审查征集贴：
http://www.cnblogs.com/BeginnerClassroom/archive/2010/07/30/1788649.html

附录征集贴：
http://www.cnblogs.com/BeginnerClassroom/archive/2010/08/04/1792175.html

欢迎各位园友对本书的某一部分内容进行拓展，将以附录的形式附在书后。

要求：

1. 紧紧围绕一两个中心展开；
2. 逻辑清晰，行文流畅；
3. 考虑到初学者的基础。
4. 写作时间最好不要少于一星期。

我写东西都是写好以后先放在那里，过段时间再读，重新修改，如此反复几次，就基本上很流畅了。

（PS：会署名，但无稿费，因为本来就没多少，不够分的。当然如果发了大财，我会分给大家的。）

标题	作者	状态
关于 RichTextBox 修改字体大小的研究	李雨来	已完稿
委托和接口的策略延迟思想	汤非凡	已完稿
XML 格式注释	张智鸣	已完稿
抽象类与接口的区别及应用	张洋	已完稿
.NET 版本变更史	张智鸣	已完稿
字符编码	赵士敬	正在写
流的应用实例	黄志斌	已完稿
通过在线评测平台磨砺 C#能力	曹如进	已完稿
正则表达式在 EmEditor 里的应用	柳永法	已完稿
C#程序编码规范	顾磊	已完稿
异步读写操作		待选
控件开发、自定义控件	MingHao_Hu	正在写
结构和类的联系与区别		待选

绘图缓存		待选
正则表达式应用实例	空军	正在写
（欢迎您提供其他附录）		

抽象类与接口的区别及应用

（本文由张洋提供）

抽象类（Abstract Class）与接口（Interface）是面向对象程序设计中两个重要的概念。由于两者在自身特性及应用方法上存在诸多相似性，如都不能实例化、都可以被继承（严格来说对于接口应该叫做实现），这么一来，在许多人心中抽象类与接口的界限非常模糊，对何时该使用抽象类、何时该使用接口更是感到困惑。

本文的目的是通过对两者的讨论与比较，帮助读者认清抽象类与接口在思想本质及应用场合方面的区别，如能做到这一点，读者便可以得心应手地根据具体情况正确选择和使用抽象类与接口。

1. 抽象类与接口是面向对象思想层面概念，不是程序设计语言层面概念

如若想正确认识抽象类与接口，首先要弄清楚的一点是，这两个概念均属于面向对象思想层面，而不属于某种程序设计语言。例如，C#中用 interface 关键字声明的语言元素，我们叫它“接口”，其实这是不准确的，准确来说，这应该叫做“接口在 C#语言中的实现机制”。

面向对象思想包含许多概念，而不同面向对象语言对这些概念的具体实现机制各有不同。例如，C++中并没有一种关键字对应于 C#中的 interface，那么 C++中就没有接口的概念了吗？非也！在 C++中，如果想定义一个接口，可以通过将一个类中所有方法定义为纯虚方法[①]来做到。

这里可以看到，同样是接口，C#中用 interface 关键字来定义，而 C++通过创建一个只包含纯虚方法的类来定义，这就是同一种概念在不同具体语言中具有不同的实现机制。类似的，C++中也没有 abstract 关键字用于定义抽象类，而是如果一个类中至少含有一个纯虚方法且它的方法不全为纯虚方法，则这个类被称为抽象类。

通过上面的分析可以看出，如果仅仅停留在语言层面去认知抽象类与接口，是无法准确理解两者的真谛的，因为不同语言对同一概念的实现机制有很大差别。

如果一个 C#初学者简单将两者理解为“用 abstract 修饰的类是抽象类，用 interface 定义的语言元素是接口”，那么当他接触 C++时一定会感到困惑，因为 C++里既没有 abstract 也没有 interface，而是通过类中纯虚方法的情况确定这是个类、是个抽象类还是个接口。

明确了上面的问题，我们就可以给出抽象类与接口的真正定义了。

抽象类是不能实例化的类，但是其中的方法可以包含具体实现代码。

接口是一组方法声明的集合，其中应仅包含方法的声明，不能有任何实现代码。

以上对抽象类和接口的定义与任何具体语言无关，而是从面向对象思想角度进行的定义，不同语言可以有不同的实现机制。

从上面的定义中，我们可以发现两者在思想层面上的一项重大区别：抽象类是类（Class），接口是集合（Set），两者从本质上不是一种东西。这是我们总结出的第一个区别。请读者受累将上面加粗的字能放声朗十遍，声音越大越好，但是如果被室友或邻居扔鸡蛋请不要找我。

