

Java课程期末作业报告——教务系统设计

项目背景介绍

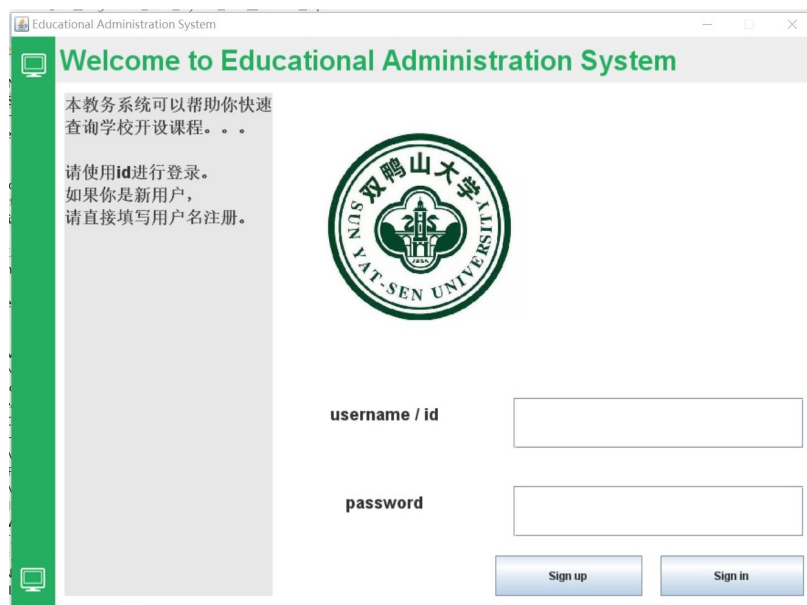
我们主要针对学生这一特定人群，设计一个信息管理系统，即我们常用的教务系统。教务系统对学生来说，就是日常生活中离不开的物品，选课、查阅信息等，对我们的学习生活发挥了很大的作用。既然教务系统对我们如此重要，我们小组就利用Java技术实现一个简单的教务系统。

