What is Data Transfer Object?

Asked 10 years, 9 months ago Active 21 days ago Viewed 154k times

199

What is a Data Transfer Object?

In MVC are the model classes DTO, and if not what are the differences and do we need both?

model-view-controller architecture dto data-transfer data-transfer-objects

81

edited Feb 16 '16 at 11:21
Soner Gönül
86.8k 29 168 306

asked Jun 26 '09 at 20:40
Yaron Naveh
20.3k 27 91 149

@yegor256 and the fact, that book in that article knows how to retrieve data from API and also how to store data into DB, and so violating SRP, is ok? – Betlista Jan 9 '19 at 12:25

7 Answers

Active Oldest Votes

205

✓

A Data Transfer Object is an object that is used to encapsulate data, and send it from one subsystem of an application to another.

DTOs are most commonly used by the Services layer in an N-Tier application to transfer data between itself and the UI layer. The main benefit here is that it reduces the amount of data that needs to be sent across the wire in distributed applications. They also make great models in the MVC pattern.

Another use for DTOs can be to encapsulate parameters for method calls. This can be useful if a method takes more than 4 or 5 parameters.

When using the DTO pattern, you would also make use of DTO assemblers. The assemblers are used to create DTOs from Domain Objects, and vice versa.

The conversion from Domain Object to DTO and back again can be a costly process. If you're not creating a distributed application, answered Jun 29 '09 at 13:09

Benny Hallett
5,221 4 22 26

7 "DTO make great models in the MVC pattern" - but shouldn't a model contain all data of the object and DTO be optimized with part of the data? If I have model A and I need to pass it to two subsystems will there be A_DTO_1 and A_DTO_2 with the relevant fields of each? "DTOs can be to encapsulate parameters for method calls" --> So every class that wraps parameters is DTO even if this is not distributed system? Are't models in MVC the domain object? – Yaron Naveh Jun 30 '09 at 21:10

2 In answer to your first question, I don't think were talking about the same thing. The model in MVC doesn't necessarily need to be a class from your Domain Model. Having said that, it well could be. Using the DTO strips out all of the unnecessary stuff. Just depends on the architecture you're going for. I'm not sure exactly how to answer your second question. Whether its across the wire or not, it's still an object that encapsualtes a bunch of data to be transferred between (sub)systems, so I'd argue it's a DTO. – Benny Hallett Jul 6 '09 at 7:49

11 "Another use for DTOs can be to encapsulate parameters for method calls. This can be useful if a method takes more than 4 or 5 parameters." This is actually an anti-pattern called a Poltergeist or Gypsy Wagon class. If your method needs 4 arguments then give it 4, don't create a class just to move an object into method or a class. – Wix Dec 10 '10 at 15:11

2 @Wix, good point. I'd argue however that this is ok if it's semantically correct (say if you pass a settings class with properties rather than the properties themselves as values). What you shouldn't do is throw in all the arguments for the sake of passing a single object, since they may very well be unrelated and cause nightmares untangling later on. – Aram Kocharyan Sep 21 '12 at 3:03

3 DTOs should not be used to encapsulate parameters for methods calls (which would make them LocalDTOs), they were introduced in the context of remote interfaces: martinfowler.com/bliki/LocalDTO.html – Rui Oct 1 '13 at 15:29

27

The definition for DTO can be found on Martin Fowler's site. DTOs are used to transfer parameters to methods and as return types. A lot of people use those in the UI, but others inflate domain objects from them.

answered Jun 26 '09 at 20:43

blu
11.3k 15 63 102

21

A DTO is a dumb object - it just holds properties and has getters and setters, but no other logic of any significance (other than maybe a compare() or equals() implementation).

