对于一个具有丰富结构化程序设计经验的程序员来说，面向对象的程序设计可能会给他们带来非常不自然的感觉。封装是实现面向对象程序设计的第一步，封装就是将数据或函数等集合在一个个的单元中（我们称之为类）。被封装的对象通常被称为抽象数据类型。在本文中，我们将详细学习属性的特性。

**封装的概念**

可以把程序按某种规则分成很多“块“，块与块之间可能会有联系，每个块都有一个可变部分和一个稳定的部

分。我们需要把可变的部分和稳定的部分分离出来，将稳定的部分暴露给其他块，而将可变的部分隐藏起来，以

便于随时可以让它修改。这项工作就是封装.

例如：在用类实现某个逻辑的时候，**类就是以上所说的块**，实现功能的具体代码就是可变的部分，而**public的**

**方法或者属性则是稳定的部分。**

**封装的意义**  
封装的意义在于保护或者防止代码（数据）被我们无意中破坏。在面向对象程序设计中**数据**被看作是一个中心的原素并且和使用它的函数结合的很密切，从而保护它不被其它的函数意外的修改。

封装提供了一个有效的途径来保护数据不被意外的破坏。相比我们将数据（用域来实现）在程序中定义为公用的（public）我们将它们(fields)定义为私有的（privat）在很多方面会更好。

私有的数据可以用两种方式来间接的控制。下面我们看一些c#例子来学习这两种方法用以封装数据。第一种方法，我们**使用传统的存、取方法**。第二种方法我们用**属性**（property）。无论我们使用哪种的方法，我们的目标是在使用数据的同时不能使它受到任何的破坏和改变。有如下好处  
  
其一: 使用者只需要了解如何通过类的接口使用类，而不用关心类的内部**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/acmilanvanbasten/article/details/_blank)**和数据组织方法。

其二：高内聚，低耦合一直是我们所追求的，用好封装恰恰可以减少耦合

其三：只要对外接口不改变，可以任意修改内部实现，这个可以很好的应对变化

其四：类具有了简洁清晰的对外接口，降低了使用者的学习过程

**用传统的读、写方法封装**  
让我们来看一个例子有一个类Department,为了操纵这个类中的数据（string departname）我们定义了一个读方法和一个写方法。  
using system;  
public class Department  
{  
private string departname;  
.......

// 读方法  
public string GetDepartname()  
{  
return departname;  
}

//写方法  
public void SetDepartname( string a)  
{  
departname=a;  
}

}  
通过上面的方法，我们可以保护私有数据不被外部程序所破坏。现在我们使用两个不同的方法来写和读数据  
public static int Main(string[] args)  
{  
Department d = new Department();  
d.SetDepartname("ELECTRONICS");  
Console.WriteLine("The Department is :"+d.GetDepartname());  
return 0;  
}  
在上面的例子中，我们不能直接访问类Department的实例d中的私有数据(string departname)，我们只能通过这两个方法来访问。

**用属性来实现封装**  
属性是c#引入的一种语言成分，只有很少的语言支持属性。通过对属性的读和写来保护类中的域。第一种方法体身也是一种好的方式，但用属性来实现封装会更方便。  
现在我们来看一个例子：  
using system;  
public class Department  
{  
private string departname;  
public string Departname  
{  
get  
{  
return departname;  
}  
set  
{  
departname=value;  
}  
}  
}  
public class Departmentmain  
{  
public static int Main(string[] args)  
{  
Department d= new Department();  
d.departname="Communication";  
Console.WriteLine("The Department is :{0}",d.Departname);  
return 0;  
}  
}  
通过上面的例子，我们可以看到如何通过属性来实现封装。属性具有两种操作get和set。Get用来返回属性域的值。Set通过value这个变量来给属性域赋值。属性可以设为只读的(read-only)。这只需属性只具有一个set操作。

**只读属性**  
using system;  
public class ReadDepartment  
{  
private string departname;  
public ReadDepartment(string avalue)  
{  
departname=avalue;  
}  
public string Departname  
{  
get  
{  
return departname;  
}  
}  
}  
public class ReadDepartmain  
{  
public static int Main(string[] args)  
{  
ReadDepartment d= new ReadDepartment("COMPUTERSCIENCE");  
Console.WriteLine("The Department is: {0}",d.Departname);  
return 0;  
}  
}  
在上面的例子中我们看到了如何来实现一个只读的属性。类ReadDepartment拥有一个Departname属性只实现了get操作。它省略了写操作。这个特别的类拥有一个构造器，用来接受一个字符串变量。类ReadDepartmain中的Main方法创建了一个新的对象d。对像d的实例使用了类ReadDepartment带有一个字符串参数的构造器。因为上面的属性是只读的，所以我们不给给域departname赋值并且我们只侧读取此域中的值。当然属性也可以是只写的（write-only），这只需属性只具有一个get操作。

**只写属性**  
  
using system;  
public class WriteDepartment  
{  
private string departname;

public string Departname  
{  
set  
{  
departname=value;  
Console.WriteLine("The Department is :{0}",departname);  
}  
}  
}  
public class WriteDepartmain  
{  
public static int Main(string[] args)  
{  
WriteDepartment d= new WriteDepartment();  
d.departname="COMPUTERSCIENCE";  
return 0;  
}  
}  
在上面的例子中我们看到了如何来实现一个只写的属性。类WriteDepartment拥有一个Departname属性只实现了set操作。它省略了读操作。  
  
总结  
  
封装是朝着面向对象程序设计迈出的第一步。本文向您展示了一些封装的知识。用传统的读、写两种方法可以实现封装，另一种实现封装的方法是使用属性。**使用属性的好处在于对象的使用者可以用一条语句来操作内部的数据。**