# 高级软件开发项目开发指南

## 分布式系统架构

Student

student-

panel.fxml

界面

数据库

java.db

StudentRepository



Student

Controller

Student

Controller

前端javafxclient

后端java-server

项目开发采用前后端分离分布式架构，前端客户端基于JavaFX+FXML 开发，javafxclient工程已经基于JavaFX+FXML 开发了前端框架，实现了与后台服务端的数据通讯，实现系统在主界面的显示交互控制，后端服务器端基于Java String Boot 开发，java-server工程基于String Boot搭建了支持前端数据请求的基础框架，实现基于JPA的数据库访问。

1、前后端数据通讯

前端与后端的通讯，采用HTTP通讯，数据格式采用Json数据格式，后端框架提供接收Http请求的服务接口：

|  |
| --- |
| @PostMapping("/getStudentList")  @PreAuthorize("hasRole('ADMIN')")  public DataResponse getStudentList(@Valid @RequestBody DataRequest dataRequest) {  String numName= dataRequest.getString("numName");  List dataList = getStudentMapList(numName);  return CommonMethod.getReturnData(dataList); //按照测试框架规范传送Map的list  } |

开发者按照接口编写规范添加接收服务的方法即可以提供对前端的数据请求，开发者可以不用深入了解JsonString结构，系统直接将JsonString 转换为Java对象。

前端采用HttpClient实现基于HTP协议的后台数据的请求，javafxclient 已经对请求方法进行了封装，只需要调用HttpRequestUtil.request的方法就可以实现数据的请求。

|  |
| --- |
| DataRequest req =new DataRequest();  req.put("numName","");  res = HttpRequestUtil.request("/api/student/getStudentList",req);  if(res != null && res.getCode()== 0) {  studentList = (ArrayList<Map>)res.getData();  } |

前端数据传输的请求数据和返回数据直接采用Java对象，框架会将Java对象序列化为JsonString, 因此也不需要关注Json结构。

2、前端javafxclient

前端通过fxml描述界面的结构，布局组件的属性，通过controller和fxml共同实现界面的显示和交互的控制，通过controller调用控件对象的方法显示后台传过来的数据，也可以从界面获取数据传递到后台，从而作为界面和后台服务进行数据交互的桥梁，Controller接收界面控件的事件的触发，响应事件处理，进行界面更新和后台数据的通讯。

3、后端java-server

主要实现为前端的界面显示提供数据服务，和业务数据的增删改操作，java-server的Controller 接收前端请求的请求参数，通过Repository方法查询数据库的数据获得实体对象或者对象集合，返回前端，用于界面显示，通过Repository方法，把从前端获得数据转换实体对象，通过Repository的Update方法更新数据库数据。

## 二、用户界面描述模板FXML介绍

#### 1、XML语言简介

（1）XML

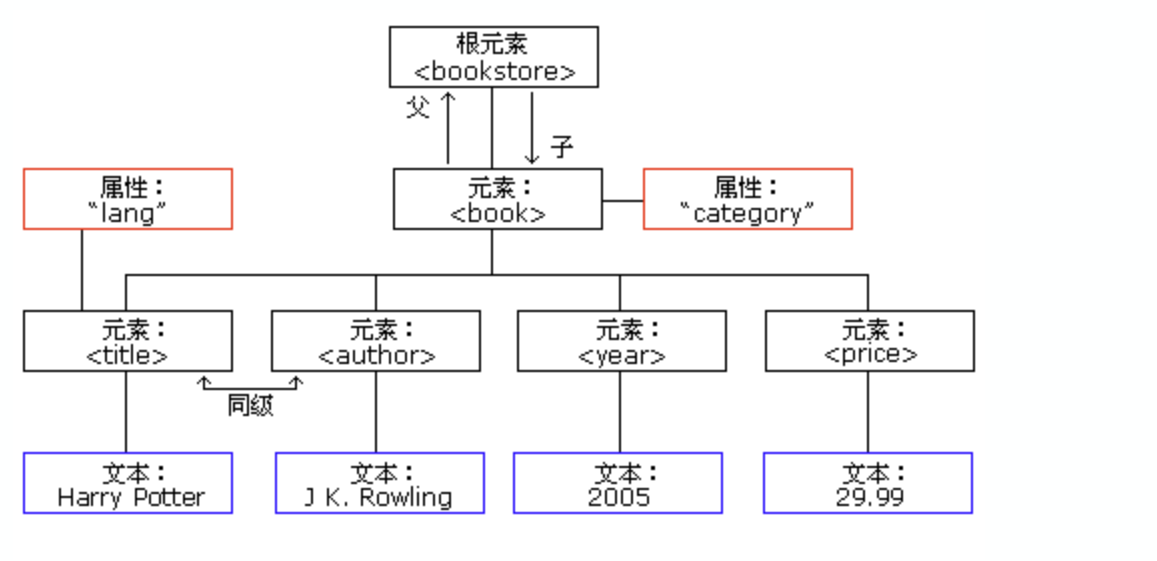
XML指可扩展标记语言（EXtensible Markup Language），XML 是一种很像HTML的标记语言。XML 的设计宗旨是传输数据，而不是显示数据。XML 标签没有被预定义。您需要自行定义标签。XML 被设计为具有自我描述性。XML 是 W3C 的推荐标准。

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <note>  <to>Tove</to>  <from>Jani</from>  <heading>Reminder</heading>  <body>Don't forget me this weekend!</body>  </note> |

XML 文档必须包含根元素。该元素是所有其他元素的父元素。XML 文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端。所有的元素都可以有子元素：

|  |
| --- |
| <root>  <child>  <subchild>.....</subchild>  </child>  </root> |

父、子以及同胞等术语用于描述元素之间的关系。父元素拥有子元素。相同层级上的子元素成为同胞（兄弟或姐妹）。所有的元素都可以有文本内容和属性



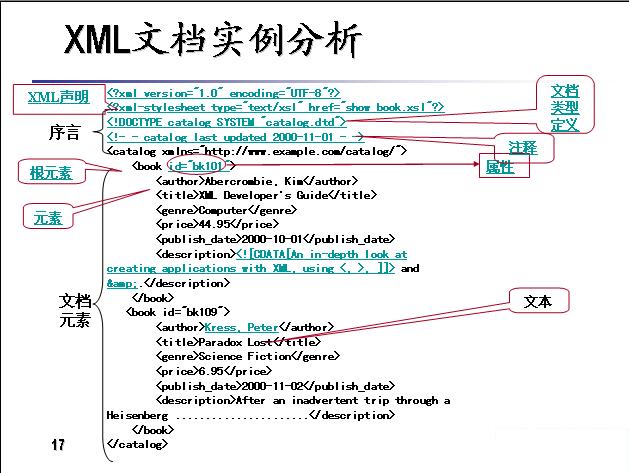
XML 的语法规则很简单，且很有逻辑。这些规则易学易用。

* **XML 文档必须有根元素：比如前面例子中的“**note **”**
* **XML 声明：**XML 声明文件的可选部分，如果存在需要放在文档的第一行
* **所有的 XML 元素都必须有一个关闭标签：**<p>This is a paragraph.</p>
* **XML 标签对大小写敏感：**XML 标签 <Letter> 与标签 <letter> 是不同的
* **XML 必须正确嵌套**

<b><i>This text is bold and italic</i></b>

在上面的实例中，正确嵌套的意思是：由于 <i> 元素是在 <b> 元素内打开的，那么它必须在 <b> 元素内关闭。

* **XML 属性值必须加引号**



（2）java-server pom.xml 文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.5.1</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>org.fatmansoft</groupId>  <artifactId>java-server</artifactId>  <version>1.0-SNAPSHOT</version>  <properties>  <maven.compiler.source>17</maven.compiler.source>  <maven.compiler.target>17</maven.compiler.target>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.session</groupId>  <artifactId>spring-session-bom</artifactId>  <version>2020.0.3</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <dependency>  <groupId>org.springframework.session</groupId>  <artifactId>spring-session-data-redis</artifactId>  </dependency>  <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>jjwt</artifactId>  <version>0.9.1</version>  </dependency>  <!-- https://mvnrepository.com/artifact/com.openhtmltopdf/openhtmltopdf-core -->  <dependency>  <groupId>com.openhtmltopdf</groupId>  <artifactId>openhtmltopdf-core</artifactId>  <version>1.0.9</version>  </dependency>  <!-- https://mvnrepository.com/artifact/com.openhtmltopdf/openhtmltopdf-pdfbox -->  <dependency>  <groupId>com.openhtmltopdf</groupId>  <artifactId>openhtmltopdf-pdfbox</artifactId>  <version>1.0.9</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->  <dependency>  <groupId>org.xerial</groupId>  <artifactId>sqlite-jdbc</artifactId>  <version>3.36.0.1</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-validation</artifactId>  </dependency>  <!-- https://mvnrepository.com/artifact/org.json/json -->  <dependency>  <groupId>org.json</groupId>  <artifactId>json</artifactId>  <version>20210307</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-relational</artifactId>  <version>2.2.2</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.4.32.Final</version>  </dependency>  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpcore</artifactId>  <version>4.4.14</version>  </dependency>  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>4.5.13</version>  </dependency>  <dependency>  <groupId>org.apache.poi</groupId>  <artifactId>poi-ooxml</artifactId>  <version>4.1.0</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <executable>true</executable>  <jvmArguments>  -Dfile.encoding=utf-8  </jvmArguments>  </configuration>  </plugin>  </plugins>  </build>  </project> |

XML的根元素为project,他下面的子元素包括modelVersion，parent，groupId，artifactId，version，properties，dependencies，build，其中 ：

<groupId>org.fatmansoft</groupId>

<artifactId>java-server</artifactId>

<version>1.0-SNAPSHOT</version>

用来说明将来生成的Java的名字 java-server1.0-SNAPSHOT.jar。

Properties 用来署名编译器的版本

Dependencies 用来书名项目要依赖的包，每个项目要引入的依赖都不同，我们的项目要引入非Java库第三方的类库，需要在这里面加入一个子元素dependency， 如我们使用Sqlite数据库，添加下面的依赖子元素

<!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->

<dependency>

<groupId>org.xerial</groupId>

<artifactId>sqlite-jdbc</artifactId>

<version>3.36.0.1</version>

</dependency>

build 元素 用来指定打包时系统配置，如添加虚拟机参数 -Dfile.encoding=utf-8。

（3）java-fx-client pom.xml 文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.teach</groupId>  <artifactId>java-fx-client</artifactId>  <version>1.0-SNAPSHOT</version>  <name>java-fx-client</name>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <junit.version>5.7.1</junit.version>  </properties>  <dependencies>  <dependency>  <groupId>org.openjfx</groupId>  <artifactId>javafx-controls</artifactId>  <version>17-ea+11</version>  </dependency>  <dependency>  <groupId>org.openjfx</groupId>  <artifactId>javafx-fxml</artifactId>  <version>17-ea+11</version>  </dependency>  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-api</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.junit.jupiter</groupId>  <artifactId>junit-jupiter-engine</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.openjfx</groupId>  <artifactId>javafx-web</artifactId>  <version>17-ea+11</version>  </dependency>  <dependency>  <groupId>org.openjfx</groupId>  <artifactId>javafx-swing</artifactId>  <version>17.0.2</version>  </dependency>  <dependency>  <groupId>org.apache.pdfbox</groupId>  <artifactId>pdfbox</artifactId>  <version>2.0.24</version>  </dependency>  <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>gson</artifactId>  <version>2.8.6</version>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-crypto</artifactId>  <version>6.0.0</version>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.1</version>  <configuration>  <source>17</source>  <target>17</target>  </configuration>  </plugin>  <plugin>  <groupId>org.openjfx</groupId>  <artifactId>javafx-maven-plugin</artifactId>  <version>0.0.6</version>  <executions>  <execution>  <!-- Default configuration for running with: mvn clean javafx:run -->  <id>default-cli</id>  <configuration>  <mainClass>com.teach.javafxclient/com.teach.javafxclient.MainApplication</mainClass>  </configuration>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

配置说明和Java-server配置的说明类似，注意一下Java的版本和依赖就可以，

Pom.xml文件组重要的就是管理类库.jar的依赖，需要新的依赖的.jar是IDEA提示添加新的依赖,添加后会自动加入pom.xml里,也可以手动修改该文件,删除或添加相呼应的依赖。

#### 2、FXML语言简介

FXML是JavaFX 2.0提供的新技术。是一种在JavaFX应用程序中定义用户界面的，基于XML的声明性标记语言。FXML非常适合用来静态布局，如表单、控件和表格。使用FXML，还可以通过脚本动态地构建布局。

FXML的优势之一是基于XML，是大多数开发人员所熟悉的，另一个优点是，FXML不是编译语言，不需要重新编译代码就可看到所做的更改。FXML的第三个优势是很容易看到应用程序的场景结构图。。

FXML是一种基于xml的语言，用于为JavaFX应用程序构建用户界面。可以使用FXML构建整个场景（ scene ）或场景的一部分。FXML允许应用程序开发人员将构建UI的逻辑与业务逻辑分离。如果应用程序的UI部分发生了更改，不需要重新编译JavaFX代码。相反，可以使用文本编辑器更改FXML并重新运行应用程序。仍然使用JavaFX使用Java语言编写业务逻辑。FXML文档是XML文档。理解FXML需要具备XML的基本知识。

使用Java编程语言开发GUI应用程序，程序员依赖于诸如高级窗口工具包(AWT)和Swings等类库。在JavaFX出现之后，这些Java程序开发就可以有效地利用丰富的类库来开发GUI应用程序。

 JavaFX 是一组图形和媒体包，使开发人员能够设计、创建、测试、调试和部署在不同平台上一致运行的富客户端应用程序。JavaFX 库是作为 Java API 编写的，因此 JavaFX 应用程序代码可以从任何 Java 库中引用 API（引用JAVA类库中其他任意包，包括swings）。

如果将用户界面 (UI) 和后端逻辑分开，可以使用 FXML 脚本语言开发 UI 界面，并为应用程序使用 Java 代码逻辑。

不编写代码的情况下设计 UI，可使用 JavaFX Scene Builder。Scene Builder 创建可移植到集成开发环境 (最好IDEA) 的 FXML 标记，开发人员可以在此之上添加业务逻辑。

**使用Java语言编写**- JavaFX库都是用Java编写的，可用于在JVM上执行的语言，跨平台,**FXML** - JavaFX采用称为FXML的语言，这是一种类似声明式标记语言的HTML。这种语言的唯一目的是定义用户界面。**Scene Builder** - JavaFX提供了一个名为Scene Builder(场景生成器)的应用程序。 在将此应用程序集成到IDE(如Eclipse和Idea)中时，用户可以使用可视化拖放设计界面，用于开发FXML应用程序(就像Swing Drag＆Drop和DreamWeaver应用程序一样)。**Swing互操作性** - 在JavaFX应用程序中，可以使用Swing Node类嵌入Swing内容。 同样，可以使用JavaFX功能(如嵌入式Web内容和丰富的图形媒体)更新现有的Swing应用程序。**内置UI控件** - JavaFX库UI控件使用它可以开发一个全功能的应用程序。**类似CSS的样式**- JavaFX提供像样式的CSS。 通过使用它，可以使用CSS的简单知识改进应用程序的设计。JavaFX库提供了一组丰富的API来开发GUI应用程序，2D和3D图形等。这套API还包括Java平台的功能。因此，使用此API，可以访问Java语言的功能，如通用，注释，多线程和Lambda表达式。传统的Java集合库已增强，包括可观察列表和映射等概念。使用这些，用户可以观察数据模型中的更改。

#### 3、采用FXML语言描述配置文件实例说明一，用户登录界面login-view.fxml

|  |
| --- |
| <?import javafx.scene.control.PasswordField?>  <?import javafx.scene.control.TextField?>  <?import javafx.scene.image.Image?>  <?import javafx.scene.image.ImageView?>  <?import javafx.scene.layout.ColumnConstraints?>  <?import javafx.scene.layout.GridPane?>  <?import javafx.scene.layout.HBox?>  <?import javafx.scene.layout.RowConstraints?>  <?import javafx.scene.layout.VBox?>  <!-- 用户登录对话框FXML配置文件 对应页面交互控制类 com.teach.javafxclient.controller.base.LoginController -->  <VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.teach.javafxclient.controller.base.LoginController">  <padding>  <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />  </padding>  <HBox>  <ImageView fx:id="imageView" fitHeight="75.0" fitWidth="100.0" pickOnBounds="true" preserveRatio="true">  <image>  <Image url="@../picture/logo.jpg" />  </image></ImageView>  <GridPane>  <columnConstraints>  <ColumnConstraints halignment="RIGHT" hgrow="SOMETIMES" maxWidth="95.0" minWidth="10.0" prefWidth="59.5" />  <ColumnConstraints hgrow="SOMETIMES" maxWidth="140.5" minWidth="10.0" prefWidth="140.5" />  </columnConstraints>  <rowConstraints>  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  </rowConstraints>  <children>  <Label text="用户名">  <GridPane.margin>  <Insets right="5.0" />  </GridPane.margin></Label>  <TextField fx:id="usernameField" GridPane.columnIndex="1" />  <Label text="密码" GridPane.rowIndex="1">  <GridPane.margin>  <Insets right="5.0" />  </GridPane.margin></Label>  <PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="1" />  </children>  </GridPane>  </HBox>  <Button onAction="#onLoginButtonClick" text="登录" />  </VBox> |

