What is the difference between DAO and Repository patterns?

Asked 8 years, 1 month ago Active 2 months ago Viewed 144k times

402

What is the difference between Data Access Objects (DAO) and Repository patterns? I am developing an application using Enterprise Java Beans (EJB3), Hibernate ORM as infrastructure, and Domain-Driven Design (DDD) and Test-Driven Development (TDD) as design techniques.

hibernate domain-driven-design ejb-3.0 repository-pattern data-access-layer

205

edited Feb 23 '15 at 14:26
TylerH
17.3k 10 57 74

asked Dec 18 '11 at 6:05
Thurein
5,027 5 19 23

10 Answers

443

`DAO` is an abstraction of **data persistence**.
`Repository` is an abstraction of **a collection of objects**.

`DAO` would be considered closer to the database, often table-centric.
`Repository` would be considered closer to the Domain, dealing only in Aggregate Roots.

`Repository` could be implemented using `DAO` 's, but you wouldn't do the opposite.

Also, a `Repository` is generally a narrower interface. It should be simply a collection of objects, with a `Get(id)` , `Find(ISpecification)` , `Add(Entity)` .

A method like `Update` is appropriate on a `DAO` , but not a `Repository` - when using a `Repository` , changes to entities would usually be tracked by separate UnitOfWork.

It does seem common to see implementations called a `Repository` that is really more of a `DAO` , and hence I think there is some confusion about the difference between them.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

 **8,206** 8 35 74

 **22.5k** 6 62 83

23 Well, you wouldn't want your DAO class to literally implement your `IRepository` interface. You would want your repository to use DAO's in its implementation. Remember, a DAO will be a per-table object, whereas a Repository will almost always have to use multiple DAO's to build a single Entity. If you find that is not the case, that your Repository and Entity only need to access a single table, then you are most likely building an anemic domain. – quentin-starin Dec 18 '11 at 7:04

28 I've noticed in the .NET world specifically the term "Repository" is used to refer to what is essentially a DAO; "DAO" is more of a Java term. – Wayne Molina Apr 25 '12 at 12:29

14 @Thurein DAO-s are not per table, the pattern is only abstracting the access to your data - you can implement that however you like (per table, or per group or models). The recommended way is to always shape your DAOs based on your domain model rather than taking underlying persistence into account tough as that makes it easier/clearer to use and gives you a bit more flexibility on how you persist it (e.g. imagine you'll need a DAO that stores your data in XML files or gets it from a message queue rather than from Database ...). – Stef Mar 20 '13 at 0:15

18 @Stef I disagree. A DAO returns *data* by its very definition (a *data* access object). A repository, by its definition, returns domain objects. It should stand to reason that the repository would use DAOs and not the other way around, because in OOP we compose domain objects out of one or more data objects, and not the other way around. – Mihai Danila Apr 19 '13 at 16:21

5 Why is a Repository a "Read Only" concept while DAO is "Read and Write"? – Dennis Apr 1 '16 at 19:30

113

OK, think I can explain better what I've put in comments :). So, basically, you can see both those as the same, though DAO is a more flexible pattern than Repository. If you want to use both, you would use the Repository in your DAO-s. I'll explain each of them below:

REPOSITORY:

It's a repository of a specific type of objects - it allows you to search for a specific type of objects as well as store them. Usually it will ONLY handle one type of objects. E.g. `AppleRepository` would allow you to do `AppleRepository.findAll(criteria)` or `AppleRepository.save(juicyApple)` . Note that the Repository is using Domain Model terms (not DB terms - nothing related to how data is persisted anywhere).

A repository will most likely store all data in the same table, whereas the pattern doesn't require that. The fact that it only handles one

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

DAO - data access object (in other words - object used to access data)

A DAO is a class that locates data for you (it is mostly a finder, but it's commonly used to also store the data). The pattern doesn't restrict you to store data of the same type, thus you can easily have a DAO that locates/stores related objects.

E.g. you can easily have UserDao that exposes methods like

```
Collection<Permission> findPermissionsForUser(String userId)
User findUser(String userId)
Collection<User> findUsersForPermission(Permission permission)
```

All those are related to User (and security) and can be specified under then same DAO. This is not the case for Repository.

