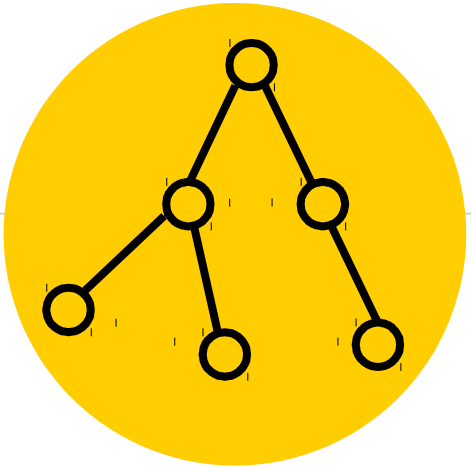
Реализация своих поставщиков запросов

Третьяков Антон

Обо мне

- Занимаюсь разработкой web приложений на .NET ~ 5 лет.
- Последние 2 года участвовал в разработке больших корпоративных B2B приложений.
- На данный момент занимаюсь разработкой инструментария.

Третьяков Антон



IQueryProvider

Что такое IQueryProvider

```
public interface IQueryProvider
{
    IQueryable<T> CreateQuery<T> (Expression Expression);

    TResult Execute<TResult> (Expression Expression);
}
```

Что такое IQueryProvider

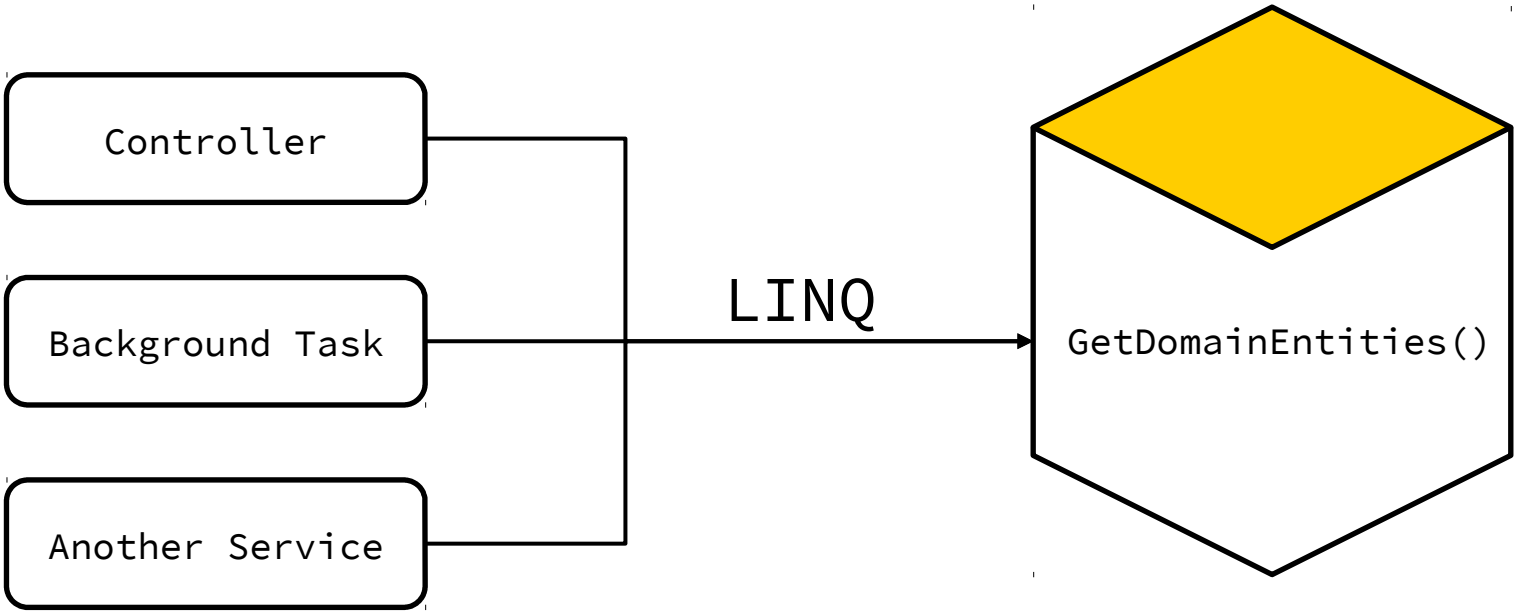
```
public interface IQueryable : IEnumerable
{
    Type ElementType { get; }
    Expression Expression { get; }
    IQueryProvider Provider { get; }
}
```

Когда можно использовать?

