谭福华_14号

任务1.1.0 跑通ChatUniTest（Python版和Java版皆可）工具并熟悉ChatUniTest源码

使用的Java插件版，跑通流程：

- git clone https://github.com/ZJU-ACES-ISE/chatunitest-maven-plugin.git
- 为防止和原版本冲突，pom.xml将groupId修改为io.github.TFH和artifactId为chatunitest-maven-plugin-explanatory以作区别
- 安装到自己的maven仓库

```
mvn clean install
```

- 查看是否install在本地仓库
- 

- 自己新建一个maven项目（为了方便测试，只有一个接口），进行测试
- 配置pom.xml加入插件的配置参数，并且加入其他依赖

```
<plugin>
    <groupId>io.github.TFH</groupId>
    <artifactId>chatunitest-maven-plugin-explanatory</artifactId>
    <version>1.0.0</version>
    <configuration>
        <apiKeys>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</apiKey
        <model>gpt-3.5-turbo</model>
        <testNumber>2</testNumber>
        <maxRounds>2</maxRounds>
        <minErrorTokens>500</minErrorTokens>
        <temperature>0.5</temperature>
        <topP>1</topP>
        <frequencyPenalty>0</frequencyPenalty>
        <presencePenalty>0</presencePenalty>
        <proxy>127.0.0.1:7890</proxy>
        <thread>true</thread>
```

```
        </configuration>
    </plugin>
```

- 运行之后发现AskGpt()服务返回code:429，参照openAI官网的API手册原因是超出配额

429 - You exceeded your current quota, please check your plan and billing details	**Cause:** You have hit your maximum monthly spend (hard limit) which you can view in the account billing section. **Solution:** Apply for a quota increase.

- 开通visa虚拟卡，在openAI官网开通账单后限速取消
- 重新运行 mvn chatunitest:project 运行正常

```
[INFO] Generating test for method < getListSong > round 1 ...
[INFO] Generating test for method < registerManager > round 1 ...
[INFO] Generating test for method < getMessage > round 1 ...
[INFO]
=========================
[ChatTester] Generating test for method < setUser > number 1...

[INFO] Generating test for method < setUser > round 1 ...
[INFO] Role: chatGpt的回答

[INFO] Content: ```java
package com.dgut.music_online.domain;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;
```

-

熟悉源码

从ProjectTestMojo开始看逐步扩展，看完全部的代码，并对95%的函数做了非常详细的注释说明，详细注释已提交到github:ChatUniTest-maven-plugin-explanatory-

工程逻辑

- 项目的入口ProjectTestMojo，其继承于AbstractMojo，同时被所有的其他Mojo所继承，在ProjectTestMojo中定义了许多配置信息，如apiKeys、thread等信息，所以excute的第一步就是初始化参数信息（其中包括用户在configuration中填写的，还有自动生成的比如tmpOutput和parseOutput等）
- 项目为了提供效率开启了多线程，在pom.xml中使用<thread>true</thread>即可配置，项目的大体运行步骤分布五步：

1. 首先是扫描所有java类，并将这些信息保存为classInfo，
2. （使用多线程）对每一个classInfo，运行ClassRunner的start方法
3. （使用多线程）对每一个ClassInfo,扫描出类的所有method,并保存为MethodInfo，运行MethodRunner的start方法
4. （使用多线程）对每一个MethodInfo，使用多线程开启testNumber（<testNumber>5</testNumber>）轮生成
5. 对于每一轮，开启maxRounds（<maxRounds>2</maxRounds>）轮测试，每一次使用generateMessages生成对chatgpt的提问（对于method有依赖要使用generatePromptInfoWithDep生成依赖，没有的则使用generatePromptInfoWithoutDep），然后则使用AskGpt类对chatgpt进行提

问，使用parseResponse对其回答进行代码提取（其回答有一定规律性）。然后使用TestCompiler对代码进行编译测试，如果不通过编译则进行下一轮修复直到通过编译为止或者达到maxRounds

- 项目还在开始之前和结束时对代码进行了备份和还原

任务1.1.1.2开销估计和限额：当前使用ChatGPT生成单元测试需要耗费很多Token，而OpenAI的Token价格依然很高，因此为了防止用户过度使用带来过高开销，需要增加开销估计和限额的功能。建议用时3天，方案分满分30（仅有方案未实现不得分）、完成分满分30，总计60分。

