程序的本质就是数据加算法

一切皆地址（本质上都是地址 地址存储在内存中）

- 变量（数据）是以某个地址为起点的一段内存中所存储的值
- 函数（算法）是以某个地址为起点的一段内存中所存储的一组机器语言指令

直接调用和间接调用

- 直接调用：通过函数名来调用函数，CPU通过函数名直接获取函数所在的地址并直接开始执行->返回
- 间接调用：通过函数指针调用函数，CPU通过读取函数指针存储的值获取函数所在的地址并直接开始执行->返回

委托（delegate）是函数指针的升级版

委托的简单使用

- Action委托
- Func委托（C#自己定义好的）
- delegate 自定义委托

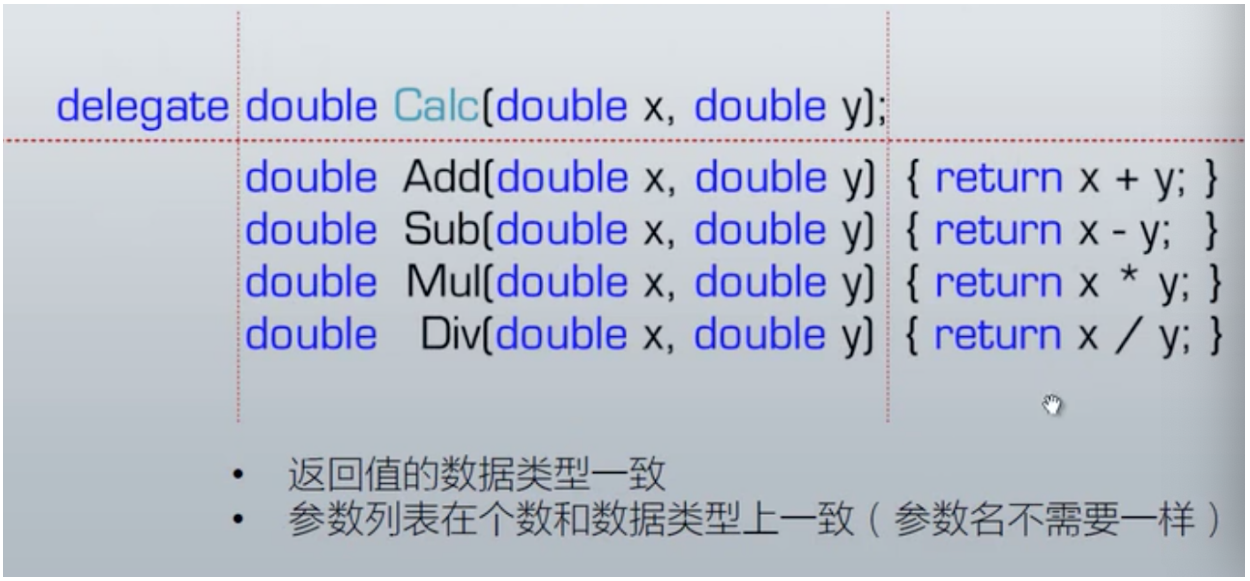
```
class Program
    {
        static void Main(string[] args)
        {
            Cal cal = new Cal();
            Action action = new Action(cal.Report);
            cal.Report();
            action.Invoke();

            Func<int, int, int> fuc = new Func<int, int, int>(cal.Add);
            Func<int, int, int> func1 = new Func<int, int, int>(cal.Sub);
            Console.WriteLine(func1(1, 2));
        }

        class Cal
        {
```

