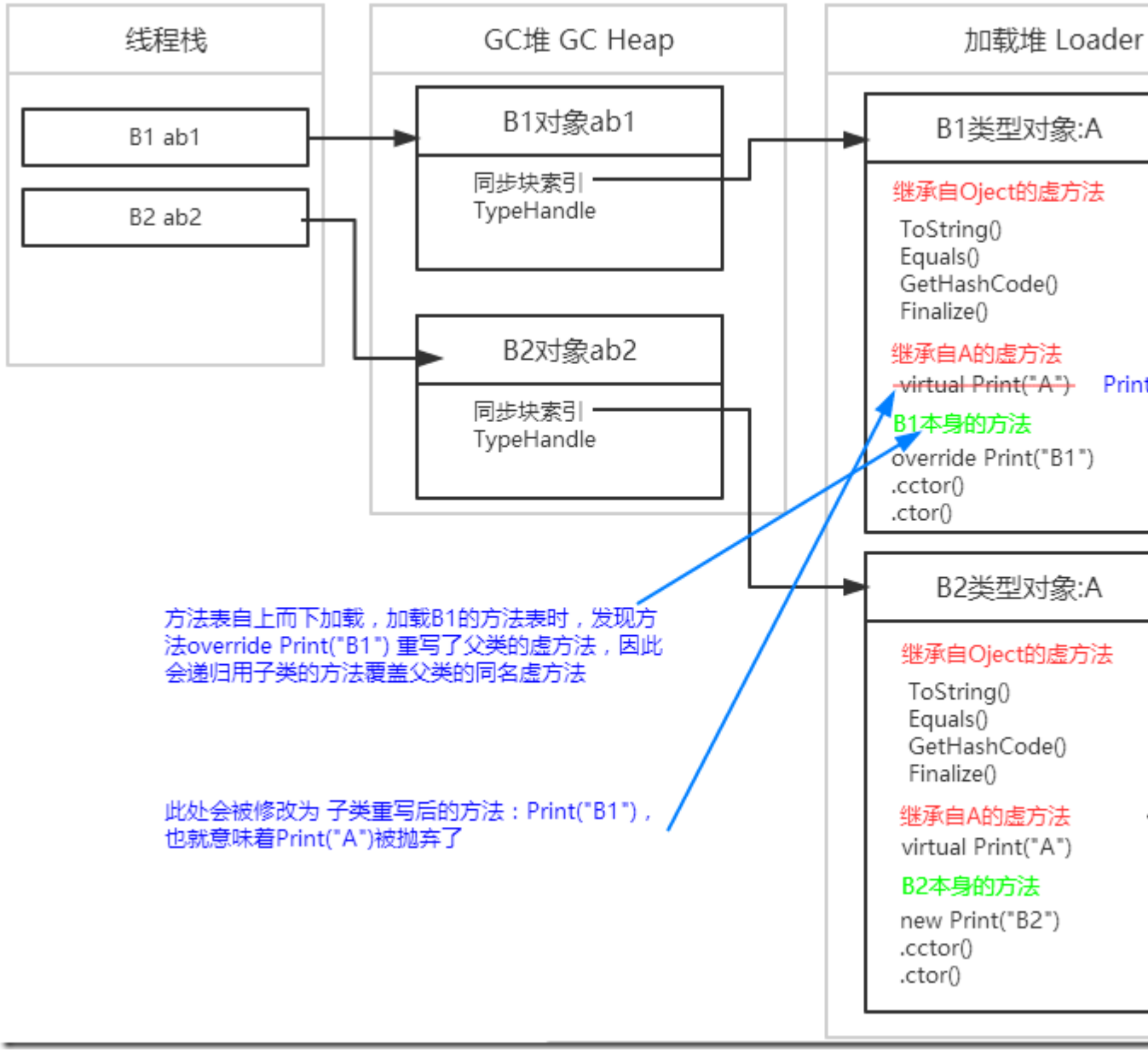


而当执行以下代码时，引用指针类型都为父类 A，引用指针类型的 offset 指向父类，如下图，按照就近查找执行原则，输出 B1、A。

```
A ab1
```

= new

```
 B1();
A ab2 = new B2();
ab1.Print(); ab2.Print();   //这里应该输出什么呢？
```



.NET 中的继承

什么是抽象类

抽象类提供多个派生类共享基类的公共定义，它既可以提供抽象方法，也可以提供非抽象方法。抽象类不能实例化，必须通过继承由派生类实现其抽象方法，因此对抽象类不能使用 new 关键字，也不能被密封。