系统功能介绍

通过教务系统我们可以方便地管理我们的信息，并可以查询相应的课程信息、成绩等。通过Java的GUI设计，我们的系统具有相对人性的系统界面。


登录界面

我们通过应用一开始的登录界面，可以进行注册用户还有登录用户两项功能。当然，如果登录时，出现密码错误或者用户不存在的情况，我们也会作出相应的提示。



个人信息修改

当然，完成注册以后，我们还可以根据需要重新修改我们的个人信息。

 modifyInfo

而且通过表格的方式呈现我们的选课信息，我们可以很方便地查询、排序、筛选过滤我们所需的信息。

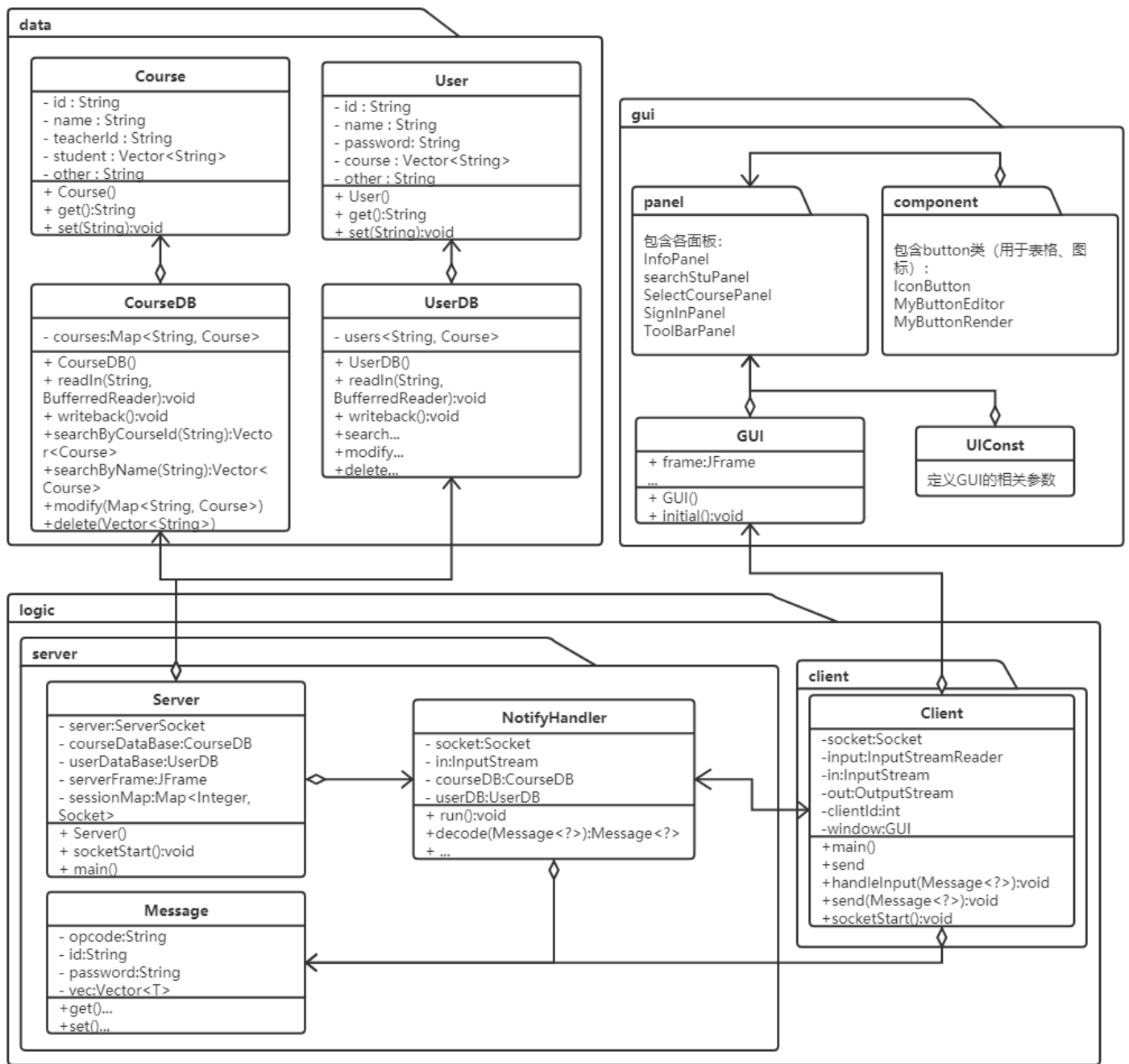
查询课程

与上面的登录密码校验不同，我们设计的教务系统实现的查询主要创新点在于，我们设计其支持模糊查询的功能，我们只需输入两三个字符，我们的课程信息表单就可以显示出对应的符合要求的课程信息。



由上面的图片可知，我们所设计的模糊查询，只要课程中有相应的关键字，即可查询到对应的信息。

系统类图



关键模块说明

我们的分成源码、数据、以及外部资源和程序截图等。
data文件夹用于持久化存储数据，使用了文件存储技术。

```

v 📁 > edu-admin-system [edu-admin-system master]
  v 📁 src
    > 📁 data
    > 📁 gui
    > 📁 logic
  > 📚 JRE System Library [jre1.8.0_231]
  v 📁 data
    📄 courseDB.txt
    📄 userDB.txt
  v 📁 resource
    > 📁 icon
    📁 screenshot
    📄 course.txt
    📄 Java课程设计_2019.pdf
    📄 LICENSE
    📄 README.md
    📄 Report.md

```

我们的项目设计主要使用了三层架构的设计模式。即，界面层（User Interface layer）、业务逻辑层（Business Logic Layer）、数据访问层（Data access layer）。这样的设计方式旨在降低程序之间的耦合度，方便我们的修改。

```

v 📁 > edu-admin-system [edu-admin-system master]
  v 📁 src
    v 📁 data
      > 📁 users
      > 📄 Course.java
      > 📄 CourseDB.java
      > 📄 User.java
      > 📄 UserDB.java
    v 📁 gui
      > 📁 component
      > 📁 panel
      > 📄 GUI.java
      > 📄 UIConst.java
    v 📁 logic
      > 📁 client
      > 📁 server
      > 📄 Configuration.java

```

数据层类

数据层的类有四个 `Course`、`CourseDB`、`User`、`UserDB`。

通过上面的系统类图可以很清晰地看出他们之间的关系。










`User` 中可以访问该用户已经选择的课程，`Course` 中也可以查看选择该课程的用户。

而数据库（包含 `UserDB` 和 `CourseDB`）存储整个系统中含有的所有用户和所有的课程。两个数据库都提供了相当多的接口，用于和逻辑层直接交互。

```


v 📄 CourseDB.java
  v 🏗️ CourseDB
    ▪ courses
    🟢 CourseDB()
    🔗 delete(Vector<String>) : void
    🔗 modify(Map<String, Course>) : void
    🟢 readln(String, BufferedReader) : void
    🔗 searchByCourseId(String) : Vector<Course>
    🔗 searchByName(String) : Vector<Course>
    🟢 writeBack() : void