- Дополнение и изменение **IQueryable**
- Конвертация **IQueryable** в запросы к другим системам
- Построение фасада для неудобного **API**

1 Расширение запросов

Расширение запросов



Расширение запросов

```
public IEnumerable<T> GetDomainEntities()
{
    return Container.GetDomain<T>()
        .Where(x=> x.Id > 1)
        .OrderBy(x=> x.Id)
        .WithAccess();
}
```

Расширение запросов

```
public static class AccessExtensions
{
    ... IQueryable<T> WithAccess<T> (this IQueryable<T> source)
    {
    Expression accessRules = Container.GetAccessRules();

    var linqWhere = GetLinqWhereMethod();

    return (IQueryable<T>)linqWhere
           .MakeGenericMethod(entityType)
           .Invoke(null, new object[] { source, accessRules });
    }
}
```

Расширение запросов

```
public static MethodInfo GetLinqWhereMethod()
{
      return typeof(Queryable).GetMethods()
         .Where(x => x.Name == "Where")
         .Select(x => new { a = x, P = x.GetParameters() })

     .Where(x => x.P.Length == 2
            && x.P[0].ParameterType.IsGenericType
           && x.P[1].ParameterType.IsGenericType
           && x.P[0].ParameterType.GetGenericTypeDefinition() == typeof(IQueryable<>)
           && x.P[1].ParameterType.GetGenericTypeDefinition() == typeof(Expression))

     .Select(x => new { M = x.a, b = x.P[1].ParameterType.GetGenericArguments() })
     .Where(x => x.b[0].IsGenericType
        && x.b[0].GetGenericTypeDefinition() == typeof(Func<,>))

          .Select(x => new { x.M, b = x.b[0].GetGenericArguments() })
     .Where(x => x.b[0].IsGenericParameter && x.b[1] == typeof(bool))
          .Select(x => x.M)
          .SingleOrDefault();
}
```

Расширение запросов

```
public IEnumerable<T> GetDomainEntities()
{
    return Container.GetDomain<T>()
        .Where(x=> x.Id > 1)
        .OrderBy(x=> x.Id)
        .WithAccess();
}
```

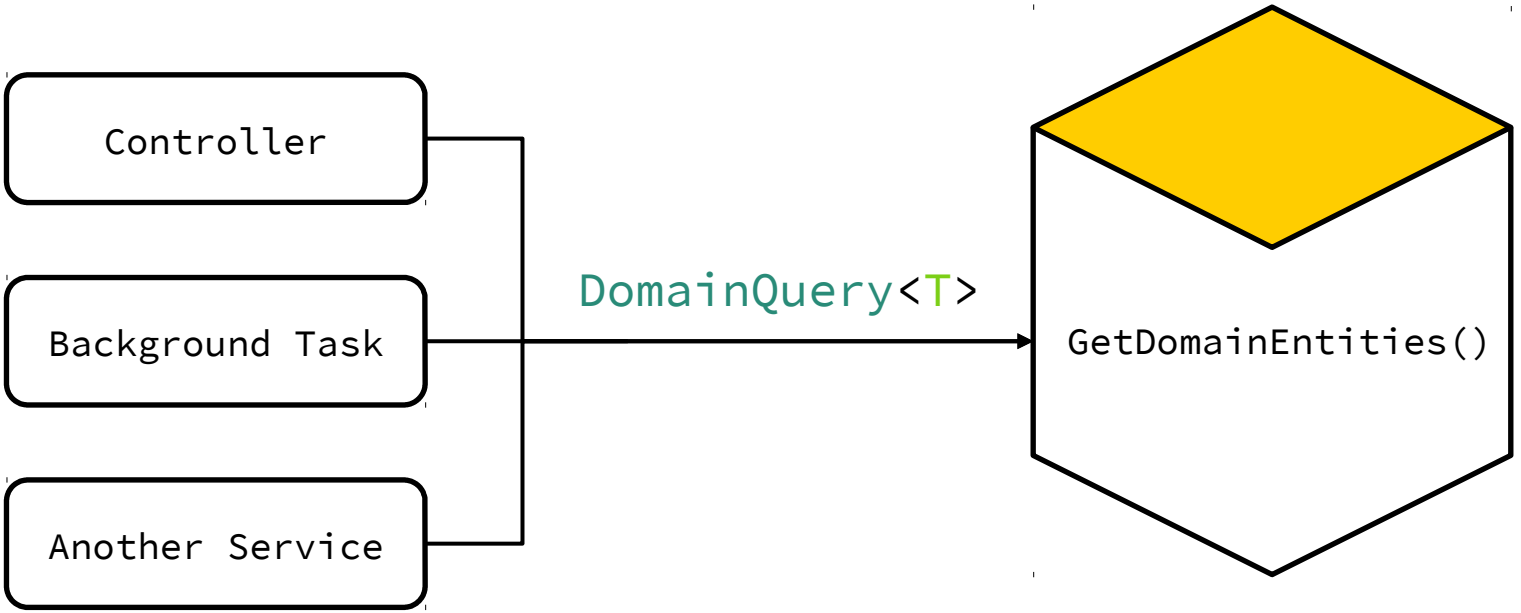
Как можно решить эти задачи используя поставщики запросов.

