Taking advantage of the ordering guarantees of the LINQ GroupBy method

devblogs.microsoft.com/oldnewthing/20181129-00

November 29, 2018

Raymond Chen

A customer wanted to group a set of data by one field, and within each group, sort the data by another field, and then sort the groups by that second field.

For example, given the following data set:

Name	Time
Charles	11
Charles	21
Alice	20
Charles	23
Alice	29
Alice	13
Charles	17
Bob	20
Alice	13
Bob	12
Alice	26
Bob	18
Charles	18
Bob	28
Alice	23

Bob	13

We group by name:

Name	Time
Alice	20
Alice	29
Alice	13
Alice	13
Alice	26
Alice	23
Bob	20
Bob	12
Bob	18
Bob	28
Bob	13
Charles	11
Charles	21
Charles	23
Charles	17
Charles	18

And then we sort each person's time, shortest first.

Name	Time
Alice	13
Alice	13
Alice	20

Alice	23
Alice	26
Alice	29
Bob	12
Bob	13
Bob	18
Bob	20
Bob	28
Charles	11
Charles	17
Charles	18
Charles	21
Charles	23

And then we sort the people by their best time. Charles's best time is 11 seconds, which is best overall, so his times go first. Bob's best time is 12 seconds, so his group goes next. Alice's best time is 13 seconds, so her group is last.

Name	**Time**
Charles	11
Charles	17
Charles	18
Charles	21
Charles	23
Bob	12
Bob	13
Bob	18
Bob	20

Bob	28
Alice	13
Alice	13
Alice	20
Alice	23
Alice	26
Alice	29

So we have a three-step LINQ query, where we group, and then sort each group, and then sort the groups.

```
var results =
    data.GroupBy(x => x.Name) // group by name
        .Select(g => g.OrderBy(x => x.Time)); // sort each group
        .OrderBy(g => g.First()) // sort the groups by best time
        .SelectMany(g => g);  // flatten the groups
```

The last step is to use `SelectMany` to convert the groups back into their individual members. This takes advantage of the fact that `IGrouping<TKey, out TElement>`, derives from `IEnumerable<TElement>`, so you can use the group as a collection.

But you can reduce this to a two-step operation: First sort globally by time, and then group them. The `GroupBy` method is documented as reporting the groups in the order of first appearance, so this ensures that the fastest group comes first.

```
var results =
    data.OrderBy(x => x.Time) // sort globally by time
        .GroupBy(x => x.Name) // group by name (best time first)
        .SelectMany(g => g);  // flatten the groups
```

It does slightly more work than the three-step query because it sorts the entire collection, even though it needed only to sort each group. But it looks slicker, and might even be easier to understand. Provided you understand that the grouping is stable.

Raymond Chen

Follow