IRBL迭代一：软件概要设计

更新日志：

时间	更新理由	责任人
2021.3.11	创建文档	程荣鑫
2021.3.12	更新软件设计	程荣鑫
2021.3.13	修复了接口设计中的一些逻辑错误	程荣鑫

1. 总述

- 迭代一：实现一个基本的IR模型（VSM）并在给定的数据上计算

本项目在迭代一过程中，只需要实现VSM模型，并在给定的数据(SWT项目)上计算。

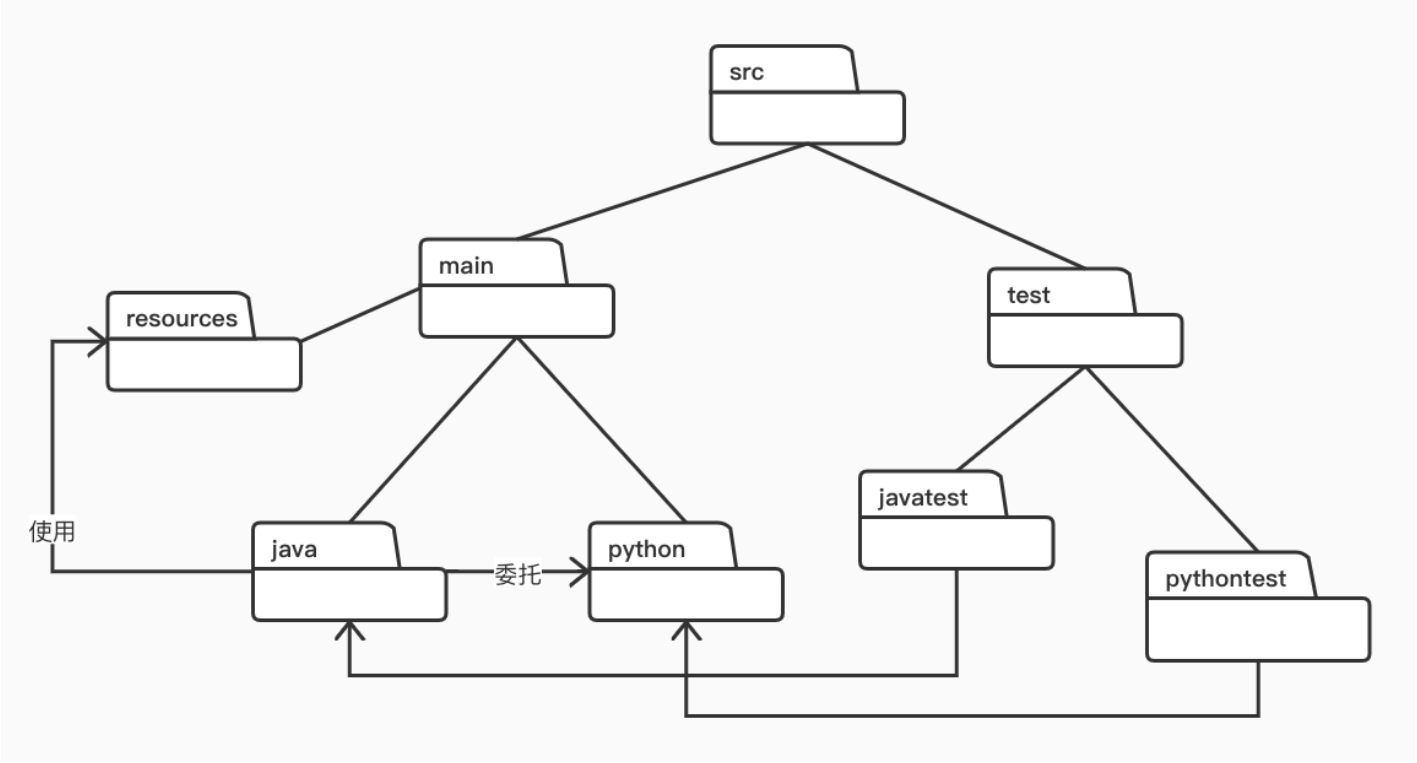
这个项目在迭代一期间没有使用SpringBoot框架的必要，甚至没有使用数据库的需要，要求比较宽松，允许使用Python语言，考虑到Python实现模型的便利性（有numpy等第三方库），计划使用Java1.8+Python3.8混合开发。