界面说明：

（1）<? /> 标记引入的Java FX控件类

（2）<!-- --> 表示XML结构的注解

（3）<padding> 描述表示控件与控件之间的间隔距离

（4）fx:controller="com.teach.javafxclient.controller.base.LoginController" 表示该描述的面板对应Java数据控制处理类为com.teach.javafxclient.controller.base.LoginController，FXMLLoader 的load() 方法加载login-view.fxnml文件时会自动创建该类的一个对象，并在构造方法里根据fxml描述的FX的Scene，创建该场景对象，初始所有内部的Node节点，设置相关的属性,构造fxml描述的Scene，，系统还会自动调用LogoinController类的initialize(),由于开发者重写该方法，显示的完成fxml配置文件没有说明的设置，最后放在要显示的舞台(Stage) 进行显示，界面呈现的效果如图所示。



（5）整个场景最外层是VBox 容器，包含2行，上面的第0行是一个HBox容器，下面的第1行是一个Button控件，上部分HBOX容器又包含两部分，左边是一个ImageView控件，内部包含Image控件，右面是一个GridPane，GridPane是一个2\*2的网格，分表是两个Label用于显示用户名、密码和一个TextFiled 接收username, 一个Password 控件 接收password，这个就是整个界面的描述结构。

（6）fx:controller="com.teach.javafxclient.controller.base.LoginController" 对应交互界面的控制类，fx:id="usernameField"，fx:id="passwordField" 对应LoginController 里面的usernameField，passwordField 属性，控制类可以通过该属性获取用户从界面上输入的用户名和密码，也可以通过他们将缺省的用户名和密码在界面显示。onAction="#onLoginButtonClick" 给Button组件注册时事件监听者对象的处理方法onLoginButtonClick，当用户点击登录按钮时，系统触发ActionEvent事件，系统会自动调用LoginController对象的onLoginButtonClick， 该方法从界面上获取用户名和密码，发送用户名和密码给java-server，java-server 接收用户名密码，查询数据库，验证密码，通过，返回用户信息，并在主框架场景显示主框架，登录不成功给出提示，继续等待用户重新登录。

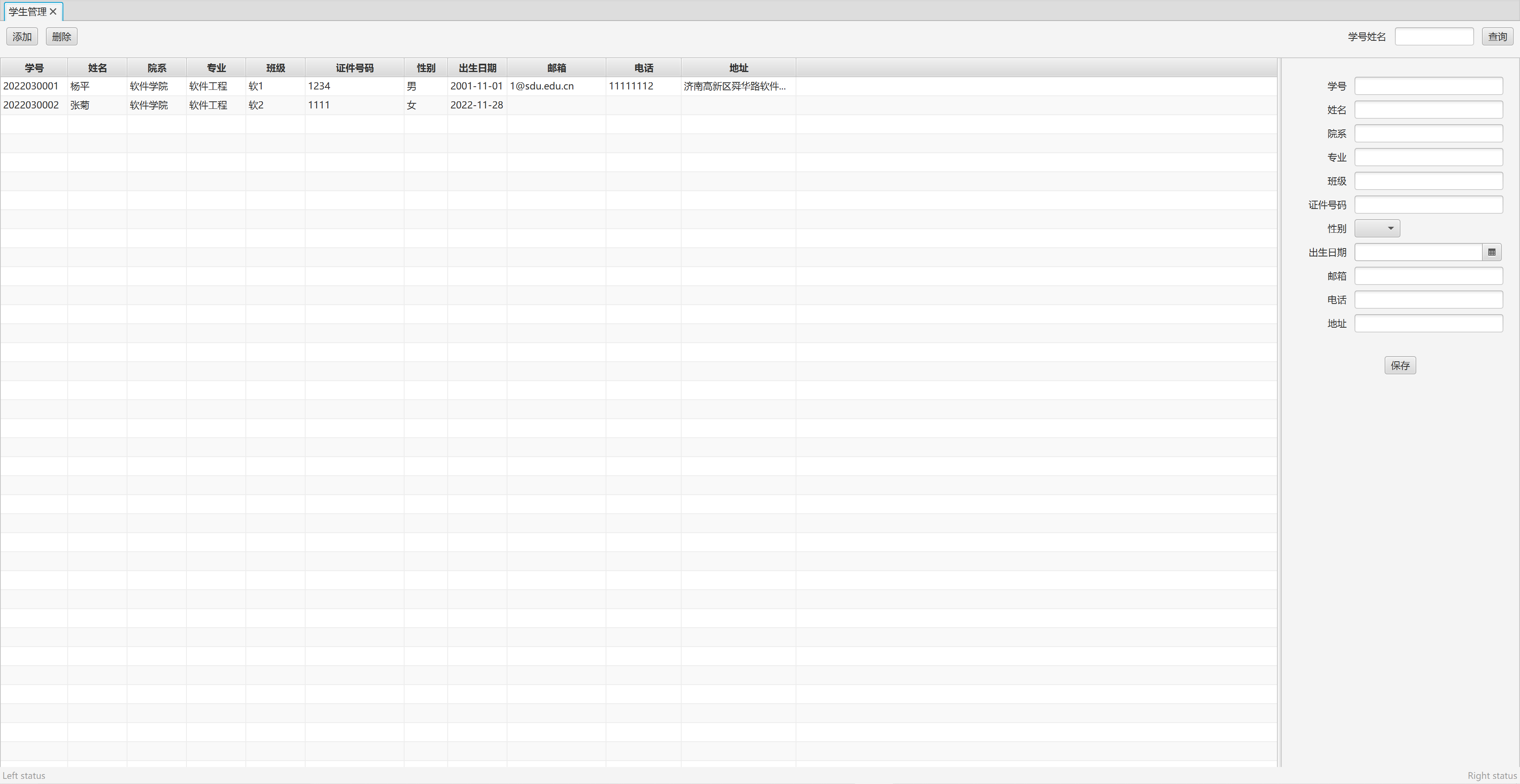
#### 3、采用FXML语言描述配置文件实例说明二，学生管理界面student-panel.fxml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?import javafx.geometry.Insets?>  <?import javafx.scene.control.Button?>  <?import javafx.scene.control.ComboBox?>  <?import javafx.scene.control.DatePicker?>  <?import javafx.scene.control.Label?>  <?import javafx.scene.control.SplitPane?>  <?import javafx.scene.control.TableColumn?>  <?import javafx.scene.control.TableView?>  <?import javafx.scene.control.TextField?>  <?import javafx.scene.layout.BorderPane?>  <?import javafx.scene.layout.ColumnConstraints?>  <?import javafx.scene.layout.FlowPane?>  <?import javafx.scene.layout.GridPane?>  <?import javafx.scene.layout.HBox?>  <?import javafx.scene.layout.Pane?>  <?import javafx.scene.layout.RowConstraints?>  <?import javafx.scene.layout.VBox?>  <!-- 学生管理 FXML 配置文件 对应的页面交互控制类 com.teach.javafxclient.controller.StudentController 如果是切换成本地开发模式 界面配置不变，将页面交互控制类 切换为 com.teach.javafxclient.controller.StudentControllerLocal-->  <BorderPane xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.teach.javafxclient.controller.StudentController">  <top>  <HBox id="HBox" alignment="CENTER\_LEFT" spacing="5.0" >  <children>  <FlowPane prefHeight="40.0" prefWidth="200.0" BorderPane.alignment="CENTER">  <children>  <Button mnemonicParsing="false" onAction="#onAddButtonClick" text="添加">  <FlowPane.margin>  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />  </FlowPane.margin>  </Button>  <Button mnemonicParsing="false" onAction="#onDeleteButtonClick" text="删除">  <FlowPane.margin>  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />  </FlowPane.margin>  </Button>  </children>  </FlowPane>  <Pane prefHeight="-1.0" prefWidth="-1.0" HBox.hgrow="ALWAYS" />  <FlowPane alignment="TOP\_RIGHT" prefHeight="40.0" prefWidth="600.0" BorderPane.alignment="CENTER">  <children>  <Label prefWidth="49.0" text="学号姓名">  <FlowPane.margin>  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />  </FlowPane.margin></Label>  <TextField fx:id="numNameTextField" prefWidth="100.0">  <FlowPane.margin>  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />  </FlowPane.margin></TextField>  <Button mnemonicParsing="false" onAction="#onQueryButtonClick" text="查询">  <FlowPane.margin>  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />  </FlowPane.margin></Button>  </children>  </FlowPane>  </children>  <padding>  <Insets bottom="3.0" left="3.0" right="3.0" top="3.0" />  </padding>  </HBox>  </top>  <center>  <SplitPane dividerPositions="0.6979797979797979" prefHeight="160.0" prefWidth="200.0" BorderPane.alignment="CENTER">  <items>  <TableView fx:id="dataTableView" prefHeight="200.0" prefWidth="200.0">  <columns>  <TableColumn fx:id="numColumn" prefWidth="85.0" text="学号" />  <TableColumn fx:id="nameColumn" prefWidth="75.0" text="姓名" />  <TableColumn fx:id="deptColumn" prefWidth="75.0" text="院系" />  <TableColumn fx:id="majorColumn" prefWidth="75.0" text="专业" />  <TableColumn fx:id="classNameColumn" prefWidth="75.0" text="班级" />  <TableColumn fx:id="cardColumn" prefWidth="125.0" text="证件号码" />  <TableColumn fx:id="genderColumn" prefWidth="55.0" text="性别" />  <TableColumn fx:id="birthdayColumn" prefWidth="75.0" text="出生日期" />  <TableColumn fx:id="emailColumn" prefWidth="125.0" text="邮箱" />  <TableColumn fx:id="phoneColumn" prefWidth="95.0" text="电话" />  <TableColumn fx:id="addressColumn" prefWidth="145.0" text="地址" />  </columns>  </TableView>  <VBox alignment="TOP\_CENTER" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="1000.0" prefWidth="300.0" spacing="20.0">  <padding>  <Insets bottom="20.0" left="20.0" right="20.0" top="20.0" />  </padding>  <GridPane hgap="10.0">  <columnConstraints>  <ColumnConstraints halignment="RIGHT" hgrow="SOMETIMES" maxWidth="124.0" minWidth="10.0" prefWidth="67.0" />  <ColumnConstraints hgrow="SOMETIMES" maxWidth="193.0" minWidth="10.0" prefWidth="193.0" />  </columnConstraints>  <rowConstraints>  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />  </rowConstraints>  <children>  <Label text="学号" GridPane.rowIndex="0" />  <TextField fx:id="numField" GridPane.columnIndex="1" GridPane.rowIndex="0" />  <Label text="姓名" GridPane.rowIndex="1" />  <TextField fx:id="nameField" GridPane.columnIndex="1" GridPane.rowIndex="1" />  <Label text="院系" GridPane.rowIndex="2" />  <TextField fx:id="deptField" GridPane.columnIndex="1" GridPane.rowIndex="2" />  <Label text="专业" GridPane.rowIndex="3" />  <TextField fx:id="majorField" GridPane.columnIndex="1" GridPane.rowIndex="3" />  <Label text="班级" GridPane.rowIndex="4" />  <TextField fx:id="classNameField" GridPane.columnIndex="1" GridPane.rowIndex="4" />  <Label text="证件号码" GridPane.rowIndex="5" />  <TextField fx:id="cardField" GridPane.columnIndex="1" GridPane.rowIndex="5" />  <Label text="性别" GridPane.rowIndex="6" />  <ComboBox fx:id="genderComboBox" GridPane.columnIndex="1" GridPane.rowIndex="6" />  <Label text="出生日期" GridPane.rowIndex="7" />  <DatePicker fx:id="birthdayPick" GridPane.columnIndex="1" GridPane.rowIndex="7" />  <Label text="邮箱" GridPane.rowIndex="8" />  <TextField fx:id="emailField" GridPane.columnIndex="1" GridPane.rowIndex="8" />  <Label text="电话" GridPane.rowIndex="9" />  <TextField fx:id="phoneField" GridPane.columnIndex="1" GridPane.rowIndex="9" />  <Label text="地址" GridPane.rowIndex="10" />  <TextField fx:id="addressField" GridPane.columnIndex="1" GridPane.rowIndex="10" />  </children>  </GridPane>  <FlowPane alignment="CENTER" columnHalignment="CENTER" prefHeight="36.0">  <Button onAction="#onSaveButtonClick" text="保存" />  </FlowPane>  </VBox>  </items>  </SplitPane>  </center>  </BorderPane> |

说明（1）界面结构最外层为BorderPane容器 ，BorderPane容器由上下左右中心5部分组成，即<top/>， <bottom/>，<left/>,<right/>,< center>,本界面仅保留上中心两部分：

1. 上部分是（top）是HBOX容器，主要实现查询部分，由三部分组成，一次为FlowPane（包含添加、删除两个Butoon，按照从左到右排列），Pane(空白作为分割用)，FlowPane（查询输入框和查询按钮，按照从右向左排列），这个是常用的一张表格查询操作布局风格
2. 下半部（Center）是一个SplitPane(是一个切分容器，中间的分割线可以用鼠标拖动，调整左右两个区域的分割比例)，左边区域是TableView，以表格的形式显示多个学生表格信息，右边区域是一个VBOX容器，包含两部分，上部分为Grid Panel，利用包含用于学生基本信息修改的组件，用于选中的单个学生信息的编辑修改，下部分是一个FlowPane，包含保存按钮，点击可实现信息提交。

界面显示效果如图所示



（2）fx:controller="com.teach.javafxclient.controller.base.StudentController" 对应交互界面的控制类，fx:id="dataTableView"和fx:id="numColumn"列控件的id对应StudentController 里面的dataTableView及 numColumn等列控件属性，控制类可以通过这些属性操作界面中的控件进行显示和数据交互。x:id="numField" 等ID 对应StudentController中的numField 等控件属性，通过他们对单个学生信息的显示和修改。onAction="#onQueryButtonClick" 点击“查询按钮”，执行onQueryButtonClick方法，将用户输入的查询传到java-server，返回查询的学生集合，并在TableView控件上显示。onAction="# onSaveButtonClick " 点击“保存”按钮，onLoginButtonClick，从界面上获取学生的信息，并将学生信息发送给java-server，java-server 接收学生信息，永久性更新数据库中的数据 ，onAction="#onAddButtonClick"执行onAddButtonClick 方法，将学生信息控件的输入清空， onAction="#onDeleteButtonClick" 对应执行删除选中学生的操作。

## 三、程序开发流程示例- 学生管理功能实现

系统开发流程分前后端开发，具体的开发流程为：

（1）前后端数据通讯协议接口的设计

（2）数据库表及相关数据类及方法的设计

（3）前端FXML和控制类的设计

（4）手动添加用菜单管理增加菜单项，进行前后端程序联调与测试。

下面以学生管理功能为例给出详细的功能的设计过程：增加菜单后端数据库表设计、实体类Model设计、数据操作接口Repository设计，Web服务处理的Controller设计：

#### 1、前后端通讯协议接口的设计

（1）前端请求Url "/api/student/getStudentList", 请求方式 Post，后端接口实现方法getStudentList， 请求参数 numName， 功能初始化时，numName为””,查询列出所有的学生信息，查询操作时numName 为界面输入的查询传，返回数据为学生列表Map对象列表集合

（2）前端请求Url "/api/student/ studentDelete ", 请求方式方式 Post，后端接口实现方法studentDelete， 请求参数 studentId， 删除主键为studentId的学生信息，由于学生信息管理Person，User,删除时一并删除，返回数据为操作是否正常完成。

