《OpenAI 代码自动评审组件》- 第3节-学习记录

来自： 码农会锁

爱海贼的无处不在 2024年07月28日 14:25

学习系列文章：

1、《OpenAI 代码自动评审组件》- 第1节-学习记录：https://articles.zsxq.com/id_bpapuj05ur03.html

2、《OpenAI 代码自动评审组件》- 第2节-学习记录：https://articles.zsxq.com/id_uokpg7mvt1uc.html

一、第2节内容加餐

在分享第3节内容之前，我先分享下关于第2节的自己的加餐内容，在作者讲完了第2节内容后，我们作为程序员就需要自己去学习背后的知识，关联的知识，而不是仅仅停留于讲师的表面内容，需要自己去探索。

在第2节中，作者讲解了如何基本使用Github Action，那么我们就可以思考既然Github Action中能够执行系统命令，那么是不是可以也可以完成一个非常简单的CodeReview呢?

答案是肯定的。

在第2节内容中，我分享了Github Action市场的东西，同时分享了Github Action中的100多个AI Code Review的公共Actions，学习后，在结合智普AI的接口文档，我们于是可以实现第一个CodeReview的方案设计，即：基于Github Action+智普AI接口+CURL命令实现的第一个CodeReview流程，流程图如下所示：

接下来看看我的实现代码吧，仅仅需要一个Github Action的工作流配置文件就可以实现一个简单版本的Github 项目的AI代码评审：

openai_codereview.yml

```yaml
name: AI LLM Code Review Workflow

on:
  push:
    branches:
      - 'xxxx'

jobs:
  openai-code-review:
    runs-on: ubuntu-latest
    name: OpenAI LLM Code Review
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
        with:
          fetch-depth: 2

      - name: Get changed files
        id: changed-files
        uses: tj-actions/changed-files@v43

      - name: Filter Java Files
        id: changed-java-files
        uses: tj-actions/changed-files@v44
        with:
          files: |
            **/*.java

      - name: List all changed files
        env:
          ALL_CHANGED_FILES: ${{ steps.changed-java-files.outputs.all_changed_files }}
        run: |
          for file in ${ALL_CHANGED_FILES}; do
            echo "$file was changed"
          done
        shell: bash
```

步骤1：检出代码仓库

步骤2：获取变更文件

步骤3：过滤Java文件

步骤4：打印下变更的文件

在这个工作流文件的上半部分中，分别设置了4个步骤：

1、检出代码

2、获取变更文件

3、过滤变更文件中的java内容

4、打印变更文件名称

接下来再看看文件的下半部分：

```yaml
- name: List all changed files
  env:
    ALL_CHANGED_FILES: ${{ steps.changed-java-files.outputs.all_changed_files }}
  run: |
    for file in ${ALL_CHANGED_FILES}; do
        echo "$file was changed"
    done
  shell: bash

- name: Open AI Review code
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
    CHATGPT_KEY: ${{ secrets.CHATGPT_KEY }}
    CHATGPT_MODEL: ${{ secrets.CHATGPT_MODEL }}
  run: |
    touch result_review.md
    for file in ${{ steps.changed-java-files.outputs.all_changed_files }}; do
        content=$(jq -Rs . "$file")
        echo "准备发送AI代码审查请求"
        json=$(jq -n --arg data "$content" '{"stream": false,"model": "glm-4-flash","messages": [ {"role": "system", "content": "你是一个Java技术专家,请对我的
        echo "发送给AI的数据结构: $json"
        airesponse=$(curl https://open.bigmodel.cn/api/paas/v4/chat/completions \
          -H "Content-Type: application/json" \
          -H "Authorization: Bearer $CHATGPT_KEY" -d "$json" | jq -r '.choices[0].message.content')

        if [ "$airesponse" == "null" ]; then
          echo "API调用失败，请检查配额和账单信息。"
          exit 1
        else
          echo "Response from OpenAI:"
          echo "$airesponse"
          echo "Github Action AI Code Review for \`$file\`, 代码评审结果如下: $airesponse " >> result_review.md
        fi
    done
  shell: bash
- name: Commit review result
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  run: |
    git config --global user.name 'github-actions[bot]'
    git config --global user.email 'github-actions[bot]@users.noreply.github.com'
    git add result_review.md
    git commit -m "Add OpenAI code review result"
    git push
```

步骤5：定义环境变量

步骤5：定义智普AI发送请求体

步骤5：解析AI返回结果

步骤5：写入结果到本地文件

步骤6：提交到当前Git仓库

在下半部分中，使用了循环发送CURL请求的方式，向智普AI的后端接口发送了代码评审的请求，官方文档关于请求参数和请求体的描述如下：https://bigmodel.cn/dev/api#http_call。

