生物测序实验管理 Java 应用开发

作者：周兵

红星电子音像出版社

ISBN 978-7-900512-46-8 pdf

题名：生物测序实验管理 Java 应用开发

作者：周兵

标识：ISBN 978-7-900512-46-8 pdf

出版者：红星电子音像出版社

出版地：江西南昌

编辑加工者：涂丽

制作单位：红星电子音像出版社

制作时间：2025 年 8 月

容量：15M

前　言

在21世纪第三个十年开启之际，生物测序技术已成为解锁遗传密码、探索生命奥秘的重要工具。从基因组的深度解析到个性化医疗的推进，生物测序技术的应用正不断拓展其边界，为科学研究与临床应用带来革命性的变化。然而，随着测序数据量的激增和实验流程的复杂化，如何高效管理生物测序实验，确保数据的准确性和可追溯性，成为亟待解决的关键问题。在这一背景下，结合强大的Java编程语言进行生物测序实验管理应用开发，显得尤为重要与迫切。

为此，我们编写了这本《生物测序实验管理Java应用开发》教材。本教材旨在为读者提供一条从理论到实践的完整学习路径，将生物测序实验管理的实际需求与Java应用开发的技术手段紧密结合。教材内容涵盖生物测序实验的基础知识、实验流程管理的关键要点，以及Java编程在生物测序数据处理、存储、分析与可视化中的具体应用。通过一系列精心设计的项目案例，读者将学习如何构建高效、稳定的生物测序实验管理系统，掌握从需求分析、系统设计、编码实现到测试部署的全流程开发技能。

本教材具有以下几个显著特点：其一，一体化教学。教材将生物测序实验管理与Java应用开发融为一体，使读者能够在理解实验管理需求的基础上，深入掌握Java编程技术的应用，实现知识的无缝衔接与综合运用。其二，实践导向。通过真实的项目案例和详细的代码示例，引导读者在实践中学习，在学习中实践。每个项目案例都经过精心设计，旨在帮助读者理解并解决生物测序实验管理中的实际问题。其三，系统性学习。内容编排由浅入深，循序渐进，从Java编程基础到高级应用开发，再到生物测序实验管理的专业实践，确保读者能够系统地构建知识体系，逐步提升开发能力。其四，跨学科融合。教材注重生物学与计算机科学的跨学科融合，旨在培养具备生物信息学背景和Java编程技能的复合型人才，以适应生物科技行业的多元化需求。

在编写过程中，我们力求内容的准确性与实用性，但鉴于生物测序技术和Java应用开发领域的快速发展，书中难免存在不足之处，恳请广大读者批评指正。我们期望这本教材能够成为生物信息学专业学生、生物科技行业从业者以及对生物测序实验管理Java应用开发感兴趣的读者的宝贵资源，助力他们在生物测序实验管理与应用开发的道路上不断前行，为生物科技的发展贡献力量。

编者

目 录

任务 1 Java 应用开发认知与安装调试运行

项目描述

开发环境搭建是进行 Java 项目开发的开始，在实际参加工作入职的第一天就是安装调试电脑的开发环境，等做完这一切后才进行项目开发。Java 的开发环境包括：Java 开发工具包 JDK、开发工具 Eclipse、项目构建工具 Maven、Web 服务器 Tomcat、数据库 Maria DB、代码版本管理 Git。

项目目标

1. 能安装配置 Java 开发环境以及创建工程项目。
2. 能安装与使用 MariaDB 数据库。
3. 能安装配置 Git 以及对代码进行版本管理。

活动一 安装配置 Java 开发环境及创建工程项目

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。项目使用 Java 语言进行开发，现在要求我们对开发电脑进行开发环境搭建，来吧，让我们一起进行开发环境搭建吧！

任务目标

1. 正确安装及配置 Java 开发需要的环境。
2. 能完成 Tomcat 安装及环境配置。
3. 能完成 Maven 安装及环境配置。
4. 能安装 Eclipse 及完成配置。
5. 创建 Eclipse 工程项目并且运行。

任务分析

1. Java 程序语言需要安装一个什么工具包才能运行？项目使用 Spring Boot3 最低版本应该使用哪个版本？
2. Java Web 服务器中间使用哪个？
3. Java 项目开发工具使用哪个？
4. 项目构建工具使用哪个？
5. 如果创建工程项目？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、Java 程序设计语言简介

Java 是 1995 年由 Sun 公司推出的一门极富创造力的面向对象的程序设计语言，它是由有“Java 之父”之称的 Sun 研究院院士詹姆斯·高斯林博士亲手设计而成的，正是他完成了 Java 技术的原始编译器和虚拟机。

