# Hello World!

注释符

变量

常量

1)单行注释：[ // ]

2) 多行注释：[ /*所要注释的内容*/ ]

3) 文档注释：[///]多用来解释类或者方法

1) 变量基本概念：用于在计算机内存中存储数据,声明变量需指明数据类型和变量名称,在全局变量无需指

2) 变量语法形式：<type> <name> = value; type：用来存放数据的盒子(数据类型),name：是这

3) 变量使用规则：如果要使用一个变量，必须先为这个数据盒子进行初始化操作，当要从某个数据盒子里做数据盒子的初始化赋值，也叫做变量的初始化赋值。

4) 变量命名规范：首先保证所定义的变量名称是有意义的、变量名称通常以驼峰命名法,我们所自定义的

5) 变量的作用域：变量的作用域一般从声明它的类或者方法内部的花括号到所对应结束的括号结束。

变量的作用范围通常分为全局变量和局部变量。

1) 全局变量：全局变量在使用前所声明地时候可以不用赋值,值类型默认值为0,引用类型默认值为null,

我们可以使用访问修饰符来进行变量数据的访问权限。

2) 局部变量：局部变量在使用前必须为其赋值,否则编译不通过,会导致编译错误,程序无法运行。(返回值boolen类型)int.TryParse(object,out 用于接收转化后的int类型变量)

该静态函数用于将一个任意object类型转换成一个整数类型,转换成功返回true,并传递当前转换的值，int.parse：将一个字符串转换成int类型。

字面值：指的是这个变量所存储的变量值通过字面意思来了解这个变量值是用来做什么的。

常量的基本概念：常量是一个固定无法重新赋值的值类型。用的常量的语法：

const 变量类型 变量名=值;

# 数据类型

值类型

float:用来存储小数;

double:用来存储整数或者小数,使用范围大于float的使用范围;

decimal:一般用于存储金钱数;

字符串类型: string:用来存储多个文本文本，也可以存储空文本，字符串类型的值需要被双引号引起来

布尔类型: bool:在C#中我们使用bool类型来描述对象对或者错，bool类型只有两个值，一个是true,

字符类型: char:用来存储单个字符，最多、最少只能存储一个字符，不能存储空字符，字符类型的值需

# 类型转换

1）隐式类型转换

基本概念：小类型转换成大类型的时候使用隐式转换。我
们要求等号两边的操作数数据类型必须一致。
如果不一致，只要满足下列条件会进行自动类型转换，或者称之为隐式类型转换。数
据类型兼容：例如：int和double（都是数字类型）
目标类型大于源类型：例如：double>int 小的转换成2）

显式类型转换

基本概念：大类型转成小类型时使用显式转换。
数据类型兼容：例如：int 和 double （都是数字类型）源类
型大于目标类型：例如：double>int 大的转换成小的基本语法：
（待转换的类型）待转换的值

3）笔记总结：

自动类型转换：int---->double
型转换：double-->int
对于表达式：如果一个操作数为double类型，则整个表达式可提升为double型。类型
如果相兼容的两个变量，可以使用自动类型转换或者强制类型转换。
但是，如果两个类型的变量不兼容，比如 string 和 int 或者是 string 和 double,这
时候我们可以使用一个叫做 Convert 的转换工厂进行转换。
注意：使用 Convert 进行类型转换时，也需要一个条件，面上必须要过得去，数据类型必须兼容，否则

# 异常处理技术

异常的理解非常容易理解，就是指我们在编写程序的过程中，程序语法上没有任何问题,但是在程序的运行
获并处理异常,通常使用try-catch语法来对程序容易出现异常的地方进行异常处理

异常基本语法

```
try
{
可能会出现异常的代码
}
```

try和catch代码块必须紧密相连，中间不允许插入代码段
catch块可以有多个,根据不同的异常进行不同异常的捕获

```
catch
{
程序出现异常后抛出异常代码段
}
catch
{
程序出现异常后抛出异常代码段
}
final
{
无论程序是否出现异常最终都要被执行的代码
}
```

我们可以自定义异常(自定义的异常必须继承于ApplicationException类)我们可以自定义异常信息。通常使用
系统预留的异常类型，以及我们自定义的异常类型都可以获取当异常发生时，所产生的错误信息，在程序的
执行过程中，如果抛出异常，则程序会进行暂停，通用中间语言CLR会通过搜索查找处理方法,如果可以使用
throw关键字对异常进行抛出,提示用户该异常为什么性质的错误。

catch通常用来捕获异常错误,并进行发布异常信息。

需要注意：（当我们在try代码块、catch代码块、final代码块使用了return我们定义的存储返回值结 系统定义存储返回值结果的局部变量，最终返回值拿到的结果其实就是系统所定义的局部变量这个内容（值

# 流程控制

(if-else)有节制的执行某段代码块：
if-基本语法：（可能会执行一次，也有可能一次也不执行，根据具体逻辑条件进行判断）
if(判断条件(关系表达式))
[逻辑条件判断，满足判断条件，执行if语句所包含的代码块]
{
[满足判断条件，所要执行的代码块]
}
if-else-基本语法：（如果逻辑条件满足执行if-语句包含的代码块，不满足执行else-语句包含的代码块if(判断条件(关系表达式))
[逻辑条件判断，满足判断条件，执行if语句所包含的代码块]
{
[满足判断条件，所要执行的代码块]
}
else(else-语句块只能有一个，不能有多个，且跟它最近的if-语句保持关联性)
{
[不满足判断条件，所要执行的代码块]
}
if else-if-基本语法：(用于区间性的逻辑语句判定)
[if-else-if-语句用于区间性的逻辑条件判断，当if-语句和else-语句都不满足逻辑条件时，会执行el if(判断条件(关系表达式))
[如果逻辑条件满足，执行if-语句包含的代码块，如果逻辑条件不满足，则执行else-if语句包含的代码块
{
[满足判断条件，所要执行的代码块]
}
else if(逻辑条件判断(关系表达式))
[既不满足if-判断条件且不满足else-逻辑条件，所要执行的代码块]
{
}
else

