mcnkbr / GenericRepository.cs

Last active 9 months ago • Report abuse

Generic Repository and AutoMapper for .NET Entity Framework

```
    GenericRepository.cs

        using System;
        using System.Collections.Generic;
        using System.Data.Entity;
       using System.Linq;
   4
        using System.Linq.Expressions;
        using Common.Extensions;
   6
        using AutoMapper;
   8
   9
        namespace Repository
  10
            public abstract class GenericRepository<TC, TM, TD> : IGenericRepository<TM, TD>
  11
  12
                where TM : class
                where TD : class
  13
                where TC : Entity.AeeeEntities, new()
            {
  15
  16
                private TC _entities = new TC();
  17
  18
                public TC Context
  19
                    get { return _entities; }
  20
                    set { _entities = value; }
  21
  22
                }
  23
  24
                public virtual List<TD> GetAll(bool mapReset = true)
                    var ent = entities.Set<TM>();
  27
                    var query = ent.ToList();
```

```
28
29
                 if (mapReset)
                     Mapper.Reset();
30
31
                 var list = MapToDtoList<TM, TD>(query).ToList();
32
                 return list;
             }
34
             public List<TD> FindBy(Expression<Func<TM, bool>> predicate, bool mapReset = true)
             {
37
                 var ent = entities.Set<TM>();
                 var query = ent.Where(predicate).ToList();
38
39
40
                 if (mapReset)
                     Mapper.Reset();
41
42
                  var list = MapToDtoList<TM, TD>(query).ToList();
43
44
                 return list;
45
             }
46
47
             public TD SingleOrDefault(Expression<Func<TM, bool>> predicate)
48
                 var ent = _entities.Set<TM>();
49
                  var query = ent.SingleOrDefault(predicate);
50
51
52
                  if (query == null)
53
                     return null;
54
55
                 Mapper.Reset();
56
                 Mapper.CreateMap<TM, TD>();
57
                  var item = Mapper.Map<TM, TD>(query);
58
59
                  return item;
             }
61
             public bool Any(Expression<Func<TM, bool>> predicate)
62
63
64
                 var result = entities.Set<TM>().Any(predicate);
65
                  return result;
```

```
66
              }
67
              public virtual void Add(TD entity)
68
69
70
                  Mapper.Reset();
                  Mapper.CreateMap<TD, TM>();
71
72
                  var item = Mapper.Map<TD, TM>(entity);
73
74
                  _entities.Set<TM>().Add(item);
75
                  Save();
76
              }
77
78
              public virtual int Add(TD entity, bool returnId, string returnName)
79
80
                  Mapper.CreateMap<TD, TM>();
                  var item = Mapper.Map<TD, TM>(entity);
81
82
83
                  _entities.Set<TM>().Add(item);
84
                  Save();
85
                  return returnId ? (int)item.GetType().GetProperty(returnName).GetValue(item, null) : 0;
              }
86
87
              public virtual void Add(TM entity)
88
89
                  _entities.Set<TM>().Add(entity);
91
                  Save();
              }
92
93
94
              public virtual TM AddGetId(TD entity)
95
                  Mapper.CreateMap<TD, TM>();
96
                  var item = Mapper.Map<TD, TM>(entity);
97
98
                  entities.Set<TM>().Add(item);
                  Save();
100
101
                  return item;
102
              }
103
```

```
104
              public virtual void Delete(Expression<Func<TM, bool>> predicate)
              {
106
                  _entities.Set<TM>().RemoveRange(_entities.Set<TM>().Where(predicate));
                  Save();
              }
110
              public virtual void Edit(TD entity, bool hasMap = false)
111
112
                  if (!hasMap)
113
                      Mapper.CreateMap<TD, TM>();
114
                  var participationDto = Mapper.Map<TD, TM>(entity);
115
                  entities.Set<TM>().Attach(participationDto);
116
                  _entities.Entry(participationDto).State = EntityState.Modified;
117
118
119
                  Save();
120
              }
121
122
              public virtual void Save()
123
              {
                  entities.SaveChanges();
124
              }
125
126
              public IQueryable<TM> List(Expression<Func<TM, bool>> filter = null, Func<IQueryable<TM>,
127
                  IOrderedQueryable<TM>> orderBy = null, List<Expression<Func<TM, object>>> includeProperties = null,
128
129
              int? page = null, int? pageSize = null)
              {
130
131
                  IQueryable<TM> query = _entities.Set<TM>();
132
                  if (includeProperties != null)
133
134
                      includeProperties.ForEach(i => { query = query.Include(i); });
135
                  if (filter != null)
136
137
                      query = query.Where(filter);
138
139
                  if (orderBy != null)
140
141
                      query = orderBy(query);
```

```
142
                  }
143
                  if (page != null && pageSize != null)
144
145
                      query = query
                           .Skip(page.Value)
146
                           .Take(pageSize.Value);
147
148
149
                  return query;
150
              }
151
152
              public IQueryable<TM> List(Expression<Func<TM, bool>> filter = null, string orderBy = null, string ascendingDescending = "ASC",
153
                  List<Expression<Func<TM, object>>> includeProperties = null,
             int? page = null, int? pageSize = null)
154
155
156
                  IQueryable<TM> query = _entities.Set<TM>();
157
158
                  if (includeProperties != null)
159
                      includeProperties.ForEach(i => { query = query.Include(i); });
160
161
                  if (filter != null)
                      query = query.Where(filter);
163
                  if (page != null && pageSize != null)
164
165
                      query = query
                           .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
166
167
                           .Skip(page.Value)
168
                           .Take(pageSize.Value);
169
170
                  return query;
              }
171
172
              public Tuple<!Queryable<TM>, int> ListWithPaging(Expression<Func<TM, bool>> filter = null, Func<!Queryable<TM>,
173
                  IOrderedQueryable<TM>> orderBy = null, List<Expression<Func<TM, object>>> includeProperties = null,
174
              int? page = null, int? pageSize = null)
176
177
                  IQueryable<TM> query = _entities.Set<TM>();
178
179
                  if (includeProperties != null)
```

```
180
                      includeProperties.ForEach(i => { query = query.Include(i); });
181
                  if (filter != null)
182
                      query = query.Where(filter);
                  if (orderBy != null)
                  {
187
                      query = orderBy(query);
                  }
                  var count = query.Count();
                  if (page != null && pageSize != null)
190
191
                      query = query
                           .Skip(page.Value)
192
193
                           .Take(pageSize.Value);
194
195
                  return new Tuple<IQueryable<TM>, int>(query, count);
196
              }
197
198
              public Tuple<IQueryable<TM>, int> ListWithPaging(Expression<Func<TM, bool>> filter = null, string orderBy = null, string ascendingD
199
                 List<Expression<Func<TM, object>>> includeProperties = null,
            int? page = null, int? pageSize = null)
201
                  IQueryable<TM> query = _entities.Set<TM>();
                  if (includeProperties != null)
                      includeProperties.ForEach(i => { query = query.Include(i); });
                  if (filter != null)
                      query = query.Where(filter);
209
                  var count = query.Count();
210
211
212
                  if (page != null && pageSize != null)
                      query = query
214
                           .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
215
                           .Skip(page.Value)
                           .Take(pageSize.Value);
217
```

```
218
                  return new Tuple<IQueryable<TM>, int>(query, count);
219
              }
220
              public IQueryable<TD> ToDtoListPaging(List<TD> list, string orderBy = null, string ascendingDescending = "ASC", int? page = null, i
221
222
223
                  IQueryable<TD> query = list.AsQueryable();
224
225
                  if (page != null && pageSize != null)
226
                      query = query
227
                           .OrderBy(orderBy ?? "Id", ascendingDescending == "ASC")
228
                           .Skip(page.Value)
229
                           .Take(pageSize.Value);
230
231
                  return query;
232
              }
233
234
              public virtual IEnumerable<TDto> MapToDtoList<TEntity, TDto>(IEnumerable<TEntity> entity)
235
                  where TEntity : class
236
                  where TDto : class
237
              {
                  Mapper.CreateMap<TEntity, TDto>();
238
                  return Mapper.Map<IEnumerable<TEntity>, IEnumerable<TDto>>(entity);
239
              }
240
241
              public virtual IEnumerable<TEntity> MapToEntityList<TDto, TEntity>(IEnumerable<TDto> dto)
243
                  where TDto : class
244
                  where TEntity : class
245
              {
                  Mapper.CreateMap<TDto, TEntity>();
246
                  return Mapper.Map<IEnumerable<TDto>, IEnumerable<TEntity>>(dto);
247
              }
248
249
250
```

```
IGenericRepository.cs

using System;
```

```
using System.Collections.Generic;
3
     using System.Linq.Expressions;
4
     namespace Repository
 6
         public interface IGenericRepository<TM, TD>
8
             where TM : class
9
             where TD : class
10
         {
11
             List<TD> GetAll(bool mapReset = true);
12
             List<TD> FindBy(Expression<Func<TM, bool>> predicate, bool mapReset = true);
13
             void Add(TD entity);
14
             void Add(TM entity);
             int Add(TD entity, bool returnId, string returnName);
15
             void Delete(Expression<Func<TM, bool>> predicate);
16
             void Edit(TD entity, bool hasMap = false);
17
18
             void Save();
19
         }
20
```