开销限额

大体方案

1. 增加一个配置参数<maxUseTokens>10000</maxUseTokens>,可在configuration里配置（默认值为10000）
2. 在AskGPT().askChatGPT()方法里对每次提问之前的message计算input token（使用TokenCounter.countToken（）方法），如果将要提问的token数和已经消耗的token数超过了maxUseTokens，则不再提问逐渐结束项目
3. 监听askChatGPT的返回，如果提问成功，则对chatgpt的返回结果进行令牌计算，将消耗的token数量加上这次提问和回答一共的token数量
4. 总结-在项目结束之前打印出消耗的token总数量，并按照openAi官网给出的价格，转化为美元进行展示

```
============================
[ChatTester] Eventually consumed 423365 Tokens
[INFO]
============================
[ChatTester] Eventually cost 0.0013645 $
[INFO]
============================
[ChatTester] Generation finished
[INFO]
```

具体代码实现

1. 增加一个配置参数<maxUseTokens>10000</maxUseTokens>,可在configuration里配置（默认值为10000）
   Config中

```
1  public static int haveAskCostTokens;
2  public static int haveResponseCostTokens;
3  public static int maxUseTokens;
```

projectTestMojo中

```
@Parameter(name = "maxUseTokens",defaultValue = "10000")
public int maxUseTokens;
```

2. 在AskGPT().askChatGPT()方法里对每次提问之前的message计算input token（使用TokenCounter.countToken（）方法），如果将要提问的token数和已经消耗的token数超过了maxUseTokens，则不再提问逐渐结束项目

```
int AskCostTokens = TokenCounter.countToken(messages);
if(AskCostTokens+Config.haveAskCostTokens+Config.haveResponseCostTokens>Con
    log.error("\n==========================\n[ChatTester] Reach the maximum
    return null;
}
```

3. 监听askChatGPT的返回，如果提问成功，则对chatgpt的返回结果进行令牌计算，将消耗的token数量加上这次提问和回答一共的token数量

```
Config.setHaveCostAskTokens(Config.haveAskCostTokens+AskCostTokens);
```

```
int ResponceCostTokens = TokenCounter.countToken(content);
Config.setHaveCostResponseTokens(Config.haveResponseCostTokens+ResponceCost
```

4. 总结-在项目结束之前打印出消耗的token总数量，并按照openAi官网给出的价格，转化为美元进行展示

```
public static double TokenTransfer2Money(int inputTokens,int outputTokens){
    double price = 0 ;
    switch (Config.model){
        case "gpt-4":
            price = inputTokens * 0.03 + outputTokens*0.06;
            break;
        case "gpt-4-32k":
            price = inputTokens * 0.06 + outputTokens*0.12;
            break;
        case "gpt-3.5-turbo":
            price = inputTokens * 0.0015 + outputTokens*0.002;
            break;
        case "gpt-3.5-turbo-16k":
            price = inputTokens * 0.003  + outputTokens*0.004;
            break;
    }
    return price/1000;
}
```

```
log.info("\n==========================\n[ChatTester] Eventually consumed "
        + Config.haveAskCostTokens+Config.haveResponseCostTokens
```

```
3           + " Tokens\n"
4           + "[ChatTester] Eventually cost "+ EstimateTokenMojo.TokenTransfer2M
5           + "\n==========================\n");
```

token预测

大体方案

1. 因为OpenAi官网对token的定价分为Ask和Response，所以需要分别对两者进行估计，而且由于chatGpt每次回答是否有编译错误是不可预测的，只能给出一个最小值~最大值的范围。预测方法：

- 第一次的Ask信息 = 直接生成
- 修复的Ask信息 = <maxPromptTokens>2600<maxPromptTokens>（由于修复的信息带有报错信息，报错信息是远远大于2600的，当前的报错信息提取方式为首先使用maxPromptTokens-methodSignature-className-info-unitTest的信息，然后按照剩余的token从前面截取）
- 第一次Response信息 = 回答的固定格式所消耗的token(如下) + 代码部分（package语句+原来类的import语句和基本的import语句+类的声明语句+预测测试代码的token(为源代码的两倍)）

```
1  Here is the JUnit test for the `test()` method in the `test` class:
2
3  \```java
4
5  \```
6
7  Make sure you have the necessary dependencies for JUnit 5 and Mockito 3 in y
```

- 修复回答的Response信息 = 回答的固定格式（如下）+代码部分（package语句+import语句+类的声明语句+预测测试代码的token(为源代码的两倍)）
-

```
1   Here is the fixed unit test:
2
3  \```java
4
5  \```
6
7  Note that I removed the `@Timeout` annotation as it was not necessary for th
```

