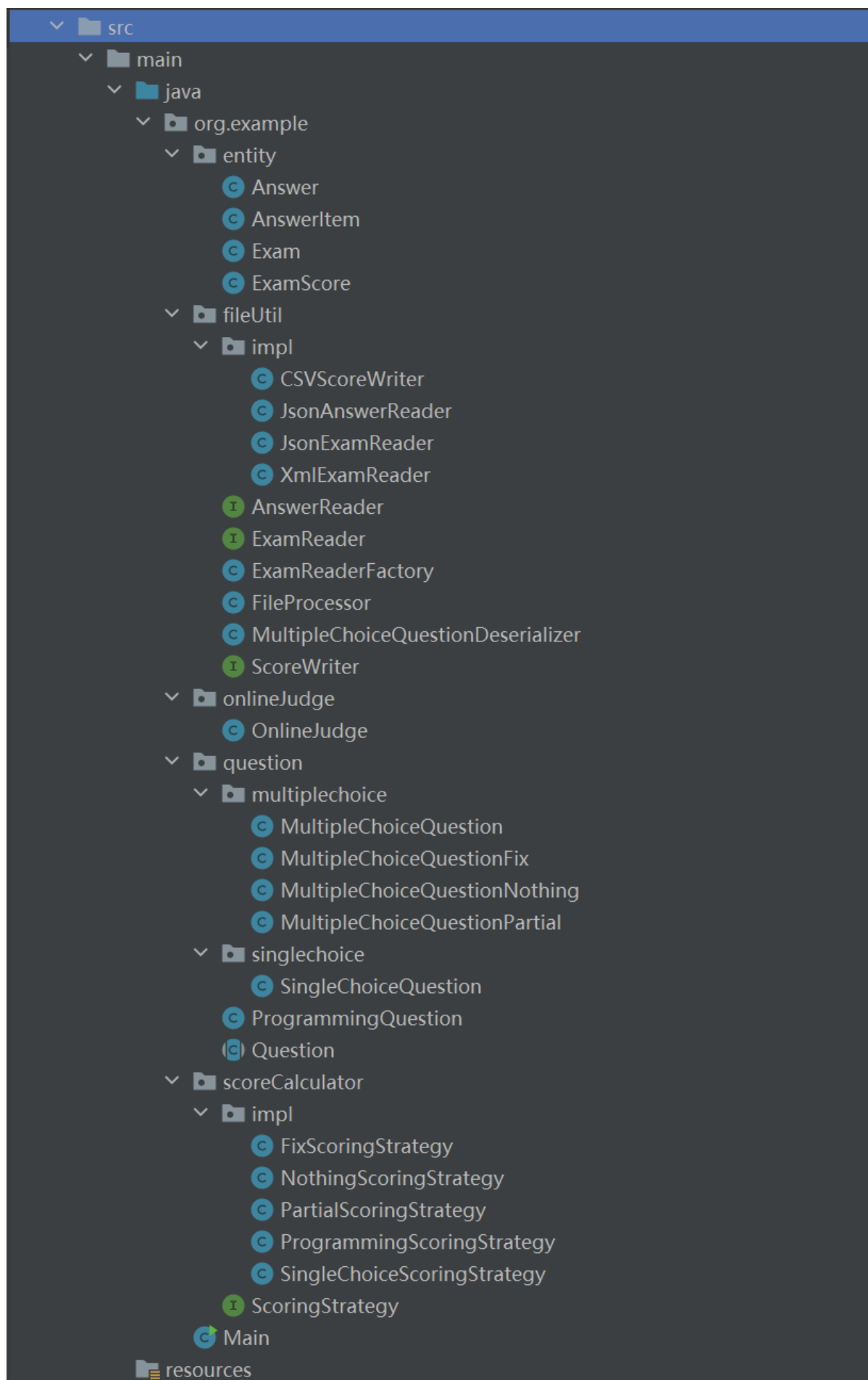


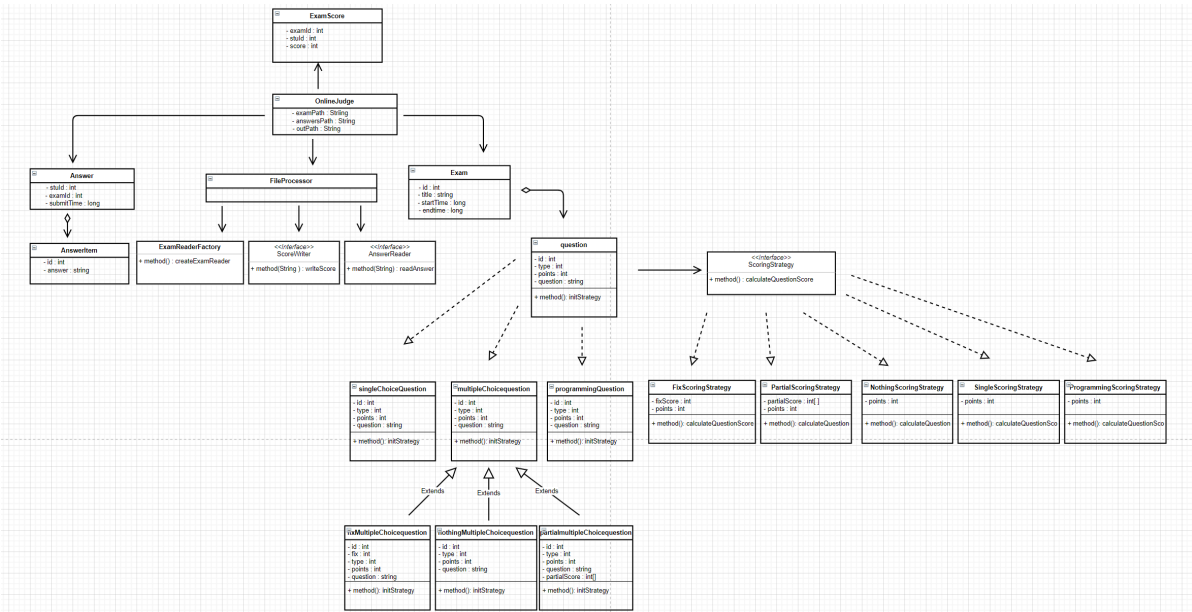
设计文档

211870287 丁旭

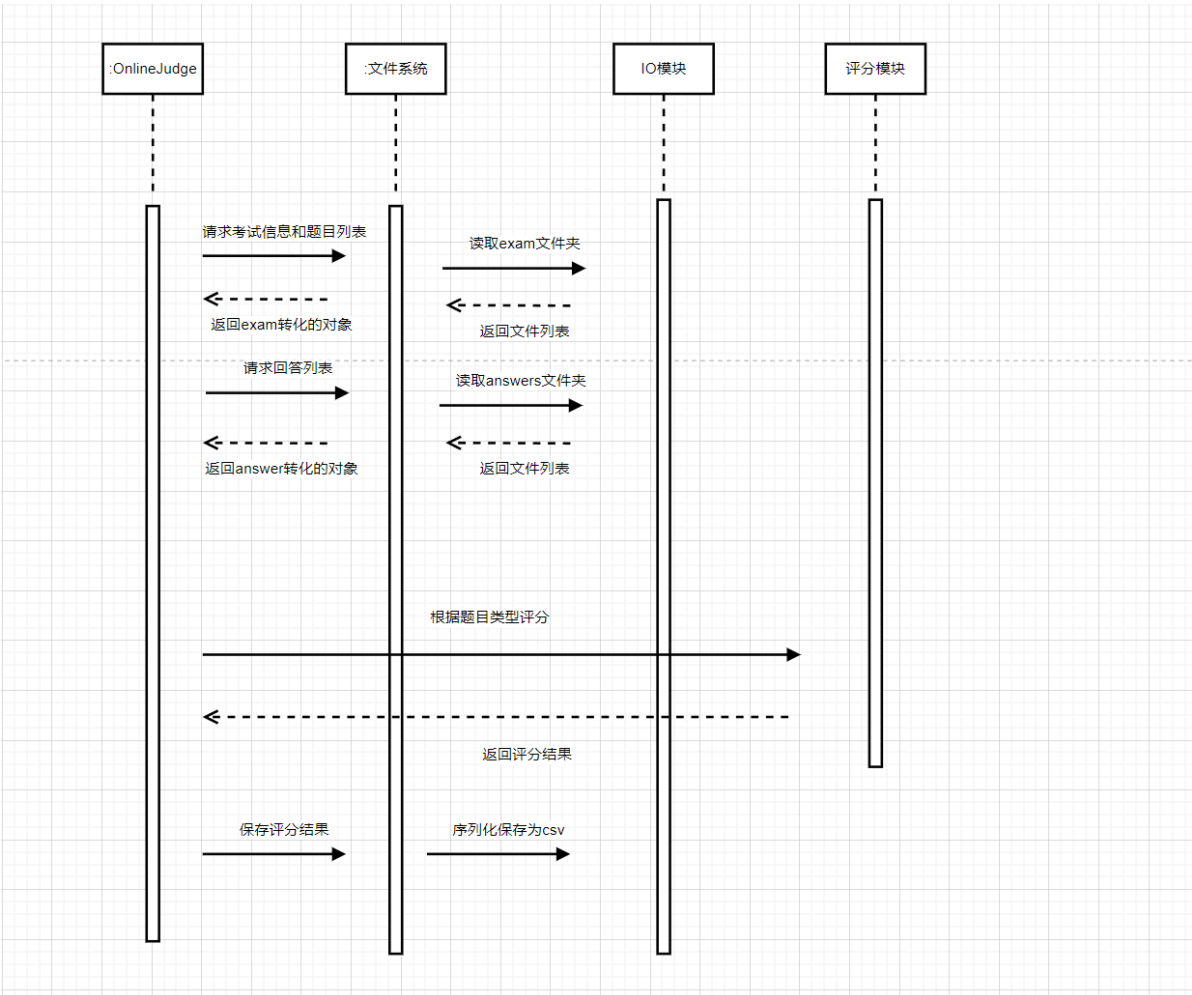
代码目录



类图



流程图



设计原则

☑ 单一职责

如ExamScore类，仅作为对csv文件的映射，将每一个学生的成绩保存在表中，作为一个实体类即成绩单

```

@Data
public class ExamScore {
    2 usages
    @JsonProperty(index = 0)
    private int examId;
    2 usages
    @JsonProperty(index = 1)
    private int stuId;
    2 usages
    @JsonProperty(index = 2)
    private int score;

    // 构造函数
    2 usages  211870287
    public ExamScore(int examId, int stuId, int score) {
        this.examId = examId;
        this.stuId = stuId;
