Паттерн Репозиторий

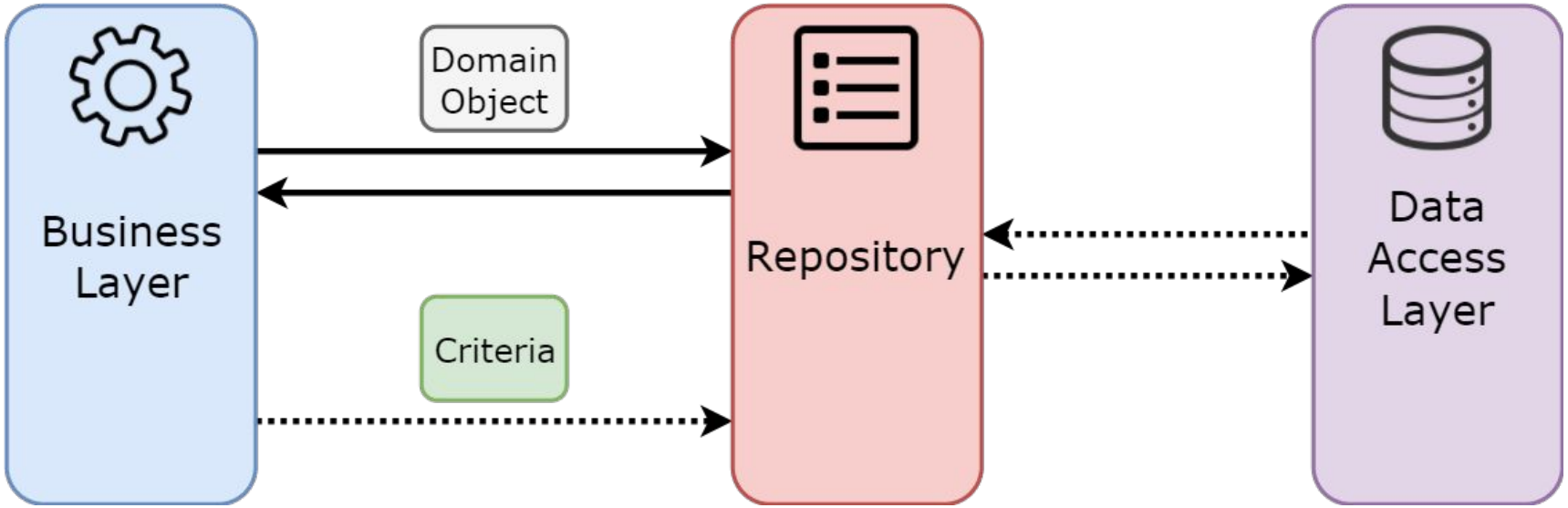
https://metanit.com/sharp/articles/mvc/11.php
https://habr.com/post/248505/
http://merle-amber.blogspot.com/2009/02/orm.html
https://habr.com/post/335856/
https://itvdn.com/ru/blog/article/b-repository

Репозиторий (Repository)

Один из наиболее часто используемых паттернов при работе с данными.

Назначение: **Разделение бизнес-логики от деталей реализации слоя доступа к данным**.

Паттерн Репозиторий стал популярным благодаря DDD (Domain Driven Design). В противоположность к Database Driven Design в DDD разработка начинается с проектирования бизнес логики, принимая во внимание только особенности предметной области и игнорируя все, что связано с особенностями базы данных или других способов хранения данных. Способ хранения бизнес объектов реализуется во вторую очередь.



Репозиторий является посредником между слоем доступа к данным и доменным слоем, работая как in-memory коллекция доменных обектов. Клиенты создают декларативные описания запросов и передают их в репозиторий для выполнения.

(свободный перевод Мартина Фаулера)

В реализацию «Репозитория» обычно включается следующий функционал (**CRUD**):

- Создание (Create). Добавление записи в БД (или другое хранилище данных);
- Чтение (Read). Выборка имеющихся записей из БД;
- Обновление (Update). Редактирование имеющихся записей;
- Удаление (Delete) имеющихся записей.