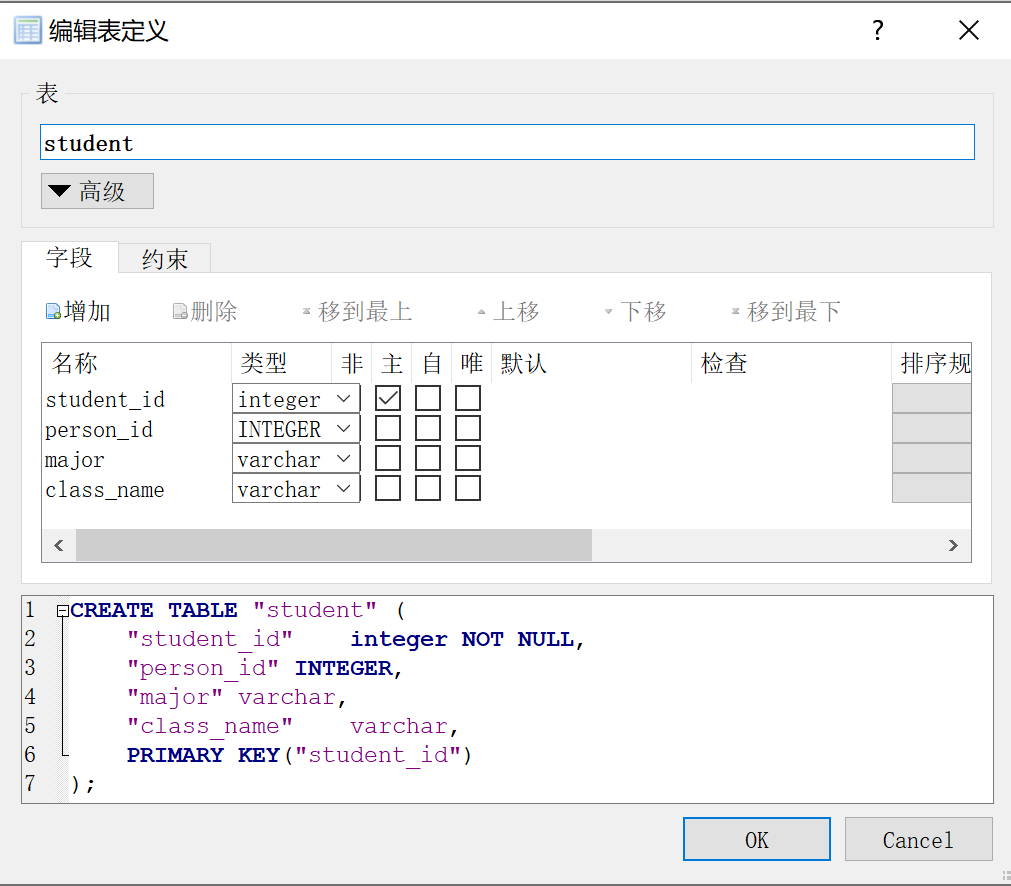
（3）前端请求Url "/api/student/ getStudentInfo ", 请求方式方式 Post，后端接口实现方法getStudentInfo， 请求参数 studentId， 学生主键ID返回数据为学生详细信息的Map对象

（4）前端请求Url "/api/student/ studentEditSave ", 请求方式方式 Post，后端接口实现方法studentEditSave， 请求参数 studentId，学生主键ID，为空表示时新添加的学生，响应添加Student,Person,User数据，不为空，修改Student 等表中相关数据，form 保存学生各属性信息的Map对象，返回数据为操作是否正常完成。

#### 2、相关数据库表类的说明

**（1） 数据库表 student**

由于学生与人员相关的信息保存在person表里面，所以student只保留学生的相关特殊的属性，和与person表主键像管理的person\_id属性，数据库表的结构如下图：



表名student， 字段名为：

student\_id –主键student\_id

person\_id: 外键关联person表的主键person\_id

major：专业

class\_name：班级

数据库表的每一条记录保存一个学生的信息，学生的主建student\_id不相同。

下面是创建student表的SQL语句

CREATE TABLE "student" (

"student\_id" integer NOT NULL,

"person\_id" INTEGER,

"major" varchar,

"class\_name" varchar,

PRIMARY KEY("student\_id")

);

**注意**：**由于这里我们使用的文件型数据库sqlit，不支持主键的自动自增，程序增加一条新的记录时需要设置主键student\_id，对于一般的常用的数据库管理系统如Mysql，则支持主键自增，增加新记录不需要设置student\_id。**

#### 2、实体类 Student

Java Spring JPA开发框架支持将对Java的对象操作映射为数据库记录的操作，开发时我们需要设计一个实体类Student 映射数据库的表student

|  |
| --- |
| @Entity  @Table(name = "student",  uniqueConstraints = {  })  public class Student {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer studentId;  @OneToOne  @JoinColumn(name="person\_id")  private Person person;  @Size(max = 20)  private String major;  @Size(max = 50)  private String className;  } |

（1）@开头是注解 @Entity 表示是一个实体类

@Table( name = "student",

uniqueConstraints = {

})

属名Student 映射 数据库表 student

（2） @Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer student\_id;

表示student\_id;是主键。

（3）@Size(max = 20) 表示该字段的长度不超过20

（4）@OneToOne 表示是一对一的关系， student表中的一条记录，对应person一条记录

@JoinColumn(name="person\_id")

private Person person; Student 对象中的person变量，对应person表中记录的Person对象，通过person对象，可以找到对应person的所有属性，

**类的每一个属性对应数据库表的每一个字段：**按照java的变量驼峰命名规范和数据库表名和字段的命名规范自动映射，如果不按照规范映射可以用注解显示的说明

dtudentId-> student\_id

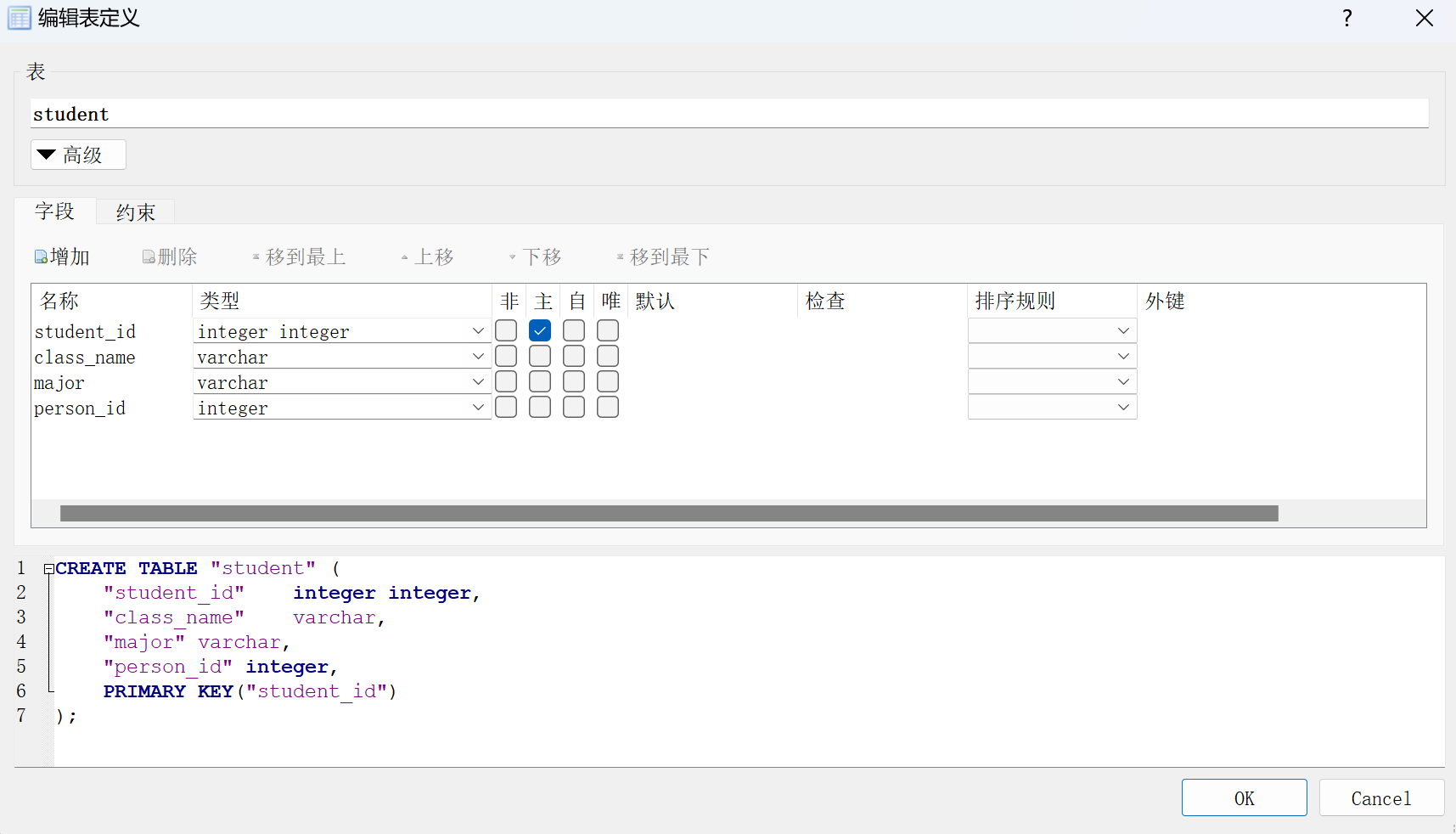
person->person\_id

major->major

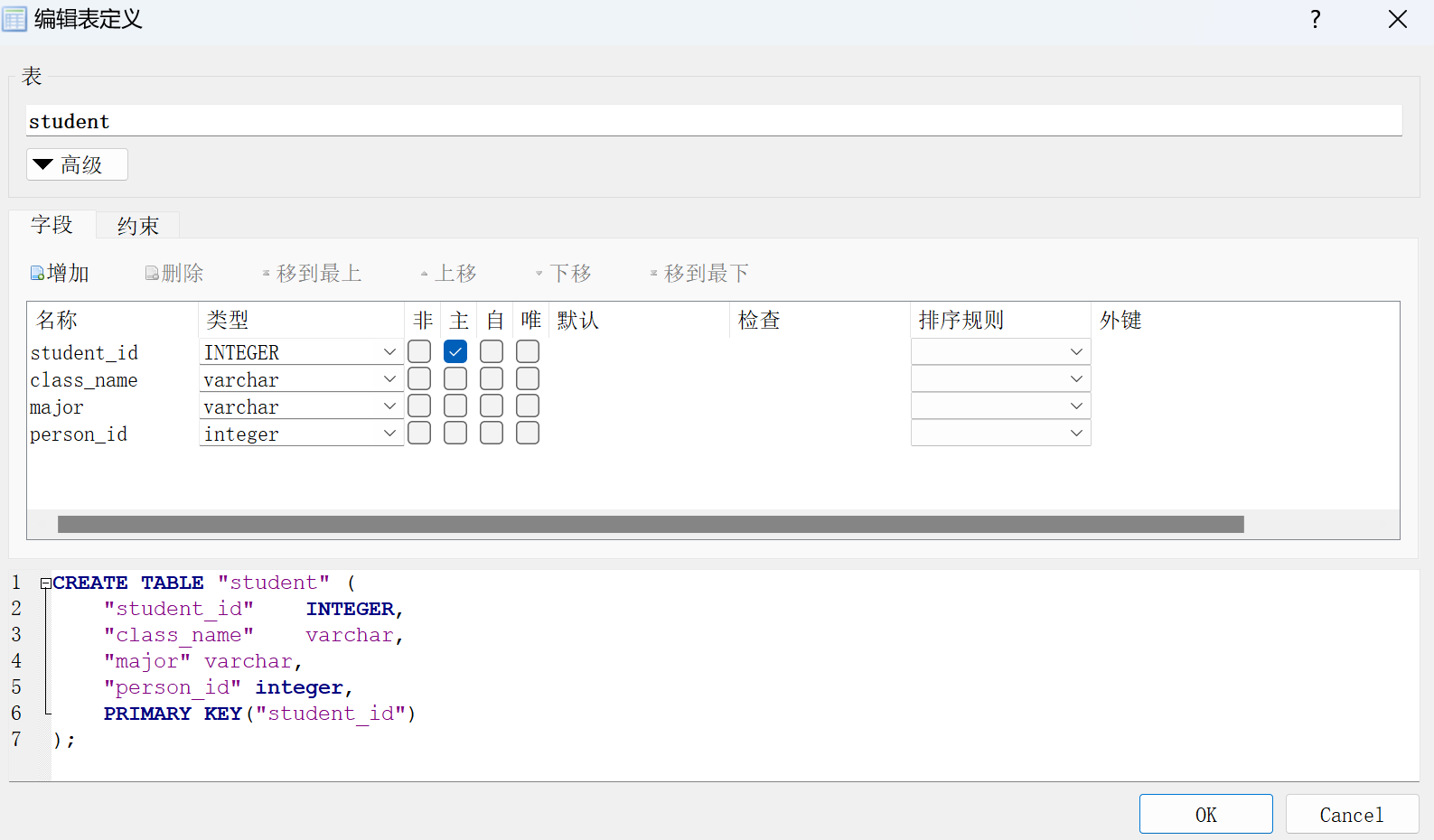
className->class\_name

**开发时只需要定义实体类Student，一般我们不需要创建和修改数据库表，程序运行时，会自动创建和修改数据库表，我们提供给大家的数据库操作的客户端软件SB browser for SQLite可以创建修改数据库表，但主要用于数据的查看和修改**。

SpringBoot JPA 对于SqlitDB 自动创建表 有一个 Bug ，如图：



Student\_id 类型是 两个 integer integer， 需要用客户端程序手工删除一个：结果如图



#### 3、数据库数据查询修改操作接口 StudentRepository

按照Java String JPA 编程规范，数据库数据的查询、修改、删除操作实际上是对实体对象的操作，具体操作的方法是在接口StudentRepository定义

|  |
| --- |
| public interface StudentRepository extends JpaRepository<Student,Integer> {  @Query(value = "select max(studentId) from Student ")  Integer getMaxId();  Optional<Student> findByPersonPersonId(Integer personId);  @Query(value = "from Student where ?1='' or person.num like %?1% or person.name like %?1% ")  List<Student> findStudentListByNumName(String numName);  } |

（1）定义StudentRepository 接口需要集成父接口JpaRepository， JpaRepository定义了很多方法，如find All(), findById(id),save(t),delete(t)，可以直接使用这些方法对实体对象操作来实现对数据库的操作， 具体参见程序示例。

（A）添加一条记录 s=new Student(), 设置属性如s.setSyudentName(“zhangsan”) ，调用save

（B）修改一条记录 findById(id) 设置属性， 调用save方法

（B）删除一条记录 findById(id) 查询得到对象 调用delete方法

（2）我们经常使用各种条件的数据查询，即根据查询条件，从数据库里查询出需要的数据，可以是单个记录，也可以是多条记录，如果JpaRepository里的方法不满足实际需求，可以根据需要在我们的接口StudentRepository里定义自己的查询方法，按照Java String JPA 开发规范，下面介绍三种方式的方法的定义：

（A）通过命名规范定义查询方法:

Optional<Student> findByPersonPersonId (Integer personId); 该方法查询人员主键为参数personId的记录，person是Student的一个属性，personId是Person实体的一个属性

（B）通过注解方式实现基于实体对象和属性的查询

@Query(value = "from Student where ?1='' or person.num like %?1% or person.name like %?1% ")

List<Student> findStudentListByNumName(String numName);

这里注解里的查询串中使用的实体类和属性

（C）通过使用标准SQL语言注解方式实现查询

@Query(value = "select \* from student s, person p where p.person\_id=s.person\_id and (?1='' or p.num like %?1% or p.name like %?1%) ", nativeQuery = true)

List<Student> findStudentListByNumNameNative(String numName);

（A）方法命名和参数的个数和类型有严格的要求， B，C方法的命名没有要求

返回类型，如果是单一记录则返回Optional<Student>, 如果多条记录，则返回List<Student>

#### 4、StudentController设计

StudentController主要用于接受前端数据请求，通过StudentRepository等数据操作接口，对数据库查询获得Student对象或者对象集合，并对数据进行相应的业务处理，再返回给前端进行显示。前端开发框架，已经实现了HttpRequest请求服务实现的封装，开发者只需在后端相应的Controller类里定义服务方法，并写好相应的注释就可以实现对前端数据请求的响应，下面给出我们用到几个方法。

**（1）StudentController类的定义：**

@CrossOrigin(origins = "\*", maxAge = 3600)

@RestController

@RequestMapping("/api/student")

public class StudentController {

@Autowired

private StudentRepository studentRepository;

