目录

[1 架构设计 2](#_Toc164330114)

[1.1 整体架构 2](#_Toc164330115)

[1.2 子系统划分 2](#_Toc164330116)

[2 前端子系统模块设计 2](#_Toc164330117)

[2.1 组件设计 2](#_Toc164330118)

[2.2 状态管理和数据流设计 3](#_Toc164330119)

[3 后端子系统模块设计 4](#_Toc164330120)

[3.1 教师业务模块 4](#_Toc164330121)

[3.2 学生业务模块 4](#_Toc164330122)

[3.2.1 接口设计 4](#_Toc164330123)

[3.2.2 主要类 4](#_Toc164330124)

[3.3 判题模块 4](#_Toc164330125)

[3.3.1 设计模式 4](#_Toc164330126)

[3.3.2 主要类 5](#_Toc164330127)

[4 数据库设计 5](#_Toc164330128)

[4.1 ER图 6](#_Toc164330129)

[4.1.1 设计描述 6](#_Toc164330130)

[5 界面设计 8](#_Toc164330131)

[6 算法设计 8](#_Toc164330132)

[6.1 判题算法流程 8](#_Toc164330133)

[6.2 相似度算法类型 8](#_Toc164330134)

[6.2.1 史密斯-沃特曼算法（Smith-Waterman algorithm） 8](#_Toc164330135)

[6.2.2 树编辑距离（Tree Edit Distance） 9](#_Toc164330136)

[6.2.3 树匹配算法（Tree Matching Algorithm） 9](#_Toc164330137)

[6.2.4 树核算法（Tree Kernel） 9](#_Toc164330138)

[6.2.5 树相似性度量（Tree Similarity Metrics） 9](#_Toc164330139)

[6.2.6 树图匹配算法（Tree Graph Matching Algorithm） 9](#_Toc164330140)

[6.3 算法选择及依据 9](#_Toc164330141)

[7 运行环境 9](#_Toc164330142)

1. 架构设计
   1. 整体架构

基于MVC实现前后端分离，具体实现采用SpringBoot集成SpringMVC。

* 1. 子系统划分

前端系统：前端系统负责展示用户界面，提供用户操作图形接口。

后端系统：后端负责具体的业务处理，提供相关接口供前端调用。

通信机制：前后端通过restful风格的接口进行通信，前后端面向统一的接口编程。

数据流：前端接收用户输入，调用相应后端接口，后端接收数据，调用相应接口，处理后返回数据，前端界面展示结果。

1. 前端子系统模块设计
   1. 组件设计

LoginPage 登录页面组件：负责用户登录功能，包括输入用户名、密码，点击登录按钮等。

MyAside 侧边栏组件：显示应用程序的侧边栏菜单，提供导航到不同页面或功能的选项。

MyHeader 顶部栏组件：显示应用程序的顶部栏，包括应用名称、用户信息、注销按钮等。

MyHomePage 主页面组件：包含侧边栏和顶部栏，作为应用程序的主要页面容器。根据侧边栏选项的不同，动态加载当前选中的题目列表、提交记录或班级列表组件。

MyProblem 题目列表组件：显示题目列表，可能包括题目名称、题目描述等信息。提供点击题目跳转到题目详情页面的功能。

MyCommit 提交记录组件：显示用户的提交记录，包括提交时间、提交状态、评测结果等信息。可能提供筛选、排序等功能，方便用户查找和管理提交记录。

MyClasses 班级列表组件：显示教师管理的班级列表，包括班级名称、班级成员等信息。提供点击班级跳转到班级详情页面的功能。

* 1. 状态管理和数据流设计

在 Vue.js 应用程序中，状态管理和数据流设计是至关重要的方面。为了实现更好的可维护性、可扩展性和可测试性，我们选择采用 Vuex 这个专为 Vue.js 应用程序设计的状态管理模式。Vuex 提供了一种集中式的状态管理方案，可以更好地管理应用程序中的各种状态，并确保不同组件之间的状态同步和一致性。