基本特点：

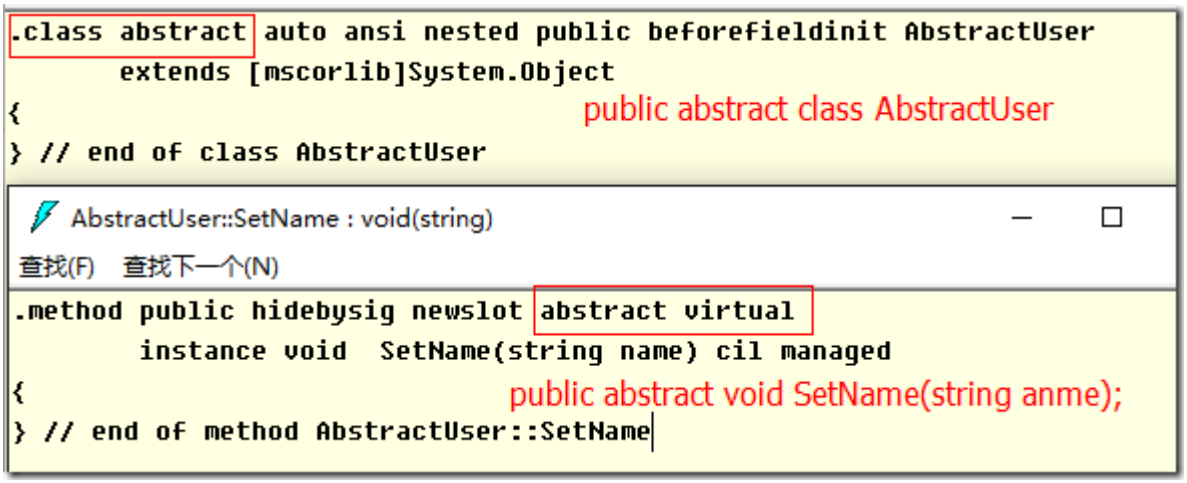
- 抽象类使用 Abstract 声明，抽象方法也是用 Abstract 标示；
- 抽象类不能被实例化；
- 抽象方法必须定义在抽象类中；
- 抽象类可以继承一个抽象类；
- 抽象类不能被密封（不能使用 sealed）；

- 同类 Class 一样，只支持单继承；

一个简单的抽象类代码：

```
public abstract class AbstractUser
{
    public int Age { get; set; }
    public abstract void SetName(string name);
}
```

IL 代码如下，类和方法都使用 abstract 修饰：



什么是接口？

接口简单理解就是一种规范、契约，使得实现接口的类或结构在形式上保持一致。实现接口的类或结构必须实现接口定义中所有接口成员，以及该接口从其他接口中继承的所有接口成员。

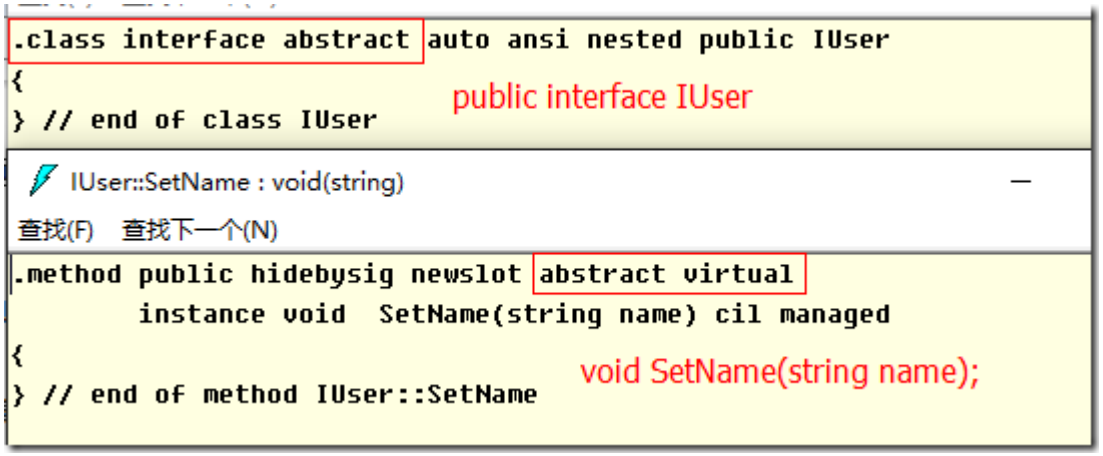
基本特点：

- 接口使用 interface 声明；
- 接口类似于抽象基类，不能直接实例化接口；
- 接口中的方法都是抽象方法，不能有实现代码，实现接口的任何非抽象类型都必须实现接口的所有成员；
- 接口成员是自动公开的，且不能包含任何访问修饰符。
- 接口自身可从多个接口继承，类和结构可继承多个接口，但接口不能继承类。