（A）类定义前面增加三行注释，说明该类可以作为支持前端请求的业务类，所有的控制类都可以包含这样的注解。

（B）@Autowired 定义注入该类中使用的数据库操作接口实现的对象，如studentRepository，

**（2）获取学生列表：getStudentList**

@PostMapping("/getStudentList")

@PreAuthorize("hasRole('ROLE\_ADMIN')")

public DataResponse getStudentList (@Valid @RequestBody DataRequest dataRequest)

**（4）删除学生：sdentDelete**

@PostMapping("/studentDelete")

@PreAuthorize(" hasRole(' ROLE\_ADMIN')")

public DataResponse studentDelete(@Valid @RequestBody DataRequest dataRequest)

**（5）获取学生信息：getStudentInfo**

@PostMapping("/getStudentInfo")

@PreAuthorize("hasRole(' ROLE\_ADMIN')")

public DataResponse getStudentInfo (@Valid @RequestBody DataRequest dataRequest)

**（6）保存学生信息：studentEditSave**

@PostMapping("/studentEditSave")

@PreAuthorize(" hasRole(' ROLE\_ADMIN')")

public DataResponse studentEditSave(@Valid @RequestBody DataRequest dataRequest)

#### 5、后端开发相关类的说明

（1）DataRequet 用于存储前端请求传入的参数， 并提供了提取参数的方法

（2）DataResponse 用于存储返回前端数据的对象

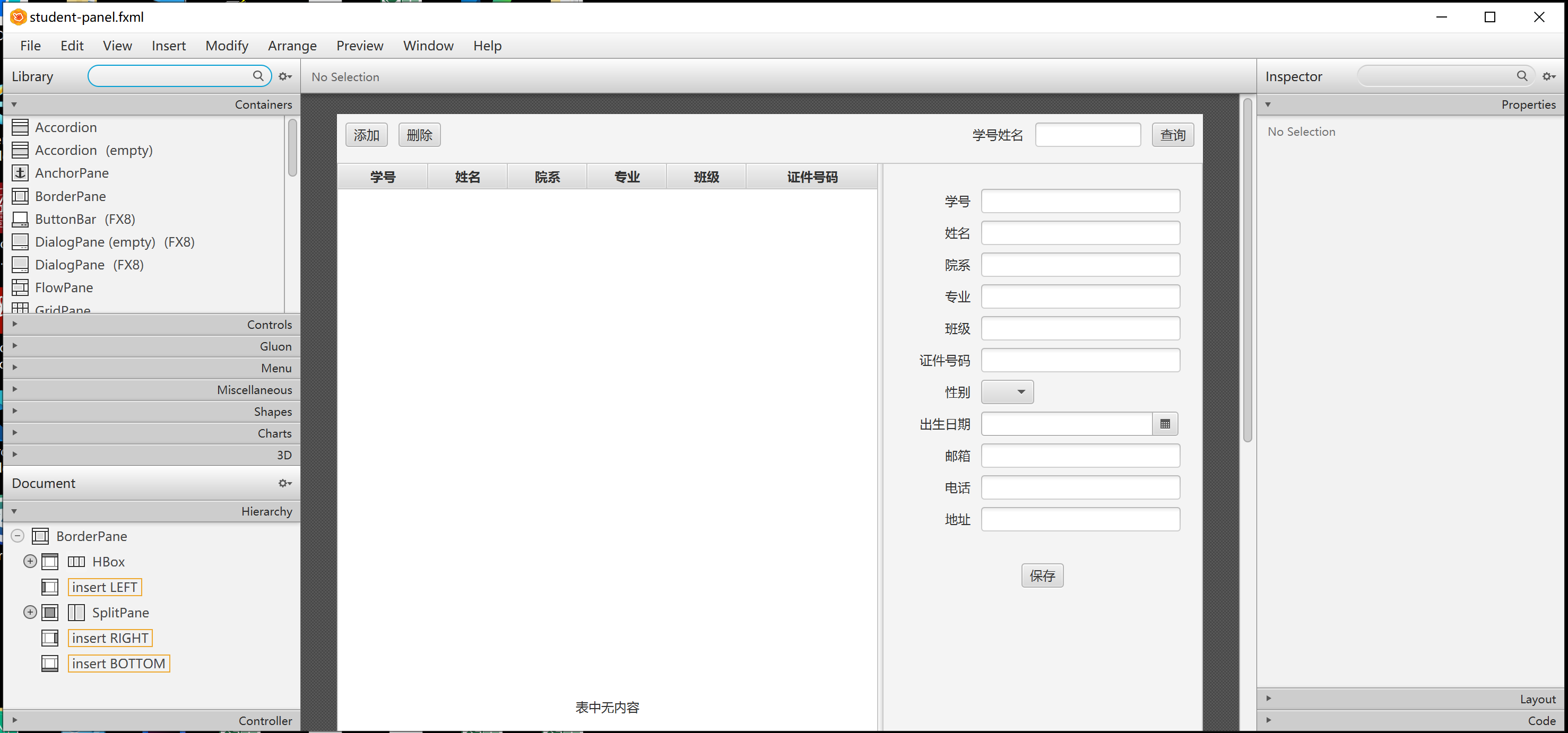
（3）DateTimeTool 用于Date和String转换和时间相关的方法

（4）CommonMethod 是一个工具类，主要包括一些常见的数据处理方法，其中主要的用于生成返回对象DataResponse的三个方法：使用 getReturnData返回数据，使用getReturnMessageOK()返回正确操作信息，使用getReturnMessageError(msg)提示错误信息。

下图是三个相关类的组织结构图，几个实体类可以共用一个controller

#### 6、前端student\_panel.fxml的设计

使用SceneBuilder 创建设计student\_panel.fxml，添加fx:controller="com.teach.javafxclient.controller.StudentController按照student\_panel容器和组件的嵌套结构，将响应的容器和组件拖拽到响应的位置，点击组件可以设置组件的属性、添加对应于StudentController中的fx:id="tableView"等组件属性，和onAction="#onSaveButtonClick"等事件处理方法，如图所示。



也可以在IDEA里直接编辑修改student\_panel.fxml，开发设计的新功能的xml，可以参考框架已经设计好的fxml文件编辑修改。

#### 7、前端 Student Controller的设计

创建Student Controller类，设计相关的属性和方法，为了使Student Controller能响应编辑菜单中的新建、保存，删除等功能设计继承ToolController。

（1）实例属性

@FXML

private TableView<Map> dataTableView; //学生信息表

@FXML

private TableColumn<Map,String> numColumn; //学生信息表 编号列

@FXML

private TableColumn<Map,String> nameColumn; //学生信息表 名称列

@FXML

private TableColumn<Map,String> deptColumn; //学生信息表 院系列

@FXML

private TableColumn<Map,String> majorColumn; //学生信息表 专业列

@FXML

private TableColumn<Map,String> classNameColumn; //学生信息表 班级列

@FXML

private TableColumn<Map,String> cardColumn; //学生信息表 证件号码列

@FXML

private TableColumn<Map,String> genderColumn; //学生信息表 性别列

@FXML

private TableColumn<Map,String> birthdayColumn; //学生信息表 出生日期列

@FXML

private TableColumn<Map,Integer> emailColumn; //学生信息表 邮箱列

@FXML

private TableColumn<Map,Integer> phoneColumn; //学生信息表 电话列

@FXML

private TableColumn<Map,Integer> addressColumn;//学生信息表 地址列

@FXML

private TextField numField; //学生信息 学号输入域

@FXML

private TextField nameField; //学生信息 名称输入域

@FXML

private TextField deptField; //学生信息 院系输入域

@FXML

private TextField majorField; //学生信息 专业输入域

@FXML

private TextField classNameField; //学生信息 班级输入域

@FXML

private TextField cardField; //学生信息 证件号码输入域

@FXML

private ComboBox<OptionItem> genderComboBox; //学生信息 性别输入域

@FXML

private DatePicker birthdayPick; //学生信息 出生日期选择域

@FXML

private TextField emailField; //学生信息 邮箱输入域

@FXML

private TextField phoneField; //学生信息 电话输入域

@FXML

private TextField addressField; //学生信息 地址输入域

@FXML

private TextField numNameTextField; //查询 姓名学号输入域

private Integer studentId = null; //当前编辑修改的学生的主键

private ArrayList<Map> studentList = new ArrayList(); // 学生信息列表数据

private List<OptionItem> genderList; //性别选择列表数据

private ObservableList<Map> observableList= FXCollections.observableArrayList(); // TableView渲染的数据列表

（2）处理方法的设计，@FXML 表示是fxml相关的方法，初始化或者事件响应方法

1)@FXML public void initialize()，页面加载对象创建完成初始化方法，页面中控件属性的设置，初始数据显示等初始操作都在这里完成，其他代码都在事件处理方法里

2) public void onTableRowSelect(ListChangeListener.Change<? extends Integer> change) 点击学生列表的某一行，根据studentId ,从后台查询学生的基本信息，切换学生的编辑信息

3) @FXML void onQueryButtonClick()点击查询按钮，从后台根据输入的串，查询匹配的学生在学生列表中显示

4) @FXML protected void onAddButtonClick() 添加新学生， 清空输入信息， 输入相关信息，点击保存即可添加新的学生

5) @FXML protected void onDeleteButtonClick() 删除当前选中的学生

6) @FXML protected void onSaveButtonClick() 点击保存按钮，保存当前编辑的学生信息，如果是新添加的学生，后台添加学生

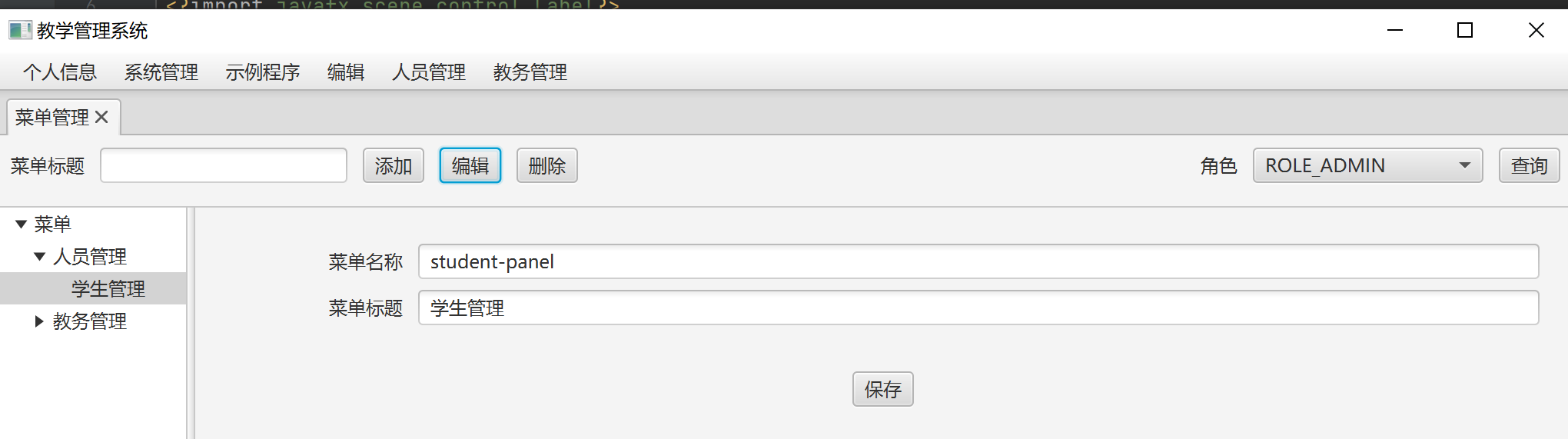
7) doNew() doSave() doDelete() 重写 ToolController 中的方法， 实现选择 新建，保存，删除 对学生的增，删，改操作

8) public void doExport() 导出学生信息表的示例 重写ToolController 中的doExport 这里给出了一个导出学生基本信息到Excl表的示例， 后台生成Excl文件数据，传回前台，前台将文件保存到本地

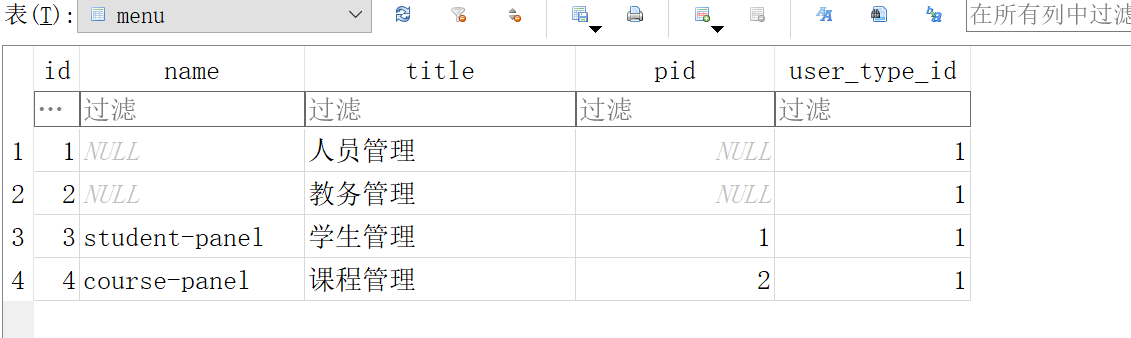
#### 8、添加学生管理功能的菜单

（1）通过菜单管理功能添加

菜单管理界面里，在菜单标题输入框中输入学生管理，点击添加按钮，添加学生管理菜单，选择学生管理菜单，点击学生管理，在菜单名输入框输入student\_panel 点击保存，可以完成菜单的添加



（2）通过数据库编辑工具进行添加，在menu表中添加一条记录就可以， user\_type\_id =1 是管理员(ROLE\_ADMIN)的菜单，pid=1 是输入人员管理的子菜单



（3）直接修改MainFrameController 方法 initialize中的代码，添加响应的菜单代码

item = new MenuItem();

item.setText("学生管理");

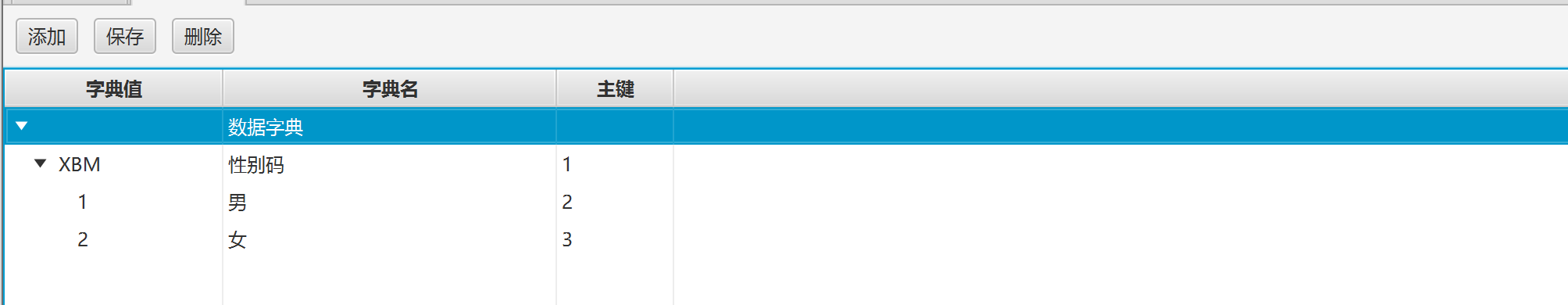
item.setOnAction(e -> changeContent("student-panel", "HTML编辑"));

menu.getItems().add(item);

#### 9、数据字典的添加

学生管理中，学生性别的选择输入是管理信息系统中对于具有特定类型集合选择输入方式，通过对于数据字典的设计实现下拉列表的选择的实现。开发框架已经在数据库中设计的数据库表dictionary, java-server 也给出了DictionaryInfo 实体类以及数据库操作接口和控制类数据处理的方法的实现，并在应用重启时将数据库中数据字典的所有数据加载到内存，为业务处理程序提供了字典列表和字典名获取方法，具体方法的使用，参见性别（gender）处理方法。为了方便大家扩展新功能所用的用于下拉列表的数据字典，可采用下面方式进行数据自点的维护。注：数据字典添加修改后需要重启前后台程序

