> **说明** 对于采用非查询表达式的语法，目前没有统一的术语，而有**方法语法**、**点式语法**、**流式语法**、**lambda 语法**等名称，之后书中会统一采用**方法语法**来代称。对于这几个术语之间的细微差别，读者无须费心分辨。

当查询变得更复杂时，方法语法依然可以从容应对。LINQ 中提供的很多方法，并没有与之对应的查询表达式语法，包括 `Select` 和 `Where` 的某些重载方法。这些重载方法返回的是元素以及元素对应的索引值。另外，如果想在查询的结尾执行一个方法调用（例如调用 `ToList()`来把结果转换成 `List<T>`对象），就要把整个查询表达式用圆括号括起来；如果使用方法语法，只需在末尾直接添加方法调用即可。

我并不是不推荐使用查询表达式。在很多情况下（包括前面那个例子在内），两种方式难分高下。编译器能够替我们处理那些隐形标识符时，也是查询表达式绽放光芒之时。虽然完全可以手动实现这一过程，但根据我的个人经验，如果自行构建那些匿名类型，然后在子句中进行拆解，会很烦琐，使用查询表达式则会简单许多。

最后，建议大家掌握两种方式。不管舍弃哪种方式，都相当于放弃增强代码可读性的可能。至此，C# 3 的所有特性介绍完毕。在介绍下一章内容之前，先谈谈这些特性是如何构建起 LINQ 世界的。

3.8 终极形态：LINQ

这部分不会介绍当前的各种 LINQ 提供器。我目前应用最多的 LINQ 技术是 LINQ to Objects，配合 `Enumerable` 静态类和委托使用。下面介绍这些特性是如何成就 LINQ 的。假设有一个查询从 Entity Framework 获取数据，代码如下所示（假设已存在某数据库和相应的表结构）：

```
var products = from product in dbContext.Products
               where product.StockCount > 0
               orderby product.Price descending
               select new { product.Name, product.Price };
```

短短 4 行代码，应用了所有新特性。

- 匿名类型，包括投射初始化器（只选择 `name` 和 `price` 这两个属性）。
- 使用 `var` 声明的匿名类型，因为无法声明 `products` 变量的有效类型。
- 查询表达式。当然对于本例可以不使用查询表达式，但对于更复杂的情况，使用查询表达式能事半功倍。
- lambda 表达式。lambda 表达式在这里作为查询表达式转译之后的结果。
- 扩展方法。它使得转译后的查询可以通过 `Queryable` 类实现，因为 `dbContext.Products` 实现了 `IQueryable<Product>`接口。
- 表达式树。它使得查询逻辑可以按照数据的方式传给 LINQ 提供器，然后转换成 SQL 语句并交由数据库执行。