具体代码实现

1. EstimateTokenMojo类作为执行方法,主要进行了一些微调，使其只生产message不进行提问

```
1  mvn chatunitestexplanatory:EstimateToken
```

2. 在ClassRunner里新增StartEstimateCostTokenClass函数，返回List<List>类型
3. 在MethodRunner里新增StartEstimateCostTokenMethod函数，返回List
4. MethodTokenCostInfo中记录了四种方式各回消耗的token数量

5. 最后对所有的Method进行合并，并生成为一个json保存在chatUniTest的根目录下


```
{
    "maxRound": 1,
    "test": [
        {
            "FirstAskCost": 191,
            "FirstResponseCost": 315,
            "FixedAskCost": 2600,
            "FixedResponseCost": 300,
            "methodName": "test",
            "className": "test",
            "firstResponseCost": 315,
            "fixedResponseCost": 300,
            "firstAskCost": 191,
            "fixedAskCost": 2600
        }
    ],
    "testNumber": 1,
    "TestApplication": [
        {
            "FirstAskCost": 232,
            "FirstResponseCost": 320,
            "FixedAskCost": 2600,
            "FixedResponseCost": 305,
            "methodName": "main",
            "className": "TestApplication",
```


```
        for(List<MethodTokenCostInfo> methodTokenCostInfoList:methodTokenCos
            String classname = methodTokenCostInfoList.get(0).className;
            for(MethodTokenCostInfo methodTokenCostInfo:methodTokenCostInfol
                minResult += methodTokenCostInfo.FirstAskCost + methodToken(
                maxResult += methodTokenCostInfo.FirstAskCost
                        + methodTokenCostInfo.FirstResponseCost
                        + (methodTokenCostInfo.FixedResponseCost+methodToke
                        * (Config.testNumber*Config.maxRounds);
                minMoney += TokenTransfer2Money(methodTokenCostInfo.FirstAsl
                maxMoney += TokenTransfer2Money(methodTokenCostInfo.FirstAsl
                        + TokenTransfer2Money(methodTokenCostInfo.FixedAskC(
                        * (Config.testNumber*Config.maxRounds);
            }
            methodTokenCostInfoMap.put(classname,methodTokenCostInfoList);
        }
```

6. 在最后进行提示，生成结果的代码，并同时进行金额换算

```
log.info("\n=========================="
        + "\n==========================\n"
        + "[ChatTester] Eventually this project will cost "+minResult+" ~ "
        + "[ChatTester] Eventually this project will cost "+minMoney+"$ ~ "

```

```
6           + "\n==========================="
            + "\n===========================\n");
```

```
==========================
==========================
[ChatTester] Eventually this project will cost 1058 ~ 6863 tokens
[ChatTester] Eventually this project will cost 0.0019045$ ~ 0.0109145$
==========================
==========================
```

任务1.1.1.3服务化改造：当前ChatUniTest主要是以客户端工具的形式来使用，我们希望可以通过Web服务的形式对外提供服务，这样客户端可以是各种IDE插件，客户端只需要负责用户交互，测试用例生成的主体功能则由服务端完成。建议用时5天，方案分满分50（仅有方案未实现不得分）、完成分满分50，总计100分。

大体实现方案

1. 使用vue+spring boot+mysql搭建了一套前后端分离的web服务系统
2. 用户可以通过邮箱验证码注册登录进入（通过次记录用户所提交的工程，用户也可以查看工程的进展状态），后续可以绑定支付功能（因为大型工程每次生成会消耗巨量的token是一笔不小的开销）
3. 用户可以在主页选择为整个Project生成测试还是估计Token消耗的数量，并上传自己的工程文件和输入工程名
4. 用户可以在两个状态页分别查看工程状态

