|  |  |
| --- | --- |
| Extension Method | Examples and Notes |
| All(Predicate) | [managers.All(p => p.Age < 40)] |
| Any() | [managers.Any()] |
| Any(Predicate) | [managers.Any(p => p.Name == “Name 1”)] |
| Average(Selector) | managers.Average(p => p.Age)  The input selector must return a value of any type that has predefined or user-defined addition and division operators. |
| Concat(IEnumerable) | managers.Concat(otherPersons)  An implicit reference conversion must exist between types of items of concatenated enumerations. |
| Contains(Object) | managers.Contains(otherPersons.First()) |
| Count() | managers.Count() |
| Count(Predicate) | managers.Count(p => p.Age > 30) |
| Distinct() | managers.Distinct() |
| First() | managers.First() |
| First(Predicate) | managers.First(p => p.Age > 30) |
| FirstOrDefault() | managers.FirstOrDefault() |
| FirstOrDefault( Predicate) | managers.FirstOrDefault(p => p.Age > 30) |
| GroupBy(Selector) | managers.GroupBy(p => p.Age)  or  managers.GroupBy(  p => new  {  Age = p.Age,  Count = p.Children.Count()  })  This method returns an enumeration of group objects. Each group has a unique key defined by the input selector and contains items of the source enumeration associated with this key. You can access the key of a group instance using the Key property. You can treat a group itself as an enumeration of items that the group contains. |
| Last() | managers.Last() |
| Last(Predicate) | managers.Last(p => p.Age > 100) |
| LastOrDefault() | managers.LastOrDefault() |
| LastOrDefault( Predicate) | managers.LastOrDefault(p => p.Age > 100) |
| Max( ComparableSelector) | managers.Max(p => p.Age) |
| Min( ComparableSelector) | managers.Min(p => p.Age) |
| OrderBy( ComparableSelector) | managers.OrderBy(p => p.Age)  or  managers.OrderBy(p => p.Age)  .ThenByDescending(p => p.Name)  or  managers.OrderBy(p => p.Age)  .ThenByDescending(p => p.Name)  .ThenBy(p => p.Children.Count())  This method returns an enumeration ordered by a single key. To specify additional ordering keys, you can use the following extension methods of an ordered enumeration:   * ThenBy(ComparableSelector) * ThenByDescending(ComparableSelector) |
| OrderByDescending( ComparableSelector) | managers.OrderByDescending(p => p.Age)  or  managers.OrderByDescending(p => p.Age)  .ThenByDescending(p => p.Name)  or  managers.OrderByDescending(p => p.Age)  .ThenByDescending(p => p.Name)  .ThenBy(p => p.Children.Count())  See the previous note. |
| Single() | managers.Single() |
| Single(Predicate) | managers.Single(  p => p.Name == “John Smith”) |
| SingleOrDefault() | managers.SingleOrDefault() |
| SingleOrDefault( Predicate) | managers.SingleOrDefault(  p => p.Name == “John Smith”) |
| Skip(int) | managers.Skip(10) |
| SkipWhile(Predicate) | managers.SkipWhile(p => p.Age < 21) |
| Sum(Selector) | managers.Sum(p => p.Children.Count())  The input selector must return a value of any type that has a predefined or user-defined addition operator. |
| Take(int) | managers.Take(5) |
| TakeWhile(Predicate) | managers.TakeWhile(p => p.Age < 50) |
| Union(IEnumerable) | managers.Union(otherPersons)  An implicit reference conversion must exist between types of items of united enumerations. |
| Where(Predicate) | managers.Where(p => p.Age > 18) |