[所有条件都不满足时，所要执行的代码块]
{
}
switch-case：
用来处理多条件的固定取值的范围判断(同于if-else-if，但要比if-else-if程序结构更清晰)
switch-cash的语法结构
switch(要进行匹配判断的值(可以是任意类型))
{
[需注意：在判断值是否匹配成功时，被判断的值必须与匹配条件的值类型必须保持一致]
case 匹配的值：所要执行的代码块;
break;
case 匹配的值:所要执行的代码块;
break;
case 匹配的值:所要执行的代码块;
break;
.....
default:(默认执行的代码块，其他匹配值都不满足条件的情况下执行该代码块)
所要执行的代码块;
break;
}
(自己分析总结)

swtich-case语句执行过程：
当程序执行到switch()处时，首先与switch代码块包含的值进行足一匹配，如果有当前所匹配到的值，则执
循环结构（循环结构基本概念：重复执行相同的代码。）[break(跳出离break关键字最近
while循环while-
基本语法
while(判断条件(关系表达式))
[循环条件，当循环条件成立时执行while循环体，当循环条件不成立时，循环结束]
{
[满足循环条件，所要执行的代码块]
}
循环条件的终止使用break关键字来跳出循环，或者循环条件不成立时终止当前循环(必须有一个索引++，

什么叫做死循环？
while(true)
{
}
这就是一个死循环，没有任何循环终止条件，所以这个循环体会一直执行，除非强制终止这个程序。
break关键字该放在哪里？
break关键字不会单独出现，一般跟if-else来配合使用产生对一个循环结构的影响，break用于跳出循环
continue关键字该放在那里？
continue不会单独出现，也是配合着if-else语句来使用，但循环体内一个代码块某个逻辑条件不成立时do-while循环
do-while语法
do
{
[满足循环条件，所要执行的代码块]
}while(循环条件(关系表达式))
[do-while先执行，后判断，无论循环条件成立与否，都会先执行一次，条件成立继续执行，条件不成立跳出do-while的死循环情况和while的情况一样，循环条件的终止使用break关键字来跳出循环，或者循环条do
{
}while(true)
这就是一个死循环，没有任何循环终止条件，所以这个循环体会一直执行，除非强制终止这个程序。
break关键字该放在哪里？
break关键字不会单独出现，一般跟if-else来配合使用产生对一个循环结构的影响，break用于跳出循环
continue关键字该放在那里？
continue不会单独出现，也是配合着if-else语句来使用，但循环体内一个代码块某个逻辑条件不成立时for循环的使用
for循环语法
for(表达式一；表达式二；表达式三；)
[表达式一：要迭代的变量；表达式二：循环条件(循环条件成立，执行循环体，条件不成立，不执行循环体
{
循环体；
}
程序执行到for-循环处的时候，首先执行表达式一，再执行循环体，执行完循环体，执行表达式三，执行完表
break关键字的使用：和if-else-语句配合使用，用作跳出当前循环。
continue关键字的使用：和if-else-语句配合使用，用作终止当前循环，继续执行下次循环。
return关键字的使用：直接跳出整个循环.
foreach迭代循环结构
其实通过foreach循环遍历数据，实际上是调用了一个"枚举器"来遍历数据，和foreach没有一点关系，for和
如果我们自己写个类想实现通过foreach循环遍历这个类中的某个自定义属性存放的数组，我们可如何使用
枚举器?

1) 将我们需要实现枚举器的类添加IEnumerable接口
该接口中包含的获取枚举器功能函数,该函数需要一个枚举器的返回类型。
2) 创建一个实现了枚举器接口的具体类，实现接口之后进行相对应的业务逻辑编写。
2.1 创建构造函数
2.2 编写接口包含的函数体
2.3 进行测试
不是用枚举器如何实现foreach循环遍历我们的自己写的类中属性数据存储的数组或者集合？
只要方法中包含一个具有IEnumerator返回值并且函数名为GetEnumerator的函数名即可实现免去我们
IEnumerable这个返回值类型可以理解为返回一个数组对象,因为数组也是继承了这个接口,当我们使用f

## 枚举类型(普通枚举的值是互斥的)

枚举是一个自定义类型，跟普通的预定义类型（int,double,string）有区别,区别在于枚举的声明。就当我们在一个游戏中要切换游戏状态或者设定怪物类型或者一个菜单选项，我们都可以使用枚举来设定这枚举
基本语法:

```
访问修饰符 enum 枚举名称
 {
     枚举值,
     枚举值,
     枚举值
}
```

枚举命名规范：遵循帕斯卡命名规则。
枚举声明法则：要将枚举声明到命名空间下，类的外面，使这个命名空间下的所有类都可以访问这个枚举**枚举转换机制**
我们可以使用枚举与int和string类型进行数据互相转换。
与int类型转换：枚举类型默认跟int类型是兼容的，我们可以使用强制类型转换将枚举类型转换成int类当转换了一个枚举中不存在的值的时候，不会抛出异常，而是将要转换的值显示出来。
与string类型转换：
将枚举类型转换成字符串类型：枚举同样也可以转换成字符串类型，只需要调用Tostring().
将字符串转换成枚举类型：调用 Parse()方法就是为了让我们将一个字符串类型转换成一个对应的枚举
实用语法：声明枚举 = (枚举类型)Enum.Parse(C#反射机制typeof(枚举类型),转换的字符串);
如果转换的字符串是数字：即使是枚举中不存在的值，也不会抛出异常。如果转换的字符串是文本：如果是枚举中不存在的值，则会抛出异常。

## 结构体类型（面向过程）

结构基本概念：
可以帮助我们一次声明多个不同类型的变量。
结构体内定义的不叫变量，而叫做字段(新手叫做变量，专家叫做字段)。
结构语法：

```
访问修饰符 struct 结构体名称
{
  _字段
}
```

变量在程序中只能存储一个值，而字段可以存储多个值。

## *程序调试

1)、写完一段程序后，想看一下这段程序的执行过程。
2)、当你写完这段程序后，发现，程序并没有按照你想象的样子去执行。
调试方法:
1)、F11逐语句调试(单步调试)
2)、F10逐过程调试
3)、断点调试

## 数据结构

数组

数组基本概念：可以帮助我们一次声明多个相同类型的变量。数组语法：

数据类型[] 数组名称 = new 数据类型[数组长度];

int []nums = new int[10];

如果你写了上边这样的一段代码，就代表你在内存中连续开了10块空间，我们管每个块称之为数组的元素

如果你想要访问数组中的一个元素，需要通过数组的下标或者索引来进行访问。

如果数组的长度被固定了，无法更改数组的长度。

nums[i]：

对于这句代码的理解方式：

1. 代表当前数组中所循环的元素
2. 代表数组中的每个元素

数组是包含若干相同类型的变量，这些变量都可以通过索引进行访问，数组中的变量称为数组的元素。数组能够容纳的元素称为数组的长度

数组是通过制定数组的元素类型，数组的秩（维数）及数组每个维度的上限和下限来定义的。当一个函数需要传递数组参数的时候

(两种传递方式：

1. 参数数组（params 需在）：在使用这种参数方式时，我们可以同时定义多个数组元素，编译器会自动
2. 数组参数：在调用时，我们需手动创建一个数组

这两种方式不同于在函数的调用)

## 数组的初始化

初始化数组的时候必须要用new .

