SINGULARIS LAB

iT-34

Деревья выражений на страже спецификации

ведущий инженер-программист

Анатолий Крыжановский

IT Meetup’18, 24 ноября 2018, Волгоград

О чем мы поговорим

- Предметная область:
  - ORM.
- Содержание:
  - Паттерн «репозиторий».
  - Паттерн «спецификация».
  - Дружба «репозитория» и «спецификации».

Паттерн «репозиторий»

- Определение (wiki):

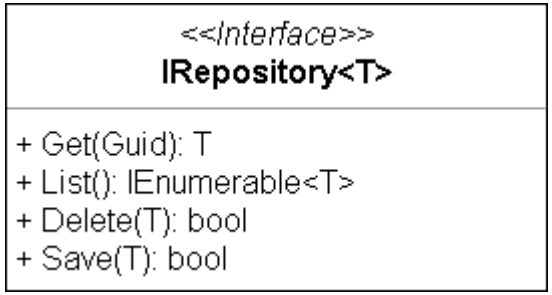
Репозиторий обычно используется как хранилище данных, часто для обеспечения безопасности или сохранности.

- Простыми словами:

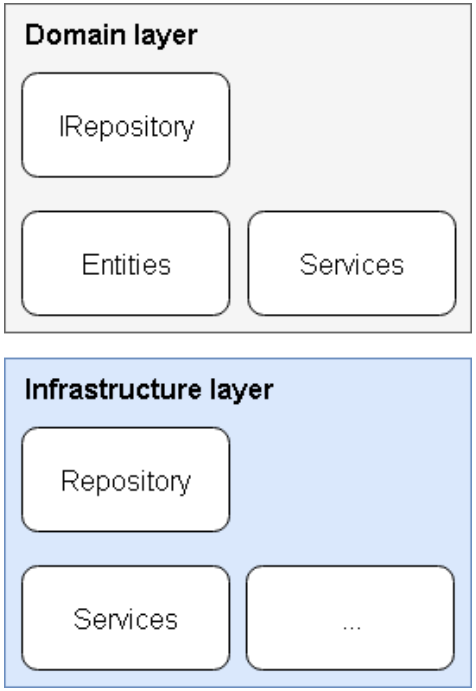
Репозиторий - это абстрактный слой между каким-либо хранилищем и вашим приложением или бизнес-логикой.

Паттерн «репозиторий»

Реализация:



Место обитания:

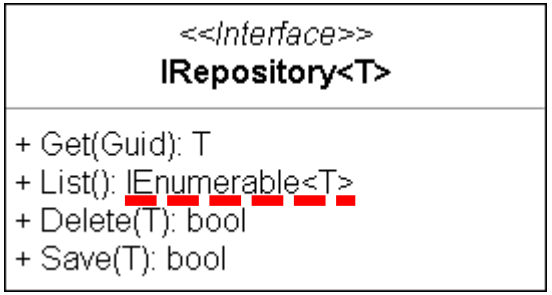


Паттерн «репозиторий»

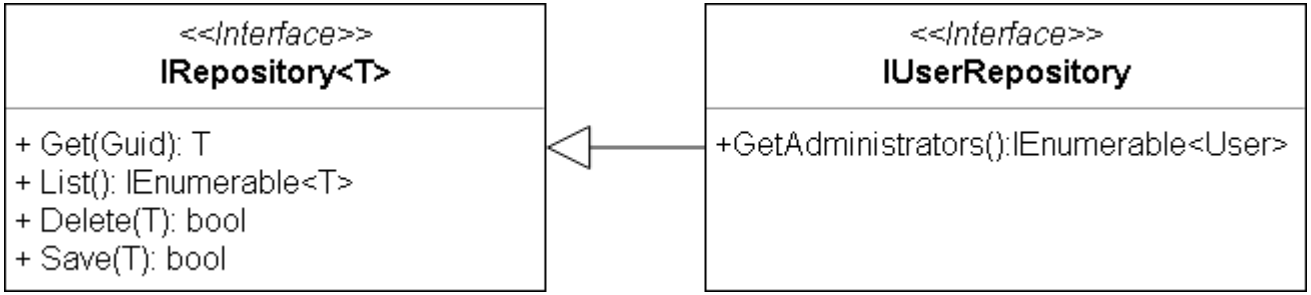
```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetAdministrators()
    {
        return _userRepository
            .List()
            .Where(x => x.Role.HasFlag(EUserRole.Administrator))
            .ToArray();
    }
}
```