Расширение запросов

```
public static class AccessExtensions
{
    ... IQueryable<T> WithAccess<T> (this IQueryable<T> source)
    {
        Expression accessRules = Container.GetAccessRules();

        return source.Provider.CreateQuery<T>(
      Expression.Call(null,
         CachedReflectionInfo.Where_TSource_2(typeof(T)),
             source.Expression,
             Expression.Quote(accessRules))
        );
    }
}
```

Расширение запросов



Расширение запросов

```
public class DomainQuery<T> : IQueryable<T>
{
      Expression AccessRules { get; }
      IQueryable<T> Original { get; }
      DomainQueryProvider Provider { get; }
}
```

Когда будет вызван поставщик запросов с бизнес-логикой?

Расширение запросов

```
public static class Queryable
{
    IQueryable<T> Where<T>(IQueryable<T> source, Expression<Func<T,bool>> expression)
    {
     return source.Provider.CreateQuery(
        Expression.Call(null,
         CachedReflectionInfo.Where_TSource_2(typeof(T)),
             source.Expression,
             Expression.Quote(expression))
        );
    }
}
```

Расширение запросов

```
public class DomainQueryProvider : IQueryProvider
{
    IQueryable<T> CreateQuery<T>(Expression expression)
    {
     Expression protected = Expression.Call(null,
         CachedReflictionInfo.Where_TSource_2(typeof(T)),
             expression,
             Expression.Quote(domainQuery.AccessRules))
        );

     var newExternal = domainQuery
         .Original.Provider.CreateQuery<T>(protected);

     return new DomainQuery(newExternal);
       }
    }
}
```

Расширение запросов

CreateQuery

Создаёт новый IQueryable, с переписанным деревом выражения

Не использует рефлексию, нет скрытых вызовов провайдера.

Инкапсулирует логику формирования запроса, позволяет забыть о явных вызовах методов расширения.