下面代码是定义了一个一位数组，并通过foreach语句显示该数组的内容

```
int[] arr = {1,2,3,4,5}
foreach(int in in arr)
{
Console.WriteLine("{0}",n+" ");
}
```

二维数组声明：type[,]  arrayName;

动态二维数组：定义部分和初始化部分分别写在不同的语句中.

（二维数组，动态二维数组，一位数组，都是差不多的。）

Rank（得到数组的行数） GetUpperBound((Rank-1)+1)(得到数组的列数)

得到列数 为什么要线-1 再加1呢？

因为 维 是从1开始的，而索引是从0开始的。如果不减去1 那么它将会超出索引，报错。所以必须是先 数组的反序交换

int[] nums={9,8,7,6,5,4,3,2,1,0}; 0 1 2 3 4 5 6 7 8 9

如果当前这个数i大于它前一位的数就与前一位数就进行交换，

if(i>i+1)交换

int temp =i; i=i[length-i-1];

i[length-i-1]=temp;

| | | |
|---|---|---|
| 第一趟比较：8 7 6 5 4 3 2 1 0 9 交换了9次 | i=0 | j=nums.Length-1-i |
| 第二趟比较：7 6 5 4 3 2 1 0 8 9 交换了8次 | i=1 | j=nums.Length-1-i |
| 第三趟比较：6 5 4 3 2 1 0 7 8 9 交换了7次 | i=2 | j=nums.Length-1-i |
| 第四趟比较：5 4 3 2 1 0 6 7 8 9 交换了6次 | i=3 | j=nums.Length-1-i |
| 第五趟比较：4 3 2 1 0 5 6 7 8 9 交换了5次 | i=4 | j=nums.Length-1-i |
| 第六趟比较：3 2 1 0 4 5 6 7 8 9 交换了4次 | i=5 | j=nums.Length-1-i |
| 第七趟比较：2 1 0 3 4 5 6 7 8 9 交换了3次 | i=6 | j=nums.Length-1-i |
| 第八趟比较：1 0 2 3 4 5 6 7 8 9 交换了2次 | i=7 | j=nums.Length-1-i |
| 第九趟比较：0 1 2 3 4 5 6 7 8 9 交换了1次 | i=8 | j=nums.Length-1-i |

数组中的冒泡排序

***需要重点理解***

冒泡排序：就是将一个数组中的元素按照从大到小或者从小到大的顺序进行排列。

nt[] nums={9,8,7,6,5,4,3,2,1,0};

如果当前这个数i大于它前一位的数就与前一位数就进行交换，

if(i>i+1)交换

int temp =i;

i=i[i+1];

i[i+1]=temp;

升序排序法

```
int[] nums = { 1, 7, 5, 25, 1, 24, 4, 55, 2 };
        for (int i = 0; i < nums.Length -1; i++)
        {
            for (int j = 0; j < nums.Length - 1 - i; j++)
            {
                if (nums[j] > nums[j + 1]): 降序排序法只需把>变成<
                {
                    int temp = nums[j]; nums[j]
                    = nums[j + 1]; nums[j + 1] =
                    temp;
                }
            }
        }
        for (int i = 0; i < nums.Length;i++)
        {
            Console.WriteLine(nums[i]);
        }
        Console.ReadKey();
```

## ArrayList类

ArrayList类位于System.Collections命名空间下，它可以动态的添加和删除元素。可以将ArrayLis

数组的容量是固定的，但ArrayList的容量可以根据需求扩充。

ArrayList只能是一维的，数组可以为多维。

## ArrayList的方法：

Add：将对象添加到ArrayList集合的结尾处。Insert：

将元素插入ArrayList集合的指定索引处。Clear ：移

除ArrayList中所有的元素

Remove ： 移除指定对象的第一个匹配项

RemoveAt ：移除指定索引处的元素

RemoveRange ：移除一定范围的元素

## 哈希表

哈希表基本概念：

哈希表属于键值对集合。

在键值对集合中，我们根据键去找值。键

值对对象［键］＝值

注意：键值对集合中 键必须是唯一的，而值是可以重复的

Hashtable 通常称为哈希表，它表示键/值对的集合（键不能为空，但是值可以）

Hashtable 元素的添加

public virual void Add(Object key,Object value)

## 哈希表的查找

Contains方法：

public virtual bool Contains (Object key)

判断哈希表中是否包含这个键

ContainsValue方法:

判断哈希表中是否包含特定值
public virtual bool ContainsValue(Object value)

# 方法(函数)

## 函数的基本概念（函数的命名规范[帕斯卡命名法]）

1☞)(函数就是将一些重复的代码进行封装的一种机制,函数可以认为就是一个程序中的功能模块,函数就

2☞)(函数编写完成,如果没有使用者调用,这个函数将毫无作用。)

3☞)(函数的功能应保持单一性,基于设计模式的单一职责,一个函数只负责处理该处理的事情)

4☞)函数的基本语法

[访问修饰符] 返回值 函数名称 ([参数列表])
{
  这个函数所包含的函数体(也可以认为是这个函数所包含的行为事件);
}

## 返回值的详解

1☞)函数的返回值类型[无返回值写void, 有返回值写对应的数据类型]

2☞)返回值该怎么写？

2.1 ☞)如果需要返回多个相同类型的数据,可以考虑使用一个数组或者集合;

2.2 ☞)如果需要返回多个不同类型的数据,这时候数组就不管用了,我们可以考虑通过使用关键字out进行返回值的类型为值类型的时候· 假定当前函数有返回值,程序编译时,系统默认定义一个局部变量用于存储回,也就是把我们定义的局部变量栈中所存储的值复制一份给了系统定义的局部变量一份。

## 参数的详解

1☞)函数的参数列表(无论是形参还是实参只要定义都会在栈中开辟内存存储参数的值)[参数列表被称为

2☞)形参列表的关键字(out、ref、 params)