下面一个简单的接口定义：

```
public interface IUser
{
    int Age { get; set; }
    void SetName(string name);
}
```

下面是 IUser 接口定义的 IL 代码，看上去是不是和上面的抽象类 AbstractUser 的 IL 代码差不多！接口也是使用`.Class ~ abstract`标记，方法定义同抽象类中的方法一样使用`abstract virtual`标记。因此可以把接口看做是一种特殊的抽象类，该类只提供定义，没有实现。



另外一个小细节，上面说到接口是一个特殊的类型，不继承`System.Object`，通过 IL 代码其实可以证实这一点。无论是自定义的任何类型还是抽象类，都会隐式继承`System.Object`，AbstractUser 的 IL 代码中就有“extends [mscorlib]System.Object”，而接口的 IL 代码并没有这一段代码。

关于继承

关于继承，太概念性了，就不细说了，主要还是在平时的搬砖过程中多思考、多总结、多体会。在.NET 中继承的主要两种方式就是**类继承和接口继承**，两者的主要思想是不一样的：

- 类继承强调父子关系，是一个“IS A”的关系，因此只能单继承（就像一个人只能有一个 Father）；
- 接口继承强调的是一种规范、约束，是一个“CAN DO”的关系，支持多继承，是实现多态一种重要方式。

更准确的说，类可以叫继承，接口叫“实现”更合适。更多的概念和区别，可以直接看后面的答案，更多的还是要自己理解。

题目答案解析:

1. 所有类型都继承 System.Object 吗？

基本上是的，所有值类型和引用类型都继承自 System.Object，接口是一个特殊的类型，不继承自 System.Object。

2. 解释 virtual、sealed、override 和 abstract 的区别

- virtual 申明虚方法的关键字，说明该方法可以被重写
- sealed 说明该类不可被继承
- override 重写基类的方法
- abstract 申明抽象类和抽象方法的关键字，抽象方法不提供实现，由子类实现，抽象类不可实例化。

3. 接口和类有什么异同？

不同点：

1、接口不能直接实例化。

2、接口只包含方法或属性的**声明**，不包含方法的实现。
3、接口可以**多继承**，类只能单继承。
4、类有**分部类**的概念，定义可在不同的源文件之间进行拆分，而接口没有。
5、表达的含义不同，接口主要定义一种**规范**，统一调用方法，也就是规范类，约束类，类是方法功能的实现和集合
相同点：

1、接口、类和结构都可以从多个接口继承。

2、接口类似于抽象基类：继承接口的任何非抽象类型都必须实现接口的所有成员。

3、接口和类都可以包含事件、索引器、方法和属性。

4. 抽象类和接口有什么区别？

1、继承：接口支持多继承；抽象类不能实现多继承。

2、表达的概念：接口用于规范，更强调契约，抽象类用于共性，强调父子。抽象类是一类事物的高度聚合，那么对于继承抽象类的子类来说，对于抽象类来说，属于"Is A"的关系；而接口是定义行为规范，强调“Can Do”的关系，因此对于实现接口的子类来说，相对于接口来说，是"行为需要按照接口来完成"。

3、方法实现：对抽象类中的方法，即可以给出实现部分，也可以不给出；而接口的方法（抽象规则）都不能给出实现部分，接口中方法不能加修饰符。

4、子类重写：继承类对于两者所涉及方法的实现是不同的。继承类对于抽象类所定义的抽象方法，可以不用重写，也就是说，可以延用抽象类的方法；而对于接口类所定义的方法或者属性来说，在继承类中必须重写，给出相应的方法和属性实现。

5、新增方法的影响：在抽象类中，新增一个方法的话，继承类中可以不用作任何处理；而对于接口来说，则需要修改继承类，提供新定义的方法。

6、接口可以作用于值类型（枚举可以实现接口）和引用类型；抽象类只能作用于引用类型。

7、接口不能包含字段和已实现的方法，接口只包含方法、属性、索引器、事件的签名；抽象类可以定义字段、属性、包含有实现的方法。

5. 重载与覆盖的区别？

重载：当类包含两个名称相同但签名不同(方法名相同,参数列表不相同)的方法时发生方法重载。用方法重载来提供在语义上完成相同而功能不同的方法。

覆写：在类的继承中使用，通过覆写子类方法可以改变父类虚方法的实现。

主要区别：
1、方法的覆盖是子类和父类之间的关系，是垂直关系；方法的重载是同一个类中方法之间的关系，是水平关系。

2、覆盖只能由一个方法，或只能由一对方法产生关系；方法的重载是多个方法之间的关系。
3、覆盖要求参数列表相同；重载要求参数列表不同。
4、覆盖关系中，调用那个方法体，是根据对象的类型来决定；重载关系，是根据调用时的实参表与形参表来选择方法体的。