```

- ▼  UserDB.java
 - ▼  UserDB
 -  pathname
 -  users
 -  UserDB()
 -  delete(Vector<String>) : boolean
 -  getNewId() : String
 -  importUser(String, BufferedReader) : void
 -  isMatched(String, String) : boolean
 -  modify(Map<String, User>) : boolean
 -  pwdMatched(String, String) : boolean
 -  searchById(String) : Vector<User>
 -  searchByName(String) : Vector<User>
 -  writeBack() : void

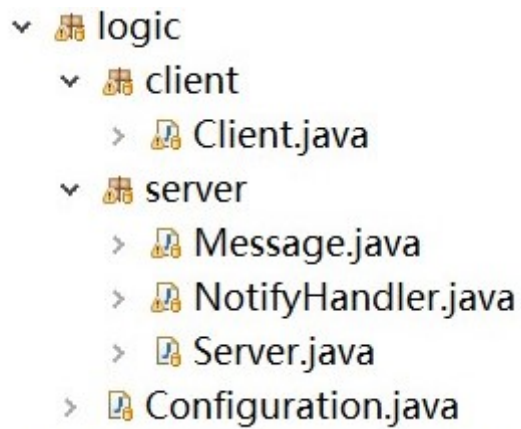
界面层类

界面层类里主要有 GUI 一个主类，UIConst 主要记下 GUI 界面所用到的相关设置。gui 内有一个名为 panel 的包，存储我们所需要跳转的相关页面；还有一个 component 的包，存储为实现一些特定功能的而特别设计的组件。

- ▼  gui
 - ▼  component
 - >  IconButton.java
 - >  MyButtonEditor.java
 - >  MyButtonRender.java
 - ▼  panel
 - >  InfoPanel.java
 - >  SelectCoursePanel.java
 - >  SignInPanel.java
 - >  ToolBarPanel.java
 - >  GUI.java
 - >  UIConst.java

业务逻辑层类

业务逻辑层主要实现了网络编程的相关功能。logic 包内只有一个文件 Configuration，它提供地址服务器的地址和端口，用于客户端和服务端之间进行沟通。



```

  ▾ logic
    ▾ client
      > Client.java
    ▾ server
      > Message.java
      > NotifyHandler.java
      > Server.java
    > Configuration.java

```

客户端

客户端只有相应的 `Client` 类，主要作用为创建 `socket` 并进行相应的接收、处理、发送信息，与服务器进行交互，并根据服务器发来的信息调用 `GUI` 的方法，对 `GUI` 进行调整。

 UMLClient

服务器端

`server` 包下有三个类：

1. `Server` 类：用于运行服务器。新建客户端时，`Server` 都新建一个线程 `NotifyHandler`，用于监听客户端的信息。
2. `Message` 类：主要用于信息传递。

```

v 📄 Message.java
  v 🧑 Message<T>
    ▫ id
    ▫ opcode
    ▫ password
    ▫ vec
    📄 Message()
    📄 Message(String, String, String)
    📄 Message(String, String, String, Vector<T>)
    📄 Message(String, Vector<T>)
    ● getId() : String
    ● getOpcode() : String
    ● getPassword() : String
    ● getVec() : Vector<T>
    ● setId(String) : void
    ● setOpcode(String) : void
    ● setPassword(String) : void
    ● setVec(Vector<T>) : void

```

比较特别是，`Message` 类为了实现能够传递不同类型的信息。我们将其设计为泛型设计，并通过 `Vector` 实现能够传递任意数量的变量。例如，要进行注册的操作，我们只需将 `T` 设为 `String`，将用户名和密码压入 `Vector` 中即可传给服务器进行处理。