接下来程序会把结果写入到文件，然后提交到当前的代码仓库中。这样完成了我们的第一个AI评审的实践。仅仅一个文件而已。

二、正文

接下来开始分享今天的学习，作者在第3节的教程中，作者首先手把手的分享了如何编写新的Github Actions的工作流脚本:

```yaml
on:
  push:
    branches:
      - *
  pull_request:
    branches:
      - *

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2
        with:
          fetch-depth: 2

      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          distribution: 'adopt'
          java-version: '11'

      - name: Build with Maven
        run: mvn clean install

      - name: Copy openai-code-review-sdk JAR
        run: mvn dependency:copy -Dartifact=plus.gaga.middleware:openai-code-review-sdk:1.0 -DoutputD

      - name: Run code Review
        run: java -jar ./libs/openai-code-review-sdk-1.0.jar
```

然后教大家如何用Java的Process API代码执行系统的Git命令，最终获取到了当前代码中变更代码的字符串。

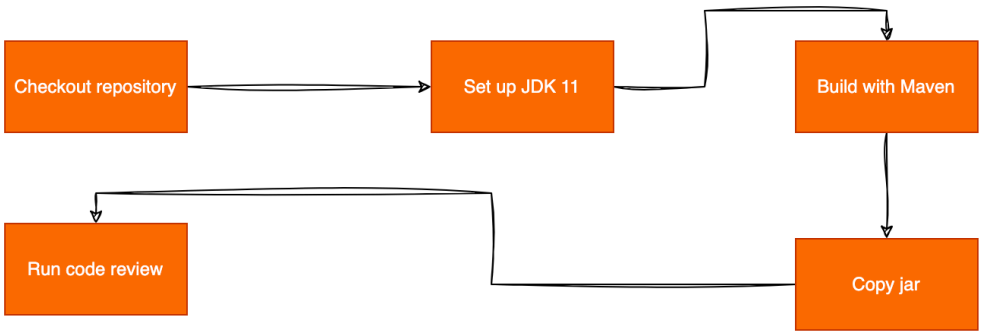
AI代码评审的本质就是将这种字符串传递给AI大模型，让大模型替我们进行整理、归纳、总结。

```
开始执行CodeReview
变更代码:diff --git a/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
 b/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
index 3b46a26..3b7d236 100644
--- a/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
+++ b/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
@@ -19,6 +19,9 @@ public class ApiTest {
     @Test
     public void test() {
         System.out.println(Integer.parseInt("aaaa1"));
+        System.out.println(Integer.parseInt("zzzz1"));
+        System.out.println(Integer.parseInt("zzzz2"));
+        System.out.println(Integer.parseInt("zzzz3"));
     }

 }
```

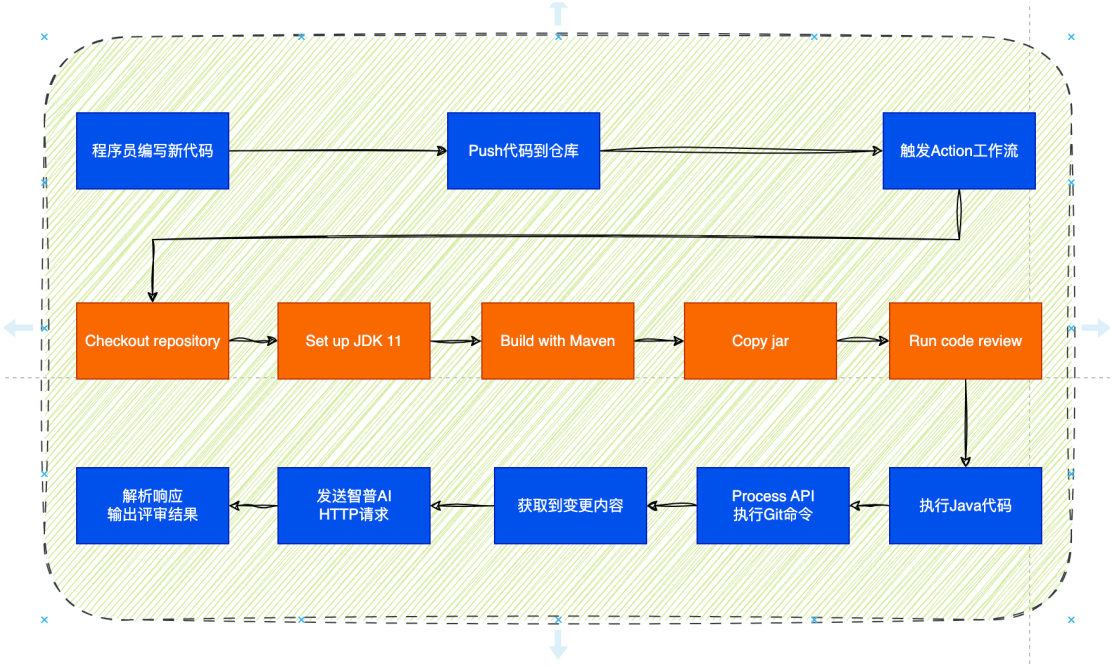
然后将新的Action发布到Git上后，可以看到执行成功的效果:

```
Run code review
1  ▸ Run java -jar ./libs/openai-code-review-sdk-1.0.jar
6  开始执行CodeReview
7  变更代码:diff --git a/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java b/openai-code-revie
   test/java/cn/aijavapro/middleware/test/ApiTest.java
8  index 3b46a26..3b7d236 100644
9  --- a/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
10 +++ b/openai-code-review-test/src/test/java/cn/aijavapro/middleware/test/ApiTest.java
11 @@ -19,6 +19,9 @@ public class ApiTest {
12      @Test
13      public void test() {
14          System.out.println(Integer.parseInt("aaaa1"));
15 +        System.out.println(Integer.parseInt("zzzz1"));
16 +        System.out.println(Integer.parseInt("zzzz2"));
17 +        System.out.println(Integer.parseInt("zzzz3"));
18      }
19
20  }
21 执行退出:0
```