Паттерн «репозиторий»

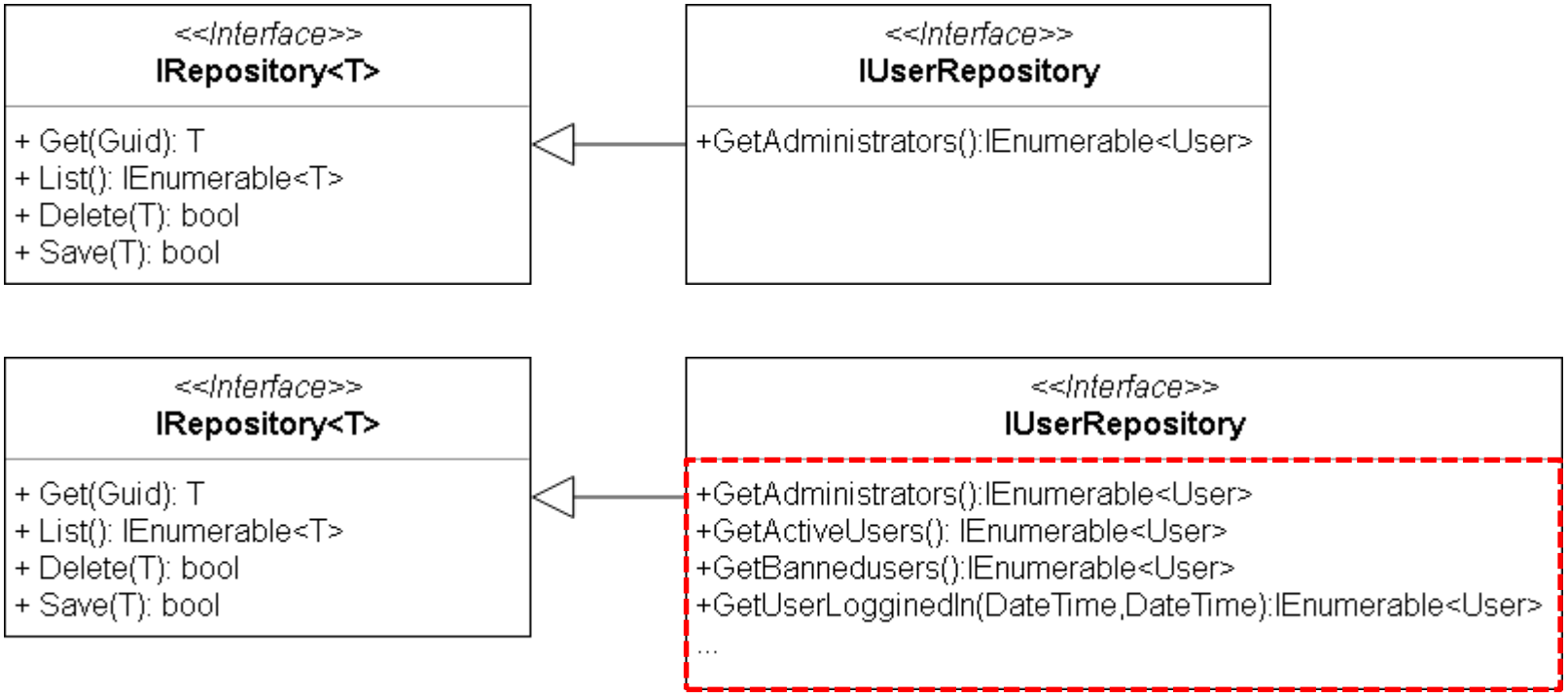
```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetAdministrators()
    {
        return _userRepository
            .List()
            .Where(x => x.Role.HasFlag(EUserRole.Administrator))
            .ToArray();
    }
}
```