最开始 java 并没有名字，高斯林注意到自己办公室外一棵茂密的橡树，这种树在硅谷当时很常见，所以他们就将新语言命名为橡树(Oak)，但是，在他们申请商标时，发现 Oak 是另外一个公司的名字，所以不得不重新取名。在命名征集会上，大家提出了很多名字，他们当时正喝着印尼爪哇(Java)岛出产的咖啡，其中有一位成员灵机一动，说叫 Java 咋样，大家都表示喜欢。

Java 程序首先被编译成字节码，然后可以在 Java 虚拟机(JVM)上运行，JVM 可以解释执行字节码，也可以将字节码直接执行，其语法规则和 C++类似。同时，Java 也是一种跨平台的程序设计语言，用 Java 语言编写的程序，可以运行在任何平台和设备上，如跨越 IBM 个人电脑、MAC 苹果计算机、各种微处理器硬件平台，以及 Windows、UNIX、OS/2、Mac OS 等系统平台，真正实现“一次编写，到处运行”。Java 非常适于企业网络和 Internet 环境，并且已成为 Internet 中最具有影响力、最受欢迎的编程语言之一。

与目前常用的 C++相比，Java 语言简洁得多，而且提高了可靠性，除去了最大的程序错误根源，此外它还有较高的安全性，可以说，它是有史以来最为卓越的编程语言。

Java 语言编写的程序既是编译型的，又是解释型的。程序代码经过编译之后转换为一种称为 Java 字节码的中间语言，Java 虚拟机(JVM)将对字节码进行解释和运行。编译只进行一次，而解释在每次运行程序时都会进行。编译后的字节码采用一种针对 JVM 优化过的机器码形式保存，虚拟机将字节码解释为机器码，然后在计算机上运行。Java 语言程序代码的编译和运行过程如图 1-1-1 所示。

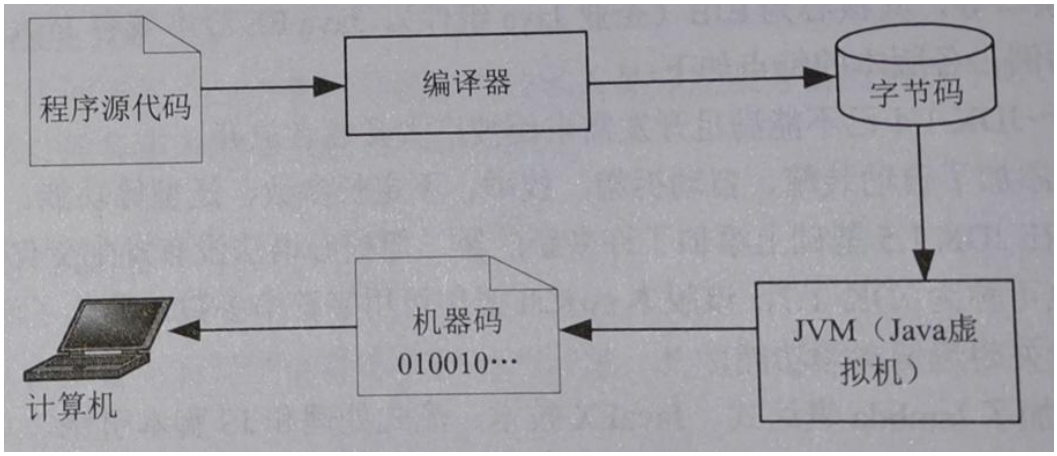


图 1-1-1 Java 程序的编译和运行过程

二、Java 的开发工具包安装及环境配置

1. 下载 JDK

要编译和执行 Java 程序，JDK（Java Developers Kits）是必备的，Oracle JDK 是最完善的商业 JDK，在 Oracle 官网可以下载 JDK 安装包，下载地址：https://www.oracle.com/java/technologies/downloads/，浏览器打开如图 1-1-2 所示。