Хорошей практикой считается создание отдельных репозиториев для каждого бизнес-объекта (POCO) или контекста, например: BooksRepository, UsersRepository, AdminRepository.

Традиционный CLR объект (**Plain Old CLR Object - POCO**) – это класс, который используется для представления класса бизнес сущности (модели), содержащий свойства и методы, характерные для бизнес-сущности, и не содержащий специфического кода для доступа к данным.

```
public interface IPostsRepository
{
    void Save(Post mypost);
    Post Get(int id);
    PaginatedResult<Post> List(int skip,int pageSize);
    PaginatedResult<Post> SearchByTitle(string title,int skip,int pageSize);
}
```

Можно создавать **Generic Repository,** но только если приложение работает с данными одинаково.

```
public interface IRepository<T> where T:class
    {
        IEnumerable<T> GetAll();
        T Get(int id);
        IEnumerable<T> Find(Func<T, Boolean> predicate);
        void Create(T item);
        void Update(T item);
        void Delete(int id);
    }
```

Можно использовать обобщенный интерфейс в качестве базового для других

```
public interface IRepository<T> : IDisposable where T : class
    {
        void Create(T item);
        void Delete(T item);
        void Update(T item);
        T GetById(int id);
        IEnumerable<T> GetAll();
    }

public  interface IDoctorRepository : IRepository<Doctor>
{
}


public  interface IPatientRepository:IRepository<Patient>
{
}
```

Пример реализации интерфейса с использованием в качестве хранилища данных базы данных MS Sql Server

```
class PatientRepository : IPatientRepository
    {
        SqlConnection connection;
        public PatientRepository(string connectionString)
        {
            connection = new SqlConnection(connectionString);
        }

        public void Create(Patient item)
        {
            connection.Open();
            SqlCommand command = new SqlCommand("CreatePatient", connection);
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.Add("@lastname", SqlDbType.NVarChar).Value =
                                                item.Lastname;
            command.ExecuteNonQuery();
            connection.Close();
        }

        и т.д.
```

Использовать ли паттерн Репозиторий, если используется **ORM**…

ORM (Object-Relational Mapping, *объектно-реляционное отображение*, или преобразование) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»

ORM позволяет:

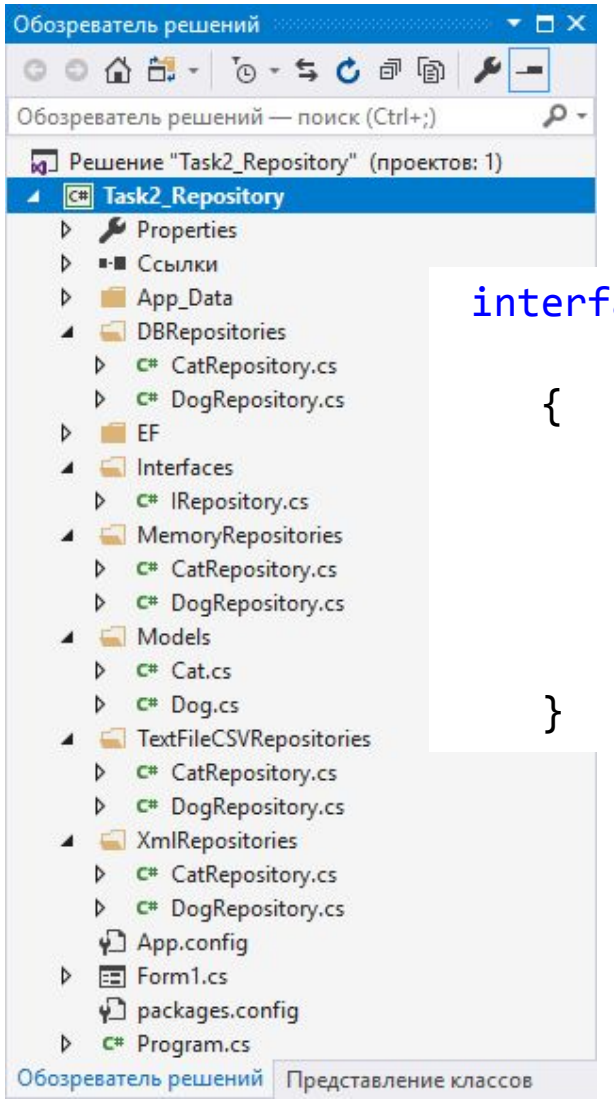
- работать с данными, оперируя бизнес-объектами (POCO).
- легко сменить поставщика данных (SQL Server на MySql и т. д.)