        this.score = score;
    }

    211870287
    @Override
    public String toString() {
        return "ExamScore{" +
            "examId=" + examId + '\n' +
            ", stuId=" + stuId + '\n' +
            ", score=" + score +
            '}';
    }
}

```

☑ 开闭原则

使用抽象类question作为所有子类的父类，包含最基本的属性id, type, question, points, scoringStrategy（对问题的计算策略，这里的scoringStrategy是一个接口，根据不同question的计分策略按需实现）

```

@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage 5 overrides 211870287
    public void initStrategy(){}

    6 overrides 211870287
    @Override
    public String toString() {...}
}

```

☑ 里氏代换原则

这里使用到Question父类进行计分策略的初始化，可以用其任意子类来进行各自的计分策略的初始化。

```

1 usage 211870287
private void initQuestionScoreStrategy(){
    for (Exam exam:idToExamMap.values()){
        exam.getQuestions().forEach(Question::initStrategy);
    }
}

```

☑ 依赖倒转原则

Exam依赖于抽象类的列表List,而非具体的 多选题、单选题、编程题

```

@Data
public class Exam {
    1 usage
    private int id;
    1 usage
    private String title;
    1 usage
    private long startTime;
    1 usage
    private long endTime;

    1 usage
    private List<Question> questions;

    211870287
    @Override
    public String toString() {...}
}