Oracle 公司推出的 JDK 已经升级到了很多个版本，根据我们项目开发的框架我们选择 JDK 17 就可以了。

根据电脑系统，我们现在 Windows 下面的 x64 Installer 进行下载，下载完成的文件是可执行文件，直接双击运行即可开始安装。

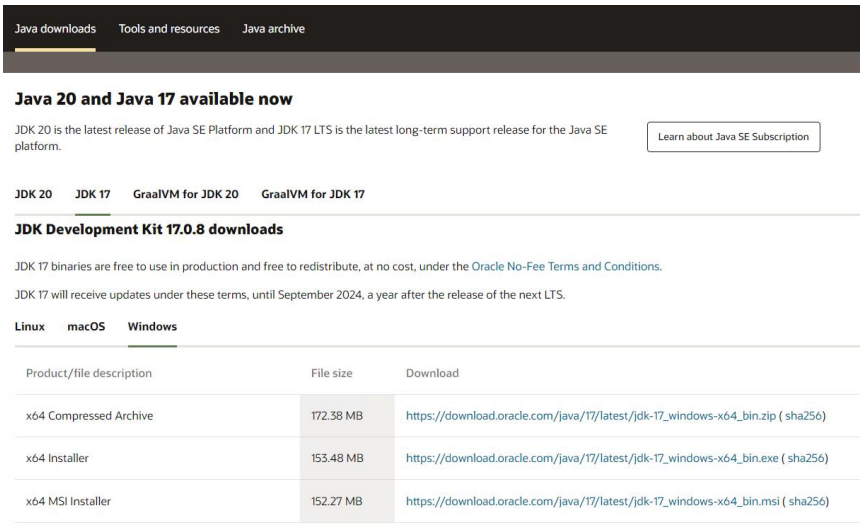


图 1-1-2 Oracle JDK 的下载页面

2. 安装 JDK

双击下载的可执行文件，出现安装界面如图 1-1-3 所示。默认安装，一路选择【下一步】按钮即可完成安装。JDK 默认安装在 C:\Program Files\Java\jdk-17 路径下。

图 1-1-3 JDK 安装界面

3. 配置环境变量

在电脑上找到我的电脑，右击【此电脑】→【属性】→【高级系统设置】→【环境变量】→【系统变量(s)】→【新建】出现新建环境变量窗口，如图 1-1-4 所示。

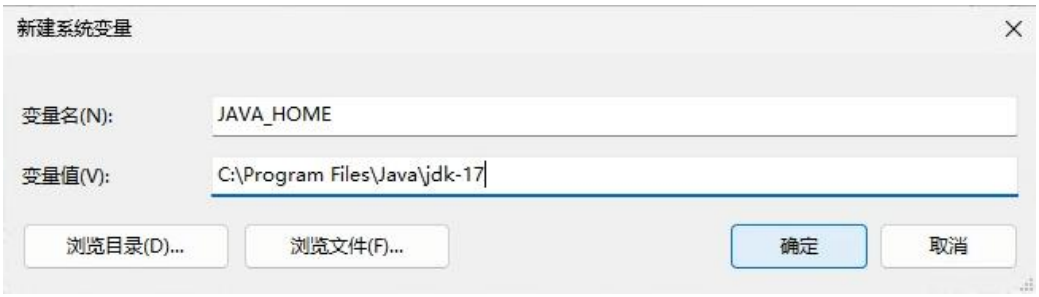


图 1-1-4 新建环境变量窗口

在【变量名】和【变量值】中分别输入 JAVA_HOME 和 C:\Program Files\Java\jdk-17，然后点击【确定】，注意这里的变量值是 JDK 安装的实际路。