Typically model classes in MVC (assuming .net MVC here) are DTOs, or collections/aggregates of DTOs

Eric Petroelje
55.3k 8 113 170

2 What you are describing is a LocalDTO: martinfowler.com/bliki/LocalDTO.html – Rui Oct 1 '13 at 15:30

2 One case where it is useful to use something like a DTO is when you have a significant mismatch between the model in your presentation layer and the underlying domain model. In this case it makes sense to make presentation specific facade/gateway that maps from the domain model and presents an interface that's convenient for the presentation. – Amitābha Aug 14 '15 at 3:21

11

In general **Value Objects** should be Immutable. Like *Integer* or *String* objects in Java. We can use them for transferring data between software layers. If the software layers or services running in different remote nodes like in a microservices environment or in a legacy Java Enterprise App. We must make almost exact copies of two classes. This is the where we met DTOs.

```
|-----------|                                                   |--------------|
| SERVICE 1 |--> Credentials DTO >--------> Credentials DTO >-- | AUTH SERVICE |
|-----------|                                                   |--------------|
```

In legacy Java Enterprise Systems DTOs can have various EJB stuff in it.

I do not know this is a best practice or not but I personally use **Value Objects** in my Spring MVC/Boot Projects like this:

```
        |------------|         |------------------|                             |-------
-----|
-> Form |            | -> Form |                  | -> Entity                   |
|
        | Controller |         | Service / Facade |                             |
Repository |
<- View |            | <- View |                  | <- Entity / Projection View |
|
        |------------|         |------------------|                             |-------
-----|
```

Controller layer doesn't know what are the entities are. It communicates with **Form** and **View Value Objects**. Form Objects has JSR 303 Validation annotations (for instance @NotNull) and **View Value Objects** have Jackson Annotations for custom serialization. (for instance @JsonIgnore)

Service layer communicates with repository layer via using Entity Objects. Entity objects have JPA/Hibernate/Spring Data annotations

```
User Service ----> XX CANNOT CALL XX ----> Order Service
```

Some **ORM** Frameworks have the ability of projection via using additional interfaces or classes. So repositories can return View objects directly. There for you do not need an additional transformation.

For instance this is our User entity:

```
@Entity
public final class User {
    private String id;
    private String firstname;
    private String lastname;
    private String phone;
    private String fax;
    private String address;
    // Accessors ...
}
```

But you should return a Paginated list of users that just include id, firstname, lastname. Then you can create a View Value Object for ORM projection.

```
public final class UserListItemView {
    private String id;
    private String firstname;
    private String lastname;
    // Accessors ...
}
```

You can easily get the paginated result from repository layer. Thanks to spring you can also use just interfaces for projections.

```
List<UserListItemView> find(Pageable pageable);
```

Don't worry for other conversion operations `BeanUtils.copy` method works just fine.

edited Aug 19 '17 at 15:17

answered Oct 27 '12 at 18:06
Fırat KÜÇÜK
3,638 1 40 46

10

1. To me the best answer to the question ***what is a DTO*** is that *DTO's are simple objects that should not contain any business logic or methods implementation that would require testing*.
2. Normally your model (using the MVC pattern) are intelligent models, and they can contain a lot of/some methods that do some different operations for that model specifically (not business logic, this should be at the controllers). However, when you transfer data (eg. calling a REST (`GET` / `POST` /whatever) endpoint from somewhere, or consuming a webservice using SOA, etc...) you do not want to transmit the big sized object with code that is not necessary for the endpoint, will consume data, and slow down the transfer.

edited Apr 17 '19 at 10:18
Surjit SD
1,209 1 11 20

answered Mar 18 '15 at 12:52
Thiago Burgos
719 1 12 14

Why should business logic be in controllers? – AlexioVay Jan 30 at 18:57

6

With MVC data transfer objects are often used to map domain models to simpler objects that will ultimately get displayed by the view.

From Wikipedia:

> Data transfer object (DTO), formerly known as value objects or VO, is a design pattern used to transfer data between software application subsystems. DTOs are often used in conjunction with data access objects to retrieve data from a database.

edited Mar 6 '17 at 11:26
Liam
20.2k 16 87 139

answered Jun 26 '09 at 20:41
Dan
16.8k 3 32 37

2 A value object is not a DTO. – Pavan Jan 22 '19 at 21:40

0

Data transfer object (DTO) describes “an object that carries data between processes” (Wikipedia) or an “object that is used to encapsulate data, and send it from one subsystem of an application to another” (Stack Overflow answer).

answered Mar 3 at 17:10