《软件体系结构与设计》

第三章

作业3

班　　级：111171

学　　号：20171000970

学 生 姓 名：董安宁

指 导 教 师：尚建嘎

中国地质大学地理与信息工程学院软件工程系

2019 年 10 月

第 3 章 软件体系结构风格

作业 3

1. 参考以下链接给出的源码

http://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET

使用 Java 语言进行实现，并结合该例子分析使用层次架构的优点和缺点。

三层架构的架构图如图 1 所示

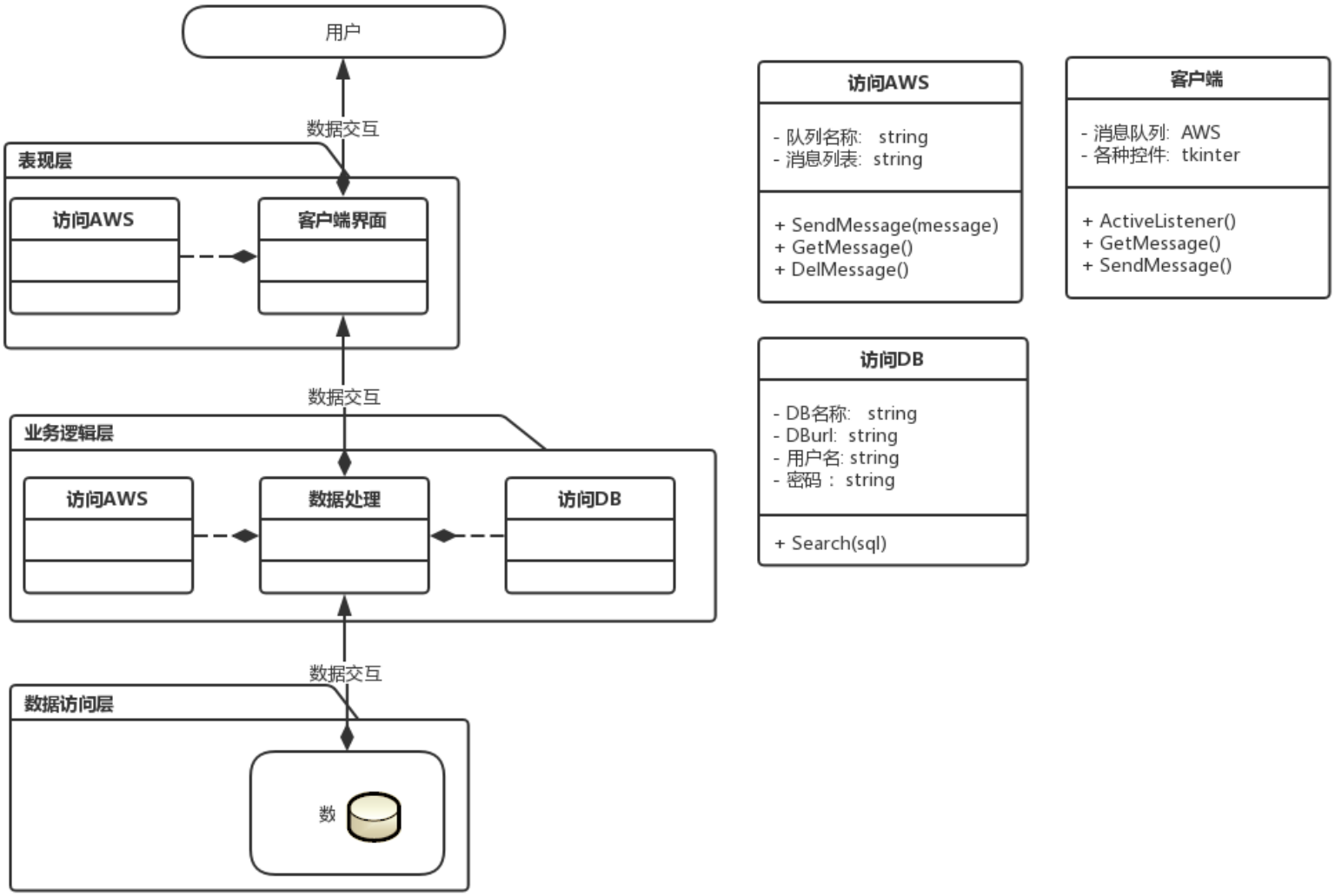

图 1 架构图

在本例中，我分别写了三个应用程序，每个程序之间都是通过 AWS 来进行消息传递的，他们都有自己对外传递和接收参数的接口，这种方式减少了各个层之间的**依赖**，每一层只是通过接口相互链接，可以在相同功能的构件之间替换使用，提高了各层的**复用性**。