2

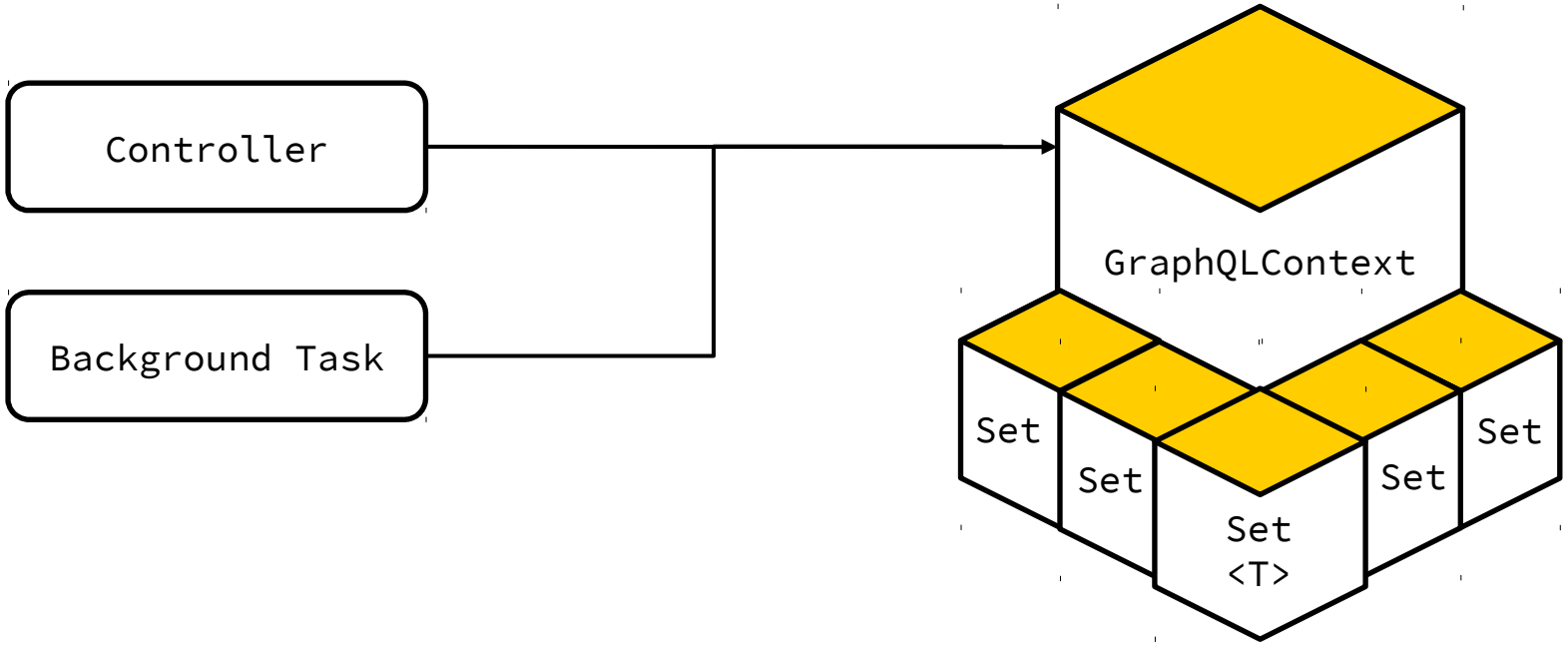
Конвертация запросов

GraphQL

```
entity(filter : “value”){
    property,
    reference {
        property
    }
}
```

Entity Framework Core

Расширение запросов



DomainQuery<T>

Конвертация запросов

```
public class NodalContext : GraphQLContext
{
    public GraphQLSet<User> Users { get; set; }
    public GraphQLSet<Thread> Threads { get; set; }
    public GraphQLSet<Post> Posts { get; set; }
}
```

Конвертация запросов

```
using(var ctx = new NodalContext("http://graphql.nodaljs.com/graph"))
{
    var data = ctx.Threads.Select(t => new
               {
                  t.Id,
                  t.Title,
                  user = new
                  {
                     t.User.Id,
                     t.User.UserName
                  },
                  posts = ctx.Posts.Select(p => new
                  {
                     p.Body,
                     user = new { p.User.UserName },
                     p.Created
                  }).Where(p => p.Body.Contains("hello"))
               }).Where(x => x.Id > 1).ToList();
}
```

ToList()

```
public static List<T> ToList<T>(this IEnumerable<T> source)
{
    return new List<T>(source);
}
```