2.1 ☞)out:使用out关键字修饰的参数用于返回多个不同类型的值,out参数必须在函数体内进行初始化,

2.2 ☞)ref:使用ref关键字修饰的参数用于传值和取值,ref参数可以不在函数体内进行初始化赋值,调用前操作)

2.3 ☞)params:使用params关键字所修饰的参数必须位于形参列表的末端,params所修饰的参数是一个

## 函数的重载

1☞)函数之间可以重名,但是函数参数(个数、数据类型)不同,就构成了函数的重载;

2☞)函数的重载和返回值没有任何关系;

3☞)函数是否构成重载技巧(一看参数数据类型,二看参数个数);

## 函数的递归调用

1☞)函数的递归调用就是一个函数自己调用自己,但必须有一个终止调用的条件,可以通过声明一个局部变

## 函数的使用方式

1☞)如果是实例函数成员,则通过对象名.函数名称来调用;如果是静态函数成员,则通过类名.静态函数名

2☞)函数的取值和赋值

2.1 ☞)函数的调用者想要得到当前所调用的函数的值,那么就需要通过这个函数的返回值来接受,接收这个

2.2 ☞)当函数被调用时需要取得值,这时候需要通过形参列表来接收传递过来的实参的值,由函数体内的局

## 扩展方法(是指在原有类的基础上增加新的函数内容而不修改原来类中函数的代码)

扩展方法的写法是指在当前类.然后对该类进行一个扩展,扩展的类必须为一个静态类,在静态类中如果要扩展的函数也是可以进行函数重载;
扩展的函数的参数只能指定一个要扩展的类,且该参数必须放在参数列表的首位参数。
扩展函数的类必须和要扩展的类在同一程序集或者命名空间下。
扩展函数的语法:

```
    public static void 扩展函数名称(this 扩展的类 参数名称)
        {
            。。。函数体
        }
```

要注意的是:

扩展方法看起来像是某个对象的函数,当实质上并不是该对象的函数,经过反编译之后,我们可以看到,这个

扩展方法同时也无法调用该对象的私有成员。

扩展方法不能和原有类中的函数产生重载性质，编辑器不知道调用的是哪个函数。

# 面向对象

面向过程----->面向对象

面向过程：(用白话说就是人的思维过程 用人的思维考虑程序如何去写)面向的是完成这件事儿的整个过程，强调的是完成 面向对象：着在写出一个通用的代码 屏蔽差异。

封装可以保护我们的数据安全访问性(可以使用属性封装 也可以使用函数封装 封装的手段有很多 需要在以后书写代码时 继承可以减少我们的代码重复性 可以抽离出一个父类模板 来进行对多个抽象对象之间的成员变量进行统一规划设计，降低多态可以实现多个类之间共同的函数成员拥有不同实现方式；

对象查找的诀窍：一般被动词为对象，对象必须是看的见摸得着的。

我们把这些具有相同属性的和相同方法得对象进行进一步的封装，抽象出来类这个概念。类就是个模子，确定了对象应该具对象是根据类创建出来的。

写好了一个类之后，我们需要创建这个类的实例对象。

当我们使用类创建了一个对象之后 我们需要对这个对象的属性进行赋值，这个过程叫做对象的初始化。那么 我们管创建对象的过程叫做类的实例化 (使用关键字 new)。

this:表示当前类中的数据成员和函数成员。

base:表示父类中的数据成员和数据成员。类是不占内存的，而对象是占内存的。

类的概念：

同类型对象的共同特征的组合 类是抽象的 只是一个模版而已 一个类模板创建完成时 会有一张类成员表单存储在内存 用软件开发中共同拥有的属性(特征)和方法(行为)在一个单元模板中，就称为类模板，一个类模板就是一个类模子，类不类的

简单理解化：

类规定能存储的数据类型有哪些？能够执行的方法或能够完成的任务是什么？

设计类就像设计了一套模板 把一些抽象对象的特征和行为抽离出来，形成一套共有的属性和方法，进行重复使用，减少了代 设计一个类模板之前 可以使用类图来规划类的设计。

根据具体的需求进行系统的分析类中包括哪些数据类型，包括哪些成员变量，包括哪些实例成员，包括哪些静态成员。类的定义仅仅是一个开始，具体实现是放在对象中。

类是一种引用类型，包含一系列数据成员，函数成员。

# 判断两个对象是否相同

基本概念：

在程序中我们需要判定一个或者多个对象是否为相同对象 我们需要通过 .Net类库中所提供的三种手段来判断对象与对象之如果是值类型的两个对象 变量名称不参与判定 我们需要判定的是这两个对象所包含的值是否相同 并通过vs的即时窗口来如 果是引用类型的两个对象引用 我们需要观察即时窗口 我们通过观察调试这两个引用类型所指向堆中的内存是否为同一 1)通过Equals来判定两个对象所包含的值是否相同：基本语法：bool b=Equals.(object o,object o)、通过==来 回true，否则返回false)

2)通过bool b=ReferenceEquals(object o,object o)来判定两个对象所指向引用的实例是否相同(唯一准确无误地

==符号是用来比较两个对象所包含的值是否一样，

object.ReferenceEquals() 和 Equals()函数的默认实现都是比较两个对象的地址引用是否是同一个地址

string 类型是特殊的引用类型。

# 静态成员和非静态的成员

在一个实例类中 既可以有实例成员 也可以有静态成员。在调用实例成员时 我们需要通过对象名点成员变量；

在调用静态成员时，我们需要通过类名点静态成员变量；总结：

静态成员必须通过类名点成员名称去调用 实例成员必须通过对象点成员名称去调用。静态函数中 只能访问静态成员 无法且不允许访问实例成员。

实例函数成员中 即可以访问实例成员 也可以访问静态成员。静态类中只允许静态成员的存在，不允许实例成员的存在。

如果你想要把你的类变成一个工具类去使用，这时候可以考虑把类写成一个静态类。

静态类在整个项目中做资源共享的使用 比如说多个成员之间存在一个共同特性 比如银行的存储利息 当存储利息发生改变静态类直到程序结束才会去释放资源，静态成员在计算机内存中存储在一块单独的内存区域，称之为静态存储区。

# 构造函数

1) 实例构造函数基本概念

作用：帮助我们初始化对象 (给对象每个属性依次的赋值)

