Is it necessary to create an repository and a service for each entity? [duplicate]

Asked 4 years, 9 months ago Active 4 years, 9 months ago Viewed 5k times

6

2

This question already has an answer here:

Are you supposed to have one repository per table in JPA? (1 answer)

Closed 2 years ago.

I'm using Hibernate+Spring and a database to persist my entities. I'm already using JpaRepository to create my repositories but even then, it seems I must create one interface extending JpaRepository for each entity. Worst, I'm creating one service to each entity. All of them very similar.

There's any way to create a generic service and a generic repository? Is it really necessary to implement each one of them?

By the moment, I have repositories like this:

```
@Repository
public interface PhaseRepository extends JpaRepository<Phase, Serializable> {

}
```

and services like this:

```
@Service
public class PhaseService {
    @Autowired
    PhaseRepository repository;

    @Transactional
    public Phase create(Phase entity) {
        return repository.save(entity);
    }

    @Transactional(rollbackFor = EntityNotFound.class)
    public Phase delete(int id) throws EntityNotFound {
        Phase deleted = repository.findOne(id);
        if (deleted == null) {
            throw new EntityNotFound();
```

```
        }
        repository.delete(deleted);
        return deleted;
    }

    @Transactional(rollbackFor = EntityNotFound.class)
    public Phase update(Phase entity) throws EntityNotFound {
        Phase updated = repository.findOne(entity.getId());

        if (updated == null) {
            throw new EntityNotFound();
        }

        repository.saveAndFlush(entity);
        return updated;
    }

    public Phase findById(int id) throws EntityNotFound {
        Phase entity = repository.findOne(id);

        if (entity == null) {
            throw new EntityNotFound();
        }

        return entity;
    }
}
```

I'm using 12 entities and everyone has the same service methods.

Thanks!

java spring hibernate jpa

asked Jun 8 '15 at 20:47

Leandro Lima

881 ● 1 ● 16 ● 38

2 Answers

Active | Oldest | Votes

3

Probably you'll need the 12 repositories. But maybe you won't need 12 services. A service could handle the access to several repositories. It depends on your logic and how "important" are every entity.

For example, if you had the entities User and Address you could have UserRepository and AddressRepository. But only UserService, with methods like addAddress(User user, Address address)...

All in all, I'd recomend you to organize your services accordingly your business logic instead of a bunch of CRUDs

edited Jun 21 '15 at 15:16

answered Jun 8 '15 at 21:28

polypiel

1,894 ● 1 ● 16 ● 19

1

You need 12 repositories, because Spring initializes instances of your declared interfaces. Spring already did a good job, I don't see a problem here. Imagine a situation back the days, when you had to implement these repositories by boilerplate JDBC code.

But you don't automatically need services for your repositories. Create services for your application purposes and autowire necessary repositories inside them. But I have created an example code, if you want generic service without copy-pasting code:

```
public abstract class AbstractService<T extends AbstractEntity, K extends Serializable>
{

    protected JpaRepository<T, K> repository;

    public AbstractService(final JpaRepository<T, K> repository) {
        this.repository = repository;
    }

    @Transactional(rollbackFor = EntityNotFound.class)
    public T delete(final K id) throws EntityNotFound {
        final T deleted = this.repository.findOne(id);
        if (deleted == null) {
            throw new EntityNotFound();
        }
        this.repository.delete(deleted);
        return deleted;
    }

    ...

}
```

Implementing class would be:

```
@Service
public class PhaseService extends AbstractService<PhaseEntity, Integer> {

    @Autowired
    public PhaseService (final PhaseRepository repository) {
        super(repository);
    }

}
```

Also a few tips:

- You don't need EntityNotFound exception. Just check if entity is null.
- Instead of Serializable use primary key strong type - an Integer for example.

answered Jun 21 '15 at 15:52

Gondy
3,566 ● 2 ● 27 ● 39