QueryableExtensions.cs using System.Linq; using System.Linq.Expressions; using System.Reflection; 3 4 5 namespace Common.Extensions 6 { public static class QueryableExtensions 8 { public static IQueryable<TM> OrderBy<TM>(this IQueryable<TM> source, string orderByProperty, 9 bool asc) where TM : class 10 11 12 var command = asc ? "OrderBy" : "OrderByDescending"; 13 var type = typeof(TM); PropertyInfo property; 14 if (!orderByProperty.Contains('.')) 15 16 property = type.GetProperties().FirstOrDefault(w => w.Name == orderByProperty);

```
17
                 else
18
                     var arrg = orderByProperty.Split('.');
19
                     property = type.GetProperty(arrg[0]).GetType().GetProperty(arrg[1]);
20
21
22
                 var parameter = Expression.Parameter(type, "p");
                 if (property == null) return null;
23
                 var propertyAccess = Expression.MakeMemberAccess(parameter, property);
24
25
                 var orderByExpression = Expression.Lambda(propertyAccess, parameter);
                 var resultExpression = Expression.Call(typeof (Queryable), command, new[] {type, property.PropertyType},
27
                     source.Expression, Expression.Quote(orderByExpression));
28
                 return source.Provider.CreateQuery<TM>(resultExpression);
29
             }
30
31
```

○ UserRepository.cs

```
using Dto.Dto;
     using Dto.GeneralModel;
     using Dto.RequestModel;
4
     using System;
     using System.Collections.Generic;
6
     using System.Linq;
7
     using Entity;
8
     using Dto.Enums;
9
     namespace Repository
10
     {
11
         public class UserRepository : GenericRepository<Entity.AeeeEntities, Entity.User, UserDto>
12
13
         {
             //Add, Edit, Delete, GetAll, FindBy methods created by Generic Repository
14
             //base.Add(...);
15
16
17
```



GutsFun commented on Dec 10, 2014

Your OrderBy extension will not work correctly for nested properties. I suggest you write it like this: