[1. Linq 中的扩展方法 2](#_Toc427075385)

[1-1. 实例 2](#_Toc427075386)

[1-1-1. 准备工作 2](#_Toc427075387)

[1-1-2. 用Where来筛选数据 3](#_Toc427075388)

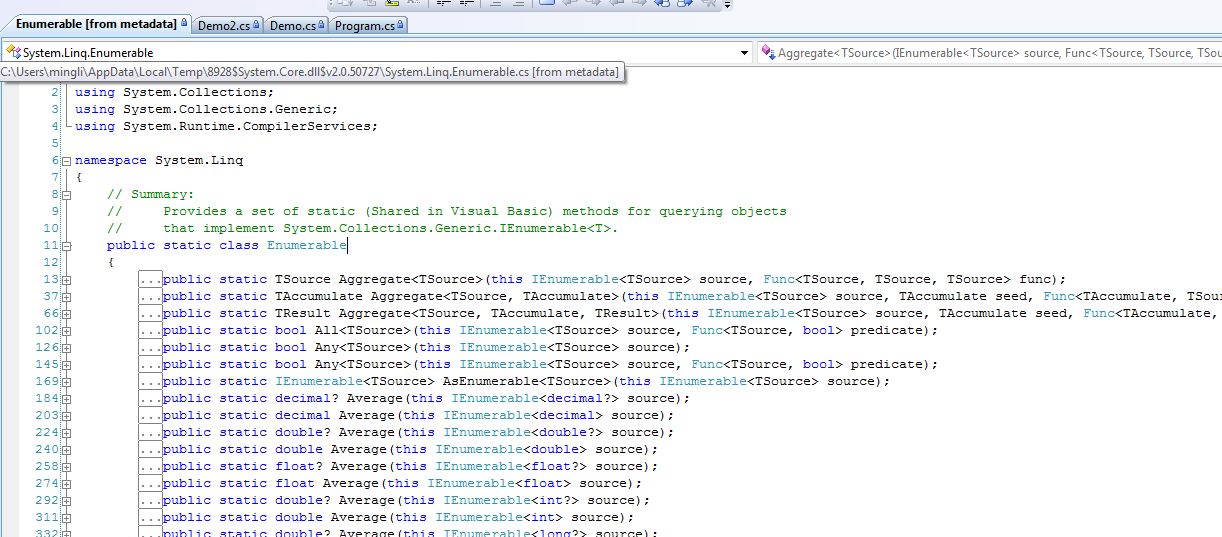
[1-1-3. 用Select进行投影操作 3](#_Toc427075389)

[1-1-4. 用OrderBy进行排序 4](#_Toc427075390)

[1-1-5. 将方法连接在一起 4](#_Toc427075391)

# Linq 中的扩展方法

Linq中存在大量的扩展方法。在System.Linq命名空间中，存在两个使用极其广泛的类Enumerable和Queryable，它们中就含有许许多多的扩展方法