1. 通过字典管理功能添加数据字典。



菜单管理、数据字典管理也为大家提供了TreeView和TreeTableView 组件的使用的示例。

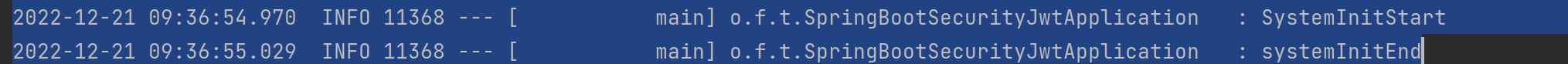
1. 通过直接在数据库dictionary表中添加记录



#### 9、前后端链接调试。

（1）启动java-server

运行SpringBootSecurityJwtApplication主类，控制台最后出现下面提示，表示正常启动，可以接收前端HTTP请求：



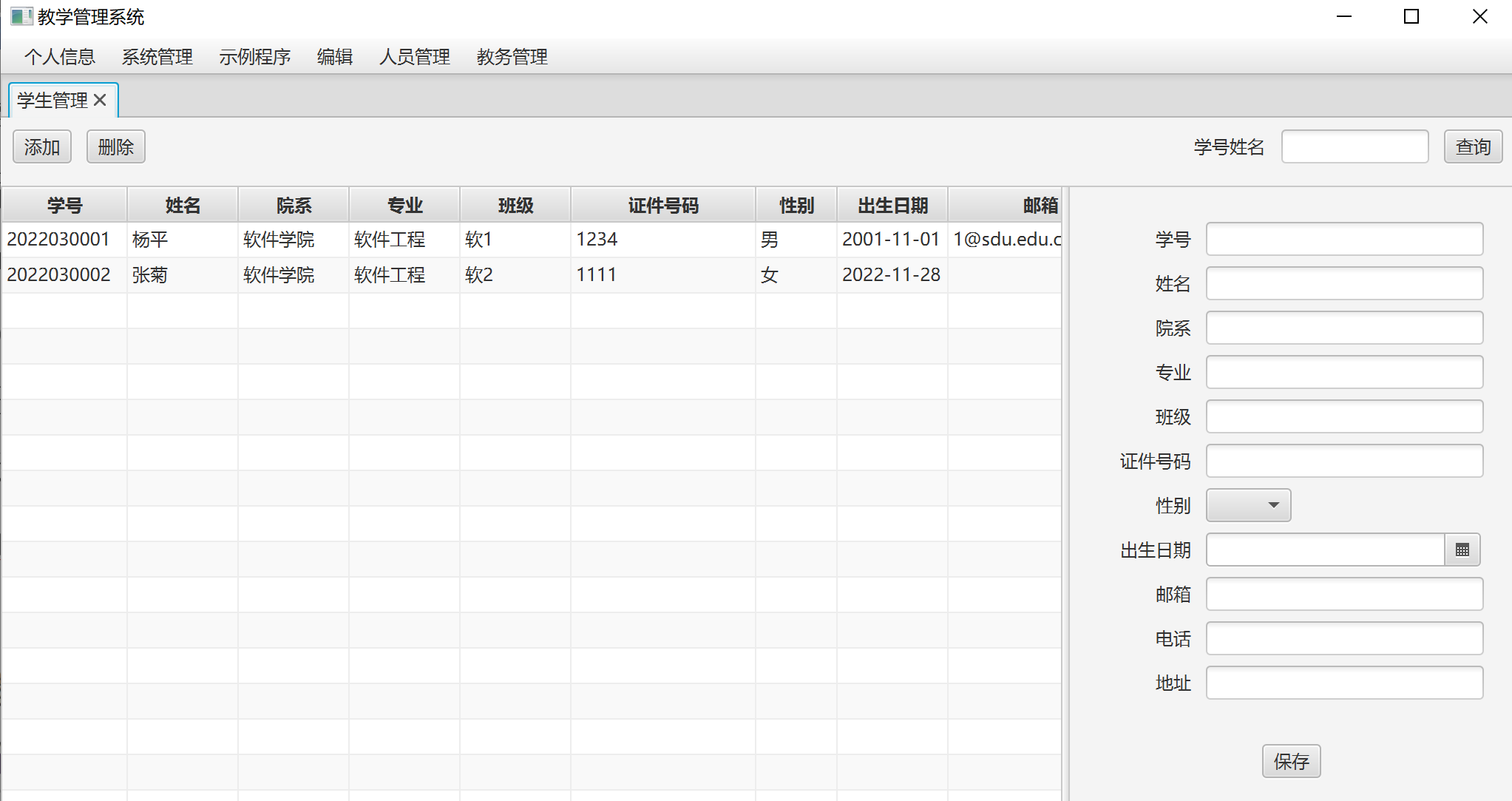
1. 如果有类有编译错误，则无法正常启动，请修正编译错误
2. 可以启动，但是程序报错，这个主要是由于Repository中的接口方法命名或者ＳＱＬ注解有问题，这些都不能使系统正常启动，请修改错误的代码后重启
3. java-server　增加的类和方法，除了基本的Java代码，类的继承和类、属性、方法都有严格的＠注释的编程规范的限制，请严格按照示例程序给出的注释规范进行开发。

（２）启动javafxclient

运行MainApplication主类，出现登陆界面，选择相应角色的账号和密码点击登录出现主框架：



点击学生管理出现：



初始化如果有数据请求，查看是否进入相应的StudentController处理函数，Debug查看数据是否正确，前端debug查看返回的数据是否正确，观察界面显示的信息是否正确。

注意点击相应的按钮调试每一个功能。

## 四、开发测试环境

#### 1、 开发环境 IDEA和创建工程，建议大家安装JDK17

系统开发环境列表

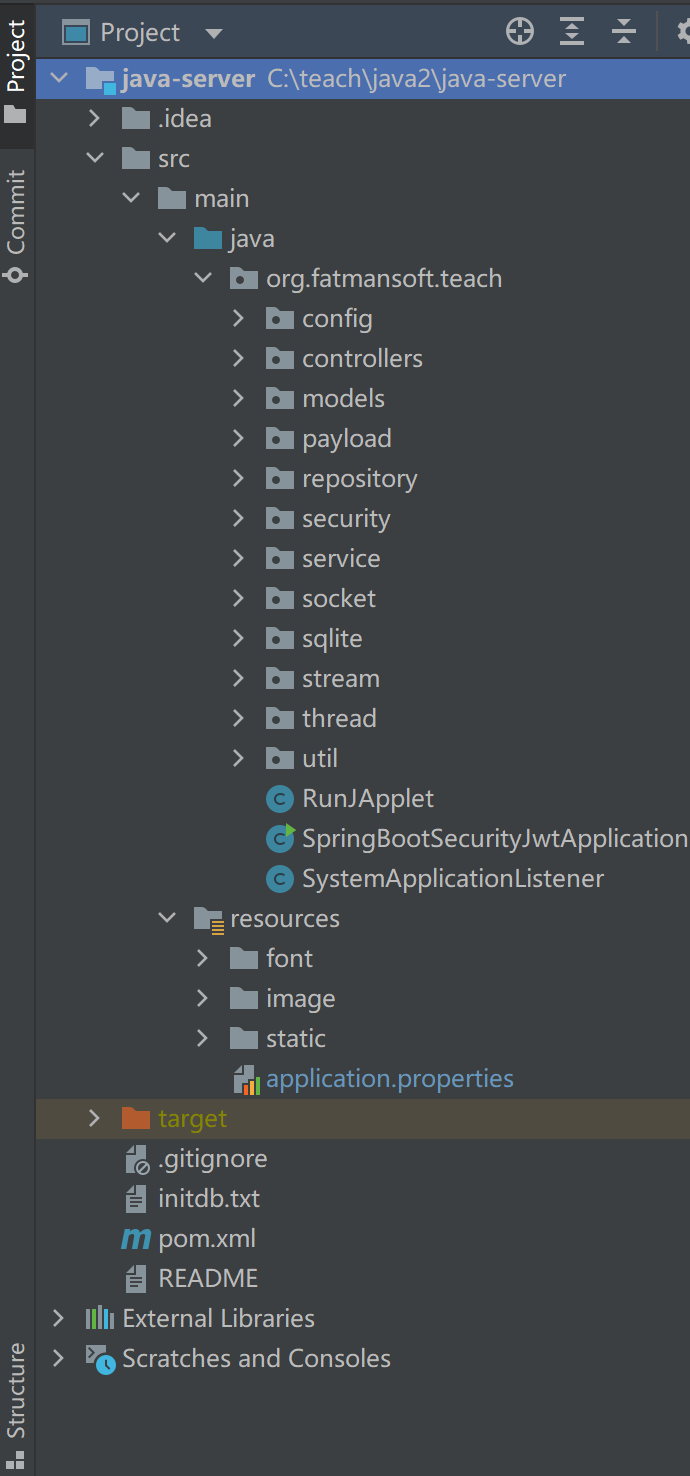
|  |  |  |
| --- | --- | --- |
| 软件 | 功能 | 备注 |
| ideaIC-2021.1.2.exe | IDEA 继承开发环境 | 上学期已经使用，不用重新安装，最好不要安装中文版 |
| jdk17.0.1.zip | Jdk 17版 | 由于Java FX 各种版本差异比较大，开发示例程序均在17版调试成功，建议直接只用17版 |
| javafx-sdk-17.0.1.zip | Javafx17版运行库 | IDEA开发不需要此库，打包生成java-fx-client.jar运行需要此库 |
| DB.Browser.for.SQLite-3.12.2-win64.msi | Sqlite数据库管理客户端，用于数据表的设计和数据修改 |  |
| SceneBuilder-19.0.0.msi | FXML可视化的编辑工具，用于设计开发fxml文件 |  |

（1）后台项目java-server的创建，两种创建工程的方法：

1）后台程序，学生可以创建一个Maven的项目，并将java-server pom.xml的内容复制到pom.xml里，以配置好所有的依赖，并将示例工程提供相应的类和文件复制到新工程中

2）将示例程序更换一个工程名字，在此基础上添加自己的代码。

目录结构：



3）application.properties关键配置

|  |
| --- |
| # App Properties  bezkoder.app.jwtSecret= bezKoderSecretKey  bezkoder.app.jwtExpirationMs= 86400000  server.port= 9090  spring.jpa.database-platform=org.fatmansoft.teach.sqlite.SQLDialect  spring.jpa.hibernate.ddl-auto=update  spring.datasource.url = jdbc:sqlite:/teach/java2/data.db  spring.datasource.driver-class-name = org.sqlite.JDBC  spring.datasource.username = admin  spring.datasource.password = admin  spring.jpa.generate-ddl=true  spring.servlet.multipart.max-file-size=50MB  spring.servlet.multipart.max-request-size=50MB  spring.jmx.enabled=false  spring.http.encoding.charset=UTF-8  spring.http.encoding.enabled=true  spring.http.encoding.force=true  attach.folder=/teach/java2/ |

application.yml关键配置

|  |
| --- |
| # App Properties  bezkoder:  app:  jwtSecret: bezKoderSecretKey  jwtExpirationMs: 86400000  server:  port: 9090  spring:  datasource:  url: jdbc:sqlite:/teach/web/data.db  driver-class-name: org.sqlite.JDBC  username: admin  password: admin  jpa:  database-platform: org.fatmansoft.teach.sqlite.SQLDialect  generate-ddl: true  hibernate.ddl-auto: update  servlet:  multipart:  max-file-size: 200MB  max-request-size: 200MB  jmx:  enabled:false  # Mail server properties  mail:  host: smtp.sdu.edu.cn  port: 25  username: xqli@sdu.edu.cn  password: 111  properties:  mail:  smtp:  auth: true  connectiontimeout: 5000  timeout: 5000  writetimeout: 5000  starttls:  enable: true  http:  encoding:  charset: UTF-8  enabled: true  force: true  attach:  folder: /teach/web/  logging:  config: classpath:loggback.xml |

application.properties和application.yml文件的作用是相同的，只要用一个就可以，目前新的使用application.yml的比较多

(A) server.port= 9090 服务器web服务的端口，对应前端客户端

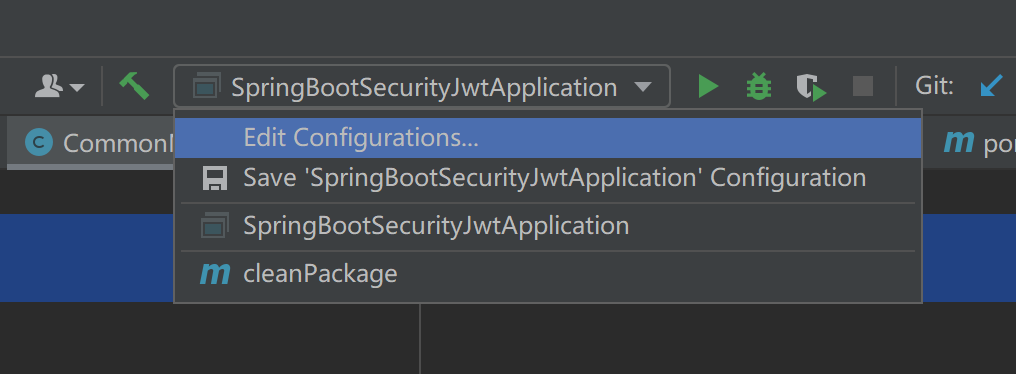
HttpRequestUtil类中的serverUrl = <http://localhost:9090>， 可以改变，只要对应就好

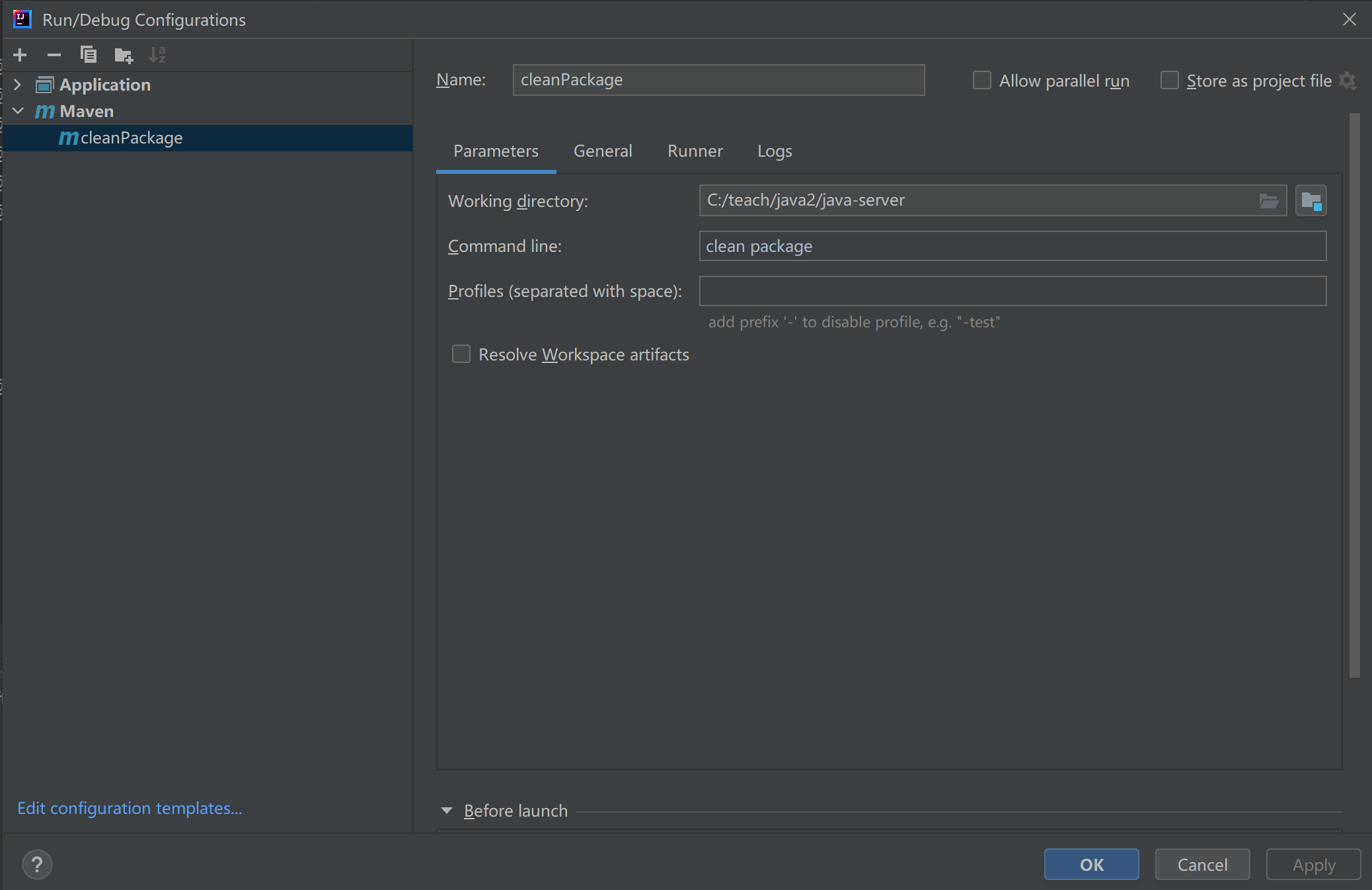
(B) spring.datasource.url = jdbc:sqlite:/teach/java2/java.db 数据库文件的路径，可以修改