```

☑ 接口隔离原则

将io处理拆分成三个细分的专业接口

1. 满足**单一职责原则**，将一组相关的操作定义在一个接口中，且在满足高内聚的前提下，接口中的方法越少越好。
2. 可以在进行系统设计时采用**定制服务**的方式，即为**不同的客户端提供宽窄不同的接口**

```

2 usages 1 implementation 211870287
public interface AnswerReader {
    1 implementation 211870287
    Answer readAnswer(String filePath) throws IOException;
}

```

```

8 usages 2 implementations 211870287
public interface ExamReader {
    3 usages 2 implementations 211870287
    Exam readExam(String filePath) throws IOException;
}

```

```

8 usages 2 implementations 211870287
public interface ExamReader {
    3 usages 2 implementations 211870287
    Exam readExam(String filePath) throws IOException;
}

```

☑ 合成复用原则

Question类组合ScoringStrategy，组合复用以使其可以黑箱复用

```
@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage 5 overrides ⓘ 211870287
    public void initStrategy(){}

    6 overrides ⓘ 211870287
    @Override
    public String toString() {...}
}
```

✅ 迪米特法则

在设计的过程中，尽可能的减少了类之间的不必要的通信，使用抽象工厂和接口来减少不同类之间的通信

```
public class FileProcessor {
    2 usages
    private ExamReaderFactory examReaderFactory = new ExamReaderFactory();
    2 usages
    private ScoreWriter scoreWriter;
    2 usages
    private AnswerReader answerReader;

    1 usage ⓘ 211870287
    public Map<Integer, Exam> readExam(String examsPath) throws IOException {...}

    1 usage ⓘ 211870287
    public void saveExamScoreList(String outputPath, List<ExamScore> scoreList) throws IOException {...}

    1 usage ⓘ 211870287
    public List<Answer> readAnswer(String answersPath) throws IOException {...}
}
```

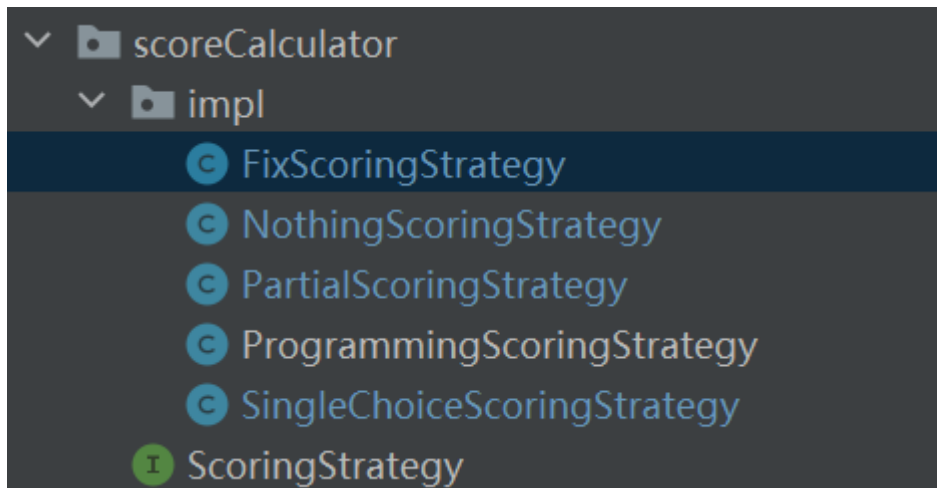
设计模式：

策略模式

定义接口

```
public interface ScoringStrategy {  
    1 usage 5 implementations 211870287  
    public int calculateQuestionScore(Object self_writtenAnswer);  
}
```

实现不同的计分策略



再由对应的Question组合复用


```

public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage 5 overrides 211870287
    public void initStrategy(){}

    6 overrides 211870287
    @Override
    public String toString() {...}
}