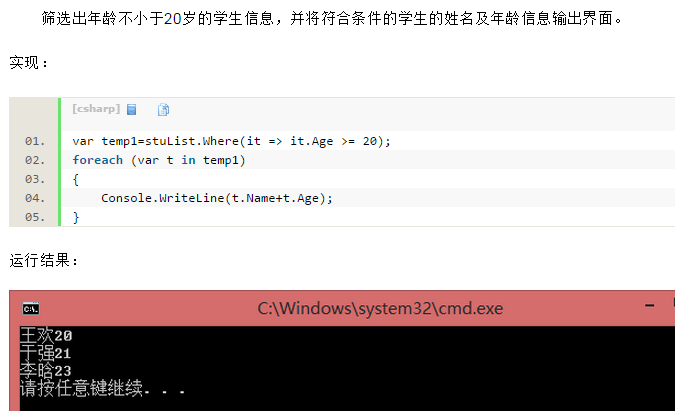


## 实例

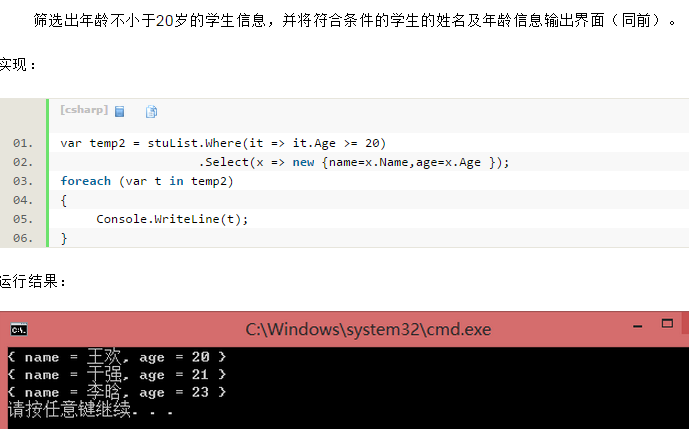
### 准备工作



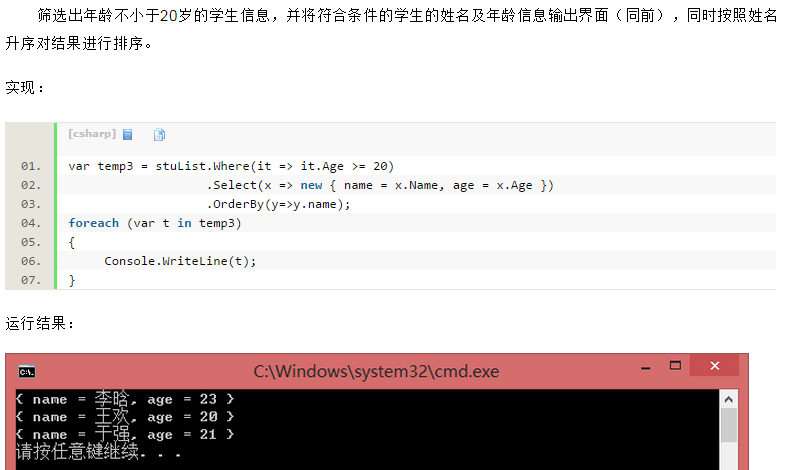
### 用Where来筛选数据



### 用Select进行投影操作



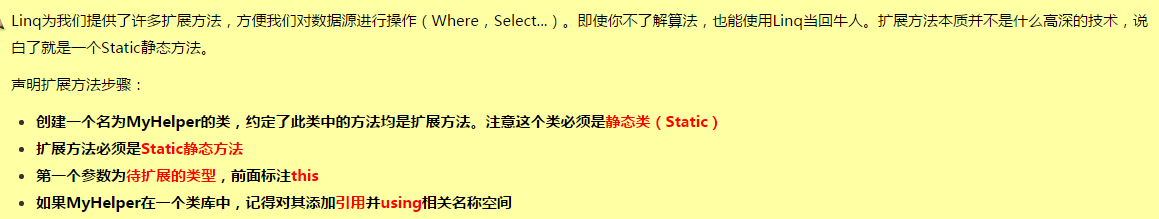
### 用OrderBy进行排序



### 将方法连接在一起

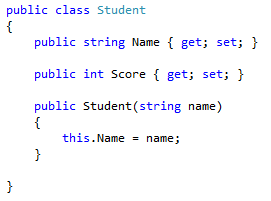
我们发现：在Linq操作中，可以将各个单独的操作链接成一条管道，然后，信息在这条管道中传输。这其实是扩展方法存在的原因之一，因为扩展方法允许将静态方法的调用链接在一起。这样做的好处是能够以一种自然的方式将静态方法链接起来