运行环境：通用计算机

2. 总体设计

因为使用了混合开发，为防混乱，将不同语言分割开。

主要代码在src/main包中，./java中是java代码，./python中是python代码，程序的入口是Java模块，运行中Java模块会将一些职责委托给Python模块。例如，在迭代一中，VSM用Python语言实现，使用Java语言调用.py，调用方式选择Process+命令行。



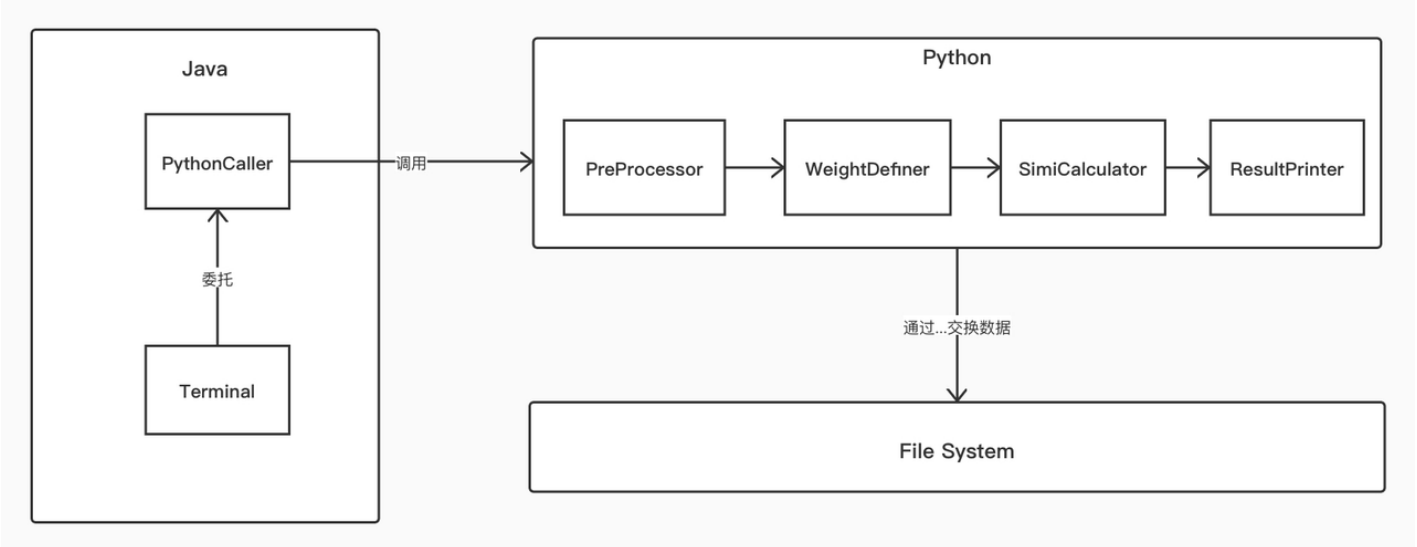
3. 外部资源

数据由教师团队提供，不需要自行挖取，数据源存放在与src目录同级的data目录中。除数据之外，本项目暂时不依赖其他外部资源，也不需要与外部系统交互。

4. 模块设计

软件体系结构

采用类流水线式软件系统设计，如下图：



说明：

用户在控制台输入命令，Terminal对象监听命令，并委托PythonCaller调度相应的处理程序，经过预处理（PreProcessor）、计算词语权重（WeightDefiner）、相似度计算（SimiCalculator），获取最终的计算结果，因为走完这一流程的耗时非常长，交互性较差，系统支持两种运行方式：