```

工厂模式

设计读取exam的简单工厂

```

public class ExamReaderFactory {
    2 usages 211870287
    public static ExamReader createExamReader(String fileExtension) {
        switch (fileExtension) {
            case "json":
                return new JsonExamReader();
            case "xml":
                return new XmlExamReader();
            default:
                throw new IllegalArgumentException("Unsupported file type: " + fileExtension);
        }
    }
}

```

根据不同的参数来获取不同的“产品”——对应的reader类

接口设计

```
public interface AnswerReader {  
    public Answer readAnswer(String filePath) throws IOException;  
    public Answer readAnswer(File jsonFile) throws IOException;  
}
```

```
public interface ExamReader {  
    Exam readExam(String filePath) throws IOException;  
}
```

```
public interface ScoreWriter {  
    void writeScore(String filePath) throws IOException;  
}
```

```
public interface ScoringStrategy {  
    public int calculateQuestionScore(Object self_writtenAnswer);  
}
```

抽象类设计

```
public abstract class Question<T> {  
  
    private int id;  
    private int type;  
    private String question;  
    private int points;  
  
    private ScoringStrategy scoringStrategy;  
  
    public void initStrategy(){}  
  
    @Override  
    public String toString() {  
        return "Question{" +  
            "id=" + id +  
            ", type=" + type +  
            ", question='" + question + '\'' +  
            ", points=" + points +  
            '}';  
    }  
}
```

实体设计

```
public class Answer {  
    @JsonProperty("examId")  
    private int examId;  
  
    @JsonProperty("stuId")  
    private int studentId;
```

```

@JsonProperty("submitTime")
private long submitTime;

@JsonProperty("answers")
private List<AnswerItem> answers;

@Override
public String toString() {
    return "Answer{" +
        "\nexamId=" + examId +
        "\nstuId=" + studentId +
        "\nsubmitTime=" + submitTime +
        "\nanswers=" + answers +
        "\n}";
}
}

```

```

public class AnswerItem {
    private int id;
    private String answer;
}

```

```

public class Exam {
    private int id;
    private String title;
    private long startTime;
    private long endTime;

    private List<Question> questions;

    @Override
    public String toString() {
        return "Exam{\n" +
            "id=" + id +
            "\ntitle='" + title + '\'' +
            "\nstartTime=" + startTime +
            "\nendTime=" + endTime +
            "\nquestions=\n" + questions +
            '}';
    }
}

```

```

public class ExamScore {
    @JsonProperty(index = 0)
    private int examId;
    @JsonProperty(index = 1)

```

```

private int stuId;
@JsonProperty(index = 2)
private int score;

// 构造函数
public ExamScore(int examId, int stuId, int score) {
    this.examId = examId;
    this.stuId = stuId;
    this.score = score;
}

@Override
public String toString() {
    return "ExamScore{" +
        "examId='" + examId + '\'' +
        ", stuId='" + stuId + '\'' +
        ", score=" + score +
        '}';
}
}

```

功能演示

题目读取

使用Jackson库中的注解来实现Java对象与JSON之间的序列化和反序列化

1. `@JsonTypeInfo`: 该注解用于指定在序列化和反序列化过程中如何处理类型信息。
 - `use = JsonTypeInfo.Id.NAME`: 指定类型信息将作为名称的一部分进行处理。
 - `property = "type"`: 指定在JSON中表示类型的属性名称。
2. `@JsonSubTypes`: 该注解用于定义多态类型的子类型。
 - `@JsonSubTypes.Type`: 用于指定一个子类型及其对应的名称。
 - `value`: 指定子类型的类。
 - `name`: 指定子类型在JSON中的名称。

使用Jackson库读取xml文件或者json文件自动转化对其对应的子类实体

1. `@JsonTypeInfo` 和 `@JsonSubTypes` 组合用于处理问题类型（SingleChoiceQuestion、MultipleChoiceQuestion、ProgrammingQuestion）的多态性。根据属性type的不同值"1"、"2"、"3"来确定问题的类型。

```

@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    property = "type")
@JsonSubTypes({
    @JsonSubTypes.Type(value = SingleChoiceQuestion.class, name = "1"),
    @JsonSubTypes.Type(value = MultipleChoiceQuestion.class, name = "2"),
    @JsonSubTypes.Type(value = ProgrammingQuestion.class, name = "3")
})
@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage 5 overrides 211870287
    public void initStrategy(){}

    6 overrides 211870287
    @Override
    public String toString() {...}
}