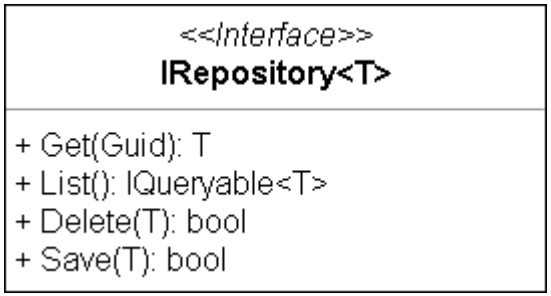
Паттерн «репозиторий» (v2)



Паттерн «репозиторий» (v2)



Паттерн «репозиторий» (v3)

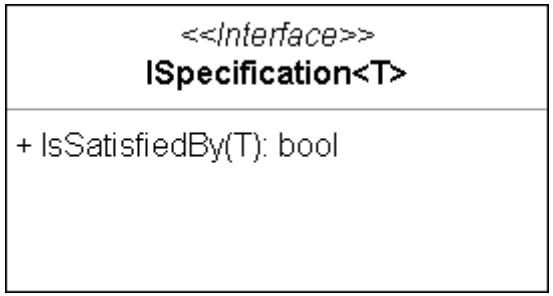



```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetAdministrators()
    {
        return _userRepository
            .List()
            .Where(x => x.Role.HasFlag(EUserRole.Administrator))
            .ToArray();
    }
}
```

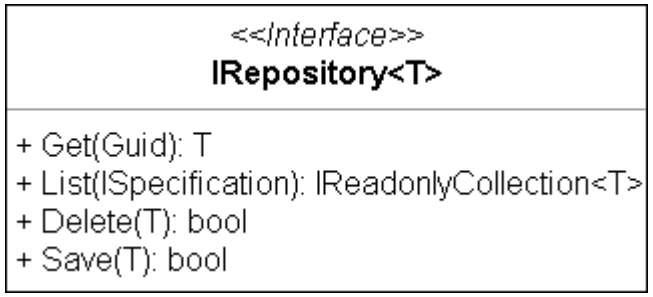
Паттерн «спецификация»

- Определение (wiki):

это шаблон проектирования, посредством которого представление правил бизнес логики может быть преобразовано в виде цепочки объектов, связанных операциями булевой логики.

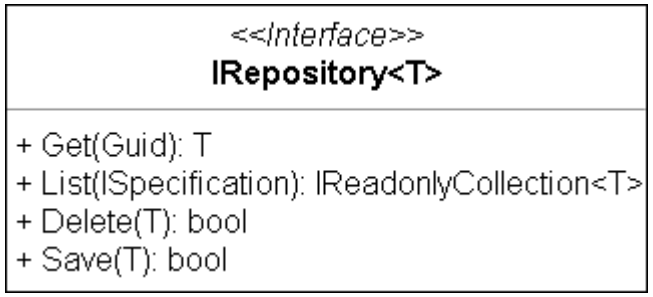


«Спецификация» в «репозитории»



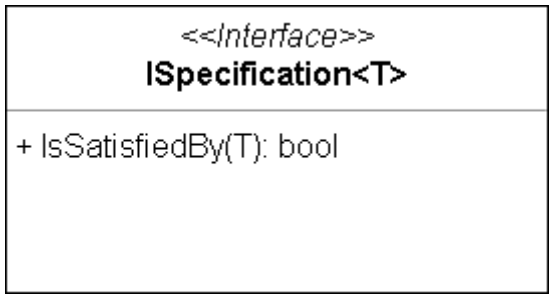

```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetAdministrators()
    {
        return _userRepository
            .List(new UsersWithAdministratorRole())
            .ToArray();
    }
}
```

«Спецификация» в «репозитории»

