链滴

领域模型中的充血模型

作者：ykz200
原文链接：https://ld246.com/article/1620710437231
来源网站：链滴

本文来源领域模型中的充血模型

本篇笑点

~~我要从甲方跳到乙方公司子。~~

被我领导知道了

你得知道这篇讲的些什么

本文1229字，阅读可能需要2分23秒

与贫血模型不一样的充血模型

说明

继续上一篇 回头看领域模型中的贫血模型继续来看一下不一样的充血模型。

充血模型

充血模型和**贫血模型**差不多，所不同的就是如何划分业务逻辑，即认为，绝大多业务逻辑都应该被放domain object里面(包括持久化逻辑)，而Service层应该是很薄的一层，仅仅封装事务和少量逻辑，和DAO层打交道。

在贫血模型中，数据和业务逻辑被分割到不同的类中。充血模型（Rich Domain Model）正好相反数据和对应的业务逻辑被封装到同一个类中。因此，这种充血模型满足面向对象的封装特性，是典型面向对象编程风格。

包结构

充血模型的实现一般包括如下包：

包	内容
model	包含了实体类信息，还包含原子服务和数持久化的逻辑
service	放置所有的服务类 组合服务 也叫事务服务

代码实现

先看model包的两个类，Account和TransferTransaction对象，分别代表帐户和一次转账事务。由它们不包含业务逻辑，就是一个普通的Java Bean，下面的代码省略了get和set方法。

```
/**
 * 账户
 **/
public class Account {
    private String accountId;
    private BigDecimal balance;

    public Account() {}
    public Account(String accountId, BigDecimal balance) {
        this.accountId = accountId;
        this.balance = balance;
    }
    // getter and setter ....

}

/**
 * 转账
 **/
public class TransferTransaction {
    private Date timestamp;
    private String fromAccountId;
    private String toAccountId;
    private BigDecimal amount;

    public TransferTransaction() {}

    public TransferTransaction(String fromAccountId, String toAccountId, BigDecimal amount,
ate timestamp) {
        this.fromAccountId = fromAccountId;
        this.toAccountId = toAccountId;
        this.amount = amount;
        this.timestamp = timestamp;
    }

    // getter and setter ....

/**
  * 转账逻辑
  * @param fromAccountId 转账人
```

```
     * @param toAccountId 收账人
     * @param amount 金额
     * @return
     * @throws AccountNotExistedException
     * @throws AccountUnderflowException
     */
    public TransferTransaction transfer(String fromAccountId, String toAccountId,
     BigDecimal amount) throws AccountNotExistedException, AccountUnderflowException {
       Validate.isTrue(0 < amount.compareTo(BigDecimal.ZERO));
         //业务逻辑略...

         // 调用DAO进行显式持久化
         return new TransferTransactionDAO().create(fromAccountId, toAccountId, amount);
    }
}
```

发现没有 在这种模型中，所有的业务逻辑全部都在TransferTransaction 中，事务管理也在TransferTansaction 中实现。

这种模型的优点：

1、更加符合OO的原则

2、Service层很薄，只充当Facade的角色，不和DAO打交道。

缺点也很明显

1、当model中包含了数据持久化的逻辑，实例化的时候可能会有很大麻烦，拿到了太多不一定需要关联model。

总结

所谓充血模型的思想 无非是要就是要大家不要盲目写代码不要盲目的一撸到底 要从全局设计出发 设清楚了再开工

附言

本篇如有错误，请及时指出，马上修改。

非常非常重要的事情

本文首发于【黑壳博客】，文章持续更新，可以微信搜索【黑壳博客】点个关注 文章第一时间阅读。