2. 抽象类是本体的抽象，接口是行为的抽象

在开始这一节之前，我想先请问各位一个问题，“我是一个人”和“我能呼吸”分别表达了“我”和“人”以及“我”和“呼吸”的关系，那么这两句话表达的是一种关系吗？如果你能很容易区分前者表示“是一个”的关系，而后者表示“能”的关系，那么恭喜你，你一定也能很容易区分抽象类和接口。

在阅读这一节时，请读者务必谨记上面这个问题以及下面这句话：

抽象类表示“是一个（IS-A）”关系的抽象，接口表示“能（CAN-DO）”关系的抽象。

请照例将上面的大声话朗读十遍。

好的，请各位擦干净头上的鸡蛋，我们继续。

从上面粗体字中我们可以看出，抽象类和接口有一个共性——它们都是“某种关系的抽象”，只不过类型不同罢了。其实如果将上面那句话的前半句中的“抽象类”改为“类”也是正确的，这并不奇怪，上文我们说过，抽象类只不过是一种特殊的类罢了。

下面我们先来解释 IS-A 关系。其实英语中的 IS-A 关系在汉语中可以解释为两种情况，当 IS-A 用在一个对象和一个类之间时，意思是“这个对象是类的一个实例”，例如关羽是一个对象，我们可以说“GuanYu IS-A General”，其中 General（将军）是个类，这表示关羽是将军类的一个实例。而当 IS-A 用在两个类之间时，我认为叫做 IS-A-KIND-OF 更为准确，表示汉语中的“是一种”，如“General IS-A Person”，表示将军这个类是人这个类的一种，换用面向对

象术语可以如下表述：General 是 Person 的子类（Sub Type），Person 是 General 的父类或超类（Super Type），General 继承自 Person。

这后一种 IS-A 关系，就是抽象类所表达的关系。分析到这里可以看出，抽象类所表达的关系其实就是面向对象三大特性之一——继承（Inheritance），也就是说，抽象类所表达的关系，与一般类与类之间的继承并无区别，而抽象类相比普通类，除了不能实例化外，也并无区别。之所以出现抽象类，是因为在较高抽象层次上，某些方法（往往是纯虚方法）无法实现，必须由其子类按照各自不同的情况具体实现。因为它含有纯虚方法，所以将这种类实例化在道理上讲不通，但我们又希望将这些子类中共有的部分抽象出来减少代码重复，于是就有了抽象类——它包含可复用部分，但又不允许实例化。

因此，抽象类的使用动机是在不允许实例化的限制下复用代码。请牢记这个动机。

接着再说说接口和 CAN-DO 关系。

我们知道，面向对象编程的基本思想就是通过对象间的相互协作，完成程序的功能。具体来说，在面向对象编程中，要求每个类都隐藏内部细节（这叫封装性），仅对外暴露一组公共方法，对象间就通过互相调用彼此的公共方法完成程序功能。

可以看到，面向对象思想中，对象和对象间根本不需要了解，调用者甚至可以完全不知道被调用者是谁，只要知道被调用者“能干什么”就行了。这就如同拨打 110 报警一样，你根本不知道对方长什么样、穿什么衣服、结没结婚、有没有孩子，你也不知道对方在哪，对象是谁，但是你知道对方一定“能接警”，所以你可以顺利完成报警。

这种“能干什么”就是 CAN-DO 关系，当我们把这种 CAN-DO 关系抽象出来，形成一个 CAN-DO 关系的集合，这就是接口了。那么使用接口的动机又是什么呢？动机之一是松散耦合。我们知道“低耦合”是面向对象程序设计中一个重要原则，而很大一部分耦合就是调用关系，面向对象中术语叫“依赖”。如果没有接口，调用者就要紧依赖于被调用者，就如同在没有 110 报警的年代，你只认识一个接警员，不知道其他接警员的电话，那么当你报警时，你必须给这个接警员打电话才行，如果哪天这个接警员休假或病了，你就无法报警了，除非你再去认识一个接警员。这时，我们说你紧依赖于这个接警员，也叫紧耦合。但有了 110 报警后就不一样了，我们将“可接警”看作一个接口，接口中有一个方法“接警”，而拨通 110 后，电话那头的人一定是实现了这个接口的，这时报警人不再依赖于具体接警员，而是依赖于“可接警”接口，这就叫做松依赖。