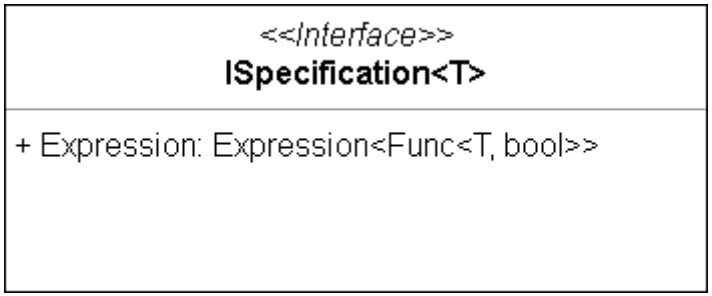



```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetTodayAdministrators ()
    {
        return _userRepository
            .List(new UsersWithAdministratorRole().And(new TodayUsers()))
            .ToArray();
    }
}
```

Паттерн «спецификация»



Дружим с деревьями выражений




```
class UserManager
{
    private readonly IRepository<User> _userRepository;

    public IEnumerable<User> GetTodayAdministrators ()
    {
        return _userRepository
            .List(new UsersWithAdministratorRole().And(new TodayUsers()))
            .ToArray();
    }
}
```

Существующие решения

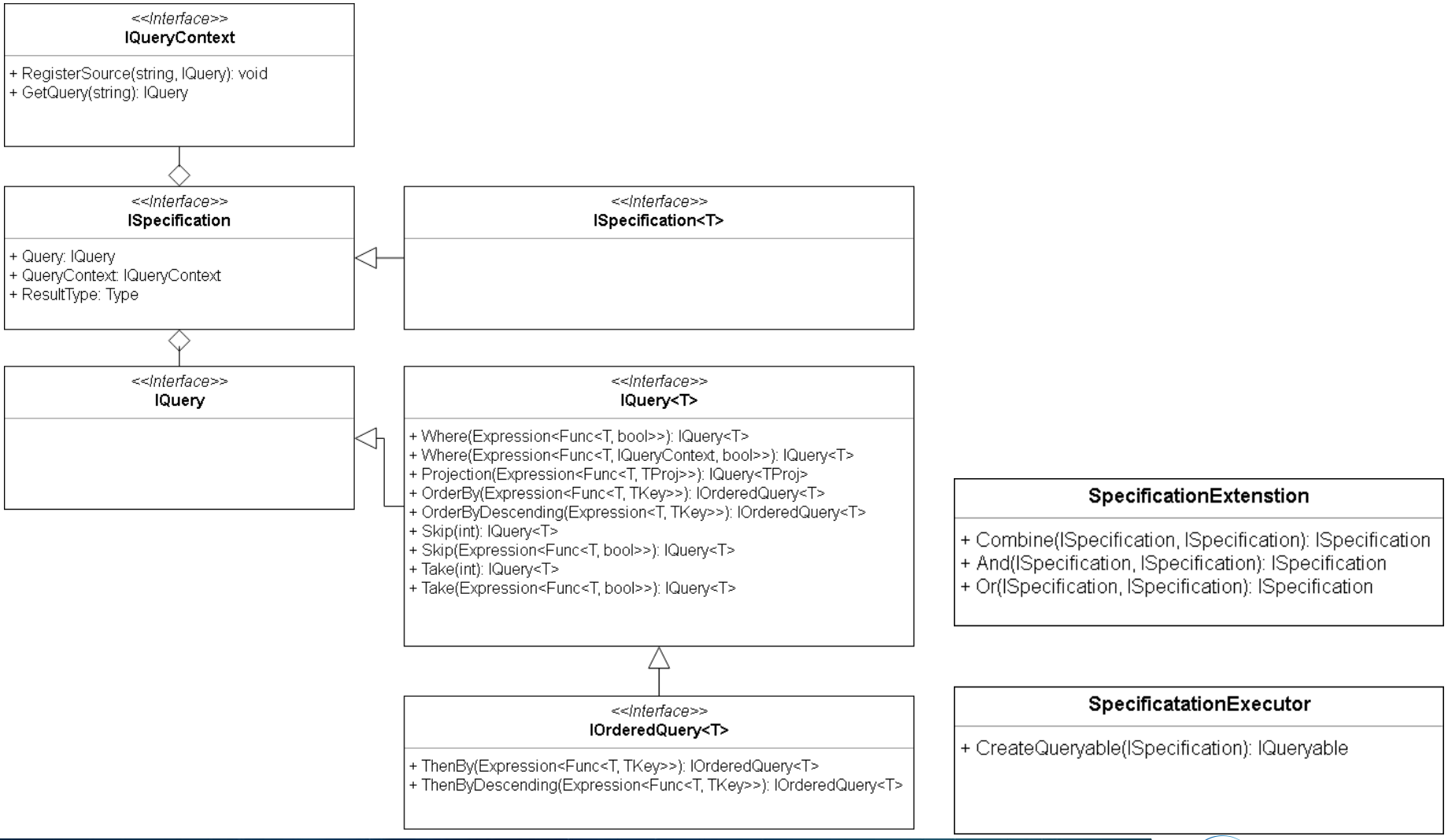
https://github.com/rjperes/DevelopmentWithADot.NHibernateSpecifications