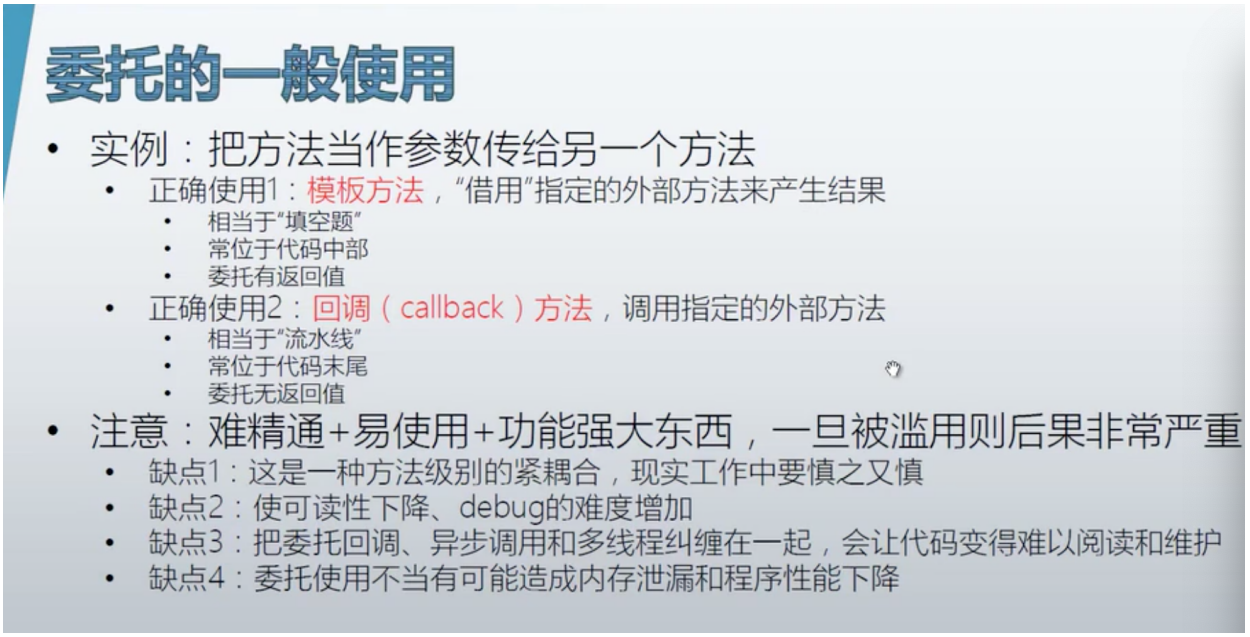
```
        public void Report()
        {
            Console.WriteLine("I have 3 methods");
        }
        public int Add(int a,int b)
        {
            return a + b;
        }
        public int Sub(int a, int b)
        {
            return a - b;
        }
    }
}
```

委托的声明（自定义委托）

- 委托是一种类（class），类是数据类型多以委托已是一种数据类型
- 它的声明方式与一般类不同，主要是为了照顾C/C++传统
- 委托与封装的方法必须类型兼容
- 注意声明位置 （嵌套类）

```
class  Program{
public delegate doubul Calc(doubule x, double y)
Program.Calc();--嵌套类
}
```





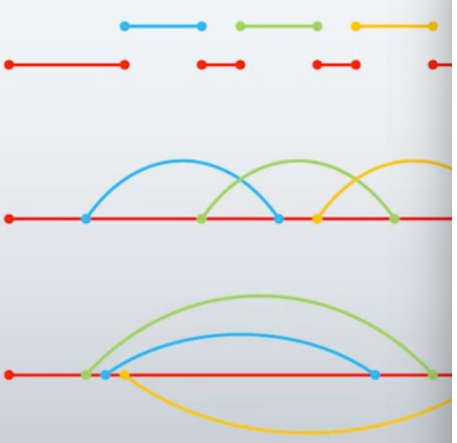
模板方法

```
class Program
    {
        static void Main(string[] args)
        {
            ProductFactory factory = new ProductFactory();
            WrapFactory wrapFactory = new WrapFactory();

            Func<Product> func1 = new Func<Product>(factory.MakePizza);
            Func<Product> func2 = new Func<Product>(factory.MakeToyCar);

          Box box1=  wrapFactory.WrapProuduct(func1);
          Box box2=  wrapFactory.WrapProuduct(func2);
            Console.WriteLine(box1.Product.Name) ;
            Console.WriteLine(box2.Product.Name) ;
        }
      class Product
        {
            public string Name { get; set; }
        }
        class Box
        {
            public Product Product { get; set; }
        }
```

```
class WrapFactory
{
    public Box WrapProuduct(Func<Product> getProduct)
    {
        Box box = new Box();
        Product product =  getProduct();
        box.Product = product;
        return box;
    }
}
class ProductFactory
{
    public Product MakePizza()
    {
        Product product = new Product();
        product.Name = "Pizza";
        return product;
    }

    public Product MakeToyCar()
    {
        Product product = new Product();
        product.Name = "ToyCar";
        return product;
    }
}
```

委托的高级使用

- 多播（multicast）委托
- 隐式异步调用
  - 同步与异步的简介
    - 中英文的语言差异
    - 同步：你做完了我（在你的基础上）接着做
    - 异步：咱们两个同时做（相当于汉语中的"同步进行"）
  - 同步调用与异步调用的对比
    - 每一个运行的程序是一个进程（process）
    - 每个进程可以有一个或者多个线程（thread）
    - 同步调用是在同一线程内
    - 异步调用的底层机理是多线程
    - 串行==同步==单线程，并行==异步==多线程
  - 隐式多线程 v.s. 显式多线程
    - 直接同步调用：使用方法名
    - 间接同步调用：使用单播/多播委托的Invoke方法
    - 隐式异步调用：使用委托的BeginInvoke
    - 显式异步调用：使用Thread或Task
- 应该适时地使用接口（interface）取代一些对委托的使用
  - Java完全地使用接口取代了委托的功能，即Java没有与C#中委托相对应的功能实体



}