```

2. @JsonTypeInfo 和 @JsonSubTypes 组合用于处理多选题的评分模式

(MultipleChoiceQuestionFix、MultipleChoiceQuestionPartial、MultipleChoiceQuestionNothing) 的多态性。根据scoreMode属性的不同值"fix"、"partial"、"nothing"来确定多选题的评分模式。

```

@JsonTypeInfo(
    use = JsonTypeInfo.Id.NAME,
    property = "scoreMode")
@JsonSubTypes({
    @JsonSubTypes.Type(value = MultipleChoiceQuestionFix.class, name = "fix"),
    @JsonSubTypes.Type(value = MultipleChoiceQuestionPartial.class, name = "partial"),
    @JsonSubTypes.Type(value = MultipleChoiceQuestionNothing.class, name = "nothing"),
})
}

@Data
public class MultipleChoiceQuestion extends Question {
    1 usage
    private String scoreMode;
    // @JacksonXmlProperty(localName = "option")
    // @JacksonXmlElementWrapper(useWrapping = false)
    1 usage
    private List<String> options;
    1 usage
    @JsonProperty("answer")
    @JsonAlias("answers")
    private List<Integer> answer;

    3 usages 211870287
    public MultipleChoiceQuestion() { this.setType(2); }

```

3.读取文件

```

public void init(String examsPath, String answersPath ) throws IOException {
    this.answerList = fileProcessor.readAnswer(answersPath);
    this.idToExamMap = fileProcessor.readExam(examsPath);
    initQuestionScoreStrategy();
}

```

评分

遍历answers的列表，找出其对应的exam，取出question列表

通过调用question的scoringStrategy属性的calculateQuestionScore方法计算每一个回答的得分

将得分累加结算每一份回答的最终得分并保持在examScoringList中

```

public List<ExamScore> calculateExamScoreList(){
    // 使用 Lambda 表达式
    answerList.forEach(item -> {
        // 在这里对每个元素执行操作
        Exam exam = idToExamMap.get(item.getExamId());
        //判断是不是超过提交时间或者提前交
        long submit_time = item.getSubmitTime();

        if (exam.getStartTime() > submit_time || exam.getEndTime() < submit_time){
            ExamScore examScore = new ExamScore(item.getExamId(),item.getStudentId(), score: 0);
            examScoreList.add(examScore);
            return;
        }

        List<Question>questionList = exam.getQuestions();
        List<AnswerItem> answers = item.getAnswers();

        int score = calculateExamScore(questionList,answers);
        ExamScore examScore = new ExamScore(item.getExamId(),item.getStudentId(),score);
        examScoreList.add(examScore);
    });
    return examScoreList;
}

1 usage 211870287
public int calculateExamScore(List<Question>questionList,List<AnswerItem> answers){
    int totalScore = 0;
    System.out.println(questionList.size()+"//"+answers.size());
    for (int i = 0; i < questionList.size(); i++) {
        Question question = questionList.get(i);
        if (question.getType() == 3){
            totalScore += question.getPoints();
            continue;
        }
        System.out.println(question.toString());
        String answer = answers.get(i).getAnswer();

        totalScore += question.getScoringStrategy().calculateQuestionScore(answer);
    }
    return totalScore;
}

```

持久化

```

public void saveScoreList(String outputPath) throws IOException {
    fileProcessor.saveExamScoreList(outputPath,examScoreList);
}

```

将其存储为csv文件

```
1 examId,stuId,score
2 1,1,100
3 1,2,0
4 1,3,0
5 2,1,95
6 2,2,70
7 2,3,90
8 3,1,100
9 3,2,90
10 3,3,70
11 4,1,70
12 4,2,80
13 4,3,90
14 |
```