Data Layer Best Practices

Asked 16 years, 4 months ago Modified 16 years, 2 months ago

Viewed 4k times

9

I am in the middle of a "discussion" with a colleague about the best way to implement the data layer in a new application.

One viewpoint is that the data layer should be aware of business objects (our own classes that represent an entity), and be able to work with that object natively.

The opposing viewpoint is that the data layer should be object-agnostic, and purely handle simple data types (strings, bools, dates, etc.)

I can see that both approaches may be valid, but my own viewpoint is that I prefer the former. That way, if the data storage medium changes, the business layer doesn't (necessarily) have to change to accommodate the new data layer. It would therefore be a trivial thing to change from a SQL data store to a serialized xml filesystem store.

My colleague's point of view is that the data layer shouldn't have to know about object definitions, and that as long as the data is passed about appropriately, that is enough.

Now, I know that this is one of those questions that has the potential to start a religious war, but I'd appreciate any

feedback from the community on how you approach such things.

TIA

.net n-tier-architecture

Share

Improve this question

Follow

asked Aug 14, 2008 at 10:23

ZombieSheep

29.9k ● 12 ● 70 ● 114

6 Answers

Sorted by: Highest score (default)

5

It really depends on your view of the world - I used to be in the uncoupled camp. The DAL was only there to supply data to the BAL - end of story.

With emerging technologies such as Linq to SQL and Entity Framework becoming a bit more popular, then the line between DAL and BAL have been blurred a bit. In L2S especially your DAL is quite tightly coupled to the Business objects as the object model has a 1-1 mapping to your database field.

Like anything in software development there is no right or wrong answer. You need to understand your requirements and future requirments and work from there. I would no more use a Ferrari on the Dakhar rally as I would a Range Rover on a track day.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:38

JamesSugrue

15k ● 10 ● 62 ● 93

I totally agree. The design of Data Access Layers etc is becoming quite a bit blurry. Whereas I would always opt in to separate your business logic from presentation layers. MVC pattern FTW ;-) – Tobias N. Sasse Aug 26, 2012 at 17:58

3

You can have both. Let data layer not know of your bussiness objects and make it capable of working with more than one type of data sources. If you supply a common interface (or an abstract class) for interacting with data, you can have different implementations for each type of data source. Factory pattern goes well here.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:28

Serhat Ozgel

23.7k ● 29 ● 104 ● 141

1

An excellent book I have, which covers this topic, is Data Access Patterns, by Clifton Nock. It has got many good explanations and good ideas on how to decouple your business layer from the persistence layer. You really should give it a try. It's one of my favorite books.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:27

Mario Marinato

4,617 ● 3 ● 30 ● 50

0

One trick I've found handy is to have my data layer be "collection agnostic". That is, whenever I want to return a list of objects from my data layer, I get the caller to pass in the list. So instead of this:

```
public IList<Foo> GetFoosById(int id) { ... }
```

I do this:

```
public void GetFoosById(IList<Foo> foos, int id) { ...
```

This lets me pass in a plain old List if that's all I need, or a more intelligent implementation of IList<T> (like ObservableCollection<T>) if I plan to bind to it from the UI. This technique also lets me return stuff from the method like a ValidationResult containing an error message if one occurred.

This still means that my data layer knows about my object definitions, but it gives me one extra degree of flexibility.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:28

Matt Hamilton

204k ● 61 ● 392 ● 321

0

Check out Linq to SQL, if I were creating a new application right now I would consider relying on an entirely Linq based data layer.

Other than that I think it's good practise to de-couple data and logic as much as possible, but that isn't always practical. A pure separation between logic and data access makes joins and optimisations difficult, which is what makes Linq so powerful.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:28

Keith

155k ● 82 ● 306 ● 446

0

In applications wherein we use NHibernate, the answer becomes "somewhere in between", in that, while the XML mapping definitions (they specify which table belongs to which object and which columns belong to which field, etc) are clearly in the business object tier.

They are passed to a generic data session manager which is not aware of any of the business objects; the

only requirement is that the business objects passed to it for CRUD have to have a mapping file.

Share Improve this answer

Follow

answered Aug 14, 2008 at 10:30

Jon Limjap
95.3k 15 103 153