# 扩展方法



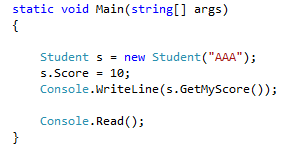
## 实例

### 声明一个Student类



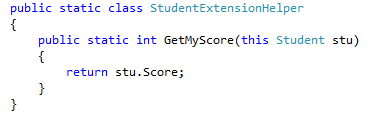
注意类中并没有GetMyScore之类的方法

### 在Program中直接调用该方法



应该会出错

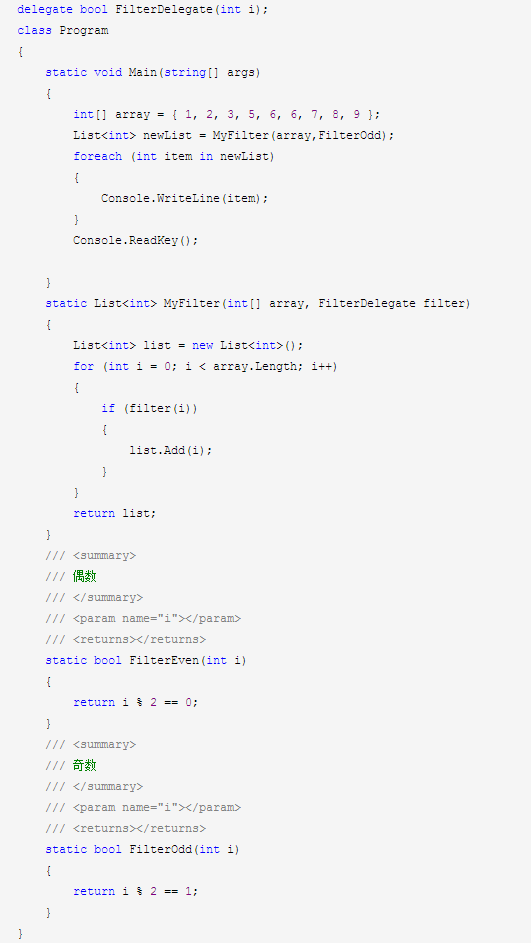
### 添加扩展类和扩展方法来解决



# Lambda表达式的前世今生

## 委托





委托要使用关键字delegate， 同时指明委托名称，返回值和参数。

注意这个返回值，在匿名函数和lambda表达式中是不需要的。因为委托是方法的声明，所以这里是一个比较全的定义。但匿名函数和lamdda表达式参数声明和函数体是一起的，函数体中可以有return语句直接返回，然后编译器会根据返回值来决定返回的参数类型，所以不需要指定。这也是C#引入var类型的原因。

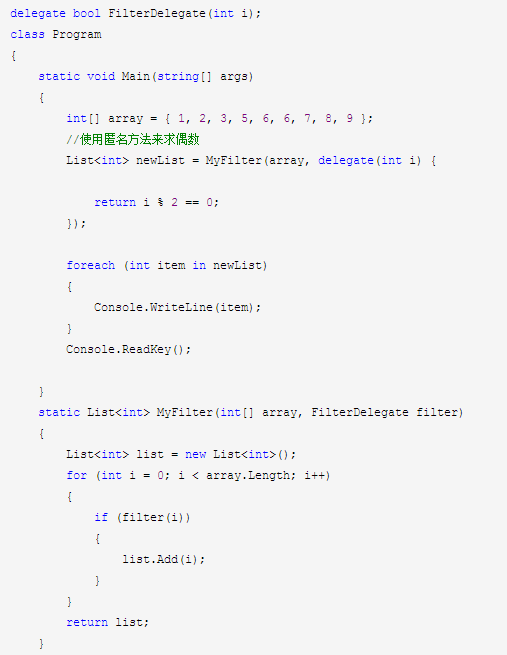
## 匿名方法

委托的定义次序是先定义委托，然后声明与委托类型相同的方法名。然后将该方法名传递给接收该委托对象的地方。

但是，如果你不想显示的创建一个方法，原因可能是（但有时候申明方法很麻烦，还要考虑方法名称不重复，所以对于一些我们只使用一次的方法，完全没有必要单独为其申明），这时候就可以用匿名函数。（委托还是要的）