Finally

Note that both patterns really mean the same (they store data and they abstract the access to it and they are both expressed closer to the domain model and hardly contain any DB reference), but the way they are used can be slightly different, DAO being a bit more flexible/generic, while Repository is a bit more specific and restrictive to a type only.

answered Mar 20 '13 at 0:48

Stef
2,045 2 12 25

If I get this right e.g. I have something like `CarDescription` that has e.g. `language_id` as foreign key - then to retrieve that I should do something like this: `CarRepository.getAll(new Criteria(carOwner.id, language.id));` which would give me all cars of a language in a specific language - is that the right way to do it? – displayname Jul 7 '15 at 14:10

@StefanFalk, have a look at Spring Data, it allows you to do much nicer calls than that. e.g. that could be written like `CarRepository.findByLanguageId(language.id)` and you wouldn't even need to write the code, you just define the interface with a method with that name and Spring Data takes care of building the default class implementation for you. Pretty neat stuff ;) – Stef Jul 8 '15 at 15:18

2 The beauty of Spring Data is that you don't actually have to write the queries, you just create an interface (like that TodoRepository in your example, which has the method `findById`). And you are practically done. What then Spring Data does is, it finds all these interfaces you've created which extend the Repository interface and creates the classes for you. You'll never see those classes and you won't be able to create new instances, but you don't need to as you can just autowire the interface and let Spring locate that repository object. – Stef Jul 9 '15 at 10:38

1 Finally, you don't have to use Spring Data, you can go the old way of writing the query methods yourself (using Criteria API etc), but you'd just make your life a bit more complex ... You might say that you would have more flexibility like that, but that's not true either as if you really want to go crazy with your queries, Spring Data allows you two ways of doing so: the @Query annotation, or if that doesn't work, you can create custom repositories which are an

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Christian Stempfer May 11 '17 at 17:52

82

DAO and Repository pattern are ways of implementing Data Access Layer (DAL). So, let's start with DAL, first.

Object-oriented applications that access a database, must have some logic to handle database access. In order to keep the code clean and modular, it is recommended that database access logic should be isolated into a separate module. In layered architecture, this module is DAL.

So far, we haven't talked about any particular implementation: only a general principle that putting database access logic in a separate module.

Now, how we can implement this principle? Well, one know way of implementing this, in particular with frameworks like Hibernate, is the DAO pattern.

DAO pattern is a way of generating DAL, where typically, each domain entity has its own DAO. For example, `User` and `UserDao`, `Appointment` and `AppointmentDao`, etc. An example of DAO with Hibernate: http://gochev.blogspot.ca/2009/08/hibernate-generic-dao.html.

Then what is Repository pattern? Like DAO, Repository pattern is also a way achieving DAL. The main point in Repository pattern is that, from the client/user perspective, it should look or behave as a collection. What is meant by behaving like a collection is not that it has to be instantiated like `Collection collection = new SomeCollection()`. Instead, it means that it should support operations such as add, remove, contains, etc. This is the essence of Repository pattern.

In practice, for example in the case of using Hibernate, Repository pattern is realized with DAO. That is an instance of DAL can be both at the same an instance of DAO pattern and Repository pattern.

Repository pattern is not necessarily something that one builds on top of DAO (as some may suggest). If DAOs are designed with an interface that supports the above-mentioned operations, then it is an instance of Repository pattern. Think about it, If DAOs already provide a collection-like set of operations, then what is the need for an extra layer on top of it?

edited Oct 21 '13 at 7:54

sajjadG
1,212 1 16 24

answered Feb 19 '13 at 21:38

Nazar Merza
2,687 14 19

2 "If DAOs already provide a collection-like set of operations, then what is the need for an extra layer on top of it?" Suppose you're modeling a pet shop and you have a table 'PetType' with different animals and their attributes (name: "Cat", type: "Mammal", etc.) referenced by a table 'Pet' of the concrete pets you have in the shop (name: "Katniss", breed: "Calico", etc.). If you want to add an animal of a type not already in the database, you could use a

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

68

Frankly, this looks like a semantic distinction, not a technical distinction. The phrase Data Access Object doesn't refer to a "database" at all. And, although you could design it to be database-centric, I think most people would consider doing so a design flaw.

The purpose of the DAO is to hide the implementation details of the data access mechanism. How is the Repository pattern different? As far as I can tell, it isn't. Saying a Repository is *different* to a DAO because you're dealing with/return a collection of objects can't be right; DAOs can also return collections of objects.