在变量名为 path 的一项中加入%JAVA_HOME%\bin，如图 1-1-5 所示。

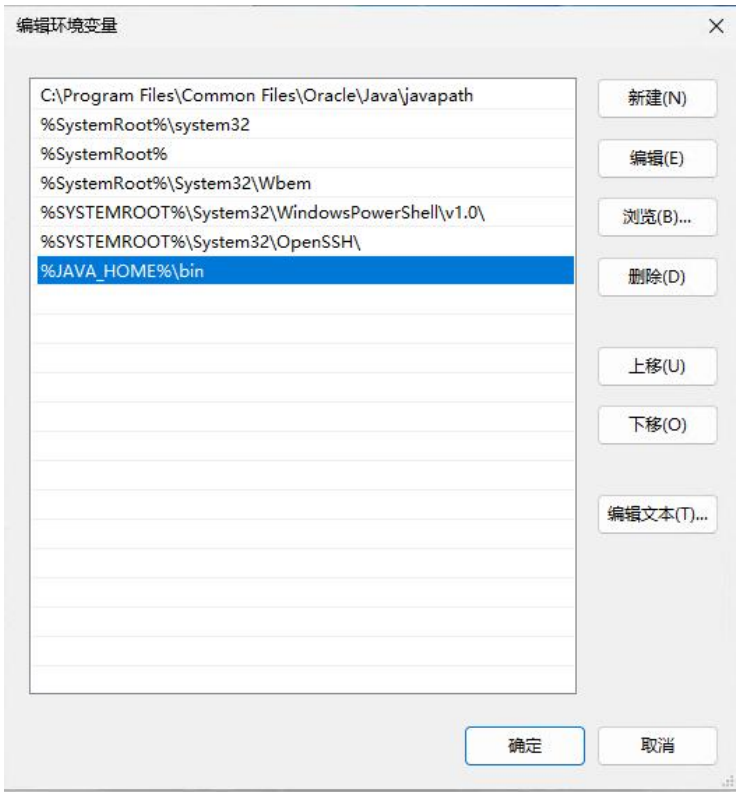


图 1-1-5 编辑环境变量 path 的值

4. 查看是否安装成功

打开命令提示符窗口，输入 java-version 命令然后回车，如出现图 1-1-6 所示，即表示安装成功。

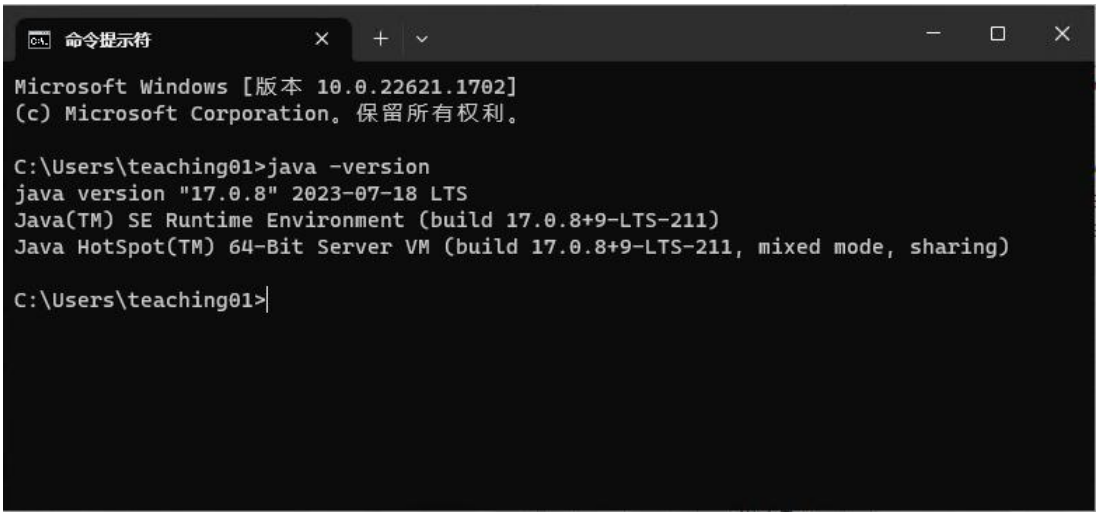


图 1-1-6 安装成功命令行窗口

三、Tomcat 安装及环境配置

1. 下载 tomcat

Tomcat 是 web 应用服务器，Java 开发的 web 项目最终需要部署到 tomcat 中运行。Tomcat 官网下载链接：https://tomcat.apache.org/，选择 tomcat10 版本，然后点击 64-bit Windows zip 进行下载，如图 1-1-7 所示。

