Introduction to LINQ: Part 5 - Iteration

Before reading this blog post, you should read Introduction to LINQ: Part 1 - Selecting Data at https://www.pdsa.com/Resources-BlogPosts/2020-02-LINQ-Select.pdf. That post will introduce you to the data and the general approach to how LINQ works.

In this blog post you learn how to apply some iteration methods to a collection of data. The ForEach() method allows you to iterate over the entire collection and apply an expression to each object. The Skip() methods allows you to bypass a certain amount of objects at the beginning of a collection. The Take() methods allows you to only take a certain amount of objects from the beginning of a collection.

Using the ForEach() Method

The ForEach() method allows you to iterate over any collection and execute an expression on each item in the collection. In the first two samples you are going to loop through the collection and get the length of the value in the *Name* property for each Product object and assign that value to a property named *NameLength* in the Product class. The second example passes each Product object to a method that sums up the sales for that product and assigns the total sales to a *TotalSales* property. Open the **Product.cs** file and add two new properties as shown in the following code snippet.

```
// Calculated Properties
public int NameLength { get; set; }
public decimal TotalSales { get; set; }
```

When using the LINQ query syntax, use the 'let' keyword and a temporary (throw-away) variable to perform the assignment of the prod.Name.Length to the prod.NameLength property. When you use the LINQ method syntax, it is much clearer that you are performing an assignment of the length of the *Name* property to the *NameLength* property.

```
public void ForEach()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products = (from prod in Products
                let tmp = prod.NameLength = prod.Name.Length
                select prod).ToList();
  }
  else {
    // Method Syntax
    Products.ForEach(prod => prod.NameLength = prod.Name.Length);
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

Calling a Method from ForEach()

If you wish to fill in the *TotalSales* property into each Product object in the Products collection, create a method to which you pass a Product object. In this method you get all of the Sales data, then perform a Where() method to filter the sales by the *ProductID* property. You then get the sum of all *LineTotal* values. NOTE: This is an inefficient design as you would not want to retrieve all the values from the Sales collection each time. This sample is just to illustrate calling a method from within a ForEach().

```
private decimal SalesForProduct(Product prod)
{
  List<SalesOrderDetail> sales;

  // Get All Sales Data
  sales = new SalesOrderDetailRepository().GetAll();

  return sales.Where(s => s.ProductID == prod.ProductID)
              .Sum(s => s.LineTotal);
}
```

Create a new method as shown below that calls the SalesForProduct() method passing in each Product object as the LINQ query runs. When using the query syntax you must use the 'let' keyword in order to perform the assignment to the *TotalSales* property from the return result of the call to SalesForProduct().

```
public void ForEachCallingMethod()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products = (from prod in Products
                let tmp = prod.TotalSales = SalesForProduct(prod)
                select prod).ToList();
  }
  else {
    // Method Syntax
    Products.ForEach(prod =>
       prod.TotalSales = SalesForProduct(prod));
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

Skip()

If you wish to skip a certain amount of items at the beginning of a collection, apply the Skip() method, passing in the total number of items to skip over as shown in the following code.

```
public void Skip()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products = (from prod in Products
                select prod).Skip(20).ToList();
  }
  else {
    // Method Syntax
    Products = Products.Skip(20).ToList();
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

SkipWhile()

If you wish to apply a condition to skip those items at the beginning of the collection, use the SkipWhile() method. In the following example, use the OrderBy() method to sort the collection by the *Name* property. Apply the SkipWhile() method to the results of the ordering and skip those items where the *Name* property starts with the letter "A".

```
public void SkipWhile()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products =
      (from prod in Products
       orderby prod.Name
       select prod)
       .SkipWhile(prod => prod.Name.StartsWith("A")).ToList();
  }
  else {
    // Method Syntax
    Products = Products.OrderBy(prod => prod.Name)
      .SkipWhile(prod => prod.Name.StartsWith("A")).ToList();
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

Take()

Instead of skipping a certain amount of items at the beginning of a collection, you might want to only return a certain amount of items the beginning of the collection. Use the Take() method to specify how many items you want to return.

```
public void Take()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products = (from prod in Products
                select prod).Take(5).ToList();
  }
  else {
    // Method Syntax
    Products = Products.Take(5).ToList();
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

TakeWhile()

TakeWhile() allows you to take a certain amount of items from the beginning of a collection based on a condition. Pass in a predicate expression to the TakeWhile() to perform some test on the objects. If the expression returns a true, the item will be taken and move to the next item in the collection. This continues until the expression returns false. At that point, no more items in the collection will be taken.

```
public void TakeWhile()
{
  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    Products =
     (from prod in Products
      orderby prod.Name
      select prod)
      .TakeWhile(prod => prod.Name.StartsWith("A")).ToList();
  }
  else {
    // Method Syntax
    Products = Products.OrderBy(prod => prod.Name)
      .TakeWhile(prod => prod.Name.StartsWith("A")).ToList();
  }

  ResultText = $"Total Products: {Products.Count}";
}
```

Summary

In this blog post you learned how to iterate over items in a collection. Using the ForEach() method you can assign values into new properties or even call a method to perform complex operations. The Skip() method will skip a specific number of items, while the SkipWhile() skips objects based on a condition. The Take() method returns a specific number of items from the start of a collection, while the TakeWhile() method returns items from the start of a collection based on a condition.

Sample Code

You can download the complete sample code at my https://github.com/PaulDSheriff/BlogPosts page.