2.LINQ Where 操作符实现

反编译 Where 操作符，可以看到如下代码:

Enumerable.cs

```
    public static IEnumerable<TSource> Where<TSource>(this
IEnumerable<TSource> source, Func<TSource, bool> predicate)
    {
      if (source == null)
        throw Error.ArgumentNull(nameof (source));
      if (predicate == null)
        throw Error.ArgumentNull(nameof (predicate));
      Enumerable.Iterator<TSource> iterator;
      if ((iterator = source as Enumerable.Iterator<TSource>) != null)
        return iterator.Where(predicate);
      TSource[] source1;
      if ((source1 = source as TSource[]) != null)
      {
        if (source1.Length != 0)
          return (IEnumerable<TSource>) new
Enumerable.WhereArrayIterator<TSource>(source1, predicate);
        return (IEnumerable<TSource>) EmptyPartition<TSource>.Instance;
      }
      List<TSource> source2;
      if ((source2 = source as List<TSource>) != null)
        return (IEnumerable<TSource>) new
Enumerable.WhereListIterator<TSource>(source2, predicate); // 重点
      return (IEnumerable<TSource>) new
Enumerable.WhereEnumerableIterator<TSource>(source, predicate);
    }
```

代码看着很复杂，我们再进行反编译一下，我们的重点类。

Where

```
   private sealed class WhereListIterator<TSource> :
Enumerable.Iterator<TSource>, IIListProvider<TSource>, IEnumerable<TSource>,
IEnumerable
    {
      private readonly List<TSource> _source;
      private readonly Func<TSource, bool> _predicate;
      private List<TSource>.Enumerator _enumerator;

      public WhereListIterator(List<TSource> source, Func<TSource, bool>
predicate)
      {
        this._source = source;
        this._predicate = predicate;
      }

      public override Enumerable.Iterator<TSource> Clone() {...}

      public int GetCount(bool onlyIfCheap) {...}

      public override bool MoveNext()
      {
        switch (this._state)
        {
          case 1:
            this._enumerator = this._source.GetEnumerator();
            this._state = 2;
            goto case 2;
          case 2:
            // 重点
            while (this._enumerator.MoveNext())
            {
              TSource current = this._enumerator.Current;
              if (this._predicate(current))
              {
                this._current = current;
                return true;
              }
            }
            this.Dispose();
            break;
        }
```

```
        return false;
      }

      public override IEnumerable<TResult> Select<TResult>(Func<TSource,
TResult> selector) {...}

      public TSource[] ToArray() {...}

      public List<TSource> ToList() {...}

      public override IEnumerable<TSource> Where(Func<TSource, bool>
predicate){...}
    }
```

重点是 MoveNext 方法。在 MoveNext 方法里，对所有数据进行了一层过滤， 代码如下:

```
while (this._enumerator.MoveNext())
{
    TSource current = this._enumerator.Current;
    if (this._predicate(current))
    {
        this._current = current;
        return true;
    }
}
```

代码看着很眼熟。

1. 遍历:

```
while (this._enumerator.MoveNext())
```

这行代码，很容易理解，就是和我们之前学习的 foreach 的同等实现是一样的。

是对 this._enumerator 做了遍历操作。

2. 过滤

```
TSource current = this._enumerator.Current;
if (this._predicate(current))
{
    this._current = current;
    return true;
}
```

this._predicate 是我们在 Where 操作符传进来的条件函数。

原理非常容易理解。

如何自己实现一个 Where 操作符?

自己实现 IEnumerable 是比较复杂的，因为里边涉及了状态的切换。最简单的还是对 List 进行操作。本质是一样的，只不过本文是仅供同学理解，在真实项目中还是建议用 Where 操作符。

实现的代码很简单，Where 用了一个静态扩展关键字 this。

也就是

```
public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource>
source, Func<TSource, bool> predicate)
```

中的 this 这个关键字。

代码如下

```
/*****************************************************************************
 * http://sikiedu.com liangxie
 *****************************************************************************/
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace UniRxLesson
{
    public class WhereImplementExample : MonoBehaviour
    {
        private void Start()
        {
            var testNumbers = new List<int> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

            testNumbers.ListWhere(testNumber => testNumber % 2 == 0)
                .Select(number => number / 2)
                .ToList()
                .ForEach(resultNumber => { Debug.Log(resultNumber); });
        }
    }

    public static class WhereImplement
    {
        public static List<T> ListWhere<T>(this List<T>
sourceList,Func<T,bool> condition)
        {
            var retList = new List<T>();

            foreach (var sourceItem in sourceList)
            {
                if (condition(sourceItem))
                {
                    retList.Add(sourceItem);
                }
            }

            return retList;
        }
    }
}
```

输出结果为:

```
1
2
3
4
5
```

与之前的结果一样。

今天的内容就这些。