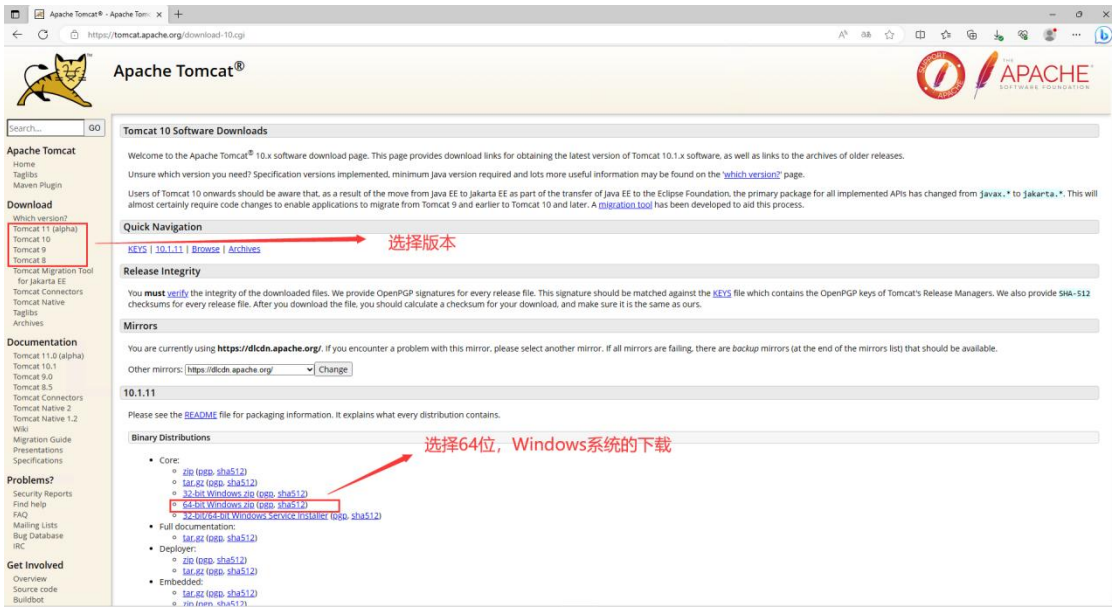


图 1-1-7 tomcat 下载页面

2. 安装 tomcat

把下载到的文件 apache-tomcat-10.1.11-windows-x64.zip 解压到路径 D:\Program

Files\apache-tomcat-10.1.11，如图 1-1-8 所示。

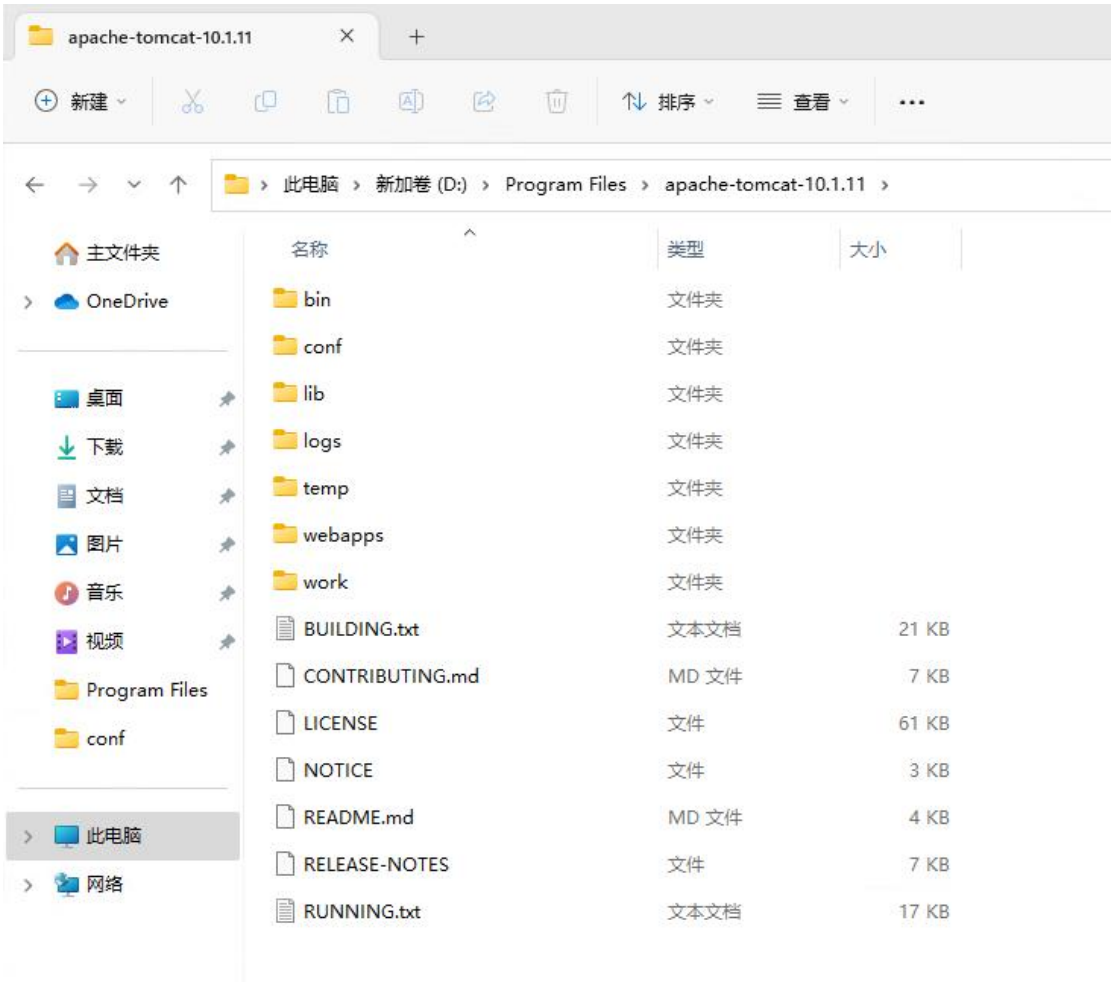


图 1-1-8 tomcat 解压路径

3. 环境变量配置

进行 tomcat 的环境变量之前需要确保 JDK 已经正确安装。

在电脑上找到我的电脑，右击【此电脑】→【属性】→【高级系统设置】→【环境变量】→【系统变量(s)】→【新建】出现新建环境变量窗口，如图 1-1-9 所示。

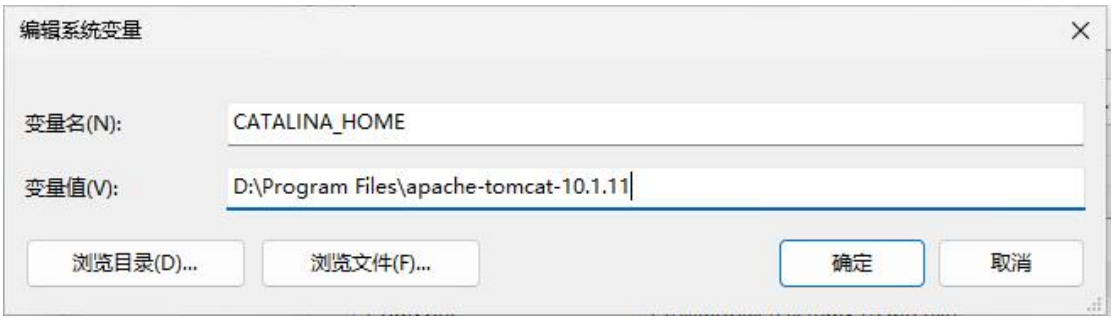


图 1-1-9 新建环境变量窗口

在【变量名】和【变量值】中分别输入 CATALINA_HOME 和 D:\Program Files\apache-tomcat-10.1.11，然后点击【确定】。

