How do I design a Data Access Layer appropriately?

Asked 9 years, 1 month ago Active 1 year, 8 months ago Viewed 19k times

8

7

I have the following Data Access Layer (DAL). I was wondering if it's set up correctly, or if I need to improve it?

```
public class User
{

}

//Persistence methods
static class UserDataAccess
{
   UsersDAL udal = // Choose SQL or FileSystem DAL impl.


   InsertUser(User u)
   {
      // Custom logic , is 'u' valid etc.

      udal.Insert(u);
   }
}

abstract class UsersDAL
{
   GetUserByID();
   InsertUser(u);
   ...
}

// implementaitons of DAL

static class UsersSQLStore : UsersDAL
{

}

static class UsersFileSystemStore : UsersDAL
{

}
```

I separated the storage layer from the User class to access methods collection which further call any custom DAL.

Is use of `static` in DAL implementation correct?

Please suggest corrections or ways I can make this better. I don't have a lot of experience with writing code in layers.

c# .net design-patterns architecture data-access-layer

edited Jan 6 '11 at 20:47
Adam Rackis
76.3k 45 237 363

asked Jan 6 '11 at 18:08
Munish Goyal
1,279 4 24 47

2 If you can't take the time to fully spell out your question (using Pl. instead of Please), then how do you expect someone to take the time to answer your question or help you out? – George Stocker Jan 6 '11 at 18:22

7 @George, I dont know if that hurts someone, but just to save people reading too much, i use that regularly. Instead i concentrated on writing down my example. That does not mean i dont appreaciate people's time and their responses. – Munish Goyal Jan 6 '11 at 18:39

Why would you want to do this instead of using an ORM like LLBLGen or Dapper? No need to reinvent the wheel. – Adam Spicer Jan 24 '12 at 18:57

3 Answers

13

None of those classes should be `static`. I don't think you should name your classes `DAL` either, because its short for Data Access Layer, and a class in itself is not a layer (in my mind at least). You can use the widely adopted term [repository](#) instead. I suggest you do something like the following:

```
public class User{

}

public abstract class UserRepository{
    public abstract void InsertUser(User user);
}

public class SqlUserRepository : UserRepository{
    public override void InsertUser(User user)
    {
      //Do it
    }
}

public class FileSystemUserRepository : UserRepository{
    public override void InsertUser(User user)
    {
      //Do it
    }
}

public class UserService{
    private readonly UserRepository userRepository;

    public UserService(UserRepository userRepository){
        this.userRepository = userRepository;
    }

    public void InsertUser(User user){
        if(user == null) throw new ArgumentNullException("user");
        //other checks
        this.userRepository.InsertUser(user);
    }
}
```

Note that the `UserService` is injected with an instance of the abstract class `UserRepository` in its constructor. You can use a Dependency Injection (DI) framework to do this for you automatically, such as Windsor Castle from Castle Project. It will allow you to specify a mapping from abstraction (`UserRepository`) to concrete implementation (eg. `SqlUserRepository`) in a configuration file, or in code.

Hope this points you in the right direction, and please ask if you need more information.

answered Jan 6 '11 at 18:25

Klaus Byskov Pedersen
94.9k 23 169 214

Thats template pattern. You are in right direction with few changes as suggested by Hoffmann and Jani – hungryMind Jan 6 '11 at 18:30

Great explanation. Why shoudlnt i use static ? If not for repositories, For UserService I can make it static i guess ? 1 more thing (may be unrelated to DAL): If existing user is tried to insert , then who should make this check , UserService or User class itself via constructors (since it doesnt know abt xistence when it is newed) – Munish Goyal Jan 6 '11 at 18:45

@Munish Goyal, you should not use `static` because there is no need for it. When used inappropriately, `static` introduces unnecessary state in your application, which makes it both harder to debug and most importantly, it makes it **very** hard to test automatically. You probably want to read up on the exact meaning of `static` here: msdn.microsoft.com/en-us/library/98f28cdx%28vs.71%29.aspx – Klaus Byskov Pedersen Jan 6 '11 at 18:51

1 @Munish Goyal, I think the `UserService` should check if the user already exists, by asking the repository if a user with that id/name already exists. You would need a method called `bool UserAlreadyExists(User user)` in your repositories for this purpose. – Klaus Byskov Pedersen Jan 6 '11 at 18:53

1 @Munish Goyal, no, that is not correct. The abstract UserRepository, the User class, and the UserService must be in the same assembly (AssemblyA). The different implementations of the repositories can be in assemblies of their own (AssemblyB/C). The program that uses the UserService will need a reference to both AssemblyA and AssemblyB (or AssemblyC). – Klaus Byskov Pedersen Jan 7 '11 at 9:23

7

My Humble Opinions

1. Use interfaces instead of abstract class if User hasn't any hierarchy.
2. Write a generic DAL so that you can reuse it as a facade for DAL layer.
3. Resolve the concrete DALs by a DI framework

edited Jan 24 '12 at 18:49

answered Jan 6 '11 at 18:25
Jahan
13.5k 4 35 64

6

Davy Brion has an excellent set of blog posts on this subject: located on GitHub

edited Jun 1 '18 at 12:01
Julian
19.6k 11 73 104

answered Jan 6 '11 at 23:25
tijmenvdk
1,738 10 18

2 The link you included is now broken. Please update it. – John Washam Feb 25 '17 at 21:01