所以说，接口又可以看作一组规则的集合，它是对调用者的保证，对被调用者的约束。如上例中，可接警对报警人（调用者）保证调用对象可接警，同时约束接警部门必须把一个实现了这个接口的人安排在接警电话前面。哪怕这是个机器人或刚进行了两个小时接警培训的保洁员都没关系。

使用接口的另一个动机就是实现多态性[②]。

下面想象你被分配到一个全新的研发小组做主管，第一天上班的早晨，一群人站在你面前等着你训话，你完全不认识他们，也不知道他们各自的职务，但是你可以说一句“都去工作吧”，于是大家作鸟兽散，程序员去写程序，会计去核对账目，业务员出门联系客户……当你这样做的时候，你就利用接口实现了多态性。因为你知道，他们都实现了“可工作”这个接口，虽然各个人员对“工作”具体的实现不一样，但这不要紧，你只要调用他们的“工作”方法，他们就各自做自己的事情了。如果你不能面向接口去利用多态性，你就要一个个说：“程序员去写程序，会计去核账，业务员快出门联系客户……”，这实在非常的费劲。

对这一节的内容做一个总结：

抽象类表示“是一个（IS-A）”关系的抽象，它抽象了类的本体，其使用动机是在不允许实例化的限制下复用代码。接口表示“能（CAN-DO）”关系的抽象，它抽象了类的行为，其使用动机是松散对象间的耦合以及实现程序多态性。

好的，照例念十遍吧，不过这次我允许你默念，因为我怕这次飞来的不是鸡蛋而是砖头。

经过上面的分析，我想你已经可以很容易在抽象类与接口间做出选择了。如果你是为了将一系列类的公共代码抽出，减少代码的重复，并且这些类与抽象出来的类可以表述为 IS-A 关系，就用抽象类；如果你是为了将一个或一组行为抽象出来，用以松散对象间耦合或实现多态性，那就用接口吧。

3. C#中抽象类与接口的探讨

这一节我们讨论 C#语言中一个是人尽皆知的区别：在 C#中，一个类最多只能继承一个抽象类，但可以实现多个接口。

如果能充分理解抽象类对应于 IS-A 而接口对应于 CAN-DO，则对这个约束不会感到奇怪。因为从逻辑上来说，一个类在所有相同抽象层次的类中只能“是其中一个”，但“能干多种事情”。这里的相同抽象层次指互相不存在继承关系的一个全集。

例如，{猪，牛，狗，猫} 可以看作具有相同抽象层次，其某个下层类只能是其中一个的子类，一个类不可能既是牛的子类又是猪的子类，但有可能既是牛的子类又是动物的子类，例如奶牛，这是因为“动物”与“牛”不在一个抽象层次上，“牛”本身就是“动物”的一个子类。

一般的，如果 ClassA 是 ClassB 的子类，同时也是 ClassC 的子类，那么一定存在 ClassB 是 ClassC 的子类或 ClassC 是 ClassB 的子类。

换句话说，一个类同时继承两个互相没有继承关系的类在逻辑上是不成立的。这就说明了为什么 C#中不允许同时继承一个以上的抽象类。如果一个类要继承

两个抽象类，那么从逻辑上来说，两个抽象类之间必然也存在继承关系，因此只需让该类继承较具体的那个抽象类即可。例如，本来的设计为“奶牛”同时继承“牛”和“动物”，但很容易发现，“牛”和“动物”已经存在继承关系，“牛”是继承于“动物”的，因此可将继承关系修改为“奶牛”只继承“牛”，而让“牛”继承于“动物”，这样就消除了多重继承。

而接口的 CAN-DO 关系在逻辑上不存在这样的矛盾，所以 C#允许实现多个接口，具体为什么请读者自己思考。

顺便说一句，C++中允许多重继承是因为 C++中非抽象类、抽象类和接口都用类来实现，而没有在语言层面区分成不同的语言元素，其实如果设计良好，也是不应该出现对抽象类的多重继承的，C#在语言层面上进行了约束，更有利于良好的设计，而 C++对这方面比较灵活，需要开发者自己把握，因此 C++对于初学者把握抽象类与方法更困难一些。

张洋

2010.08.28

leoo2sk.cnblogs.com

[①] C++中的纯虚方法指不含有实现代码的方法，对应于 C#中用 abstract 修饰的方法。

[②] 尽管使用抽象类有时也是为了实现多态性，但其动机完全没有使用接口这么直接，而且我个人更赞同将需要多态的方法抽象成接口的做法。