1. 每输入一个命令，执行流水线中的一个任务（条件是前置任务执行完）——交互性更强
2. 一次性执行完成——自动走流程

系统目前支持命令：

1. preprocess（执行预处理任务）
2. define weight（计算词语权重）
3. calculate similarity（计算相似度并保存）
4. print result [num]
5. do all [num]（连续完成四个任务）
6. exit（退出程序）

命令的接口规范在**模块详述**中进一步说明

模块通信：

1. Java程序调用Python程序使用sys.argv传参（即在python xxx.py命令后添加参数）
2. Python模块间需要交换的数据都是巨量的，因此每个模块计算完成后，需要将结果持久化保存在文件中，方便下一模块取用，通过文件系统进行通信

模块详述

Python模块间的数据交换主要通过文件IO实现，这些文件统一存访在项目目录：
src/main/python/Datapool中

1. Terminal

功能：监听控制台的命令，根据命令委托PythonCaller调用相应的python程序

输入：

- preprocess：预处理命令，调用PreProcessor功能模块
  - 调用脚本：python preprocessor.py [bug报告文件相对路径] [全部代码文件的相对路径（','分割）]，相对路径指的是从**项目根目录**开始的文件相对路径

- define weight：计算权值的命令，调用WeightDefiner功能模块
  - 调用脚本：python weight_definer.py
- calculate similarity：计算文本相似度的模块，调用SimiCalculator模块
  - 调用脚本：python simi_calculator.py
- print result [num]：输出与每个bug最相关的num个文件的相对路径，调用ResultPrinter模块
  - 调用脚本：python result_printer.py num，其中num应该设置默认值：5
- do all [num]：等价于按照预处理、计算权值、计算相似度、输出结果的顺序调用上面4个命令
  - 直接复用上面4个命令的调用代码即可（避免重复耦合）
- exit：终止程序命令，停止Java程序，终止python子进程

输出：无

2. PythonCaller

功能：Terminal的代理人，接受Terminal的委托，调用Python程序，防止Terminal模块职责过重。

输入：Python程序调用字符串

处理：开启子进程，运行Python程序

输出：无

3. PreProcessor

功能：对文本数据进行预处理工作，进行去停用词、词形还原、分割。

输入：PythonCaller传来的文件路径

处理：读取所有文件路径对应的文件，对文件内容进行预处理

输出：预处理的结果写入新建的txt格式文件，文件名称来自于原来代码文件名，bug报告使用这样的命名：{id}.txt，其中id指的是bug id。（代码文件在Datapool/class下，bug报告在Datapool/report下）

4. WeightDefiner

功能：计算预处理后的文本数据池中每个词语的权重

输入：经过预处理的文本

处理：使用TF-IDF算法计算每个词的权重，并且按字典序为每个词编号（从0开始）

输出：word-idx映射均用json格式存储，TF-IDF的权重计算结果也使用json存储，json的key为文件相对路径（**从Datapool之下开始的相对路径**），value是一个数组，数组中每个词语权重的对应索引等于word-idx映射中的词语索引

- 例如：word-idx{"crx":1, "glh":2,"hx":3, "zzy":4}, weights={"./xxx/file": [1,1,1,1]}（数组元素分别表示"crx", "glh", "hx", "zzy"的权重）

5. SimiCalculator

功能：根据TF-IDF计算结果，计算bug文本和代码文本间相似度

输入：TF-IDF计算结果、word-idx映射表

处理：相似度计算

输出：结果在逻辑上是一个矩阵，即bug-file相似度矩阵，存储格式使用json

- 例如：{"1": {"file1": 0.5, "file2": 0.4}}

6. ResultPrinter

功能：展示计算结果

输入：SimiCalculator输出的bug-file相似度矩阵

处理：输入与每个报告的bug前{num}个最相似的代码文件名（相对路径）

输出：结果同时输出控制台和.txt格式文件中