构造函数是一个特殊的方法 (构造函数没有返回值，连void都不用写，构造函数必须跟类名一样命名) 建对象时会执行构造函数

new关键字：new帮我们做了三件事：

1. 在内存中开辟一块空间
2. 在开辟的内存中创建对象

3. 调用对象的构造函数进行初始化对象

2) 静态构造函数基本概念：

静态构造函数只会且只执行一次(无论在静态类或者实例类)；

在第一次使用静态类或者静态成员变量时，会使用静态构造函数。

如果一个非静态类中有静态成员，则编译器会默认为我们提供一个无参数的静态构造函数；一个类中的静态函数成员在首次调用静态类时进行静态函数初始化工作。

静态构造函数不需要指明访问修饰符，也不允许我们为静态构造函数进行添加访问修饰符，因为静态的构造函数是由系统来 静态构造函数不需要任何参数传入；

静态构造函数的调用顺序优先于实例化构造函数。

## 析构函数

析构函数基本概念：

程序结束时才会被执行。

作用：帮助我们快速的释放资源。语法：

```
~using()
{
代码块(用于程序结束之后所要释放的对象)
}
```

## 命名空间

基本概念☞可以认为类是属于命名空间的。

如果在当前项目中没有该个类的命名空间，需要我们手动导入该个类所在的命名空间。在一个项目中调用另一个项目的类：添加引用程序集或者项目引用名称。

## 值类型和引用类型

值类型不可以由其他类来继承，也就是说值类型不能派生任何类型，并且值类型也不允许继承自其他类型。

值类型在使用的过程中，如果没有提供方式修改值类型所包含的值，那么值类型所包含的值是不会被修改的，要与string区

1) ☞值类型和引用类型在计算机内存中存储的地方不一样(值类型存储在栈中)

值类型包含{int、double、float、bool、char、decimal、struct、enum}。值类型在函数之间进行参数传递的时候，则传入的值只能在该函数的局部改变；

值类型传递参数的时候如果使用ref关键字，则将值类型当作引用类型来看待，就会将传入的值类型的值发生改变。

2) ☞在传递值类型和引用类型的时候，传递方式不一样(引用类型的物理内存路径存储在栈中，引用类型的实际数据存储在堆中)

引用类型包含{string、Array、系统类库预制的类(包括我们自定义的类)}

引用类型在函数之间进行参数传递的时候，只要传入参数，该个引用变量的值就会发生改变，前提条件必须是加上ref，否则值类型拷贝的是栈中的值(当对一个函数中的形参列表进行参数传递的时候，如果参数列表不添加ref、out关键字在函数内引用类型拷贝的是栈中的地址。

*

**引用类型和值类型都具有值传递和引用传递(参数传递默认就是值传递，加上ref就代表引用传递(ref既可以入栈，也可以出**

*

1）值传递(说白了就是实参从形参那拷贝一份栈中的内容，如果形参已经在堆中开辟了一块新的内存空间，则实参保持原来

{☞"值传递"指的是普通传递，传递的是栈中的内容，是将栈中的内容拷贝了一份给传递过来的值，将两个栈中存储的变量指 值类型进行值传递·

当一个函数传入一个值类型的实参市，这个函数的形参会拷贝这个实参的栈中的值；引用类型进行值传递·

当一个函数传入一个引用类型的实参对象时，该个函数的形参会拷贝该个实参的栈中物理路径地址，并和当前传递过来的实当该个函数内的形式参数没有在堆中开辟一块新的内存空间，则实参和形参所指向的是同一块堆中的空间，则传递过来的实 当该个函数内的形式参数在堆中开辟了一块新的内存空间，则形参执行新的一块堆中的空间，则传递过来的实参对象跟形参2)

引用传递(实参和形参虽然两个参数的变量名称不同，但是所存在的栈内存地址相同，并指向同一块堆内存空间，当一个参

{☞"引用传递(使用ref关键字)"传递的是栈中的地址，是将栈中的地址拷贝了一份给传过来的值}

值类型引用传递☞使用ref时传递的实参和形参其实就是同一个变量，两个不同变量名称，指向的是同一块栈中的内存空间。当一个函数传递过来实参的时候，是将这个实参栈中的物理地址拷贝了一份，使这个函数内的形参跟传递过来的实参指向同

引用类型引用传递

当一个函数传递过来一个实参对象的时候，如果该个函数的形参对象在函数中创建了一个新的实例对象，则将传递过来的实 当个函数传递过来一个实参对象的时候，如果这个函数的形参对象没有创建新的实例对象，则将这个函数的形参对象和传 引用类型案例解析·

如果一个函数中的参数列表为一个对象参数时，当某一时刻外部传进来一个跟该对象参数同类型的对象时，该个函数内没有 果一个函数中的参数列表为一个对象参数的时候，在某一时刻外部传入进来一个跟该对象同类型的对象时， 如果这个函数无法完成引用类型的传递。

## 装箱和拆箱

1) 具备父子关系的才能进行装箱和拆箱。

2) 装箱：把值类型隐式转换成引用类型(当一个子类传入一个函数的参数中，如果这个函数需要传入的参数是一个object

3) 拆箱：把引用类型强制转换成值类型(当一个父类强制转换成子类类型的对象时，并赋值给一个新的子类变量时，这时候

4) 装箱和拆箱是一个非常消耗性能的操作，装箱就是把一个子类包含的值，在堆中开辟了一块新的内存空间，使这个子类的

## 浅拷贝和深拷贝

浅拷贝是指一个对象指针拷贝了另一个对象指针堆中内存空间的引用，这两个指针共同指向堆中的一块内
深拷贝是指一个对象指针拷贝了另一个对象指针堆中内存空间的引用，如果在拷贝的过程中，拷贝者对象指
的对象引用之后又重新自己又重新使用拷贝过来的引用，在堆中又开辟了一块新的内存空间，表示深拷贝）深拷
贝是指将原对象指针所指向堆中的对象所包含的值拷贝了一份给新的对象指针，虽然是拷贝了一份，但深拷
贝和浅拷贝的用处比较多
深拷贝（比如说一个对象存放着一些数据，另一个对象也想引用这些数据，并且引用过后将对数据进行修改建
一个实例对象来对数据进行深拷贝，深拷贝之后通过监视面板我们观察堆中的内存地址与原对象地址发 浅拷
贝（比如说一个对象存放着一些数据，另一个对象也想引用这些数据，引用之后这些数据不进行修改，可以考虑进行对对