在变量名为 path 的一项中加入%CATALINA_HOME%\bin，如图 1-1-10 所示。

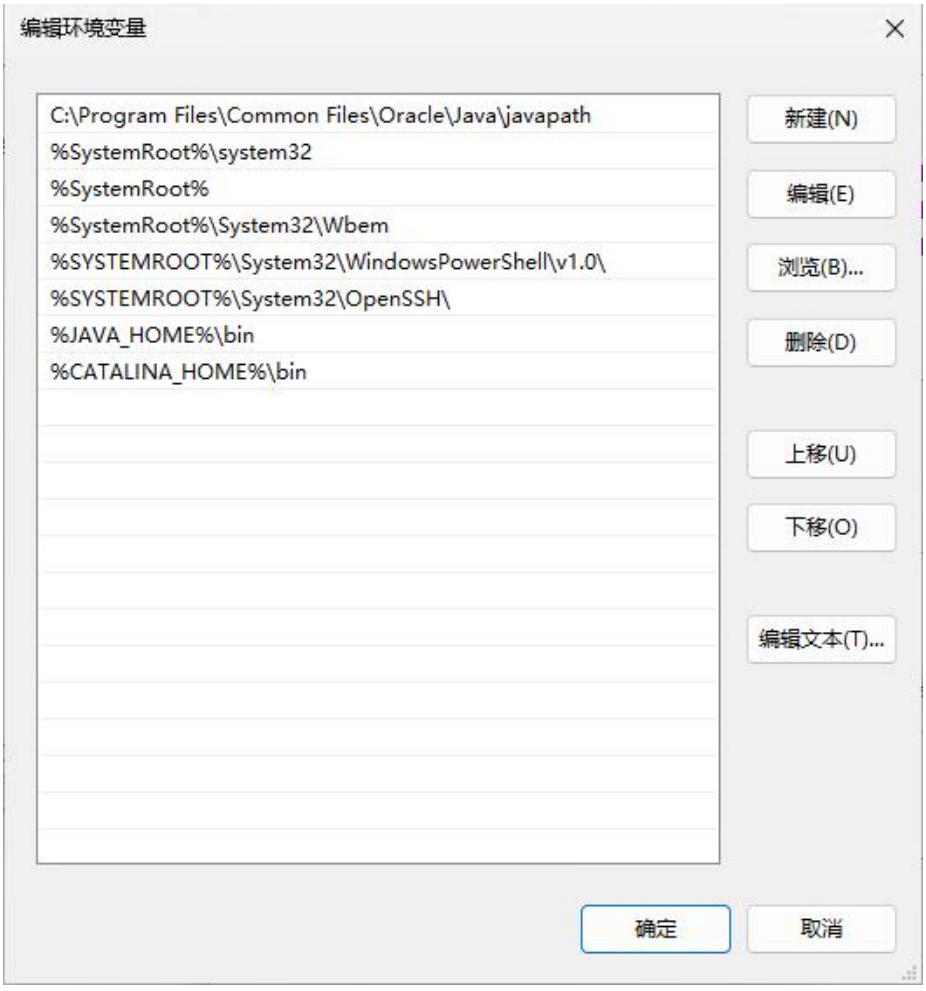


图 1-1-10 编辑环境变量 path 的值

4. 启动 tomcat

打开命令提示符窗口，输入 startup.bat 回车，如出现图 1-1-11 所示。tomcat 运行的命令行提示符窗口，如图 1-1-12 所示。

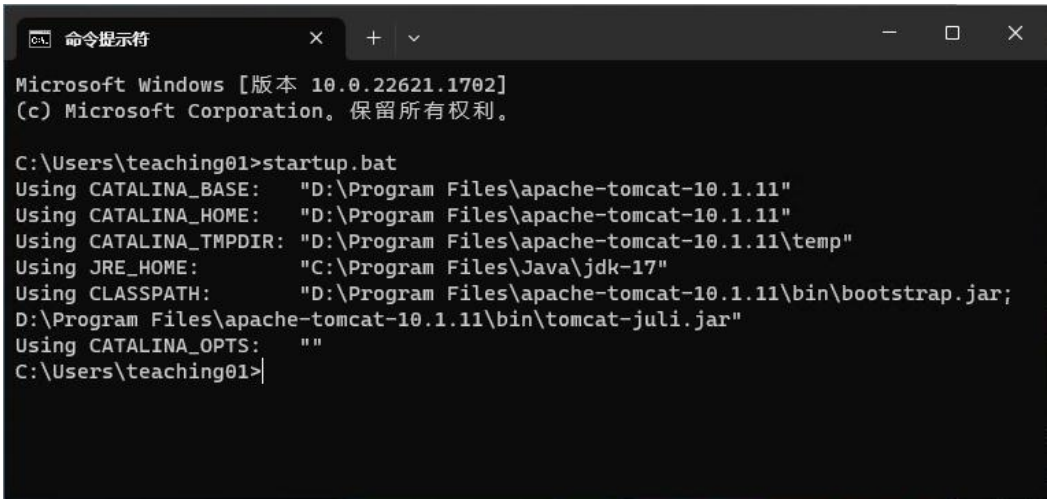


