3.LINQ Select 操作符实现

我们反编译，示例中的 Select 操作符，其定义方法如下:

```
    public static IEnumerable<TResult> Select<TSource, TResult>(this
IEnumerable<TSource> source, Func<TSource, TResult> selector)
    {
      if (source == null)
        throw Error.ArgumentNull(nameof (source));
      if (selector == null)
        throw Error.ArgumentNull(nameof (selector));
      Enumerable.Iterator<TSource> iterator;
      if ((iterator = source as Enumerable.Iterator<TSource>) != null)
        return iterator.Select<TResult>(selector);
      IList<TSource> source1;
      if ((source1 = source as IList<TSource>) != null)
      {
        TSource[] source2;
        if ((source2 = source as TSource[]) != null)
        {
          if (source2.Length != 0)
            return (IEnumerable<TResult>) new
Enumerable.SelectArrayIterator<TSource, TResult>(source2, selector);
          return (IEnumerable<TResult>) EmptyPartition<TResult>.Instance;
        }
        List<TSource> source3;
        if ((source3 = source as List<TSource>) != null)
          return (IEnumerable<TResult>) new
Enumerable.SelectListIterator<TSource, TResult>(source3, selector); // 重点
        return (IEnumerable<TResult>) new
Enumerable.SelectIListIterator<TSource, TResult>(source1, selector);
      }
      IPartition<TSource> source4;
      if ((source4 = source as IPartition<TSource>) == null)
        return (IEnumerable<TResult>) new
Enumerable.SelectEnumerableIterator<TSource, TResult>(source, selector);
```

```
      if (!(source4 is EmptyPartition<TSource>))
        return (IEnumerable<TResult>) new
Enumerable.SelectIPartitionIterator<TSource, TResult>(source4, selector);
      return (IEnumerable<TResult>) EmptyPartition<TResult>.Instance;
    }
```

代码与 Where 的定义非常类似，我们直接看SelectListIterator 的定义。

```
    private sealed class SelectListIterator<TSource, TResult> :
Enumerable.Iterator<TResult>, IPartition<TResult>, IIListProvider<TResult>,
IEnumerable<TResult>, IEnumerable
    {
      private readonly List<TSource> _source;
      private readonly Func<TSource, TResult> _selector;
      private List<TSource>.Enumerator _enumerator;

      public SelectListIterator(List<TSource> source, Func<TSource, TResult>
selector)
      {
        this._source = source;
        this._selector = selector;
      }

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

      public override bool MoveNext()
      {
        switch (this._state)
        {
          case 1:
            this._enumerator = this._source.GetEnumerator();
            this._state = 2;
            goto case 2;
          case 2:
            // 重点
            if (this._enumerator.MoveNext())
            {
              this._current = this._selector(this._enumerator.Current);
              return true;
```

```
            }
            this.Dispose();
            break;
        }
        return false;
    }

    public override IEnumerable<TResult2> Select<TResult2>(Func<TResult,
TResult2> selector){...}

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

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

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

    public IPartition<TResult> Skip(int count){...}

    public IPartition<TResult> Take(int count){...}

    public TResult TryGetElementAt(int index, out bool found){...}

    public TResult TryGetFirst(out bool found){...}

    public TResult TryGetLast(out bool found) {...}
}
```

重点还是在 MoveNext 方法里。

```
if (this._enumerator.MoveNext())
{
    this._current = this._selector(this._enumerator.Current);
    return true;
}
```

MoveNext 不用说了，就是遍历，为什么不是 while 而只是 if 判断一次?

答案很简单，Select 操作符会有多种实现，如果是先使用 Where 操作符再使用 Select 操作符，Select 操作符会自动合并成 WhereSelectListIterator ，在这个类里的 MoveNext 的核心代码如下。

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

是将 Select 和 Where 的逻辑合并在一起了。

而我们目前关注的是 SelectListIterator。

一般是作为 List 的第一个操作符的 Select 才会最终使用 SelectListIterator。

比如:

```
mList.Select(XXX);
```

因为 List 本身是到最后才进行遍历操作的。

操作符的迭代器，是可以进行组合的。组合之后再最后进行遍历操作之后，这些组合的迭代器在会具体执行其 MoveNext 方法。

总之原理很简单。我们用 List 简单实现一下。

```
/*****************************************************************************
 * http://sikiedu.com liangxie
 *****************************************************************************/
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using UnityEngine;

namespace UniRxLesson
{
    public class SelectImplementExample : 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)
                .ListSelect(number => number / 2)
                .ToList()
                .ForEach(resultNumber => { Debug.Log(resultNumber); });
        }
    }

    public static class SelectImplement
    {
        public static List<K> ListSelect<T, K>(this List<T> sourceList,
Func<T, K> convertion)
        {
            var retList = new List<K>();

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

            return retList;
        }
    }
}
```

输出结果为

```
1
2
3
```

```
4
5
```

结果一致。