## 可空值类型？

int?在值类型变量后边加上一个？表示可以为一个值类型赋值为null的处理，可空值类型在C#内部的一个泛型类NULL
null值内部处理为null值。

# 面向对象之继承

所有的类都间接或者直接地继承于Object类。
我们可能在不同的类中写重复的成员变量 如果一个基类所继承的多个派生类持有共同的属性和方法 我们可以把这些重
我们可能在多个类中同时定义一些同一种类型 同一种意义的属性或者方法，如果能把这些类中所存在的共有属性和特征，类，也
叫做父类，而扩展类称作为派生类或者叫做子类。
子类继承了父类之后，则是父类的派生类，那么子类从父类那里继承了什么？
子类继承了父类的属性的和方法 但是子类并没有继承父类的私有字段，同时也没有继承父类的构造函数，但是，创建子类无
参构造函数 再执行子类的构造函数。
当我们在父类定义了一个新的有参数的构造函数，会把父类默认存在的无参数构造函数替换掉，使之不复存在，子类因此就解
决方案：
1. 在父类中重新写一个无参数的构造函数，供子类调用父类中的无参数构造函数。
2. 在子类中显示调用父类的构造函数，使用关键字：base()来调用父类中的无参数和有参数的构造函数。继
承的特性：
继承的单根性：单继承（在C#中没有多重继承的概念，只有多接口的概念）
继承的传递性（祖宗的识早是我的。）：子类与父类之间存在一种传递性的关系，子类可以继承父类的属性和方法，同时也学
会使用查看类图可以更清晰的把类之间的继承关系一目了之。
我们可以通过类图来更好地设计一个继承关系，使用类图可以使我们编写程序的思路和意图更为清晰，学会适用类图来更好
隐藏父类中的函数成员：
使用new关键字：当一个子类继承了一个父类后 如果子类中存在了和父类中同名的函数，可以使用new关键字把父类中已使
用后果：子类所创建的对象无法调用到与父类同名成员的

## 里氏替换原则

里氏替换原则基本概念：
当一个派生类继承了一个基类之后 当我们需要创建一个对象的时候 我们可以使用一个基类川籍一个派生类的对象 也就有
的时候 我们调用一个函数时 我们需要传递一个object类型的参数 我们可以使用任何一个间接或者直接继承自objec也可
以认为：子类可以赋值给父类（如果有一个方法需要一个父类作为参数 我们可以传递一个子类对象）
如果父类中存储着一个子类对象 那么可以将这个父类显式转换成一个子类对象。子
类对象可以任意调用父类中的成员变量，但是父类对象无法使用子类对象的成员变里
斯转换判定：
1. is：一个对象与另一个对象做对比，如果可以转换成功，返回true，否则返回false。
2. as：使用as关键字可以使一个对象强制转换成另外一个对象引用，转换成功则返回对应的对象，转换失败返回null，不访
问修饰符：
protected:该修饰符用于当前所在类中的成员变量只能在当前类中或者继承自该类的对象的访问当前类中的数据成员、函

# 面向对象之多态

## 1☞)虚方法(virtual)

「当一个或者多个类有着共同的特性，我们可以通过继承关系，抽象出一个基类，基类中函数成员有具体实现（当我们可以确定类
中的虚函数，多个具体类实现这个基类中的虚函数，可以减少代码的重复性，目的是为了实现多态。]

## 2☞)抽象类(abstract)

抽象成员必须标记关键词abstract，抽象成员必须在声明定义在抽象类内，并且抽象类本身不可以有任何实现，且不能被实例
基类中可以包含虚函数成员，虚函数成员可以不被派生类重写。
派生类继承抽象类后，必须重写父类中的抽象成员（如果子类是抽象类，则可以不用重写）。
如果基类中的方法有参数，则继承基类的派生类在重写基类方法时也必须与基类的方法签名保持一致。
如果父类中的方法没有默认实现（不能确定父类所需要实现的默认方法的时候） 且父类也不需要实例化 则可以将该父类里
氏替换原则：抽象类的定义主要目的就是为了实现面向对象的多态性，抽象类如果是基类，不能创建实例对象，而是通过派

## 3☞)接口(interface)

表示接口的字面意思(大写I开头able(表示一种能力)收尾,使人从字面意义上直接了当的看出这是一个接口,而不是一个普访问修饰符 iInterface 接口名称(I.....able)
{
接口的成员变量[方法-属性-索引器-事件]
}

### 1) 接口的基本的概念:

接口其实是一个更为抽象的模板 一个规范 一种能力 一种定制的协议 定义接口是为了形成一套程序规范,目的是为了 接只参与程序功能的设计,不参与程序功能的具体实现方式,具体实现交由某个具体的对象去实现。

### 2) 接口的多态概念

接口实现多态的方式和抽象类的抽象方式大致含义上差不多,但比抽象类更为抽象,可以把多态实现的更好。
接口解决了不同类型之间的共有特征行为(比如鱼和船只都会在水里游,具体游的方式不一致,但又提取不出来一个父类来实里氏替换原则:接口不能使用new关键字,也就是接口不能被实例化创建接口对象,只能通过继承了这个接口的具体实现的来声明一个接口的指针引用,通过创建具体实例对象,来实现接口的功能。

### 3) 接口的设计

接口并不能去继承一个类,而类可以继承接口。(接口只能继承于接口,而类可以继承接口,也可以继承类)接口不能用于存储数据。
接口中不允许声明定义字段,接口中只能有[方法-属性-索引器-事件],不能有"字段"和构造函数。
接口中可以定义自动属性(get; set;不包方法体的属性叫做自动属性) 但是非自动属性不可以定义 只能定义成为未实现的接口中可以定义索引器(returnType this[int index]{get;set;}),不能有具体实现,只能定义成为未实现的索引器 接口不能使用修饰符来修饰成员 接口的成员默认修饰为public且不能被修改。
接口的函数成员不允许我们定义函数体接口中的成员不允许有任何实现,只是形式上定义了一组未实现的成员,也就是接口

### 4) 实现接口

当一个其类实现了某个接口,这个其类的派生类同样也可以实现该接口(指明继承关系 实现接口) 如果派生类想再次完成接一个类可以同时继承一个类并实现多个接口,如果一个子类同时继承父类A,并实现了接口IA,那么语法上,作为类必须写在实 现接口的子类必须实现接口全部成员。
接口被一个具体实现的类继承的时候 这个类就必须实现这个接口中所有的成员变量,否则编译无法通过。接口与接口之间可以存在多实现(继承)关系,也就是一个接口可以实现(继承)多个接口。