这个操作为后续在Java层面实现了代码评审打下了坚实的基础，作者在本次的Github Action工作流中，定义了代码的检出、Java环境的配置、 项目的打包、项目的依赖拷贝、项目的运行，从而实现了一个Github Action工作流程：



这样如果我们深入思考，可以分析当前AI代码评审的截止到目前能够实现到的一个基础的业务流程图：



既然有了这个流程图，我们就开始上手写代码吧，首先我们根据智普AI官网上的操作（https://bigmodel.cn/dev/api#sdk_install），在项目中引入智普AI的依赖包，这里我们使用最新版本的：release-V4-2.2.0

安装 Java SDK

Java SDK 地址： https://github.com/zhipuai/zhipuai-sdk-java-v4
如下方式，将SDK的依赖项加入到Maven项目：

```xml
<dependency>
    <groupId>cn.bigmodel.openapi</groupId>
    <artifactId>oapi-java-sdk</artifactId>
    <version>release-V4-2.0.2</version>
</dependency>
```

引入release-V4-2.2.0依赖后，参考官网文档，我们实现一个请求大模型的方法：

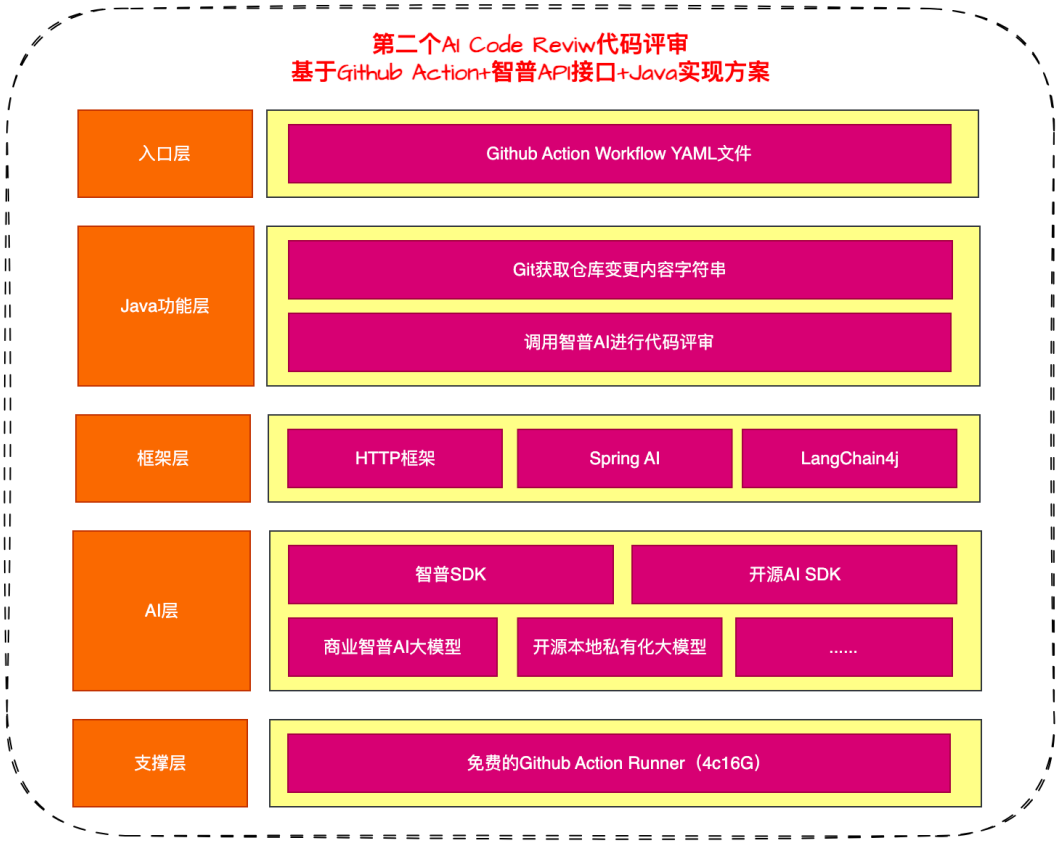
```java
public class OpenAiCodeReview {

    public static void main(String[] args) throws Exception {
        System.out.println("开始执行CodeReview");
        // 1.执行本地命令进行代码检出的操作
        String code =checkGitChangeCode();
        // 2.发送给智普AI进行代码评审
        String result = sendToZhipuAI(code);
    }
    private static String checkGitChangeCode() throws Exception {
        ProcessBuilder processBuilder = new ProcessBuilder("git", "diff", "HEAD~1", "HEAD");
        processBuilder.directory(new File("."));
        Process process = processBuilder.start();
        String diffCode = IOUtils.toString(new BufferedReader(new InputStreamReader(new
                BufferedInputStream(process.getInputStream()))));
        System.out.println("变更代码:" + diffCode);
        int result = process.waitFor();
        System.out.println("执行退出:" + result);
        return diffCode;
    }
    private static String sendToZhipuAI(String diffCode) {
        List<ChatMessage> messages = new ArrayList<>();
        String systemMessage = "你是一个非常有经验的Java架构师，非常擅长代码评审，请对用户提供的Git Diff格式的代码进行评审";
        ChatMessage systemChatMessage = new ChatMessage(ChatMessageRole.SYSTEM.value(), systemMessage);
        messages.add(systemChatMessage);
        String userMessage = "这是我的变更代码，请进行评审吧：" + diffCode;
        ChatMessage chatMessage = new ChatMessage(ChatMessageRole.USER.value(), userMessage);