Если хранение данных осуществляется не только с использованием ORM, то тогда такой слой данных инкапсулируется с помощью паттерна репозиторий.

Преимущества паттерна Репозиторий:

- Независимость бизнес-логики от способа хранения. Использование коллекций бизнес объектов (POCO), а не database related объектов и т.п. Возможность использовать разные способы хранения: ORM, cloud storage, file system и т. д, заменять их и комбинировать.
- Работая через интерфейсы, можно создать несколько реализаций репозитория.

Пример. Реализовать паттерн Репозиторий в приложении, управляющем информацией о пациентах ветеринарной клиники.




```
interface IRepository<T>
        where T : class
    {
        IEnumerable<T> GetAll (); // получение всех объектов
        T Get(int id); // получение одного объекта по id
        void Create(T item); // создание объекта
        void Update(T item); // обновление объекта
        void Delete(int id); // удаление объекта по id
    }
```

Пример реализации интерфейса для сущностей класса Dog с использованием в качестве хранилища данных текстового файла с информацией в формате CSV

```
class DogRepository : IRepository<Dog>
    {
        List<Dog> dogs;
        string pathFile;
        public DogRepository(string pathFile)
        {
            this.pathFile = pathFile;
            dogs = new List<Dog>();
        }
        public void Create(Dog item)
        {
            if (!dogs.Any(d => d.Id == item.Id))
            {
                dogs.Add(item);
                Write();
            }
            else
                throw new Exception("This dog already exists!");
        }
```

```
public void Delete(int id)
        {
            dogs.Remove(dogs.Find(a => a.Id == id));
            Write();
        }

        public Dog Get(int id)
        {
            Read();
            return dogs.Find(a => a.Id == id);
        }

        public IEnumerable<Dog> GetAll()
        {
            Read();
            return dogs;
        }
```

```
public void Update(Dog item)
        {
            int i = -1;
            i = dogs.FindIndex(a => a.Id == item.Id);
            if (i > -1)
            {
                dogs[i].Name = item.Name;
                dogs[i].Age = item.Age;
                Write();
            }
            else
                throw new Exception("This id does not exist!");
        }
```

```
private void Read()
  {
      using (StreamReader reader = new StreamReader(pathFile))
      {
            dogs.Clear();
            while (!reader.EndOfStream)
            {
               try
               {
                   string[] temp = reader.ReadLine().Split(';');
                   dogs.Add(new Dog() { Id = Convert.ToInt32(temp[0]),
                     Name = temp[1], Age = Convert.ToInt32(temp[2]) });
                }
                catch
                {
             throw new Exception("Could not read csv file in directory "
                                    + pathFile);
                }
            }
        }
  }
```

```
private void Write()
        {
            StringBuilder builder;
            using (StreamWriter writer = new StreamWriter(pathFile))
            {
                builder = new StringBuilder();
                foreach (Dog item in dogs)
                {
                    builder.Append(item.Id)
                           .Append(";")
                           .Append(item.Name)
                           .Append(";")
                           .Append(item.Age);
                    writer.WriteLine(builder.ToString());
                    builder.Clear();
                }
            }
        }
```