### 5) 显示实现接口

显示实现接口的目的和作用(显示实现接口后,这个接口的所定义的所有函数访问修饰符都为private):解决方法的重名问当具体类实现了这个显示接口,具体使用该接口定义的函数时候,必须通过接口名点函数名。
使用时段:当一个类所实现的多个接口中的函数签名相同的时候,要使用显示来实现接口。
当一个抽象类实现接口的时候,抽象类不需要实现这个接口,而是需要通过这个抽象类的派生类去具体实现接口。

### 6) 使用接口的建议

面向对象的编程:要使用抽象(父类、抽象类、接口)不直接使用具体类来设计程序,而实现方式通过具体类来进行实现。梯次向上转型(越抽象越好):接口——>抽象类——>父类——>具体类(在定义方法参数时、返回值、声明变量的时候能使用抽象的话使用接口编程,就不要使用抽象类编程,能用抽象类就不要使用父类编程,能使用父类就不要使用具体子类。
避免定义"体积庞大的接口"、"多功能接口",这样会造成"接口污染"。只把相关联的一组成员定义到一个接口中(尽量在接定义多个单一职责的接口(小接口、组合各个接口之间的使用[设计模式中的组合行为(通过组合行为接口与接口之间可以不接口的更多好处需要在以后深入的学习更多关于接口的使用方式才能慢慢体会理解。

## 4☞)泛型

泛型集合修改数组的默认长度不可更改性质,泛型集合对存储数据更灵活

1. 泛型类的定义

在C#中 泛型指的就是可以泛滥地类型,可以使用任意类型来进行数据操作,通常由一个<大写字母>来代表C#的语法形式,泛型类定义的约束:<T>where{struct(值类型),class(引用类型),也可以传入自定义类(或者自定义类的子类)},每一 的泛型参数

2. 泛型方法的定义:

定义泛型的方法时 需要在方法名的后边使用<大写字母>来声明 如果有多个参数签名,这时需要使用逗号隔开,同时参数 需要注意的时,在定义泛型方法的时候,定义的大写字母尽量不要和定义类的时候声明的大写字母保持一致,如果一致,编译

## 队列(Queue)[FIFo 先进先出]

## 栈(Stack){先进后出}(就象薯片盒子一样 放薯片的时候 一片片往里边放,取薯片的时5☞)

## 委托(委托是常规对象,委托是一种类似于类的引用类型,用来封装带有特定签名和返回 编世界里,经常面对类似的情形:要执行某想操作,但无法提前知道应该调用哪个方法或对象去执行。委托联系起来 然后在程序运行时再决定委托给哪个具体方法。
事件是对程序必须影响的"事件"这一概念的封装。事件和委托是一对紧密联系的概念,因为灵活的事件外 委托可以用作回调,这样一个类可以告诉另一个类:"执行这个操作,干完后通知我"。
(委托和字符串一样都有不可变性质,也就是声明定义委托之后不能对委托赋的初始值进行改变,如果重新象,对于需求的变化实现不同的功能,无论是客户程序调用还是自己在调用的过程中一个函数可以实现不同

委托的声明就像定义一个类一样.具有访问修饰符.委托的标识是delegate关键字.使用委托可以将函数当做
委托实例时.需要注意的是.当有一个函数赋值给委托引用时.这个函数的函数签名必须和委托的委托声明委
托类型时,使用delegate关键字,后跟一个返回类型和可以委托给它的函数的签名。
定义委托之后,通过实例化委托的方式, 传入与委托签名相匹配的函数。
C#类库中有两种预定义委托
(1、Action(普通委托),Action<T,T>(泛型委托),不论哪种委托,这个委托都没有返回值)
(2、Func<T,out result>(泛型委托).T代表参数.可以指明多个(最多支持16个).out result.表示 多播委托
(创建一个新的委托实例时.可以使用一个委托对象引用指向多个需要委托的函数.使用+=表示添
名.GetInvocationList()得到当前这个委托实例所包含的所有委托函数,返回一个委托集合,通过遍历 如
果委托函数有参数,则需要传入参数
)
匿名函数
(创建一个委托实例时.可以不用函数赋值.可以像对象初始化器一样.但是不是new关键字.而是delegate需要
而匿名函数必须有),匿名函数的作用,当有一个独立功能的且不确定名称的函数时,这时候可以考虑6☞
Lambada表达式)
Lambada表达式是一个升级版的匿名函数.如果一个函数不知道如何命名,并且功能单一.就可以使用Lamb
Lambada表达式可以在赋值的时候无需指定数据类型,即可通过一个假定的变量名来为一个委托对象实例
Lambada表达式语法
Func<T,out Result> 委托实例名 = (不需要delegate关键字)假定变量名(任意名字)(参数列表) Lambada
表达式的函数参数在函数体内可以与外部变量进行运算,如果是值类型直接运算结果,如果是字符混乱,为
程序带来严重的后果,尽量避免这么做)
## 7☞事件Event)
(委托可以通过事件来进行访问权限设置.通过事件可以控制委托触发函数的合理性.以至于程序不会出现
件为后 就可以群发布事件.事件就好像一个报社.由报社发出报纸.由人订阅报纸.简化为发布订阅。
事件中包含两个方法add和remove类似与属性中的get和set.这两个方法用于添加函数事件和移除函数时为
么事件无法在外部访问?经过反编译中之后我们可以看到.经过编译之后实质上把事件最终编译成一 事件可
能是一次操作界面的按钮按下.一次菜单选择.一次文件传输的完成。
简言之.就是有事情发生,而我们必须对此做出响应。我们无法预测时间发生的顺序.系统一般处于安静状
事件的发布和订阅·
在C#中任何对象都可以发布一组事件供其他类订阅。当发布类产生事件时.所有订阅类都会得到通知。和此
感兴趣的观察这类.这里按钮就被称作一个发布者.因为它们发布了事件.而其他类称为订阅者.因为它们观
察者设计模式·这种设计实际上实现了发布订阅模式.该设计模式的意图是定义了一种对象之间的一对
需要注意的是：在注册委托事件的时候,如果要通过委托事件集合拿到委托类型返回值的结果的值
## 8.关于委托的函数回调
使用回调方法就可以实现委托工作然后在完成后获得回调的目标.其实回调本质也是通过委托实现的.在N实
现原理·用户需要请求委托的调用列表.然后调用该列表中所有委托的BeginInvoke函数。(BeginInv当方法进
行回调时.Net框架传入IAsyncResult对象,BeginInvoke的第二个参数是我们的委托实例对象它转换为我们
的原委托类型。
转换完毕则将使用转换完成之后的委托调用EndInvoke()函数 传入接收到的作为参数的IAsyncResult对象的
好处是通过委托对象异步调用所注册的委托事件.如果一个委托事件使用线程被占用了.则下一个如果在委
托事件的使用过程中.某一个委托事件使用了线程.如果不通过异步委托调用委托事件.则其他委反之.通过
异步委托调用委托事件当一个委托事件执行完毕,则会立刻执行下一个委托事件,不会等待上一使用异步委
托进行委托事件集合遍历的时候
需要使用· 委托对象实例. BeginInvoke(new AsyncCallback(回调函数) , 委托实例对象);[进行异步回调函数
需要注意的事项

```
private void ResultReturned(IAsyncResult iaResult)[用于接收委托回调的函数]
{
    委托类型 委托实例名 = (委托类型) iaResult . AsyncState;
  调用委托函数取得委托返回结果
}
```