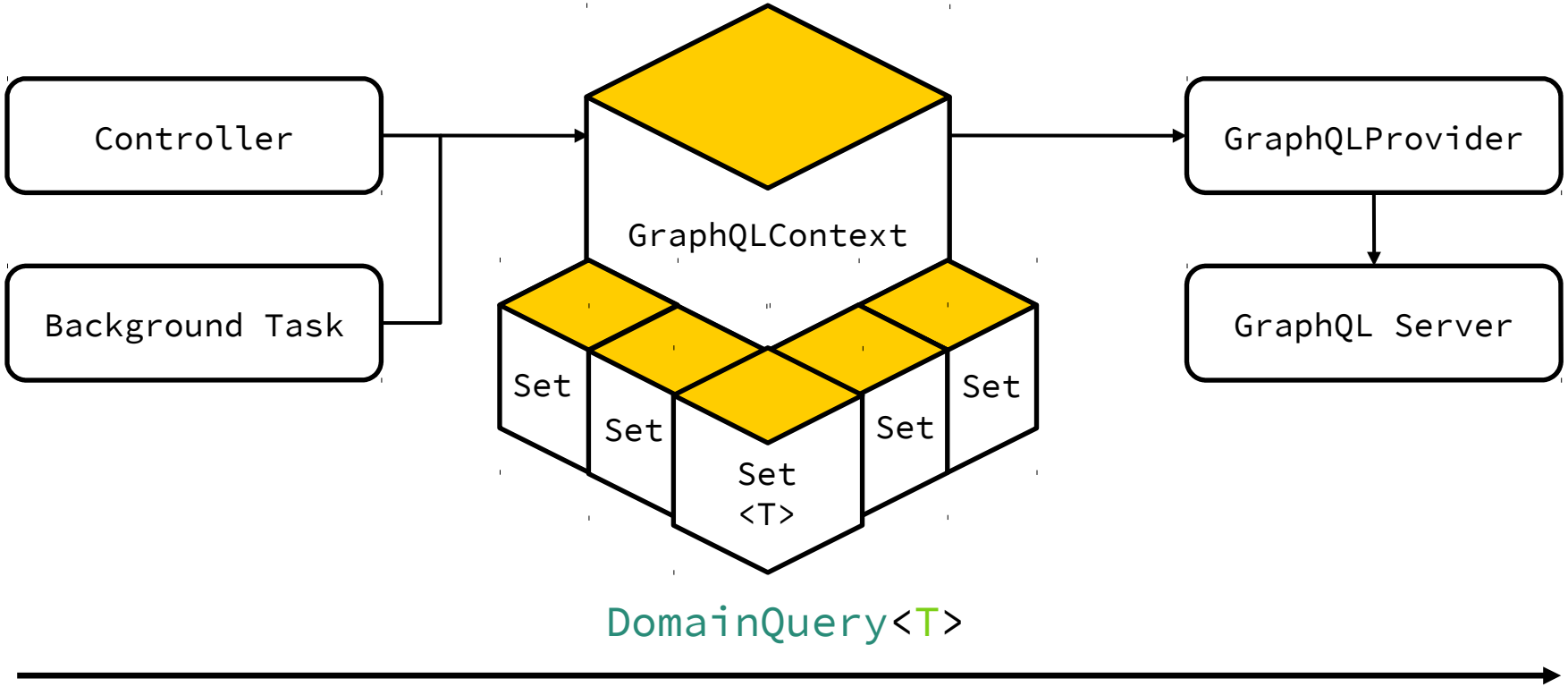
```
using(IEnumerator<T> en = enumerable.GetEnumerator())
{
    while(en.MoveNext())
    {
        _items[size++] = en.Current;
    }
}
```

Конвертация запросов

```
public class GraphQLSet<T> : IQueryable<T>
{
  public Type ElementType => typeof(T);
  public Expression Expression { get; private set; }
  public IQueryProvider Provider { get; private set; }

  public IEnumerator<T> GetEnumerator()
   => Provider.Execute<IEnumerator<T>>(this.Expression);
}
```

Расширение запросов



Конвертация запросов

```
public class GraphQLProvider : IQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • Validate


   }
}
```

Конвертация запросов

```
public class GraphQLProvider : IQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • Validate
       • Processing Expression

   }
}
```

Конвертация запросов

```
public class GraphQLQueryProperty
{
  public Type CLRType { get; set; }
  public string Name { get; set; }
}


public class GraphQLQueryTree : GraphQLQueryProperty
{
  public Filter Filter { get; set; }
  public ICollection <Property> Properties { get; set; }
}
```

Конвертация запросов

```
public class GraphQLVisitor : ExpressionVisitor
{
  protected override Expression Visit(Expression)
  protected override Expression VisitBinary(BinaryExpression)
  protected override Expression VisitNew(NewExpression)
  protected override Expression VisitConst(ConstantExpression)
  protected override Expression VisitLoop(LoopEpression)
  protected override Expression VisitGoto(GotoEpression)
  protected override Expression VisitSwitch(SwitchEpression)
}
```