使用方法是在需要委托对象的时候直接传入匿名函数。匿名函数要使用delegate关键字， 入参和方法体。方法体中有返回值。

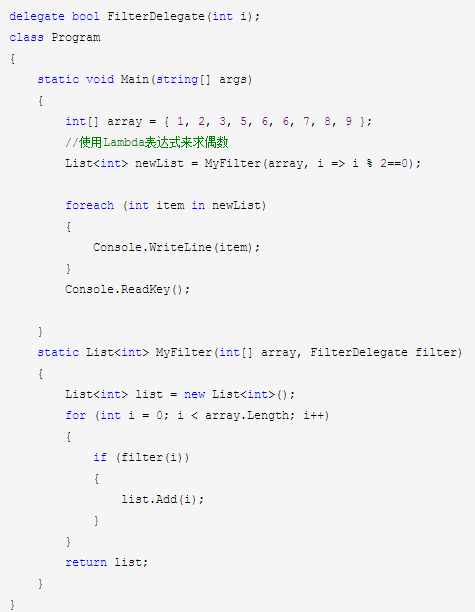
匿名方法在C# 2.0中首次引入。



## Lambda表达式

C# 2.0中加入的匿名方法，简化了我们编写事件处理函数的工作，使我们不再需要单独声明一个函数来与事件绑定，只需要使用delegate关键字在线编写事件处理代码。

而C# 3.0则更进一步，通过Lambda表达式，我们可以一种更为简洁方式编写事件处理代码，新的Lambda事件处理代码看上去就像一个计算表达式，它使用"=>"符号来连接事件参数和事件处理代码。我可以这样写：SomeEvent += 事件参数 => 事件处理代码；



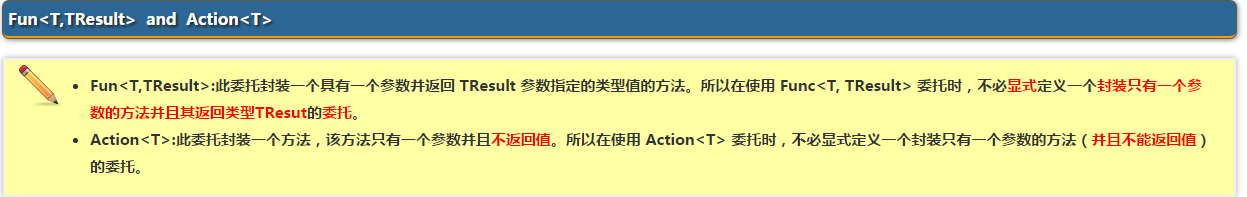
**使用Lambda表达式，"=>"之前为参数列表，如果有多个参数，则不能省略括号，比如：(s,e)=>....**

**如果方法有返回值，并且处理代码只有一行，可以简写成i=>i%2==0，等价于i=>{return i%2==0}，反之对于有多行的处理代码，则不能简写，必须写完整，比如：(s,e)=>{...程序代码块...}**

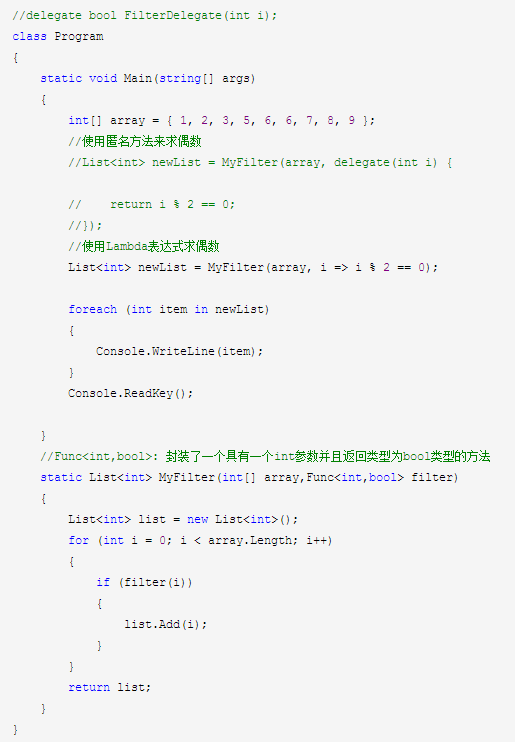
## 有没有可能连委托也不声明了：Fun 和 Action

因为要为不同的函数原型声明不同的委托太麻烦了，就跟声明方法一样的麻烦。有没有类似匿名方法那样的，让我不要再声明委托了。

答案当然是有。C# Framework 为我们定义了两个通用的委托类



其中约定的就是Fun 前面的是入参，最后一个是返回值。这样就可以用**Func<TSource, bool>这种形式替代任何需要一个委托原型的地方。**



## Linq下的扩展方法特征

