# 我亲手写出了难以维护的代码

2019-12-15

前段时间尝试了一些线下活动，加上有几天身体异常痛苦，现在稍微静下来感觉恍如隔世。最近有个项目在做最后的验收，于是搬出了一两个月前的代码，需要手动操作更新一些数据。看到这些不久之前写的代码，同样恍如隔世的感觉，如果这是别人写的，我一定会第一时间认为很垃圾，既然是我亲手写的，客观的说，它确实很垃圾。

那是一个很好记的日子，从那天开始，2019年才真正开始。我曾经接触的是老系统的代码，不得不在resin里启动项目，编译一次至少要两分钟，难以调试，曾经在测试服务器上开发和调试程序，而不是发布代码，曾经写过一些小任务，比如批量执行sql还踩了坑。我曾经看不起垃圾代码，看不起9102年还在用eclipse的程序员。

时至今日，我也变成了写出垃圾代码的程序员。

## 小需求

第一次写出垃圾代码，印象比较深的是打印两次日志：

```
catch (e) {
    System.out.println("捕获到异常");
    logger.error("捕获到异常");
}
```

logger出现了多少次，println就出现了多少次，这显然是丑陋的做法。因为当时的nohup.out和log文件夹不在同一目录，log文件夹专门配置过放在统一的目录，然后希望nohup.out也有日志输出……具体原因有点模糊了，这应该是第一次恶心到我自己的代码。后来好像再版的时候去掉了，恍恍惚惚。

同样是这个需求中，当时的老大的代码里，有一处值得借鉴和赞赏的写法，我当时怀着慷慨激昂的心情认为这是冗余的做法，有点想要挑战权威的意味，事情证明还是年轻了。

代码想不起来了 `T_T`

同样还是这个需求，需要实现可以自定义配置sql语句，类似jdbcTemplate中的`?`，或者mybatis的`#{}`，我当时用了非常生硬的做法：

```
// 需求
sql = update set a={a}, b={b}, ...
// 解决
for (i = 0; i < param.num; i++) {
    switch(match.group()) {
        case {a}: break;
        case {b}: break;
        ...
    }
}
```

当被问到参数位置是否可以调换，还是放心的，那参数能多能少吗？……

## 改需求

曾经见过一个程序，一个文件里面有一个类，这是理所当然的，这个类里面只有一个方法，方法有一千多行，是个女程序员写的。

我的任务是原先输出的报表上加几个字段，原先输出了`a,b`，现在需要`a,b,c,d`。`c,d`是我需要实现的部分，大致的程序逻辑和`a,b`一样，但是来源不同，执行不同的sql查出来的。按理说改个sql不就完了吗，问题是一千多行的方法里面嵌套了三四层循环，包含了从不同银行查数据的业务操

作，循环内外的变量名就是`list1`、`list2`，最终的结果还会涉及到对`a,b`的累加操作。

当时选择的做法是不动原来的程序，复制一份出来，分别执行完后整合一下数据。心高气傲啊，不愿意读垃圾代码。所以后来一个类里面有两个一千行代码的方法了。要命的是bug不断，复制出来的代码仍然需要从头到尾看明白，而且对两部分数据的整合也带来了不少麻烦和意想不到的漏洞……

## 完善需求

另一个需求是接手别人做到一半的项目。9102年了在spring boot里面用模板引擎写页面，我不得不用jquery和基于jquery的移动端样式库完成开发，界面丑先不说了，文档！文档前后对不上！也怪我，小看了需求，应该不明白就问，问清楚了各个环节再动手操作。结果自然好不了，也是反反复复的改……可能不会有人愿意再维护那份代码。

## 不断变更的需求

无法在一开始敲定需求，一定会产生烂代码。在理想情况下做好规划、有充裕的时间和高素质的开发人员完成工作，各种设计、模式自然是好的。现实一定不是那样。

vue的前端项目里有一种惯用的写法，是对axios的封装，在一个单独的api.js里写各种export，然后在模块中import。据说是为了统一管理api的url还是什么，总之在实际的开发过程中，发现这实在是繁琐的写法，带来了很多开发上的不便。维护更容易了吗？并没有。原因之一就是需求在不断变化，如果api里的注释不够清晰也不够准确，那单独抽出来毫无意义。

```
// api.js
export a = () => { get('a') }

// model.js
import {a} from api.js

a().then(res => {})
```

如果api中的a函数和路由a不完全对应，需要修改api.js时就不得不到对应的页面中去找是哪个函数，这和直接把路由写在页面中没有区别。或许可以试着把路由访问的函数在每个模块或者每个页面顶部封装一下，让模块或者页面里的api是自治的。

前端项目里比起api，更让人难过的是随处可见的css！更更让人难过的是项目已经成型了！同样是背景色的配置，index里写一遍，common里写一遍，model里写一遍，甚至内联的写法再来一遍……问题是各个地方的配置内容还不一样。背景色这种样式还好，可怕的是文档流，定位，浮动，边距……css真正的内功是文档流吧，假如完全不懂还撑起了整个项目，这样的代码谁敢动?

## 懒得重写的需求

随着需求的不断变更，会出现的另一个现象是以前的代码能复用就复用，以最快的速度完成现在的需求，反正不知道以后需求会不会变，要是每次都重构出整洁的代码，工作量和工作效率……这绝对是产生烂代码的途径之一。

后端开发中有个我不太喜欢的东西，bean，也叫entity，不知道是从哪个年代流传下来的做法，包括模板化的dao和service，写个IClass再写个ClassImpl有多大意思呢。bean不利于需求变更，有时候用map反而好一点。回到一开始提到要验收的项目，我写出的烂代码和bean不无关系，大概就是bean的设计包含有很多个字段，后来需求变了，字段也变了，如果要改就要改很多地方，天知道以后需求还会不会变，就强行复用了，导致有些字段含义不明，有些字段完全对不上，仅仅是占用了位置来储存数据。

当然，框架的选择失误也是产生垃圾代码的原因，这要归因于我经验不足，不得不承认架构师在项目管理中起到的核心作用。

## 开发中的困境

一个是起名字的难题，偶尔写错了接口名称，或者复制粘贴相似的逻辑，懒得改变量名，产出的代码一定难以维护，这就可见code review真是企业开发中不可缺少的一步，不过中小公司可能对代码质量的要求也没那么高，除非是要上线的、会涉及到线上业务的代码，review是必须的，搞不好会出人命的，普通的开发可能就没那么严格。

另一个是模块划分的问题，比如a页面有文件上传，b页面有文件下载，那么文件上传下载是不是应该放在同一api路径？都是文件操作，好像挺对的。可如果其他api都是按照页面划分，同一个页面的所有接口都在同一路径下，那单独把文件操作拎出来是不是有点突兀？有时也会按照数据表划分模块，一遇到联表的操作就不好约定了。各种规则混着用多半会引起混乱，让代码难堪。

？

年终之际，还是要给我的2019画上一个圆满的问号。