业务场景中相似的代码太多？试试看模版方法+泛型+多态，一份代码适配n个相似业务流

==========

很多业务比如工单系统，都会遇到一个这样的业务场景，对于不同类型的工单，他们的查询思路大同小异，如果单独为每一个账单类型写一份代码，会产生至少10份流程非常相似的代码，从代码维护和简洁性上看明显存在不小的问题；而且有让开发工作成为劳动密集型的嫌疑，不符合追求的自动化/精简工作量的原则。

其实仔细梳理一下业务流程，业务的执行过程可以抽象为一定的算法骨架：

1. 构造通用查询条件
2. 构造特定的查询条件
3. 查询数据库，包括历史库和实时库
4. 去重
5. 排序

对于算法可以抽象的流程，一般可以采用设计模式中的**模版方法**来解这个难题：**模板方法模式是一种行为设计模式**，它在超类中定义了一个算法的框架，允许子类在不修改结构的情况下重写算法的特定步骤。在这里，「构造特定的查询条件」这一步就是暴露给各具体账单子类进实现的。详细学习模版方法可以参考 [模版方法](http://cxyroad.com/”https://refactoringguru.cn/design-patterns/template-method”)

而且通过观察发现，比如第3步查询数据库，不同的类型的工单使用的查询数据库，无非就是参数的不同罢了。对于参数不同又希望函数能适用的场景，一般是通过**泛型**来解决。

关于泛型，在Java和TS等现代语言中都有很好的支持和应用，以Java为例子，简单和大家分享一下模版方法+泛型的使用：

基类中：

```

public abstract class TicketRepositoryBase<I extends
```

```
TicketDOCriteriaBase, T extends QueryTicketOptionBaseEntity> {

// 模版方法
protected final List<?> queryTicketTemplate(I criteria, T queryOption){
    1. 构造通用查询条件(本类实现)
    2. 构造特定的查询条件(子类实现)
    3. 查询数据库，包括历史库和实时库(本类实现)
    4. 去重(本类实现)
    5. 排序(本类实现)
}

// 2. 构造特定的查询条件(子类实现)
abstract void buildQueryCriterion(I criteria, T queryOption);

}
```

TicketDOCriteriaBase和QueryTicketOptionBaseEntity分别是查询构造条件和查询参数实体的基类，可以放一些通用的函数。下面的RepairTicketDOCriteria, QueryRepairTicketOptionEntity都需要分别继承他们，实现多态。

```
```

某具体类型工单中（以维修工单举例）：

```
```
// <RepairTicketDOCriteria, QueryRepairTicketOptionEntity> 可以理解为是把具体的类型当作参数传进去基类里，这样基类的I和T就有了具体的类型。

public class RepairTicketRepositoryImpl extends
        TicketRepositoryBase<RepairTicketDOCriteria,
QueryRepairTicketOptionEntity> {

 // 只需要实现的函数
        void buildQueryCriterion(RepairTicketDOCriteria criteria,
QueryRepairTicketOptionEntity queryOption){
        // 构造具体类型工单需要支持的查询条件
        }

  }


```
```

1. 构造通用查询条件(本类实现)

2. 构造特定的查询条件(子类实现)
3. 查询数据库，包括历史库和实时库(本类实现)
4. 去重(本类实现)
5. 排序(本类实现)

可以看到，在这5点里，只有第二点的代码变化较多，需要延迟到子类实现。其他的相似的代码都可以通过泛型，在父类中通过子类指定具体的类型参数，以泛型+多态的方式插入模版方法参数里。

原文链接: https://juejin.cn/post/7363453558031794212