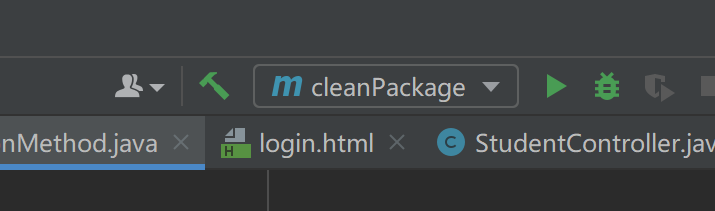
(C) attach.folder=/teach/java2/ 服务器保存文件的路径 个人照片存在/teach/java2/photo目录。

3）打包生成 java-server.jar

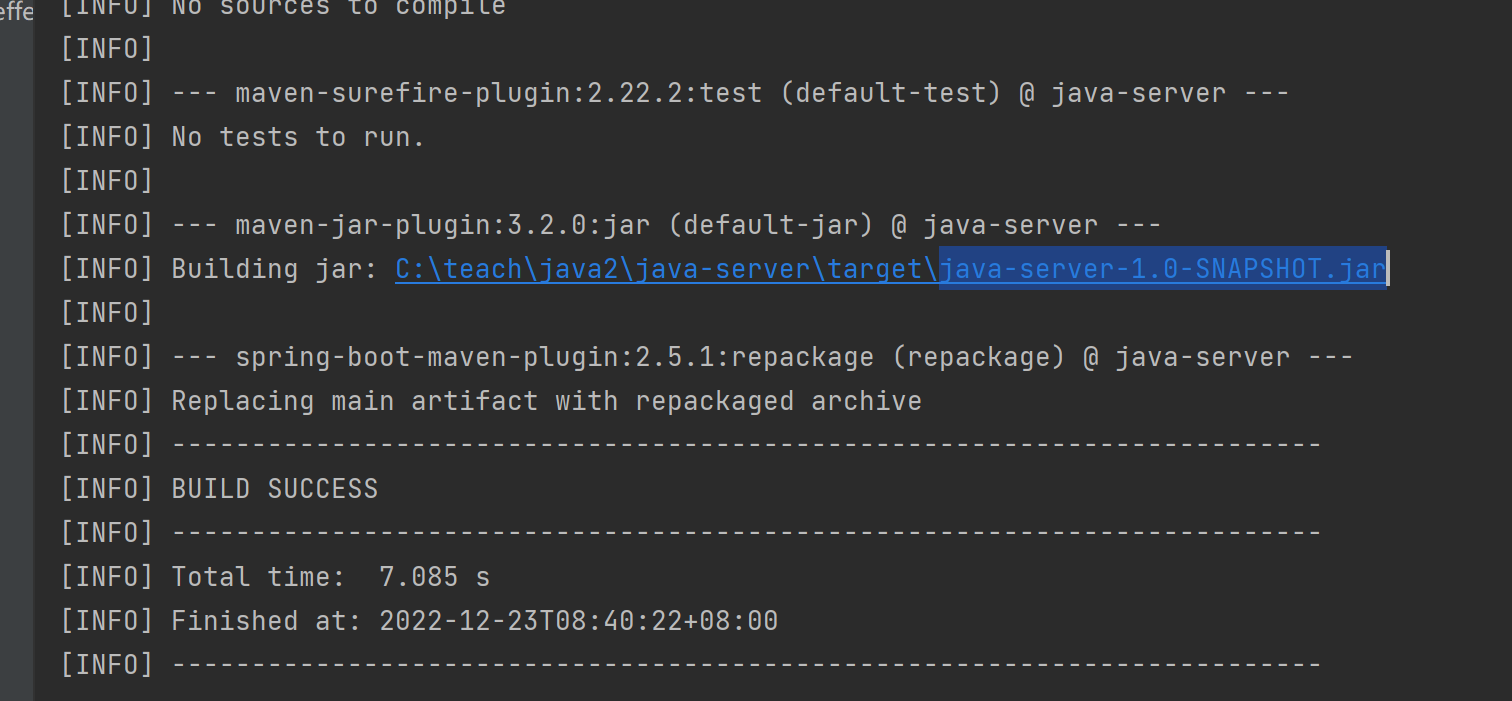
A) 选择 Edit Configurations添加 maven moudle cleanPackage 配置







运行cleanPackage 则生成可以运营的java-server-1.0-SNAPSHOT.jar



B）运行 java-server-1.0-SNAPSHOT.jar

c:\dev\Java\jdk17.0.1\bin\java -jar C:\teach\java2\java-server\target\java-server-1.0-SNAPSHOT.jar

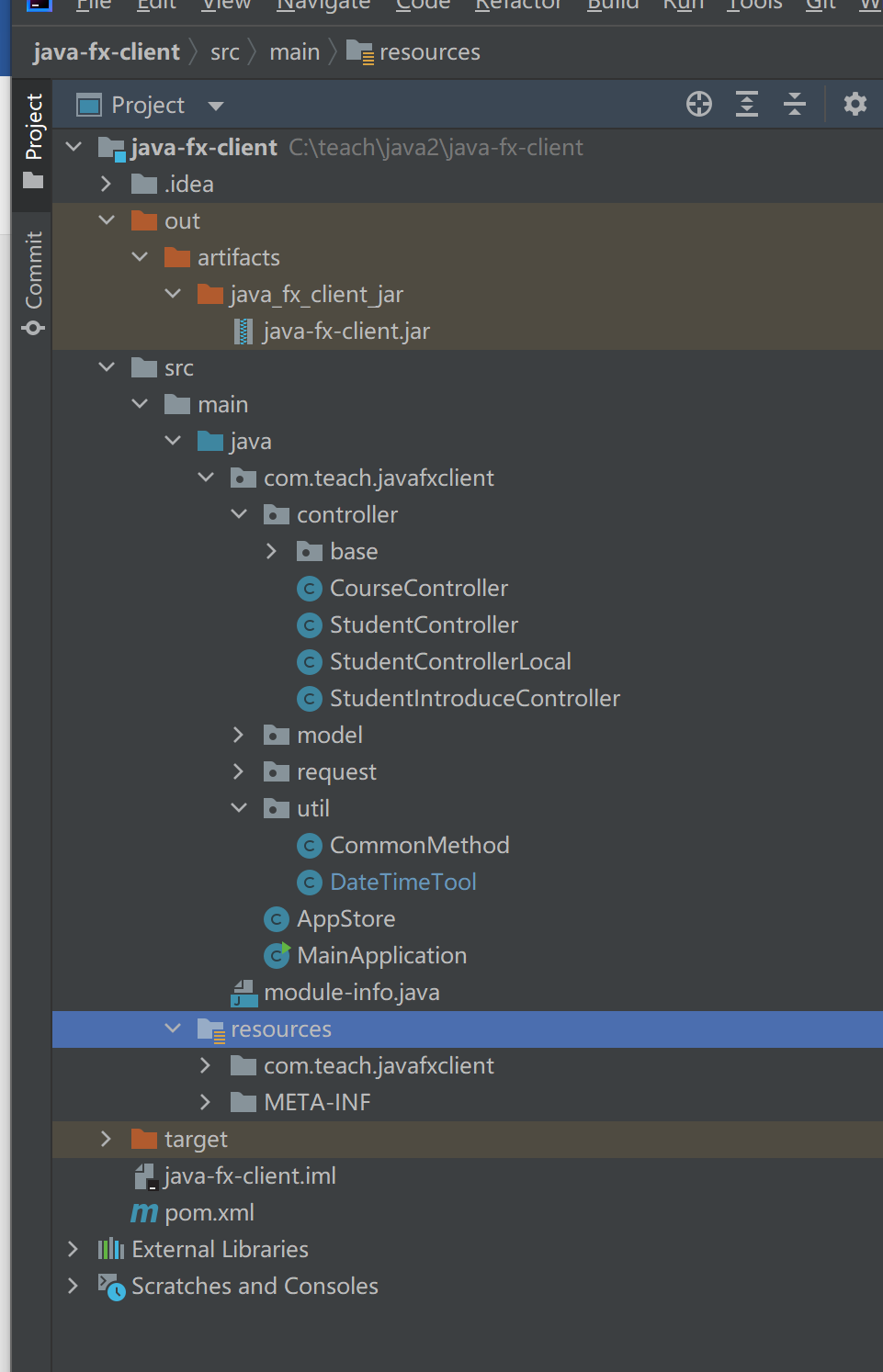
这里：c:\dev\Java\jdk17.0.1\bin\ jdk17安装路径

C:\teach\java2\java-server\target\java-server-1.0-SNAPSHOT.jar java-server-1.0-SNAPSHOT.jar的路径

（2）前台项目的创建，两种创建工程的方法：

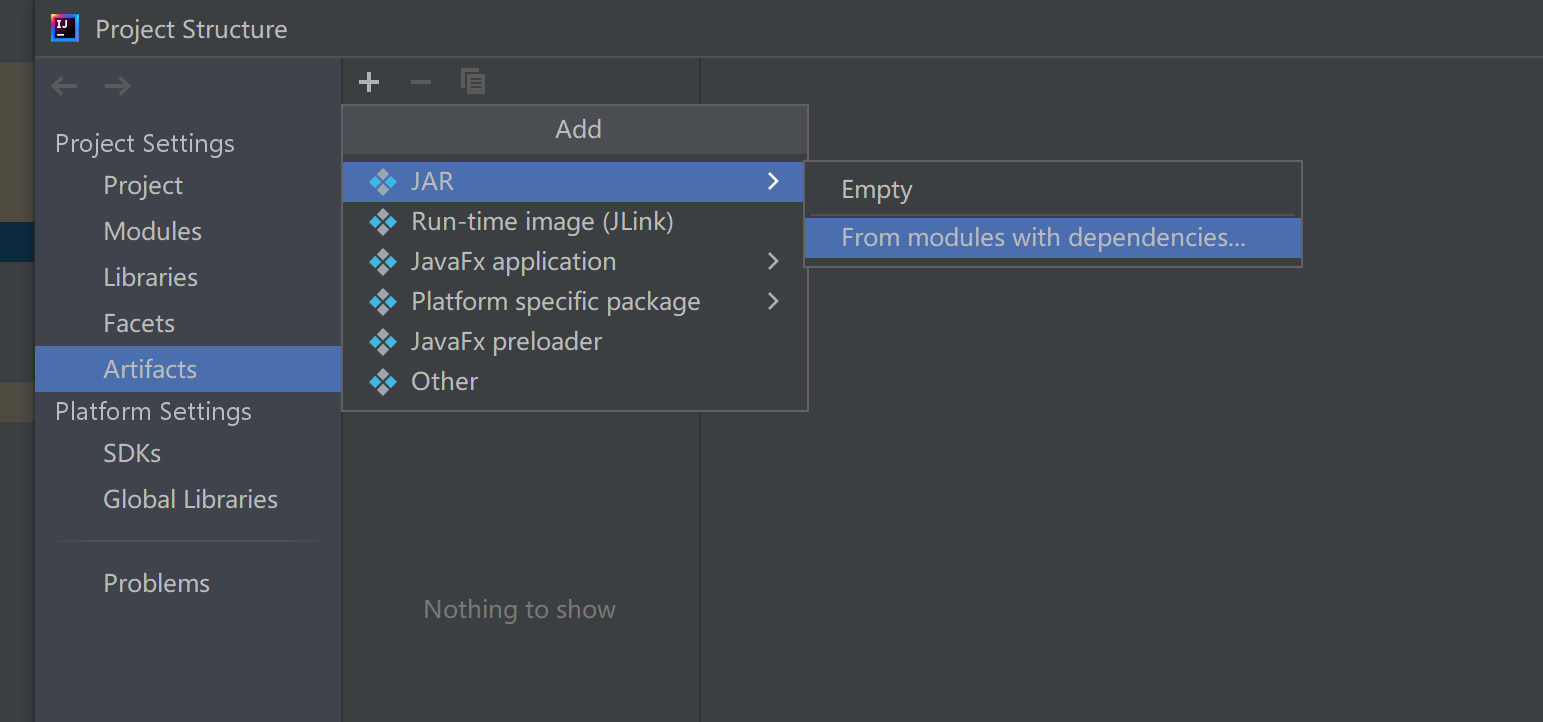
1）前台程序，学生可以创建一个JavaFX的项目，并将javafxclient中 pom.xml的内容复制pom.xml里，以配置好所有的依赖，并将示例工程提供相应的类和文件复制到新工程中