Чего не хватает:

- Вложенные запросы
- Возможность задания проекции в спецификации (хотя это немного и выходит за пределы ее основных назначений), но крайне полезно, когда вам нужно вытянуть всего 1-2 колонки из «широкой» таблицы.
- Хотелось бы использовать «фичи» ORM, такие как Fetch, FetchMany, который в LINQ-нет.

Свой велосипед

Реализация


<<Interface>>
IQueryContext
+ RegisterSource(string, IQuery): void
+ GetQuery(string): IQuery
<<Interface>>
ISpecification
+ Query: IQuery
+ QueryContext: IQueryContext
+ ResultType: Type
<<Interface>>
ISpecification<T>
<<Interface>>
IQuery
<<Interface>>
IQuery<T>
+ Where(Expression<Func<T, bool>>): IQuery<T>
+ Where(Expression<Func<T, IQueryContext, bool>>): IQuery<T>
+ Projection(Expression<Func<T, TProj>>): IQuery<TProj>
+ OrderBy(Expression<Func<T, TKey>>): IOrderedQuery<T>
+ OrderByDescending(Expression<T, TKey>>): IOrderedQuery<T>
+ Skip(int): IQuery<T>
+ Skip(Expression<Func<T, bool>>): IQuery<T>
+ Take(int): IQuery<T>
+ Take(Expression<Func<T, bool>>): IQuery<T>
<<Interface>>
IOrderedQuery<T>
+ ThenBy(Expression<Func<T, TKey>>): IOrderedQuery<T>
+ ThenByDescending(Expression<Func<T, TKey>>): IOrderedQuery<T>
SpecificationExtenstion
+ Combine(ISpecification, ISpecification): ISpecification
+ And(ISpecification, ISpecification): ISpecification
+ Or(ISpecification, ISpecification): ISpecification
SpecificatationExecutor
+ CreateQueryable(ISpecification): IQueryable

Пример использования

```
class VisibleItems: Speicifcation<Item>
{
    public VisibleItems(Team team)
    {
        Query = Source().Where(x => x.Owner == team);
    }
}

class EntityId: Specification<Item>
{
    public EntityId()
    {
        Query = Source().Projection(x => x.Id);
    }
}

class BakedData: Specification<BakedItem>
{
    public BakedData(ISpecification<Item> sourceItems)
    {
        QueryContext.RegisterSource("id", sourceItems.Query);
        Query = Source().Where((x, c) => c.GetSource("id").Any(e => e == x.Id));
    }
}

class PageSpecification: Specification<BakedItem>
{
    public PageSpecification(int start, int count)
    {
        Query = Source().Skip(start).Take(count);
    }
}
```