Everything I've read about the repository pattern seems rely on this distinction: bad DAO design vs good DAO design (aka repository design pattern).

edited Oct 13 '15 at 11:13

Matthew
7,053 8 50 75

answered Jul 8 '12 at 17:07

rakehell404
713 5 2

4 yep, totally agree, they are essentially the same. DAO sounds more DB related, but it's not. Same as Repository, it's just an abstraction used to hide where and how data is located. – Stef Mar 20 '13 at 0:03

+1 For this statement. Frankly, this looks like a semantic distinction, not a technical distinction. The phrase Data Access Object doesn't refer to a "database" at all. – Sudhakar Chavali Oct 14 '19 at 22:17

17

Repository is more abstract domain oriented term that is part of Domain Driven Design, it is part of your domain design and a common language, DAO is a technical abstraction for data access technology, repository is concerns only with managing existing data and factories for creation of data.

check these links:

http://warren.mayocchi.com/2006/07/27/repository-or-dao/ http://fabiomaulo.blogspot.com/2009/09/repository-or-dao-repository.html

edited Oct 24 '12 at 16:03

robyaw
1,964 1 16 21

answered Dec 18 '11 at 22:34

Mohamed Abed
4,475 16 29

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

6

entities. Therefore, it's common that a repository delegates the actual persistence of the aggregate roots to a DAO. Additionally, as the aggregate root must handle the access of the other entities, then it may need to delegate this access to other DAOs.

answered Jul 18 '12 at 17:06

pablochacin
691 8 23

3

Repository are nothing but well-designed DAO.

ORM are table centric but not DAO.

There's no need to use several DAO in repository since DAO itself can do exactly the same with ORM repositories/entities or any DAL provider, no matter where and how a car is persisted 1 table, 2 tables, n tables, half a table, a web service, a table and a web service etc. Services uses several DAO/repositories.

My own DAO, let's say CarDao only deal with Car DTO,I mean, only take Car DTO in input and only return car DTO or car DTO collections in output.

So just like Repository, DAO actually is an IoC, for the business logic, allowing persitence interfaces not be be intimidated by persitence strategies or legacies. DAO both encapsulates the persistence strategy and does provide the domaine-related persitence interface. Repository is just an another word for those who had not understood what a well-defined DAO actualy was.

answered Aug 12 '13 at 15:07

Cyril
59 2

First of all "ORM repositories/entities"? You mean ORM entities. There no such thing as a repository of ORMs. Second of all ORMs usually deal with entities only, ie. domain models. DAOs deal with tables directly and abstract data access. They return entities as well. Repositories are the highest abstraction, offering a collection interface to getting entities. A DAO can be a repository, ie. abstracting the actual storage engine, offering an interface to it and also offering a collection view of (cache) entities. A DAO can use an ORM to interface with the database and delegate entity ops. – Paul-Sebastian Manole Oct 17 '18 at 8:33

1 Agree with @brokenthorn. The most crucial point in his comment is "Repositories are the highest abstraction," and this abstraction becomes a necessity when you want to protect your domain code from underlying database technology. ORM/Adapter/DB Driver concepts tend to leak into DAOs. If you have an application that supports more than one database technology, or if you want your app not to be locked to a database, using DAOs directly from the domain model is a no-go. – Subhash Bhushan May 24 '19 at 23:34

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

3

DAO provides abstraction on database/data files or any other persistence mechanism so that, persistence layer could be manipulated without knowing its implementation details.

Whereas in Repository classes, multiple DAO classes can be used inside a single Repository method to get an operation done from "app perspective". So, instead of using multiple DAO at Domain layer, use repository to get it done. Repository is a layer which may contain some **application logic** like: If data is available in in-memory cache then fetch it from cache otherwise, fetch data from network and store it in in-memory cache for next time retrieval.

edited Nov 16 '19 at 7:01

answered Mar 14 '19 at 4:15

Rahul Rastogi
2,455 2 17 37

2

Try to find out if DAO or the Repository pattern is most applicable to the following situation : Imagine you would like to provide a uniform data access API for a persistent mechanism to various types of data sources such as RDBMS, LDAP, OODB, XML repositories and flat files.

Also refer to the following links as well, if interested:

http://www.codeinsanity.com/2008/08/repository-pattern.html

http://blog.fedecarg.com/2009/03/15/domain-driven-design-the-repository/

http://devlicio.us/blogs/casey/archive/2009/02/20/ddd-the-repository-pattern.aspx

http://en.wikipedia.org/wiki/Domain-driven_design

http://msdn.microsoft.com/en-us/magazine/dd419654.aspx

answered Dec 19 '11 at 6:09

javaDisciple
184 12

---

in a very simple sentence: The significant difference being that Repositories represent collections, whilst DAOs are closer to the database, often being far more table-centric.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

936 2 14 21

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.