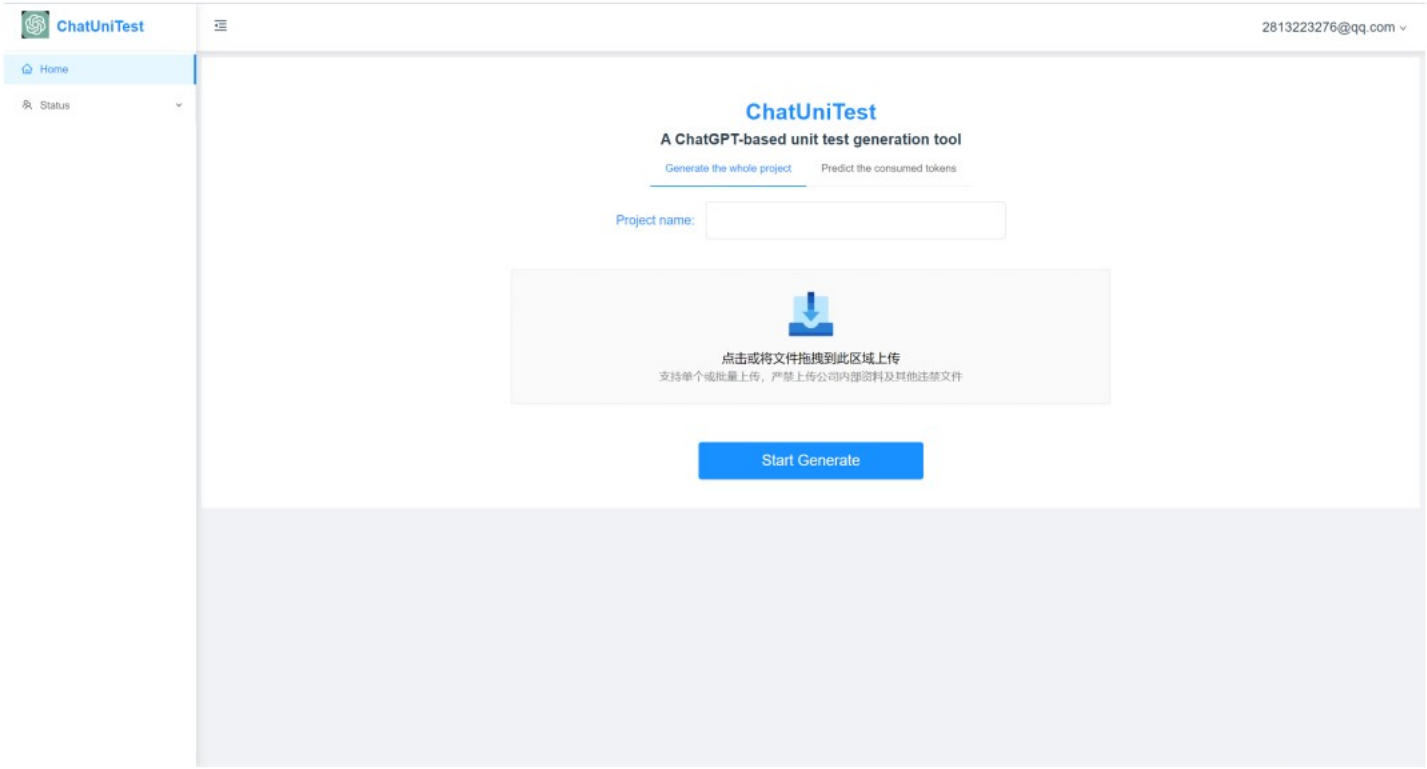
具体实现代码

- upload接口实现了接受用户上传的工程压缩包并进行解压缩
- file?filename=...实现了用户下载处理号的工程文件
- /project/start接口实现了工程的开始处理代码，具体工程