Конвертация запросов

```
protected override Expression VisitBinary(BinaryExpression node)
{
    if(!TryGetOperation(node, out var operation))
       !TryGetEntity
       !TryGetProperty
       !TryGetValue
            return base.VisitBinary(node);

       entity.Filter = new Filter
       {
      Name=property,
      Operation=operation,
      Value=value
       };
       return base.VisitBinary(node);
  }
```

```
public class GraphQLQueryProperty
{
  public Type CLRType { get; set; }
  public string Name { get; set; }
}
```

Конвертация запросов

```
public class GraphQLProvider : IQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • Validate
       • Process Expression
       • Materialize
   }
}
```

Конвертация запросов

```
public class GraphQLTreeMaterializer<T>
{

  public IEnumerator<T> Materialize(GraphQLQueryTree queryTree)
  {
    return new GraphQLResponseEnumerator<T>(SendRequestAsync(queryTree));
  }


  public Task<GraphQLResponse<T>> SendRequestAsync<T>(.. queryTree)
  {
       •Serialize
       •Async HttpPost
       •Deserialize<T>
  }
}
```

Конвертация запросов

```
public class GraphQLResponseEnumerator<T> : IEnumerator<T>
{
  public GraphQLResponseEnumerator(Task<GraphQLResponse<T>> deferred);

  public T Current => enumeratorFromDeferred.Current;

  public bool MoveNext() => enumeratorFromDeferred.MoveNext();
}
```

Конвертация запросов

```
data.ToList();
```

```
GraphQLSet<T>.GetEnumerator<T>()
```

```
GraphQLProvider.Execute<T>
```

```
GraphQLTreeMaterializer<T>
```

```
GraphQLResponseEnumerator<T>
```

Конвертация запросов

```
using(var ctx = new NodalContext("http://graphql.nodaljs.com/graph"))
{
    var data = ctx.Threads.Select(t => new
               {
                  t.Id,
                  t.Title,
                  user = new
                  {
                     t.User.Id,
                     t.User.UserName
                  },
                  posts = ctx.Posts.Select(p => new
                  {
                     p.Body,
                     user = new { p.User.UserName },
                     p.Created
                  }).Where(p => p.Body.Contains("hello"))
               }).Where(x => x.Id > 1).ToList();
}
```

Конвертация запросов

```
threads(threads__id__gt:1) {
    id,
    title,
    posts(threads__posts__body__contains:"hello") {
        body,
        created,
        user {
          username
        }
    },
    user {
      id,
      username
    }
}
```

Конвертация запросов

```
(local variable) System.Collections.Generic.List<'a> data

Anonymous Types:
    'a is new { int Id, string Title, 'b user, IQueryable<'c> posts }
    'b is new { int Id, string UserName }
    'c is new { string Body, 'd user, DateTime? Created }
    'd is new { string UserName }
```

Конвертация запросов

Execute

Имеет удобное API и слабо связное отношение с **`IQueryable`**.

Возможность работы через **`ExpressionVisitor`**.

Позволяет структурированно преобразовать любой **`IQueryable`** в нужную вам структуру данных.