然而缺点也是有的，由于不同应用层之间需要信息传递，这种需要额外添加的数据传递导致了**效率的下降**，以及在修改代码的中会出现级**联修改问题**，添加额外的工作量。

最终程序的结果如图 2 所示，左侧没有界面的的便是业务处理层和数据访问层了。

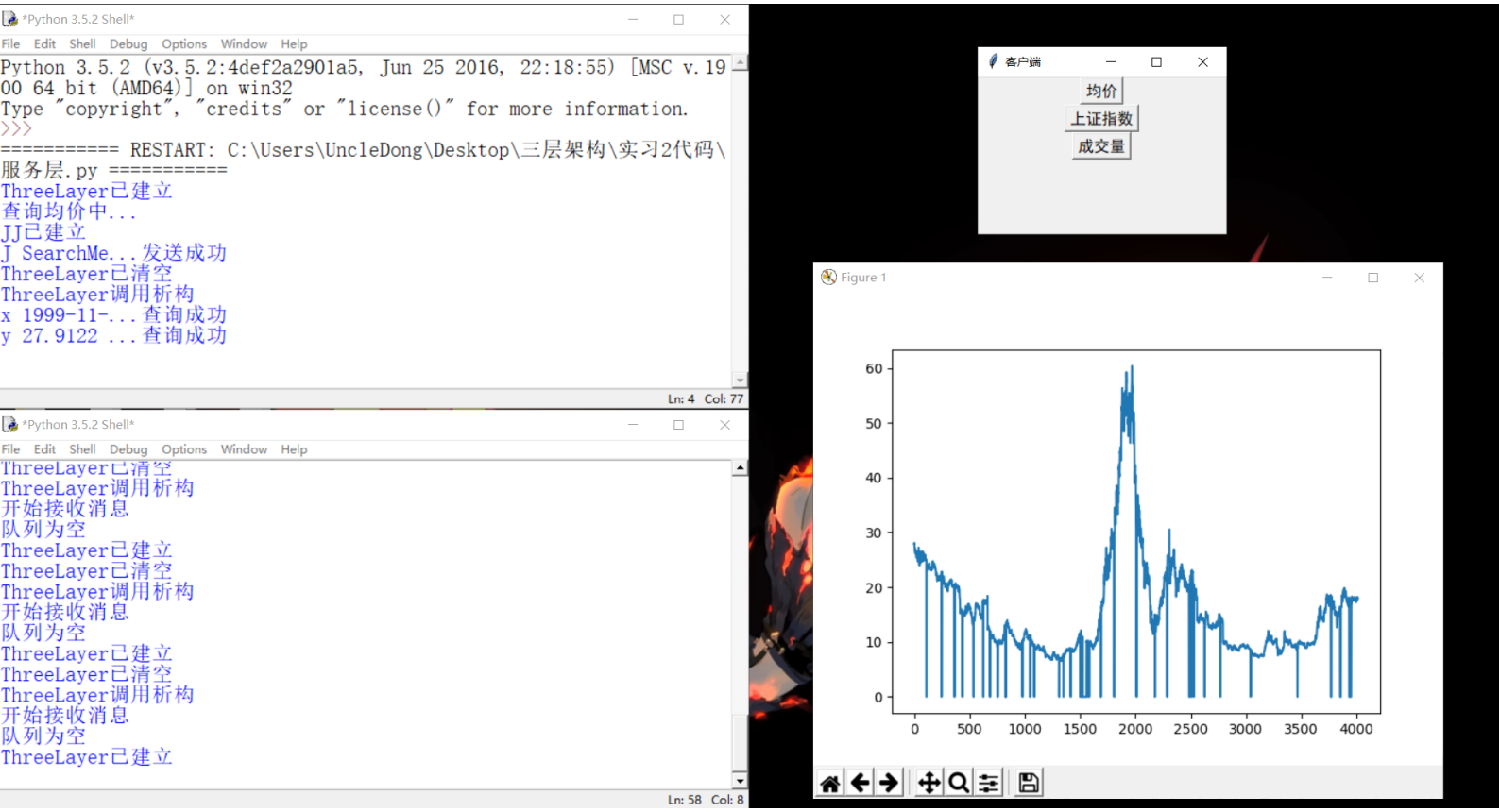

图 2 三层架构结果

2. 针对解释器的三种策略：

- 传统解释器(traditionally interpreted)
- 基于字节码的解释器 (compiled to bytecode which is then interpreted)
- Just-in-Time (JIT)编译器