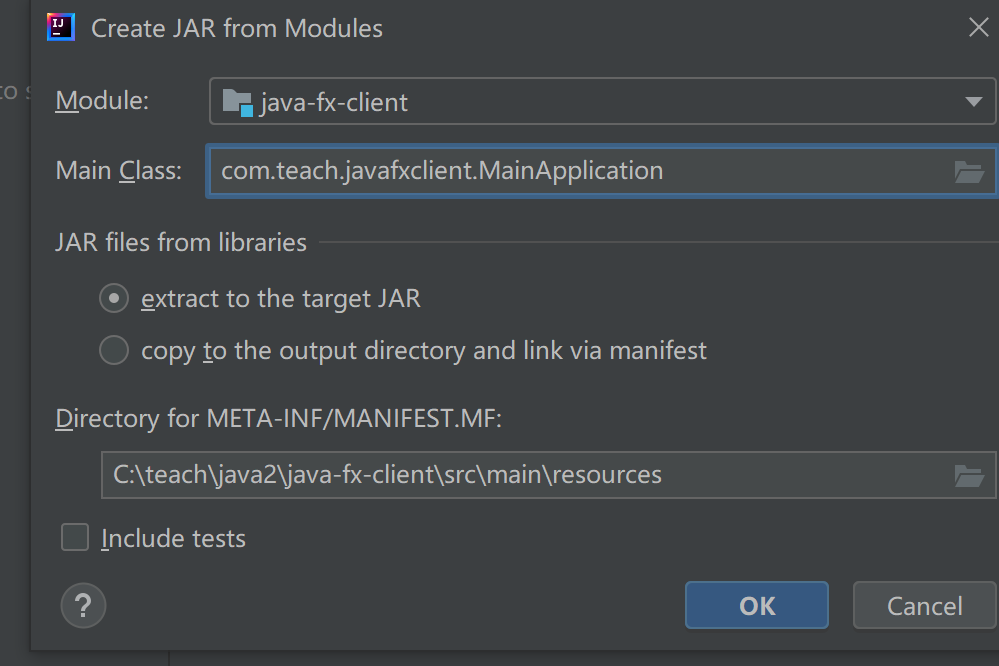
2）将示例程序更换一个工程名字，在此基础上添加自己的代码。目录结构为

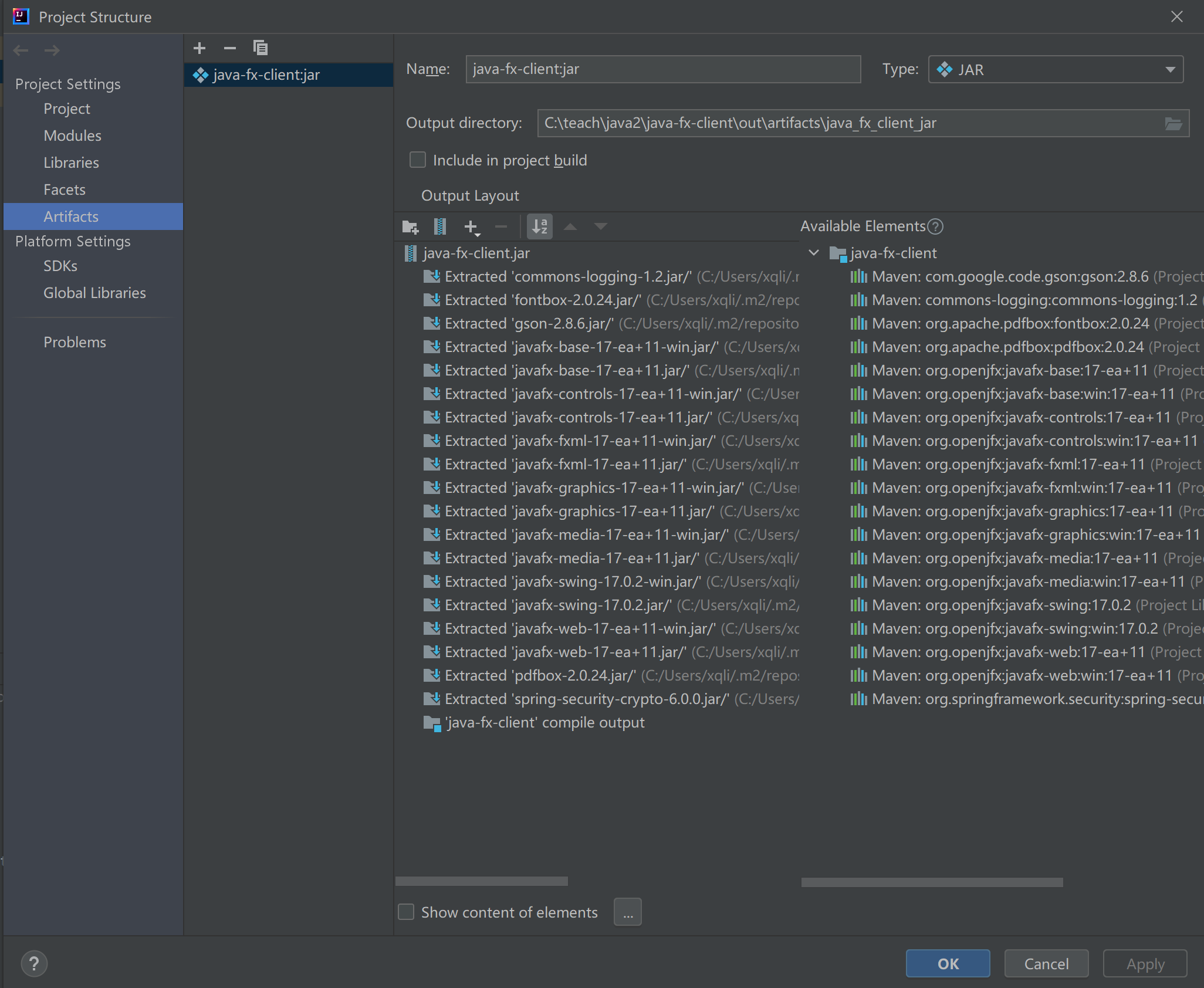


3）打包生成可运行的java-fx-client.jar

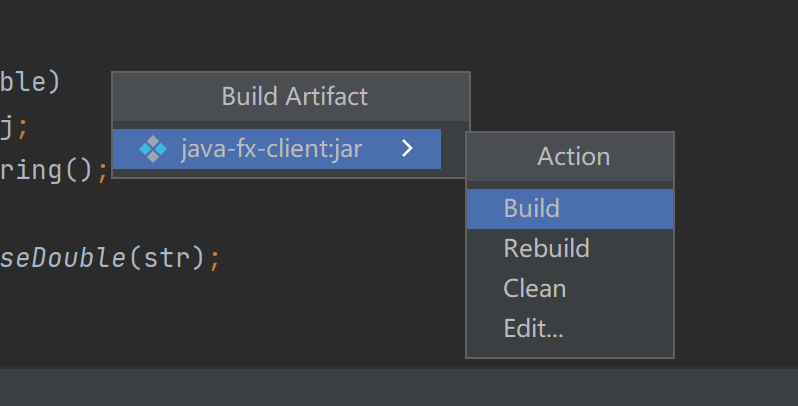
A) 选择菜单file 中的 Project Structure 选择Artifacts 添加jar



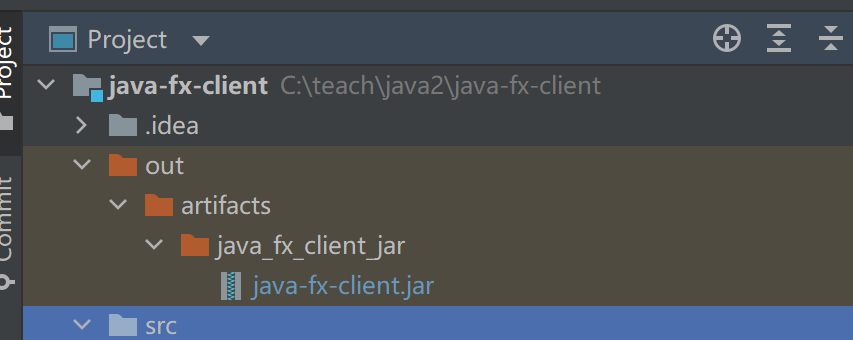




B)选择building->builder Artifacts 出现下面的界面。



选择Rebuild Action 则可以重新生成java-fx-client.jar



C)运行java-fx-client.jar

c:\dev\Java\jdk17.0.1\bin\java --module-path "C:\dev\Java\javafx-sdk-17.0.1\lib" --add-modules javafx.controls,javafx.fxml -jar C:\teach\java2\java-fx-client\out\artifacts\java\_fx\_client\_jar/java-fx-client.jar

其中：c:\dev\Java\jdk17.0.1\bin 17版本jdk的安装路径

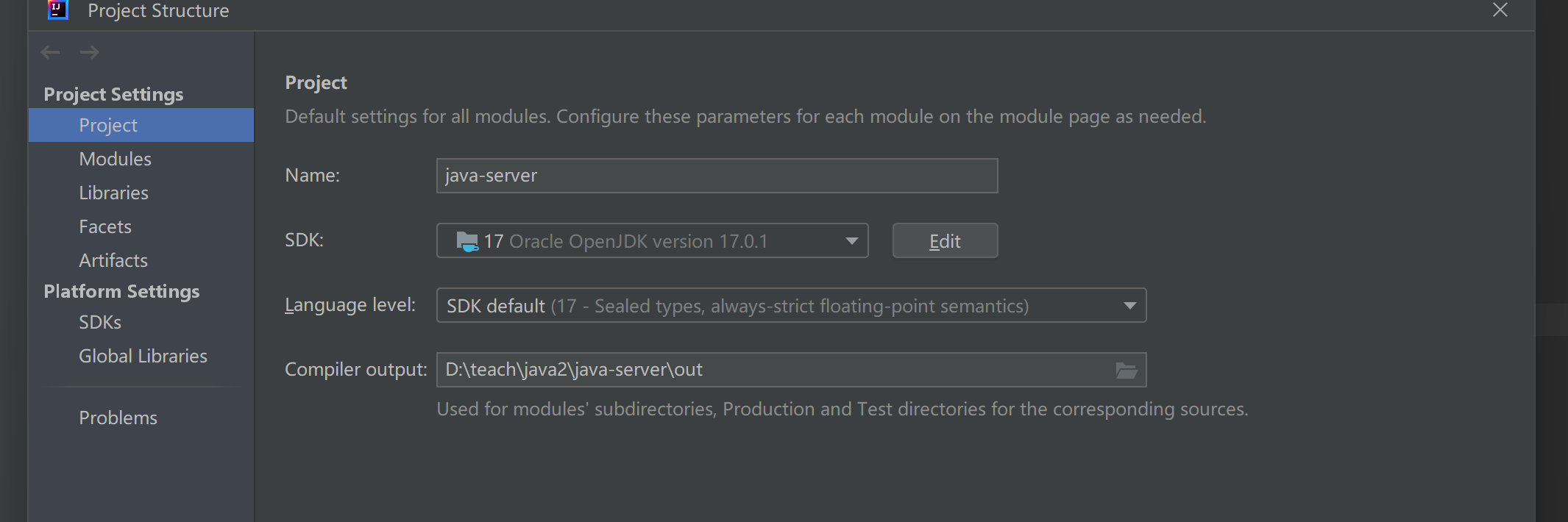
C:\dev\Java\javafx-sdk-17.0.1\lib 17版本 JavaFX的运行路径

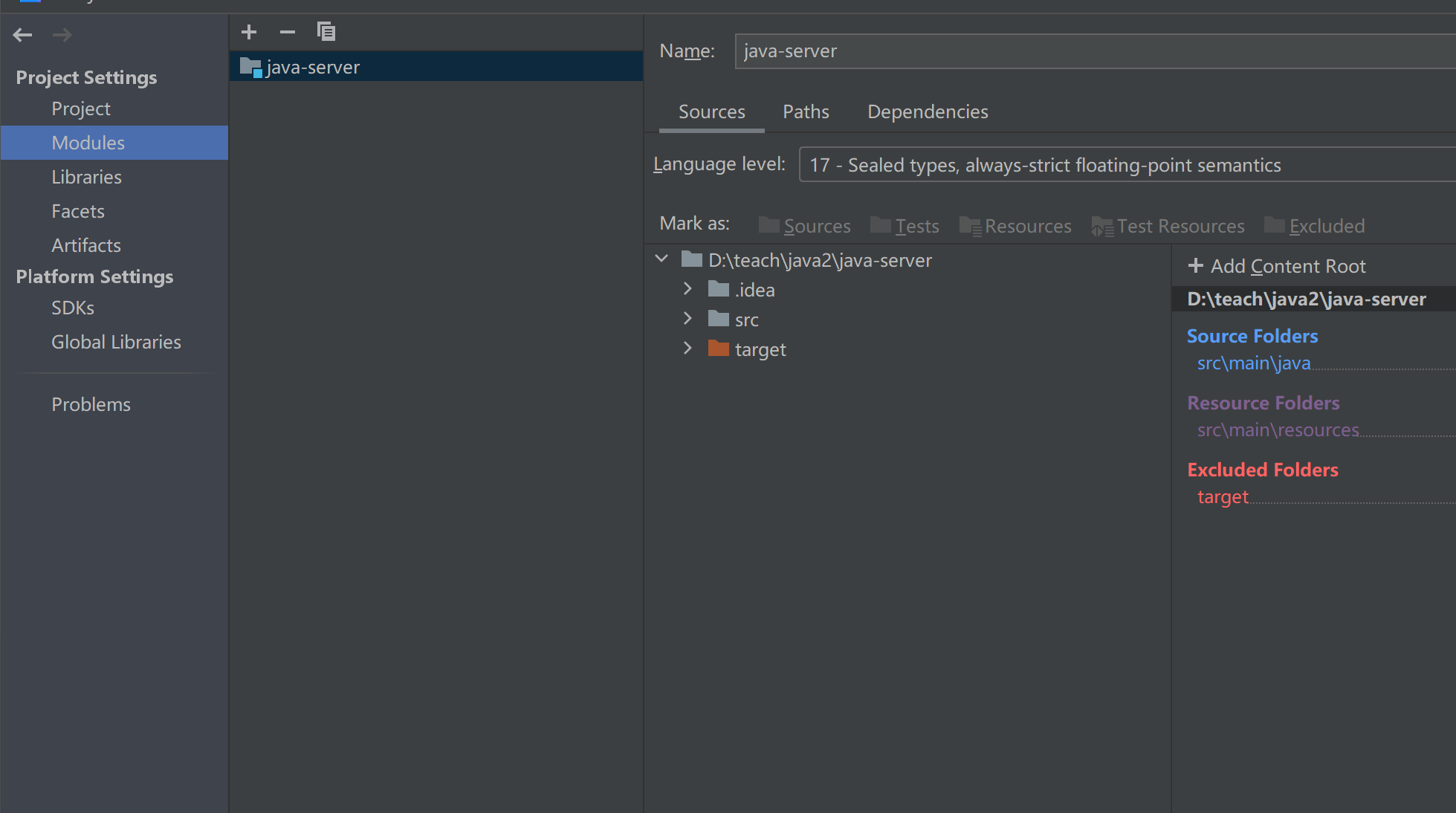
C:\teach\java2\java-fx-client\out\artifacts\java\_fx\_client\_jar/java-fx-client.jar 客户端java-fx-client.jar路径