Пример использования

```
var visibleItemsIds= new VisibleItems(user.Team)
    .Combine(new EntityId());

var bakedDataSpecification = new BakedData(visibleItemsIds)
    .Combine(new PageSpecification(10, 10));

var items = _repository.List(bakedDataSpecification);
```

```
select
  bakeditem0_.Id as id1_3_,
  bakeditem0_.Data as data4_3_
from BakedItems bakeditem0_
where bakeditem0.Id in (
  select item1_.Id
  from Items item1_
  where item1_.Owner=@p0)
ORDER BY CURRENT_TIMESTAMP
OFFSET @p1 ROWS FETCH FIRST @p2 ROWS ONLY

@p0='F43D79B3-2D93-4E03-8066-4E0C71338DB5',
@p1=10,
@p2=10
```

В недрах executor’a

Ответственность:

- Specification.Source() -> Session.Query()
- IQuery - > IQueryable
- Expression<Func<T, IQueryContext, bool>> -> Expression<Func<T, bool>>

В недрах executor’a

```
var result = QueryMethodInfo
    .MakeGenericMethod(queryType)
    .Invoke(session, null) as IQueryable;

foreach (var part in expressions)
{
    switch (part.Type)
    {
        case EQueryType.Where:
            result = ApplyWhereExpression(session, result, part, queryContext);
            break;

        case EQueryType.Projection:
            result = ApplyProjectionExpression(result, part);
            break;

        ...
    }
}

private IQueryable ApplyProjectionExpression(IQueryable source, QueryParameter queryExpression)
{
    var parameters = (SelectQueryParameter)queryExpression;
    var select = SelectMethodInfo.MakeGenericMethod(parameters.InType, parameters.OutType);
    return (IQueryable)select.Invoke(source, new object[] { source, parameters.Expression });
}
```

В недрах executor’a

```
private IQueryable ApplyWhereExpression(
    ISession session,
    IQueryable source,
    QueryParameter queryExpression,
    QueryContext queryContext)
{
    var parameters = (WhereQueryParameter)queryExpression;
    var where = WhereMethodInfo.MakeGenericMethod(parameters.InType);
    var expression = (LambdaExpression)parameters.Expression;
    if (expression.Parameters.Any(x => x.Type == typeof(IQueryContext)))
    {
        var entityArgument = expression.Parameters.First(x => x.Type == parameters.InType);
        var visiter = new SourceReplacerVisitor(this, queryContext, session);
        var updatedExpression = (LambdaExpression)visiter.Visit(expression);
        expression = Expression.Lambda(updatedExpression.Body, entityArgument);
    }

    return (IQueryable)where.Invoke(source, new object[] { source, expression });
}
```