6. 在继承中 new 和 override 相同点和区别？看下面的代码，有一个基类 A，B1 和 B2 都继承自 A，并且使用不同的方式改变了父类方法 Print（）的行为。测试代码输出什么？为什么？

```
public void DoTest()
{
    B1 b1 = new B1(); B2 b2 = new B2();
    b1.Print(); b2.Print();      //按预期应该输出 B1、B2

    A ab1 = new B1(); A ab2 = new B2();
    ab1.Print(); ab2.Print();   //这里应该输出什么呢？输出 B1、A
}
public class A
{
    public virtual void Print() { Console.WriteLine("A"); }
}
public class B1 : A
{
    public override void Print() { Console.WriteLine("B1"); }
}
public class B2 : A
{
    public new void Print() { Console.WriteLine("B2"); }
}
```

7. 下面代码中，变量 a、b 都是 int 类型，代码输出结果是什么？

```
int a = 123;
int b = 20;
var atype = a.GetType();
var btype = b.GetType();
Console.WriteLine(System.Object.Equals(atype,btype));          //输出 True
Console.WriteLine(System.Object.ReferenceEquals(atype,btype)); //输出 True
```

8.class 中定义的静态字段是存储在内存中的哪个地方？为什么会说她不会被 GC 回收？

随类型对象存储在内存的加载堆上，因为加载堆不受 GC 管理，其生命周期随 AppDomain，不会被 GC 回收。

.NET 面试题解析(00)-开篇来谈谈面试 & 系列文章索引

参考资料:

书籍：CLR via C#

书籍：你必须知道的.NET

Interface 继承至 System.Object？：
http://www.cnblogs.com/whitewolf/archive/2012/05/23/2514123.html
关于 CLR 内存管理一些深层次的讨论[下篇]
[你必须知道的.NET]第十五回：继承本质论
深入.NET Framework 内部，看看 CLR 如何创建运行时对象的

后记：本文写的有点难产，可能还是技术不够熟练，对于文中的“继承中的方法表”那一部分理解的还不够透彻，也花了不少时间（包括画图），一直犹豫要不要发出来，害怕理解有误，最终还是发出来了，欢迎交流、指正！

分类: C#.NET

/*梦里花落知多少*/
关注 - 5
粉丝 - 136
+加关注

« 上一篇：.NET 面试题解析(03)-string 与字符串操作

posted @ 2016-03-07 00:51 /*梦里花落知多少*/ 阅读(629) 评论(6) 编辑 收藏

评论列表

#1 楼 2016-03-07 02:27 大树 v587
支持，做了这么几年 c#要我来搞这些基础还是好虚。

支持(2)反对(0)

#2 楼 2016-03-07 10:41 王浩兴

A ab1 = new B1(); A ab2 = new B2();

ab1.Print(); ab2.Print(); ab2 输出 A 我没看懂为什么，能简单再给解释下吗

支持(0)反对(0)

#3 楼[楼主] 2016-03-07 15:11 /*梦里花落知多少*/

@王浩兴

这个一两句话也说不清楚，建议可以再看看上面的文章（和本系列以前的）。

要想搞清楚，最好还是多看看书，比如文末提到的参考资料。

支持(0)反对(0)

#4 楼 2016-03-07 15:44 丨渊丨

@王浩兴

B2 类没有 override Print(),是 new Print(),这里 new 的意思是隐藏，隐藏子类从父类那里继承过来的同名方法，ab2 是 A 类的实例，所以会调用 A 类的 Print 方法。

支持(0)反对(0)

#5 楼 2016-03-07 16:50 丨渊丨

静态类 还有静态字段 不是在静态存储区中储存吗

静态存储区：内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。它主要存放静态数据、全局数据和常量。

支持(0)反对(0)

#6 楼[楼主] 2016-03-07 21:44 /*梦里花落知多少*/

@丨渊丨

ab2 是 A 类的实例，所以会调用 A 类的 Print 方法。

---

ab2 和 b2 都是 B2 的实例哦，