使用 Vuex，我们可以将应用程序的状态（state）、动作（actions）、突变（mutations）和模块（modules）进行统一管理。具体而言：

**状态（State）：** 我们可以将应用程序中的所有数据状态统一存储在 Vuex 的状态树中。这包括用户信息、应用程序配置、页面状态等。通过定义清晰的状态结构，我们可以更方便地管理和访问应用程序的数据。

**动作（Actions）：** 动作用于触发状态的变化，可以执行异步操作、提交突变或调用其他动作。在 Vuex 中，我们可以定义各种动作来处理用户交互、异步请求等操作，从而实现复杂的状态管理逻辑。

**突变（Mutations）： 突变用于直接修改状态，但是只能同步地执行。通过提交突变，我们可以确保状态的变化是可追踪和可控的，避免了直接在组件中修改状态带来的问题。**

**模块（Modules）： 为了提高代码的组织性和可维护性，我们可以将应用程序的状态分解为多个模块，并在 Vuex 中进行统一管理。每个模块可以拥有自己的状态、动作、突变等，从而实现状态的模块化管理。**

1. 后端子系统模块设计
   1. 教师业务模块
   2. 学生业务模块

应用MVC分层架构，前端请求首先根据接口URL路径到达相应的学生业务controller，再根据URL调用相应的业务方法，再调用service层的对应service处理具体业务逻辑，如果具体业务需要调用mapper层的数据库调用接口完成相应数据库。