В недрах executor’a

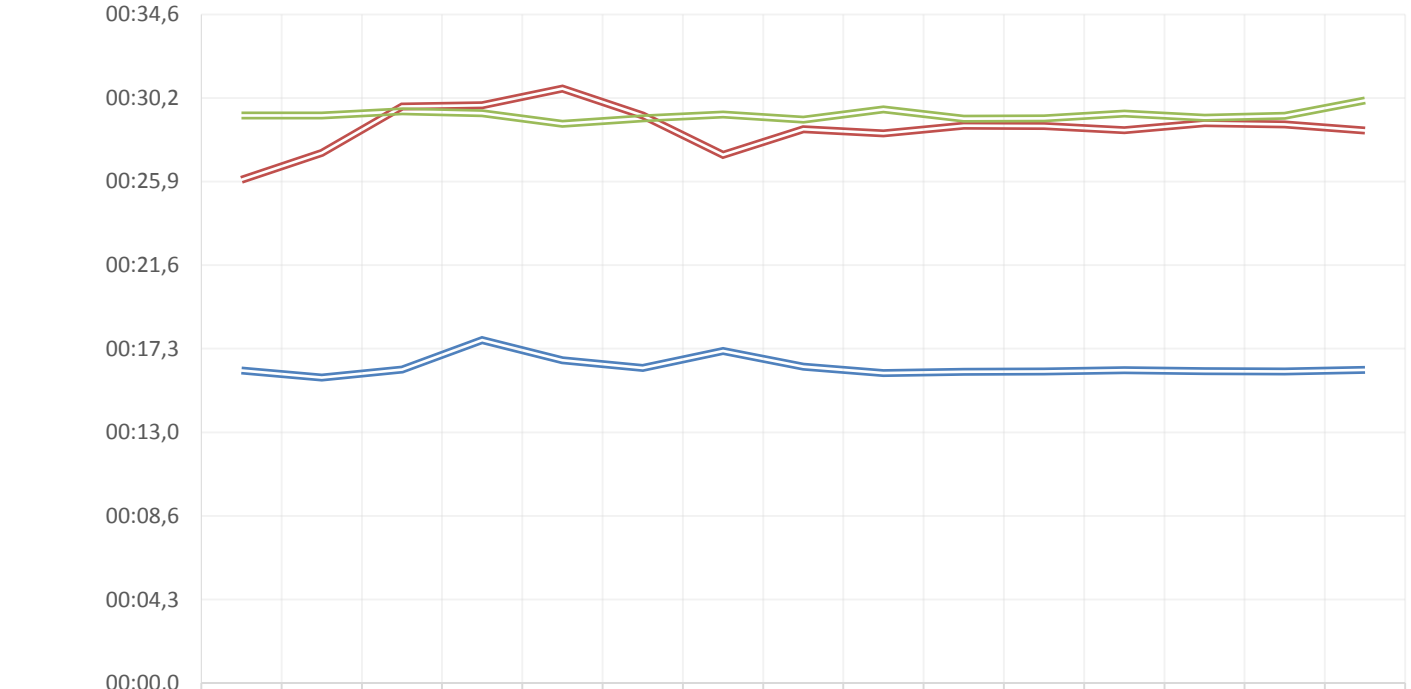
```
protected override Expression VisitMethodCall(MethodCallExpression node)
{
    if (node.Arguments.Count == 1 &&
        node.Object != null &&
        node.Object.Type == typeof(IQueryContext))
    {
        var value = (string)((ConstantExpression)node.Arguments[0]).Value;
        var query = _context.GetQuery(value);
        var source = _executor.CreateQueryable(_session, (Query)query, (QueryContext)_context);
        var sourceExpression = Expression.Constant(source, source.GetType());

        return sourceExpression;
    }

    return base.VisitMethodCall(node);
}
```

Профит

- Простой способ описания бизнес-правил для выборки объектов.
- Отсутствие дублирования кода.
- Возможность построения более сложных цепочек с помощью булевых операций.
- Бизнес правила остались на слое домена.
- Запросы выполняются на стороне БД.
- Потенциальная расширяемость IQuery.
- Каждой технологии свой SpecificationExecutor.

Идеальное решение?

- Производительность:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SQL	00:16,2	00:15,8	00:16,2	00:17,7	00:16,7	00:16,3	00:17,2	00:16,4	00:16,0	00:16,1	00:16,1	00:16,2	00:16,1	00:16,1	00:16,2
IQueryable	00:26,0	00:27,4	00:29,8	00:29,9	00:30,7	00:29,3	00:27,3	00:28,6	00:28,4	00:28,8	00:28,8	00:28,6	00:28,9	00:28,9	00:28,6
Specification	00:29,3	00:29,3	00:29,6	00:29,5	00:28,9	00:29,2	00:29,4	00:29,1	00:29,7	00:29,2	00:29,2	00:29,4	00:29,2	00:29,3	00:30,1

Идеальное решение?

- Нет предела совершенству:
  - Расширение IQuery (Fetch, GroupBy, …).
  - Оптимизация рефлексии в SpecificationExecutor.
  - Кэширование IQueryContext.GetSource.
  - Проверка runtime -> compiletime.
- Можно сделать лучше:

https://github.com/anatoly-kryzhanovsky/singularis.specification/tree/develop

Вопросы?

anatoly.kryzhanovsky@singularis-lab.com

iT-34

vk.com/it34_community