说明各自的工作原理，对比分析各自的优缺点。

传统解释器：

解释器直接读取源代码并加以执行。

是最原始的代码解释方式，不会出错，缺点是在不同平台都需要重新编译运行。

字节码解释器：

不是直接将源程序交给解释器给解释器解释，而是先经过一个编译过程，把源程序翻译成一种特定的高度压缩和优化的字节码，但并不是真正的机器目标代码，再把这个字节码文件交给解释器来解释执行。

优点是转换成字节码后可以在不同平台运行，与硬件平台无关。缺点是过程多了一步，更加繁琐。

实时编译：

模糊了解释器、字节码解释器和编译器之间的边界与区分，只有在执行某段函数的时候，该段代码才会被编译执行。JIT 并不是编译全部的代码，而是只编译那些被频繁执行的代码段。

优化了处理过程，减少了编译的工作量，不过还是很繁琐。

基于规则引擎 Drools 开发一个简单的专家系统：

（1）根据百分制成绩进行评级(A-F)，转化规则如下：

A (90-100)

B+ (87-89)

B (80-86)

C+ (77-79)

C (70-76)

D+ (67-69)

D (60-66)

F (0-59)

首先参考教程安装了 Drools 系统[1]，并学习了相关语法，了解了基本的运作原理。编程思路简述如下：

代码主程序配置：

① 新建 Score 的类用来配置规则文件，如需对其内容进行规则判断只需要将该对象加载入执行引擎即可。

② 设置界面，为用户提供输入输出接口。

③ 建立规则引擎的一系列初始化，首先通过请求获取 KieServices，通过 KieServices 获取 KieContainer，KieContainer 加载规则文件并获取 KieSession，KieSession 来执行规则引擎。

规则文件配置：

drools 有专门的规则语法 drl，就是专门描述活动的规则是如何执行的，即描述指定的数据在符合什么条件的时候，去做什么事情。当我们想要修改规则的时候，我们只需要修改规则文件，然后重新加载即可生效。

最终结果如图 3 所示：

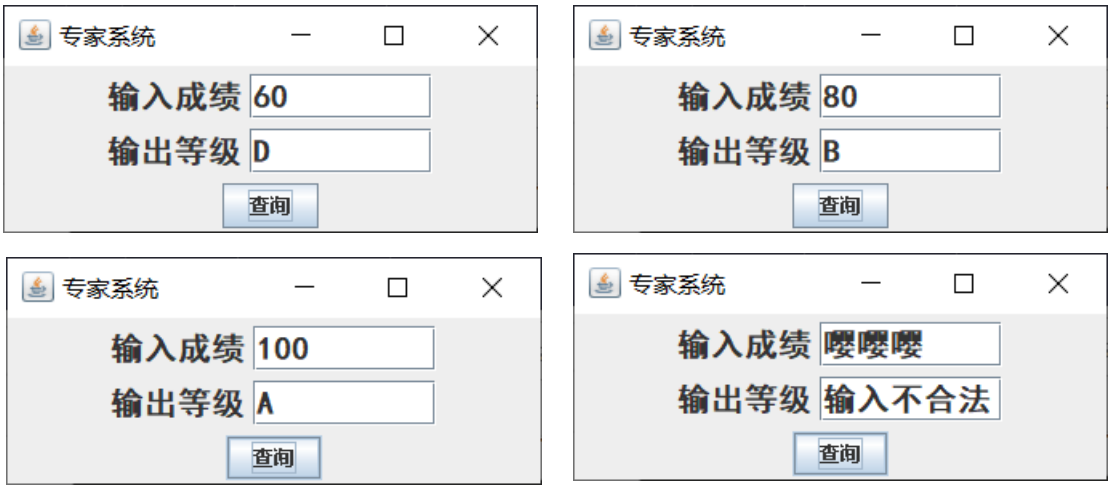

图 3 专家系统成果展示

代码已上传至 GitHUb：https://github.com/UncoDong/SAInCug

（2）分析基于规则的系统适用于什么情况，其优缺点是什么？

优点：

属于声明式编程，代码简单好理解，利于项目组交流。

逻辑和数据分离，代码结构更直观。

扩展性高，修改速度快。

缺点：

代码项目的复杂度提高。

需要学习新的规则脚本语法。

引入新的组件可能导致在不同环境运行失败。

参考资料：

[1] Drools 入门：https://www.jianshu.com/p/c2c0f78f416a