* + 1. 接口设计

请查看后端API接口文档的学生管理部分

* + 1. 主要类

StudentController：负责提供接口接收学生业务请求，根据URL调用对应方法。

StudentService：提供接口供StudentController调用，封装具体业务逻辑。

StudentMapper：提供学生管理业务相关数据库接口，完成数据库操作。

* 1. 判题模块
     1. 设计模式

采用策略模式实现根据前端传入的判题类型调用对应的判题类，实现插件调用。

采用工厂模式实现判题类的创建，使得controller无需关心插件创建细节。

采用饿汉单例模式创建插件和具体工厂类。

采用模板方法模式设计判题流程。

* + 1. 主要类

**JudgeService接口**——提供判题相关接口，一层统一的抽象。

**JavaJudgeService实现类**——实现Java代码相关判题接口。

**PythonJudgeService实现类**——实现Python代码相关判题接口。

**ShellJudgeService实现类**——实现Shell代码相关判题接口。

**JudgeServiceCreator抽象类**——提供创建判题类的统一工厂方法接口，提供创建具体JudgeServiceCreator类的静态方法。

**JavaJudgeServiceCreator实现类**——实现创建JavaJudgeService的工厂方法。

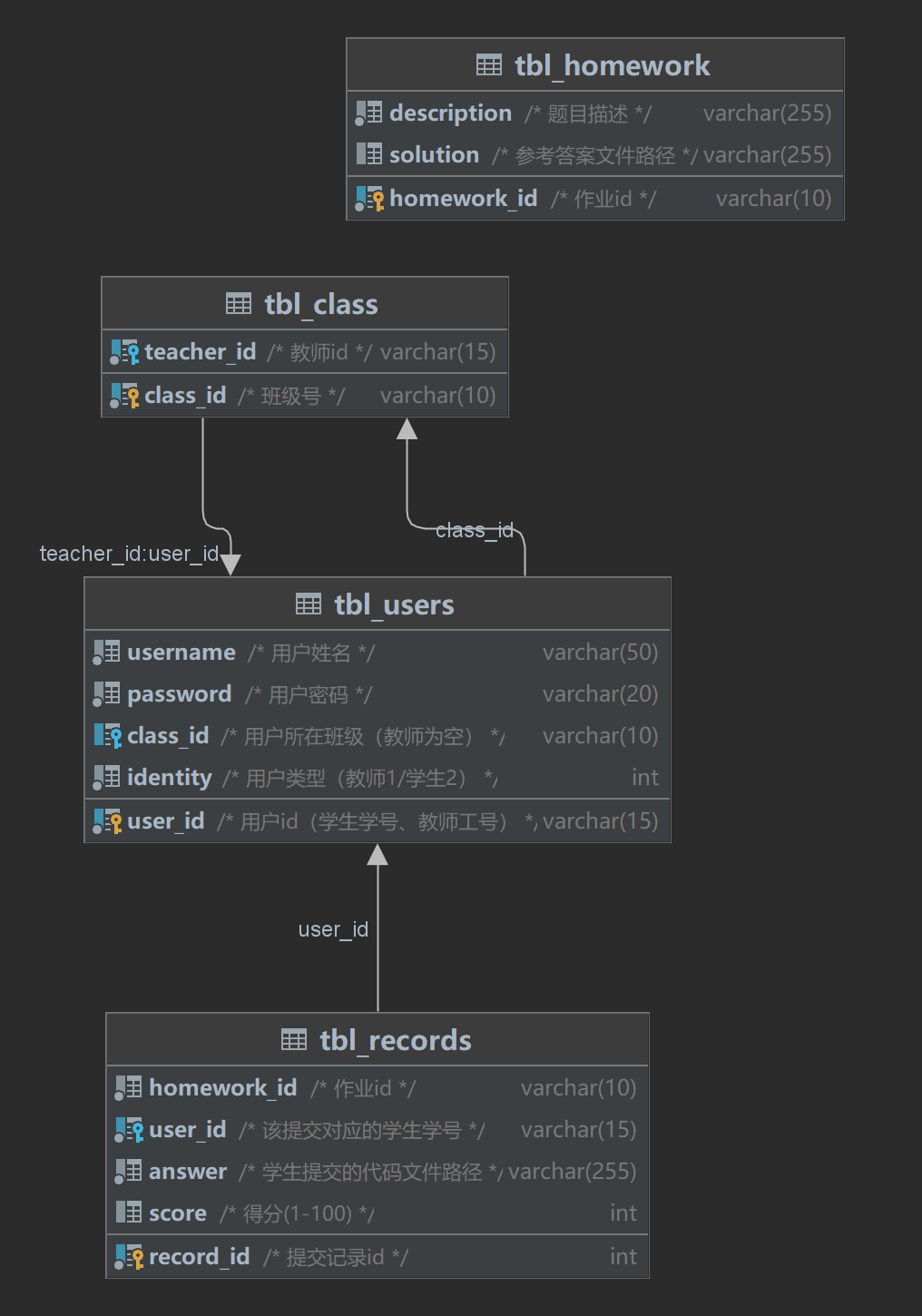
**PythonJudgeServiceCreator实现类**——实现创建PythonJudgeService的工厂方法。