3. `NotifyHandler` 类：用于识别、响应处理客户端发来的请求。进行 `UserDB` 和 `CourseDB` 之间的交互。


```
▼ 📄 NotifyHandler.java
  ▼ 📄 NotifyHandler
    ▫ courseDB
    ▫ in
    ▫ socket
    ▫ userDB
    📄 NotifyHandler(Socket, UserDB, CourseDB)
    ● decode(Message<?>) : Message<?>
    ● deleteCourse(Message<String>) : Message<String>
    ■ deleteUser(Message<String>) : Message<String>
    ● exit(Message<?>) : Message<?>
    ● login(Message<String>) : Message<Integer>
    ● modifyCourse(Message<Course>) : Message<String>
    ■ modifyUser(Message<User>) : Message<String>
    ● other() : void
    ● register(Message<String>) : Message<Integer>
    📄 run() : void
    ● searchCourseById(Message<String>) : Message<Course>
    ● searchCourseByName(Message<String>) : Message<Course>
    ■ searchUserById(Message<String>) : Message<User>
    ■ searchUserByName(Message<String>) : Message<User>
```

知识点应用说明

1. 面向对象编程知识点：

○ 类和对象

使用JAVA进行编程的时候，我们大多使用的是面向对象的编程方式，离不开使用类和对象。例如数据层使用了不同的类，存储用户和课程的信息。

○ 接口及其实现

JAVA提供了很多组件供我们实现图形化界面，但仅靠默认提供的API，很可能无法实现我们所需要的需求。因此，我们对GUI的很多组件进行了一定接口的实现，在前人的工作基础上进行拓展。

```

1 package gui.component;
2
3
4 import java.awt.Component;
10
11 public class MyButtonRender implements TableCellRenderer
12 {
13     private JPanel panel;
14
15     private JButton button;
16

```

例如，上图所展示的 `MyButtonRender`，实现了 `TableCellRenderer` 的接口，这让我们能在表格中添加按钮。

- 超类与继承

父类的继承和接口的实现算是同样道理，我们项目同样也继承了一些父类。

```

1 package gui.component;
2
3 import java.awt.Color;
17
18 /**
19  * 自定义一个往列里边添加按钮的单元格编辑器。最好继承DefaultCellEditor，不然要实现的方法就太多了。
20  *
21  */
22 public class MyButtonEditor extends DefaultCellEditor
23 {
24
25     /**
26      * serialVersionUID
27      */
28     private static final long serialVersionUID = -6546334664166791132L;
29
30     private JPanel panel;
31
32     private JButton button;
33
34     private JTable table;
35

```

如，上图中的 `MyButtonEditor`，能让我们定义点击表格中的按钮，对应的渲染行为。

- 网络编程

信息管理系统离不开网络编程，离不开客户端和服务端之间的信息交流。因此，我们设计相应的客户端和服务端，实现了两套系统相应的行为。

- 多线程

不同的客户端连接到服务器时，服务器需要根据不同的客户端进行相应的响应。因此，服务器对每一个客户端都建立了一个线程，处理其发出的请求。

```

while (true) {
    Socket socket = server.accept();
    if(socket != null) {
        System.out.println("Connect to client!");
        sessionMap.put(i, socket);

        ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
        oos.writeObject(i);

        Thread thread = new Thread(new NotifyHandler(socket, userDataBase, courseDataBase));
        thread.setDaemon(true);
        thread.start();
        i++;
    }
}

```

- 异常处理

当客户端断开和服务器的连接时，会产生断开连接的异常。此时，我们的服务器可以根据这个异常进行相应的处理。

- 文件存储

作为信息管理系统，我们实现了数据的持久化操作。关闭服务器时，我们会调用数据写回的函数，将数据写到服务器的本地磁盘中。

2. 扩展知识点（鼓励挑战）：

- Java 图形界面

关于图形界面的设计，上面已经有所介绍。实现出的效果如下：

- Java 设计模式

项目的设计模式使用的是三层架构。

创新点、技术难点说明

1. 项目的难点主要是为了提高用户使用的便利性，要向很多组件添加一些细节行为。例如：

- 没有登录的时候，我们实现了：对工具栏的按钮灭活，此时无法对页面跳转。
- 查看个人信息，实现对密码的隐藏，以及对用户信息的实时修改。

2. 表格操作

我们的信息主要通过表格进行呈现。因此，我们对表格的操作增加了相应的操作。如：

- 通过点击表头，我们可以对表格进行排序显示。
- 通过某些关键字，我们可以对相应的表格信息进行模糊查询。

3. GUI设计

我们设计了相对人性的图形化界面，主要增加了侧边导航栏的功能方便我们快速的进入我们所需要的功能界面。

存在的尚未解决问题及可改进的方向

工作量展示