图 1-1-11 启动 tomcat 命令提示符窗口

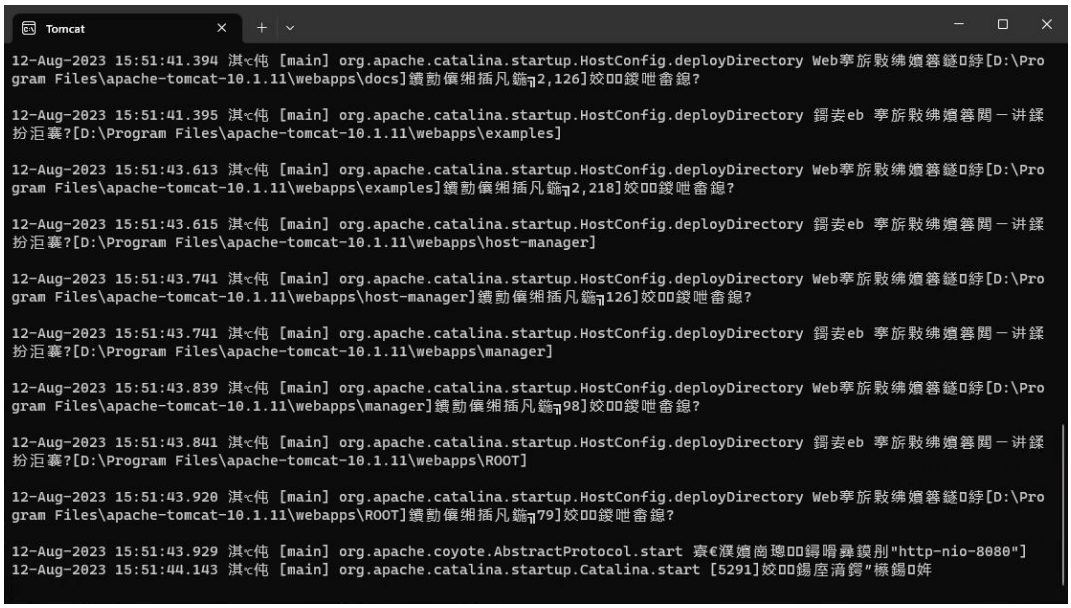


图 1-1-12 tomcat 运行窗口

5. 访问 tomcat

打开浏览器输入 http://localhost:8080 进行访问，访问结果如图 1-1-13 所示则表示 tomcat 启动成功。

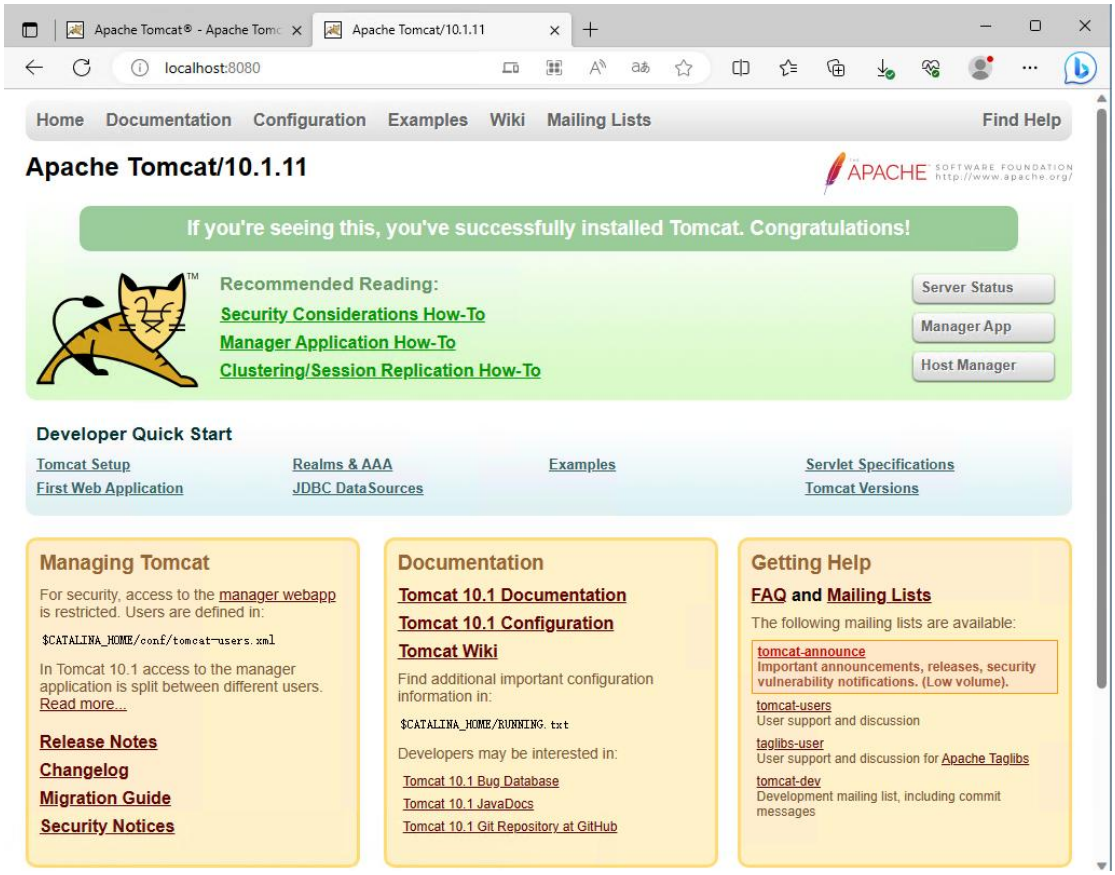