        messages.add(chatMessage);
        ChatCompletionRequest chatCompletionRequest = ChatCompletionRequest.builder()
                .model("glm-4-flash")
                .stream(Boolean.FALSE)
                .invokeMethod(Constants.invokeMethod)
                .messages(messages)
                .build();
        // 为了安全我们从系统的环境变量中获取
        String key = System.getProperty("GLM_KEY");  动态读取秘钥
        if(key == null || key.trim().length() == 0){return "";}
        ClientV4 client = new ClientV4.Builder(key).build();
        ModelApiResponse invokeModelApiResp = client.invokeModelApi(chatCompletionRequest);
        try {
            System.out.println(JSON.toJSONString(invokeModelApiResp));
            String content = invokeModelApiResp.getData().getChoices().get(0).getMessage().getContent().toString();
            return content;
        } catch (Exception e) {
            e.printStackTrace();
            return "";
```

在这个代码中，我们使用了System的getProperty方法，让Java从系统环境中动态获取秘钥信息。
然后我们在更改下Github Action的脚本信息，增加上环境变量的读取，与-D参数的使用，秘钥的信息可以在Github的Secret中变量进行维护：

```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v2
    with:
      fetch-depth: 2
  - name: Set up JDK 11
    uses: actions/setup-java@v2
    with:
      distribution: 'adopt'  # 你可以选择其他发行版，如 'adopt' 或 'zulu'
      java-version: '11'

  - name: Build with Maven
    run: mvn clean install

  - name: Copy openai-code-review-sdk jar
    run: mvn dependency:copy -Dartifact=cn.aijavapro:openai-code-review-sdk:1.0 -Doutpu

  - name:  Run code review
    env:
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      CHATGPT_KEY: ${{ secrets.CHATGPT_KEY }}
    run: java -DGLM_KEY=$CHATGPT_KEY -jar ./libs/openai-code-review-sdk-1.0.jar
```

然后提交到Github仓库中，我们就能实现基于Github Action+Java的代码评审了。可以在Action日志中看到效果：

这样基于作者的讲解，与我们的学习，我们的第2个迭代的AI代码评审的实战例子就通过了，于是就可以梳理出一个新的架构图了：



三、总结

本文基于第3节的内容，记录了自己的思考、实践、梳理，目前根据作者的讲授的知识，已尝试进行2个AI代码评审CodeReview的实践，希望可以帮助到有需要的小伙伴。