代码（编程题作答结果）的预处理和执行

- 创建接口 CodeProcessor 来实现代码的编译和执行
- 接口方便添加其它语言文件类型的代码的处理

```java
public interface CodeProcessor {
    /**
     * 预处理代码
     * @param codeFilePath 代码文件路径
     */
    void preprocess(String codeFilePath) throws Exception;
    String execute(String className, String[] args) throws Exception;
}
```

- java代码的编译 执行 实现类如下：

```java
public class JavaCodeProcessor implements CodeProcessor {

    @Override
    public void preprocess(String codeFilePath) throws IOException,
InterruptedException {
        // 编写代码预处理逻辑，将 Java 代码编译成可执行文件
        // 调用命令行命令来编译代码

        // 获取文件的可编译路径
        String Path = PathUtil.getCompilerPath(codeFilePath);

        Process process = Runtime.getRuntime().exec("javac " + Path);

        int exitCode = process.waitFor();
        if (exitCode != 0) {
            throw new RuntimeException("Compilation failed");
        }
    }

    @Override
    public String execute(String className, String[] args) throws IOException,
InterruptedException {
        // 编写代码执行逻辑，执行编译好的 Java 类并传入参数
        // 调用命令行命令来执行代码
        String Path =  PathUtil.getClassPath();
        Process process = Runtime.getRuntime().exec("java -cp " + Path + " " +
className + " " + String.join(" ", args));
```

```
        // 读取代码执行的输出
        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
        StringBuilder output = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            output.append(line);
        }

        // 等待执行完成
        int exitCode = process.waitFor();
        if (exitCode != 0) {
            throw new RuntimeException("Execution failed");
        }

        return output.toString();
    }
}
```

- 其中对路径处理创建了关于路径的工具类 来将答题文件路径 转化为可以编译 执行的路径
-

-
  ```
  public class PathUtil {
      public static String getCompilerPath(String Path) {
          String res = "target" + File.separator
                  + "test-classes" + File.separator
                  + "cases" + File.separator
                  + "answers" + File.separator
                  + Path.substring(0, Path.lastIndexOf("/")) + File.separator
                  + Path.substring(Path.lastIndexOf("/") + 1);
          return res;
      }

      public static String getClassPath(){
          return "target" + File.separator
                  + "test-classes" + File.separator
                  + "cases" + File.separator
                  + "answers" + File.separator
                  + "code-answers" + File.separator;
      }

  }
  ```

并发需求

对于并发要求 自定义了线程池 来实现并发处理

```
/**
 * 自定义线程池
 */
public class CustomThreadPool {
    // 线程池中线程的数量为5
    private final int numThreads = 5;
    private final List<WorkerThread> threads;
```

```
    private final LinkedList<Runnable> taskQueue;

    public CustomThreadPool(int numThreads) {
        this.threads = new LinkedList<>();
        this.taskQueue = new LinkedList<>();
        initializeThreads(numThreads);
    }

    private void initializeThreads(int numThreads) {
        for (int i = 0; i < numThreads; i++) {
            WorkerThread thread = new WorkerThread();
            thread.start();
            threads.add(thread);
        }
    }

    public void submit(Runnable task) {
        synchronized (taskQueue) {
            taskQueue.addLast(task);
            taskQueue.notify(); // 唤醒等待的线程
        }
    }

    private class WorkerThread extends Thread {
        public void run() {
            Runnable task;
            while (true) {
                synchronized (taskQueue) {
                    while (taskQueue.isEmpty()) {
                        try {
                            taskQueue.wait(); // 等待任务
                        } catch (InterruptedException e) {
                            return;
                        }
                    }
                    task = taskQueue.removeFirst(); // 取出任务
                }
                try {
                    task.run(); // 执行任务
                } catch (RuntimeException e) {
                    // 处理任务执行异常
                    e.printStackTrace();
                }
            }
        }
    }
}
```

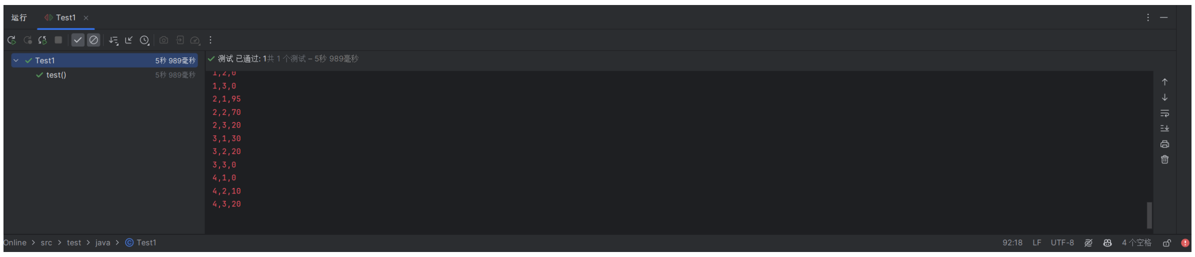
评分

评分对于之前 的代码题的处理器 策略进行了实现 来完成评分逻辑的实现

```
@Override
public int calculateScore(Question question, String answer) {
    if (answer == null){
        return 0;
    }

    CodeQ codeQ = (CodeQ) question;
    JavaCodeProcessor javaCodeProcessor = new JavaCodeProcessor();
    try {
        javaCodeProcessor.preprocess(answer);
        for (String input : codeQ.getSamples().keySet()) {
            String output = codeQ.getSamples().get(input);
            String result =
javaCodeProcessor.execute(answer.substring(answer.lastIndexOf("/")+1
,answer.lastIndexOf(".")), new String[]{input});
            // 任一样例不通过则返回0分
            if (!output.equals(result)) {
                return 0;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
        return 0;
    }
    return question.getScore();
}
```

运行成功示例



测试 已通过: 1共 1 个测试 – 5秒 989毫秒

```
1,2,0
1,3,0
2,1,95
2,2,70
2,3,20
3,1,30
3,2,20
3,3,0
4,1,0
4,2,10
4,3,20
```