# LINQ数据查询

LINQ用做数据查询;

LINQ表达式-语法

(使用枚举迭代器接收结果)IEnumerable<T> result = from 迭代变量名称 in 要查询的集合
(查询条件) where 迭代变量包含的属性>
(查询结果并返回)select 迭代变量名称;

lambada表达式-语法

(使用枚举迭代器接收结果)IEnumerable<T> result=要查询的集合.Where(迭代变量=>迭代变量.属LINQ联合查询(如果想多个集合同时查询相同的一条数据,则可以选择使用联合查询)

(使用枚举迭代器接收结果)IEnumerable<T> result = from 迭代变量名称 in 要查询的集合

(查询条件) where 迭代变量包含的属性>
(查询结果并返回)select 迭代变量名称;
(查询结果返回多条数据需要创建一个对

LINQ联合查询(复杂语法,反编译之后还是跟普通方法一样,没有区别)

var result = 查询的集合.SelectMany(拟定指向的集合变量=> 所要联合的集合, (查询的集合,
=> new { gaoshou, kongfu })(多集合联合查询需要创建一个新实例) (过滤
查询) .Where(
新实例名称=> 新实例名称.属性.属性所包含的数据== 新实例名称.属性.属性所包
&& 新实例名称.属性.属性所包含的数据 > 95);

LINQ查询的排序(orderby 迭代变量名.属性 descending[只加 orderby是从小到大正序排,两个都加LINQ查询的排序扩展方法:.OrderBy(),.ThenBy()

## 字符串(String类库)

1) 字符串的不可变性质:字符串是不可变性的。
当你给一个字符串重新赋值时，旧值并未被销毁，而是开了一块新空间存储新值，我们不需要知道如何清理这个垃圾，因为我们可以把字符串看成是一个char类型的只读数组。
2) 字符串对象池:
不要直接使用字符串常量来存储大量的字符串,会大量的消耗堆中的对象池内存空间,导致程序运行效率降低,应该将文本信我们声明定义了一个字符串常量的时候,系统默认会把该字符串常量添加到字符串的字符池中,供以后的字符串来使用;每次声明定义了一个新的字符串常量,字符串对象池首先会通过字符池搜来寻找有没有相同内容长度的字符对象,如果有了 字符串是引用类型,在程序中会存在大量的字符串对象,如果每次都创建一个新的字符串对象,会比较浪费内存,性能低下,因当我们使用一个或者多个进行字符串变量名称来拼接字符串的时候,拼接完成的新字符串,不会添加到字符串池塘中,则会在 3) 字符串对象的释放阶段:字符串对象通常在程序退出或者关闭时,进行资源释放。
4) 字符串类添加了sealed不可以被继承的关键性质:子类如果继承了字符串类,当继承了字符串后,可能对字符串的特性 的运行效率和性能。
5) 拼接字符串的时候建议使用StringBuilder 6)
字符串的使用
ToCharArray()将字符串转成char数组
new string (char[] char):能够将char数组转换成字符串
.length:方法获得当前字符串中所包含的字符个数
ToUpper():将字符串转换成大写形式
ToLower():将字符串转成小写形式
string.Split():主要用于字符串的分割,以什么指定字符分割;
Equals()比较两个字符串是否相同,相同返回为 true ,不相同返回false.e
Format:主要用于将字符串数据格式化成指定的格式。
Substring主要用于截取字符串中指定位置和指定长度的字符。
Insert:主要用于向字符串任意位置插入新的字符元素。
Remove 主要用于删除字符串中指定位置及以后的字符串 以及指定要删除的字符。
string.Copy: 将一个字符串复制到另一个新的字符串中。
string.CopyTo:将一个字符串中的某一部分复制到另一个字符数组中IndexOf:
判断在字符串中出现的第一次位置,有的话返回1,没有的话返回-1。
LastIndexOf: 判断在字符串中最后一次出现的位置,有的话返回1,没有的话返回-1。
StartsWith:判断以。。。。。。。。开始
EndsWith: 判断以。。。。。。。。结束Replace():
将字符串中某部分替换成新的字符。Contains():
判断字符串是否包含指定的字符。Trim():去掉字
符串中前后的空格;

string.IsNullOrEmpty():判断当前字符串是否为空值
string.Join():将数组按照指定形式隔开"|"

## StringBuilder

使用StringBuilder创建字符串对象,创建完成之后,字符串的性质是可以改变的,可以把StringBuilder当作一个对象池,我们读取一个文本文件的时候,可以考虑使用StringBuilder来接受读取到的文本数据,这样会节省程序读取文本文件的当我需要拼接字符串对字符串的内容进行修改时,可以考虑使用StringBuilder来进行操作。

StringBuilder的使用好处,效率高处理快.

### 文件操作(常用类库)

File:操作文件(静态类,对文件整体操作 拷贝、删除、剪切等)
Directory:操作文件目录(静态类), 文件夹
DirectoryInfo(操作文件目录下的一个类(用来描述一个文件夹对象,用来获取文件目录信息)) 返回一个DirectoryIn
Fileinfo:用来获取一个文件夹对象,指定获取一个文件目录下的所有文件,返回一个Fileinfo数组
Path:对文件路径或文件目录路径进行操作
Steeam(文件流(抽象类))
1)FileStream(文件流) MemoryStream(内存流) NetworkStream(网络流) 2)StreamReader(快速读取文本文件(适用于操作文本文件, 只适用于文本的读取))
3)StreamWriter(快速写入文本文件(适用于操作文本文件, 只适用于文本的写入))

对于文件的拷贝:先通过读取文件原路径位置,然后使用文件流进行读取,读取完毕,将读取到的实际数据在写入流,通过流加