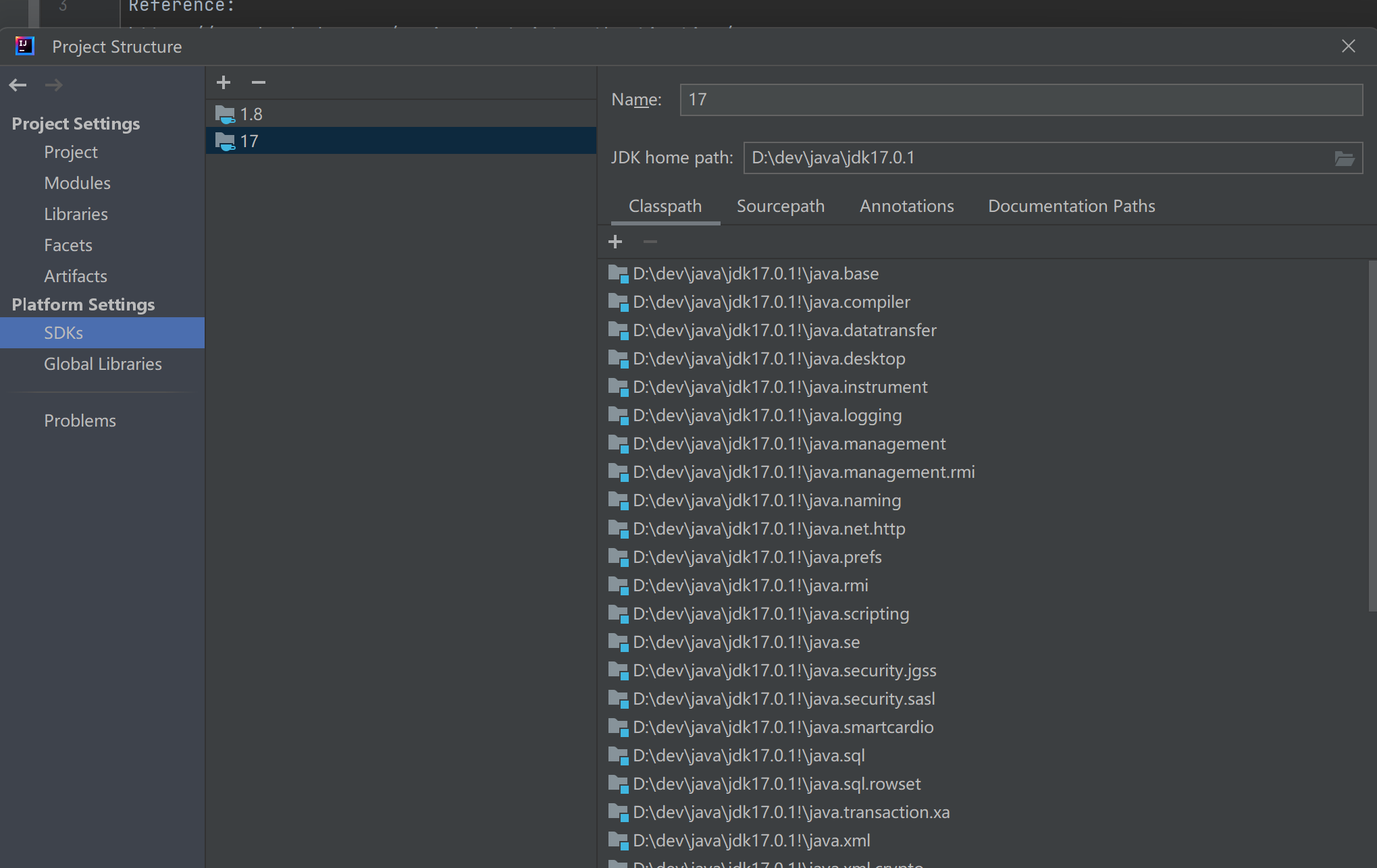
#### 2、JDK的相关配置和系统自动安装依赖

（1）JDK17的相关配置

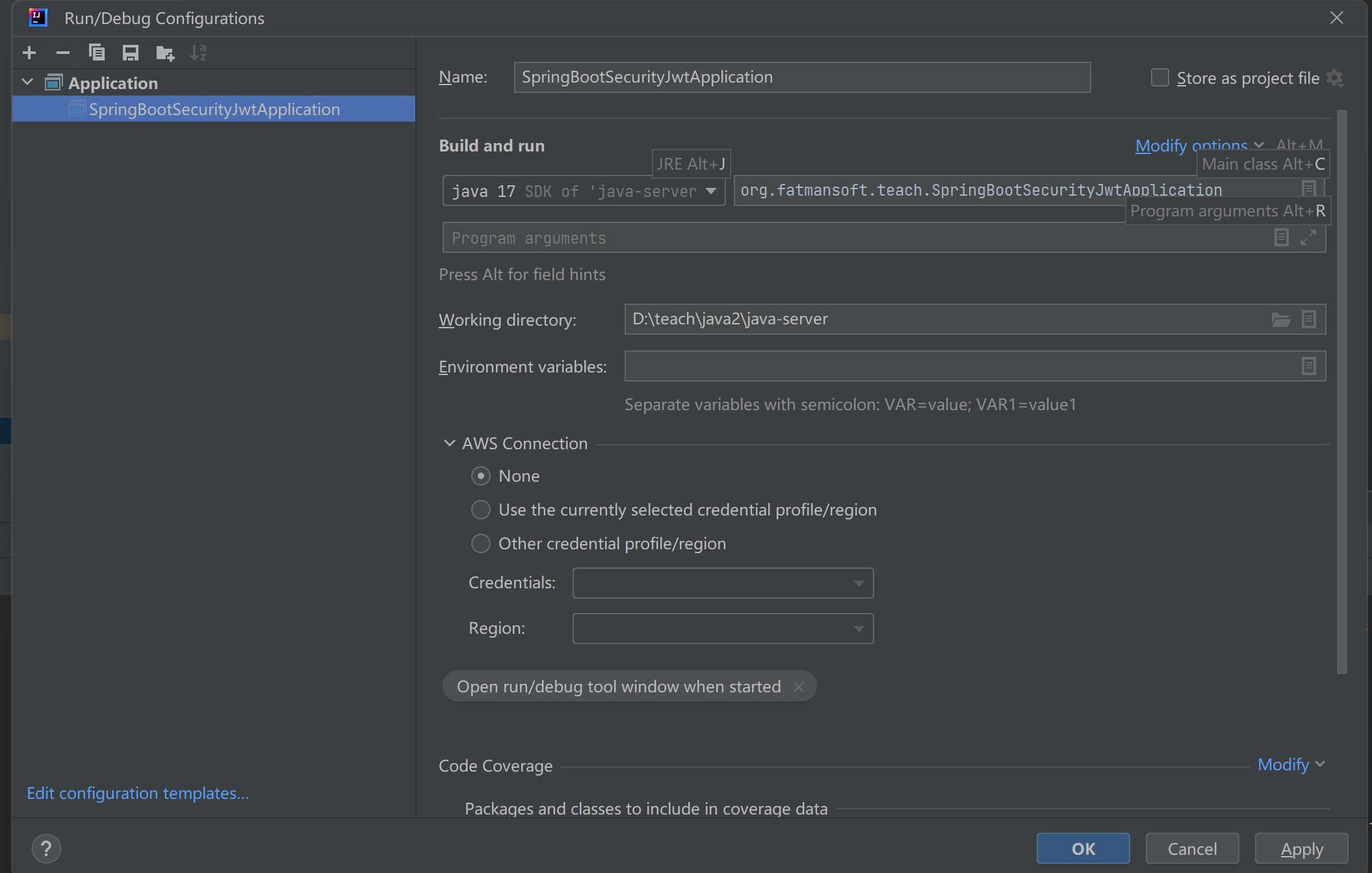
（A） files->Project Setting



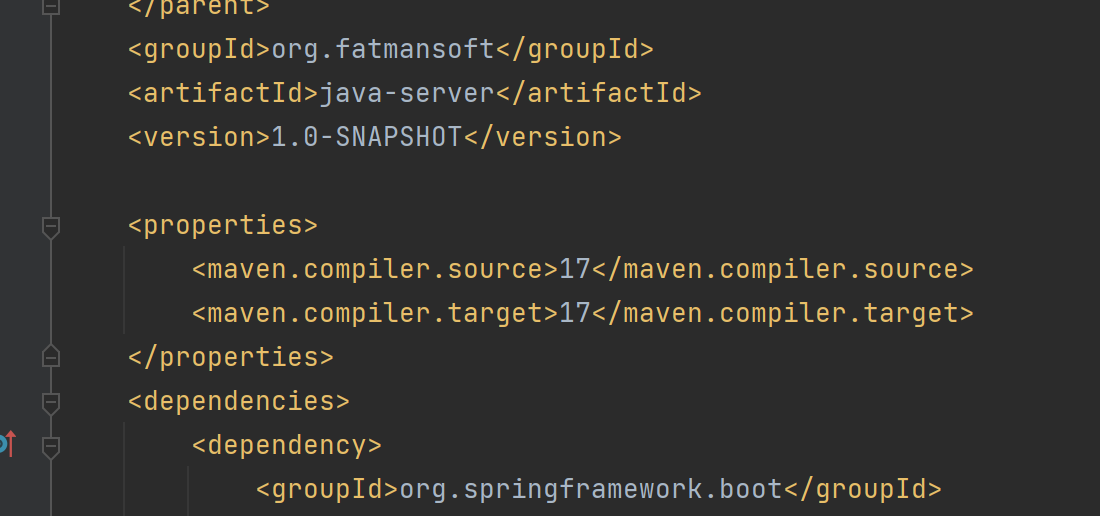




Run/DebugConfigure



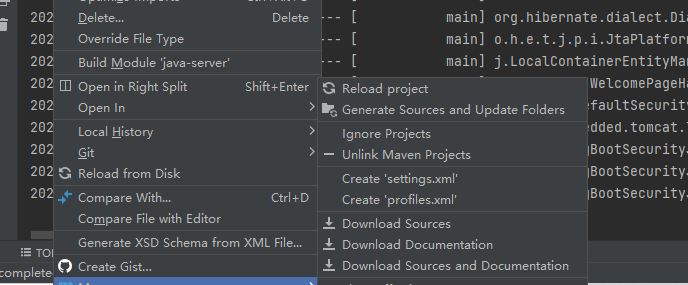
java-server：Pom.xml 配置



java-fx-client: pom.xml



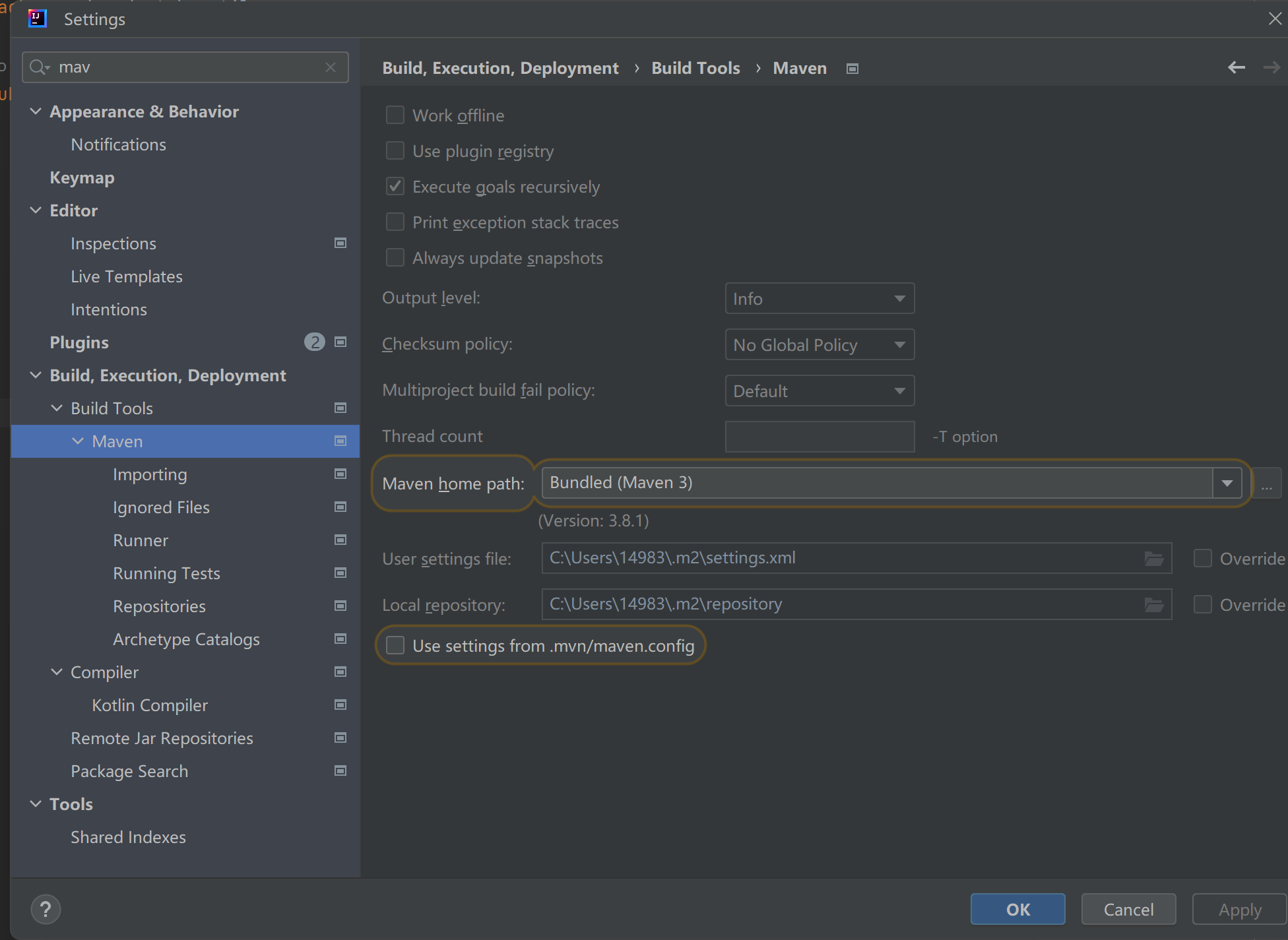
(2) 按照依赖，修改pom.xml文件，增加依赖的说明，点击pom.xml 右键菜单 maven->reload Prokject



请耐心等待，等安装完成，如果不能顺利更新，可多次执行reload 直到编译不再报错，即可以运行， 两个工程依赖的包有所不同，两个工程第一次都需要安装依赖，一旦依赖安装好，后面就不再需要安装依赖了。

如果无法maven reload 更新依赖的.jar 尝试删除本地的maven 依赖的库

选择File-Settings 搜索maven 找到repositories 的 local 路径，



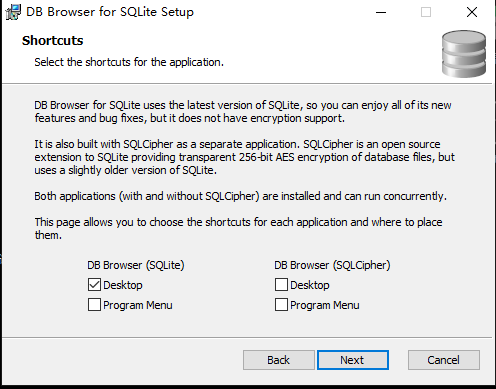
C:\users\14983\.m2\repository(每个用户的路径略有差别，这里用户是14983)

删除 respository里的所有文件，重新执行。

#### 3、DB.Browser.for.SQLite客户端

点击DB.Browser.for.SQLite-3.12.2-win64.msi，开始安装，按照提示一步步操作既可以完成安装， 安装后就可以用此工具打开数据，数据的文件在项目所在的目录下java.db文件

安装时注意SQLite Desktop 选中就可以。



**重要提示：程序运行时请注意不要用客户端打开数据库进行操作，打开后对文件进行加锁，我们的程序就无法正常对数据库进行修改删除创建操作了。**

我们给大家提供的java.db已经进行了基本的初始操作，大家可以在此基础上进行操作，如果想回到原始的状态，只需要用这个文件替换当前的数据库文件就好。

另外通常一般项目开发可以用开源的MYSQL数据库这个需要下载相关软件，进行安装，开发时要首先启动Mysql servcer，有兴趣的同学可以尝试，我们的工程的java-server :application.properties.Xml也需要调整。

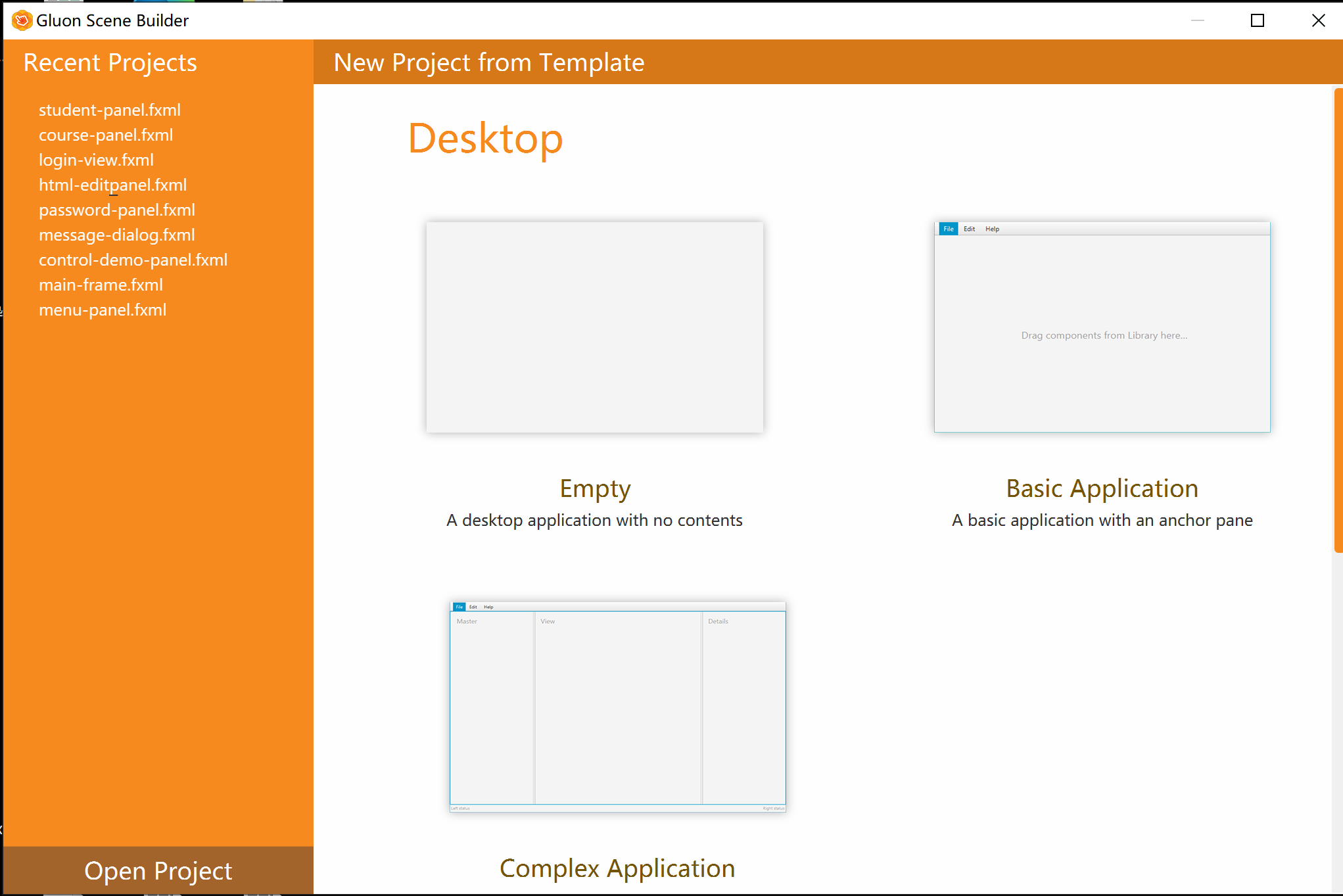
SQLite数据源的配置

spring.datasource.url = jdbc:sqlite:/teach/java2/java.db  
spring.datasource.driver-class-name = org.sqlite.JDBC  
spring.datasource.username = admin  
spring.datasource.password = admin

Mysql数据源的配置

spring.datasource.url= jdbc:mysql://localhost:3306/java\_db  
spring.datasource.username= root  
spring.datasource.password= qlscadmin

4、SceneBuilder-19.0.0.msi 安装，直接安装就可以使用



## 附一、Student实现的案例代码

#### 源代码文件说明

（1）java-server项目

1) Student.jar

2)StudentRepository.jar

3)StudentController.jar

（2）javafxclient项目

1) student-panel.fxml

2)StudentController.jar

## 附件二、技术文档参考链接

#### Java FX,FXML语言介绍

https://www.runoob.com/xml/xml-intro.html

https://www.w3cschool.cn/xml/dict

Learn JavaFX 8.pdf

#### 2、SQl 语句介绍

https://www.runoob.com/sql/sql-intro.html

#### 3、Java String JPA Java数据库操作文档

https://docs.spring.io/spring-data/jpa/docs/2.5.3/reference/html/#jpa.query-methods.at-query