3

Построение фасада API

Построение фасада API

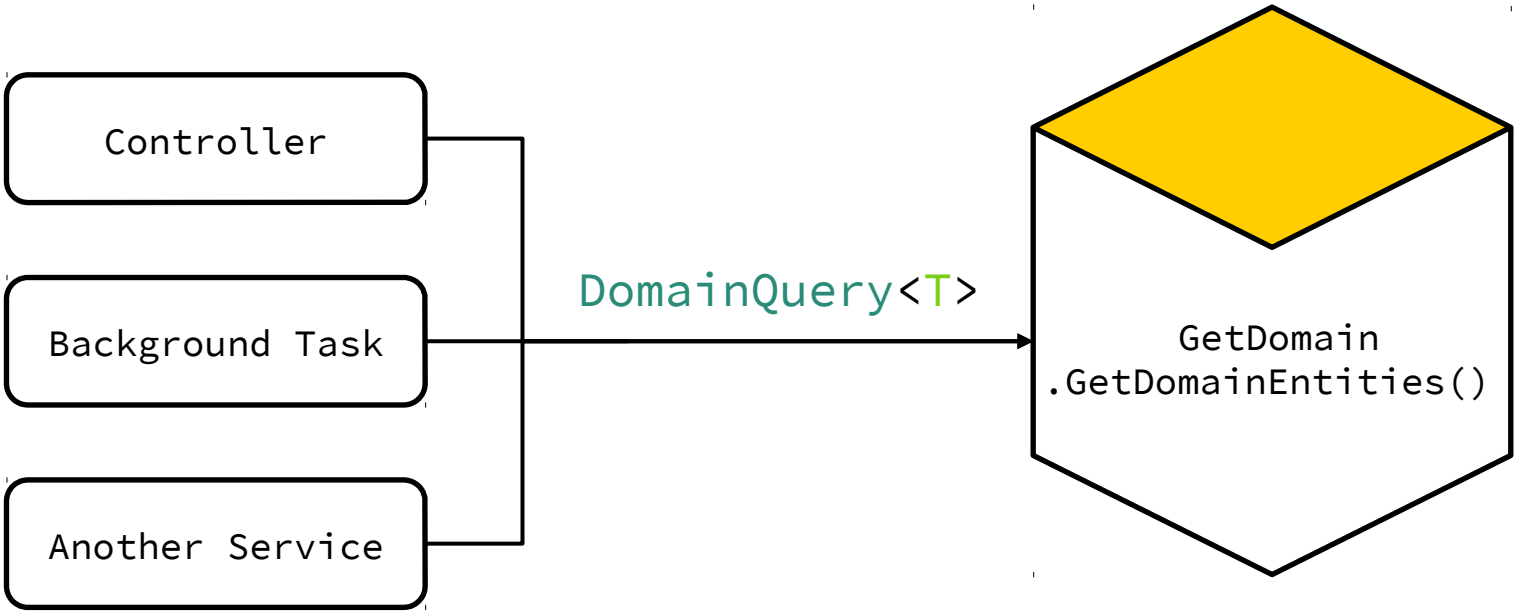
```
public class WrongUserApi
{
  public TAccessor GetAccessor<TAccessor>()
   where TAccessor : IAccessor
   { .. }
}
```

Построение фасада API

```
public class FirstUserPartAccessor : IAccessor
{
    public string[] GetUsersIdAsync(Search search) { network call }
}
```

```
public class SecondUserPartAccessor : IAccessor
{
    public UserWithoutId GetUserAsync(string userId) { network call }
}
```

Расширение запросов



Построение фасада API

```
public IQueryable<T> GetDomain<T>()
{
   var wrongApi = Container.Resolve<WrongUserApi>();

   Search search = "id_greater_1“;

   var userIds = wrongApi
      .GetAccessor<FirstUserPartAccessor>()
      .GetUsersId(search);

   List<ApiUser> users = userIds.Select(userId=>
   {
      var userPart = wrongApi.GetAccessor<SecondUserPartAccessor>(userId);
      return new ApiUser(){ Id = Int.Parse(userId), .. userPart });
   });

   return users.AsQueryable();
}
```

Построение фасада API

- Невозможность фильтрации в момент выполнения
- Множество сетевых запросов
- Полная загрузка данных в память

Каким хочется видеть API для запроса к стороннему сервису

Построение фасада API

```
var apiFacade = Container.Resolve<ApiUserService>();

var users = apiFacade.Where(x=> x.UserName.Contains(“alex"))
    .ForCompany(Companies.Microsoft)
    .IncludeBlocked()
    .RegisteredAfter(DateTime.Now.AddDays(-1))
    .OrderByDescending(x=>x.RegisteredDate)
    .Skip(5)
    .Take(10)
    .ToList();
```

Построение фасада API

```
public class ApiUserService : IQueryable<ApiUser>
{
  public Type ElementType => typeof(ApiUser);
  public Expression Expression { get; private set; }
  public IQueryProvider Provider { get; private set; }
}
```

Построение фасада API

```
public static IQueryable<ApiUser> ForCompany(this .. apiQuery, Companies company)
{
    return apiQuery.Where(x=>x.Filter.SupportedCompanies.Contains(company));
}
```

```
public static IQueryable<ApiUser> IncludeBlocked(this .. apiQuery)
{
    return feedQuery.Where(x=>x.Filter.CanBlocked == true);
}
```

Построение фасада API

```
public sealed class ApiUser
{
    public long Id { get; set; }
    public string UserName { get; set; }
       …
    internal readonly SearchInfo Filter
         = new SearchInfo();
}
```