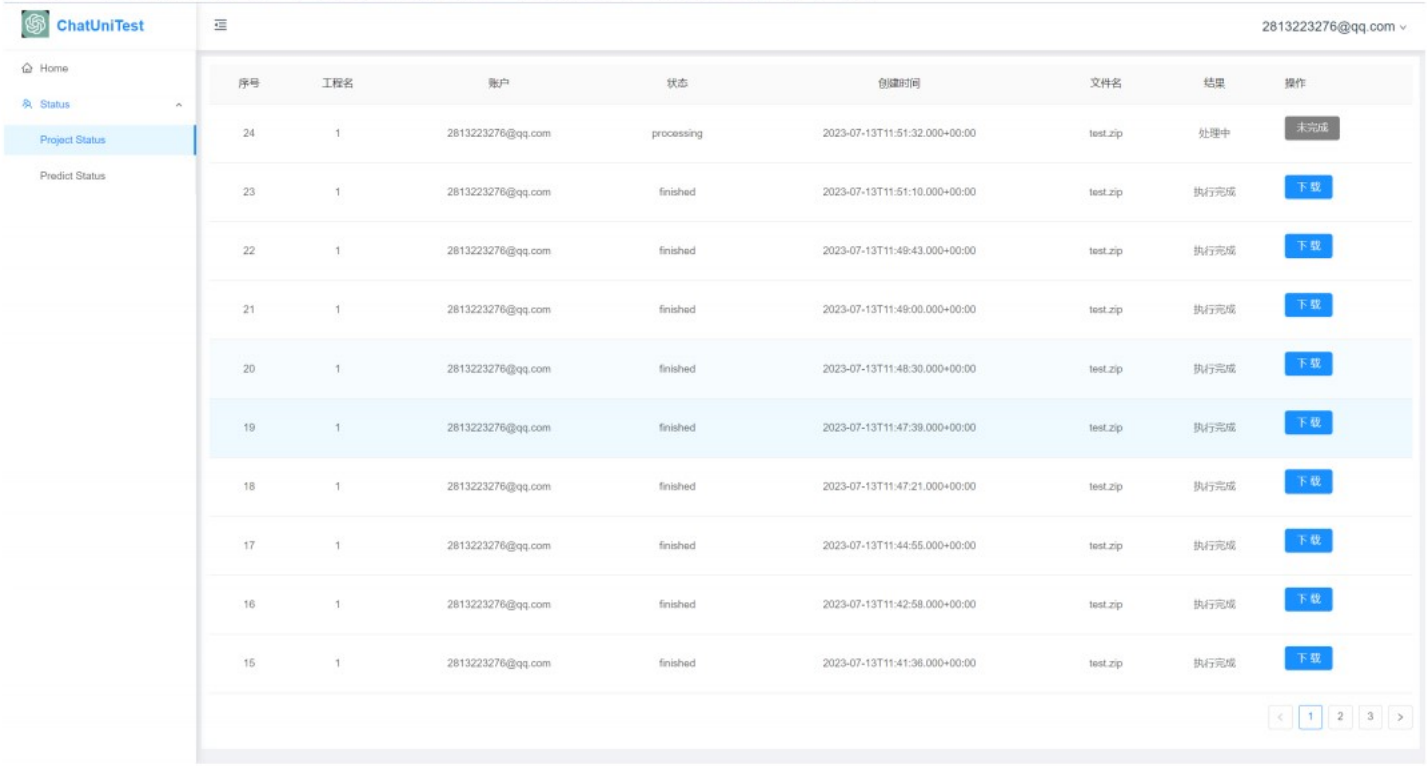
1. 检查目录-查看用户上传的工程文件是否完整
2. 检索pom.xml文件位置，并根据次获得工程的根目录
3. 读取pom.xml信息，并将其备份，将我们需要的dependency和plugin依次添加到pom.xml，重新写pom.xml
4. 开启一个processBuilder后台启动一个命令行，首先切换到工程根目录下，然后执行相应的命令（mvn chatunitest:project或者mvn chatunitest:EstimateToken）

最终效果

- 上传界面，在此输入工程名并且上传工程压缩包



- 查看工程状态界面，已经处理好的工程点击下载按钮即可下载



- 登录界面，测试阶段账户名：2813223276@qq.com 密码：123456789

- 注册界面，输入邮箱后，点击发送即可收到验证码

任务1.1.2.3 提炼错误信息：当前ChatUniTest需要根据错误信息对测试用例进行修复，由于程序执行报错信息可能过长，所以使用了简单的截断策略，需要探索使用更加智能的方式，从错误信息中提取有用的信息。建议用时7天，方案分满分70（仅有方案未实现，乘以0.5系数）、完成分满分50，总计120分。

实现方案

通过观察发现，TestCompiler的报错信息重复性巨多，无用性很多（主要体现在报错路径中），因此主要从这两方面优化

- 去重，根据每段error的哈希值去除掉重复的报错信息

```
public static String processErrorMessage(List<String> msg, int allowedTokens
    if(allowedTokens<=0)
        return "";
    ErrorParser errorParser = new ErrorParser();
    TestMessage testMessage = errorParser.loadMessage(msg);
    TokenCounter tokenCounter = new TokenCounter();
    List<String> errors = testMessage.getErrorMessage();
    String errorMessage = String.join(" ",errors);

    List<String> save = new ArrayList<>();
    List<String> hashResult = new ArrayList<>();
    for(String error:errors){
        // 查看是否有相同的哈希值
        String tempHash = calculateHash(error);
        boolean haveSame = false;
        for(String hash:hashResult){
            if(tempHash.equals(hash)){
                haveSame = true;
                break;
            }
        }
        if(!haveSame) {
            save.add(error);
            hashResult.add(tempHash);
        }else{

        }
    }

    return processErrorMessage(save,allowedTokens);
}
```

- 使用正则表达式去除所有路径信息

```
public static String removePaths(String text) {
    String pattern = "/.*?/";
    Pattern regex = Pattern.compile(pattern);
    Matcher matcher = regex.matcher(text);
    String result = matcher.replaceAll("");
    return result;
}
```

- 对所有报错信息进行相似度对比，对相似度非常高的语句提取出差别关键字，然后对这些相似度高的语句进行合并
- 最后根据要求的maxPromptToken，进行截断