**ShellJudgeServiceCreator实现类**——实现创建ShellJudgeService的工厂方法。

1. 数据库设计

如果软件系统涉及数据库，需要详细描述数据库的结构、表设计、关系模式等。

* 1. ER图



图表 1 ER图

* + 1. 设计描述

这个数据库包含了四张表：tbl\_records、tbl\_users、tbl\_class 和 tbl\_homework。

* + - 1. tbl\_records

record\_id（提交记录id）：主键，唯一标识每一次提交记录。

homework\_id（作业id）：外键，与 tbl\_homework 表中的作业id关联，表示提交的作业。

user\_id（用户id）：外键，与 tbl\_users 表中的用户id关联，表示提交作业的学生学号。

answer（学生提交的代码文件路径）：保存学生提交的代码文件路径。

score（得分）：保存该次提交的得分，取值范围为1到100之间。

表中有一个索引 user\_id，用于加速根据学生学号查询提交记录的操作。

有一个外键约束，关联到 tbl\_users 表的 user\_id 字段，确保提交记录的 user\_id 是有效的学生学号。

* + - 1. tbl\_users

user\_id（用户id）：主键，唯一标识用户，可以是学生学号或教师工号。

username（用户姓名）：保存用户的姓名。

password（用户密码）：保存用户的登录密码。

class\_id（用户所在班级）：外键，与 tbl\_class 表中的班级号关联，表示用户所在的班级。

identity（用户类型）：表示用户的身份，1 表示教师，2 表示学生。

表中有一个索引 class\_id，用于加速根据班级号查询用户的操作。

有一个外键约束，关联到 tbl\_class 表的 class\_id 字段，确保用户所在班级的 class\_id 是有效的班级号。

* + - 1. tbl\_class

class\_id（班级号）：主键，唯一标识班级。

teacher\_id（教师id）：外键，与 tbl\_users 表中的教师id关联，表示该班级的班主任教师。

有一个外键约束，关联到 tbl\_users 表的 teacher\_id 字段，确保班级的教师id是有效的教师工号。

* + - 1. tbl\_homework

homework\_id（作业id）：主键，唯一标识作业。

description（题目描述）：保存作业的描述信息。

solution（参考答案文件路径）：保存作业的参考答案文件路径。

* + - 1. 表关系描述

tbl\_records 表与 tbl\_users 表之间是一对多关系，一个用户可以有多条提交记录，但每条提交记录只属于一个用户。

tbl\_records 表与 tbl\_homework 表之间是一对一关系，一条提交记录只对应一个作业，而一个作业可以有多条提交记录。

tbl\_users 表与 tbl\_class 表之间是一对多关系，一个班级可以有多个学生和一个教师，但一个用户只属于一个班级。

tbl\_class 表与 tbl\_users 表之间是一对多关系，一个教师可以管理多个班级，但一个班级只有一个教师。

这种数据库设计能够有效地表示学生、教师、班级和作业之间的关系，并支持学生提交作业和教师管理班级的功能。

1. 界面设计

参考用户界面原型设计文档。

1. 算法设计
   1. 判题算法流程

解析源代码生成抽象语法树-->采用相似度算法计算提交代码与正确代码的抽象语法树的相似度->解析抽象语法树，给出相应提示->返回判题结果

* 1. 相似度算法类型

关键点、关键路径

* + 1. 史密斯-沃特曼算法（Smith-Waterman algorithm）

一种进行局部序列比对（相对于全局比对）的算法，用于找出两个核苷酸序列或蛋白质序列之间的相似区域。该算法的目的不是进行全序列的比对，而是找出两个序列中具有高相似度的片段。

* + 1. 树编辑距离（Tree Edit Distance）

类似于编辑距离，用于比较两棵树之间的相似度。它通过计算将一棵树转换为另一棵树所需的最小编辑操作（插入、删除、替换节点）来衡量它们之间的相似度。

* + 1. 树匹配算法（Tree Matching Algorithm）

这些算法尝试找到两棵树之间的最大子树或子结构的匹配，然后根据匹配的结果来计算相似度。常见的树匹配算法包括最大公共子树算法、最长公共子序列算法等。

* + 1. 树核算法（Tree Kernel）

这些算法将树映射到高维特征空间中，并通过计算特征空间中的内积来衡量树之间的相似度。树核算法通常基于子树或者子结构的出现频率来计算相似度。

* + 1. 树相似性度量（Tree Similarity Metrics）

这些度量方法将树映射到一个特征向量空间，并使用距离度量来计算两个树之间的相似度。常见的树相似性度量包括树编辑距离、树结构相似度等。

* + 1. 树图匹配算法（Tree Graph Matching Algorithm）

将树转换为图，并利用图匹配算法来比较两个树之间的相似度。这些算法通常使用图匹配算法来找到树之间的最大匹配子图，然后根据匹配子图的大小来衡量相似度。

* 1. 算法选择及依据

1. 运行环境

数据库——MySQL8.0.36

JDK版本——jbr-17.0.9

SpringBoot版本——2.7.3

JavaParser版本——3.24.2

"core-js": "^3.8.3",

"element-plus": "^2.6.3",

"vue": "^3.2.13",

"vue-router": "^4.3.0",

"vuex": "^4.1.0"