Построение фасада API

```
var apiFacade = Container.Resolve<ApiUserService>();

var users = apiFacade.Where(x=> x.UserName.Contains(“alex"))
    .ForCompany(Companies.Microsoft)
    .IncludeBlocked()
    .RegisteredAfter(DateTime.Now.AddDays(-1))
    .OrderByDescending(x=>x.RegisteredDate)
    .Skip(5)
    .Take(10)
    .ToList();
```

Построение фасада API

```
public class ApiUserQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • VisitExpression => Search;


   }
}
```

Построение фасада API

```
public class ApiUserExpressionVisitor : ApiUserVisitor
{
  public override Expression VisitMethodCall(MethodCallExpression node)
  {
    if (["Skip", "Take"].Contains(node.Method.Name))
    {
       return new ApiUserSkipTakeVisitor().Visit(node);
    }

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

Построение фасада API

```
public class ApiUseSkipTakeVisitor : ApiUserVisitor
{
   public override Expression VisitMethodCall(MethodCallExpression node)
   {
     this.EnsureCount(node);
    // this.Visit => this.VisitConstant => this.count=node.value;

     if (node.Method.Name==“Skip")
     {
    this.Search.Skip = this.count;
     }

        return node.Arguments.First(); // rewrite node
     }
}
```

Построение фасада API

```
public class ApiUserQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • VisitExpression => Search;
       • WrongUserApi.Search(Search) => DeferredEnumerator

   }
}
```

Построение фасада API

```
public class ApiUserQueryProvider
{
   public T Execute<T>(Expression expression)
   {
       • VisitExpression => Search;
       • WrongUserApi.Search(Search) => DeferredEnumerator
       • Is Sync?
   }
}
```

Построение фасада API

```
var apiFacade = Container.Resolve<ApiUserService>();

var users = apiFacade.Where(x=> x.UserName.Contains(“alex"))
    .ForCompany(Companies.Microsoft)
    .IncludeBlocked()
    .RegisteredAfter(DateTime.Now.AddDays(-1))
    .OrderByDescending(x=>x.RegisteredDate)
    .Skip(5)
    .Take(10)
    .CanSync()
    .ToList();
```

Построение фасада API

```
protected void NotEvaluated(MethodCallExpression node)
{
   this.AddApplier(x=>
    Expression.Call(node.Method, x, node.Arguments.Last()));
}

internal IQueryable<ApiUser> GetNotEvaluated(IQueryable<ApiUser> from)
{
    Expression root = Expression.Constant(from);
    appliers.Reverse();
    foreach (var applier in appliers)
    {
         root = applier(root);
    }
    return Expression.Lambda(root).Compile().DynamicInvoke();
}
```

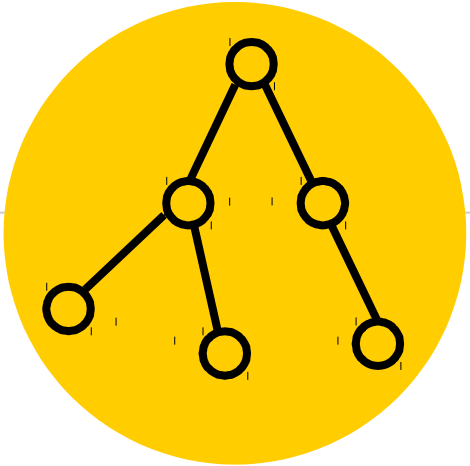
Построение фасада API

```
var users = …
    . CanSync();
    . ToList();


System.Collections.Generic.List`1[ApiUser]
    .OrderByDescending(x => x.RegisteredDate)
```

Построение фасада API

- Можем фильтровать запрос в момент выполнения
- Уменьшим количество сетевых запросов
- Нагрузка на память будет значительно снижена.



Итоги

Итоги

- Использование `IQueryProvider` избавляет от ненужных использований reflection.
- Позволяет единообразно и удобно конвертировать запросы ко внешним источникам.
- Использование выражений позволяет создавать удобный фасад для извлечения информации.

Спасибо!

Со мной можно связаться:

- anete.anetes@gmail.com
- https://github.com/anetegithub

API : https://github.com/barsgroup/NuGet.Querying

GraphQL : https://github.com/anetegithub/Linq.GraphQL