图 1-1-13 访问结果页面

6. 关闭 tomcat

打开命令提示符窗口，输入 shutdown.bat 回车，如出现图 1-1-14 所示。

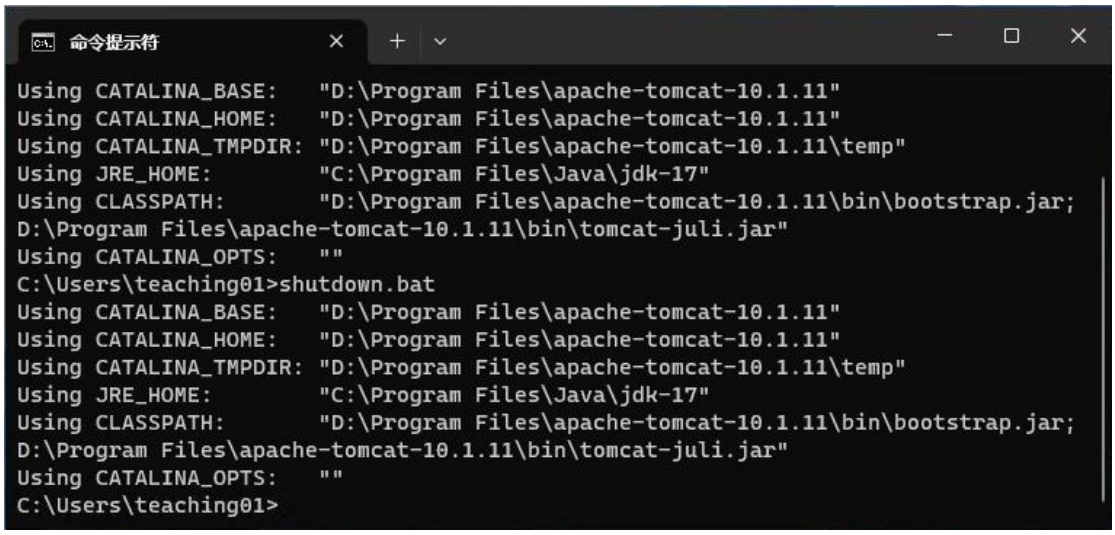


图 1-1-14 关闭 tomcat 命令提示符窗口

四、Maven 安装与环境配置

1. 下载 Maven

打开网址 https://maven.apache.org，在去到下载页面选择 apache-maven-3.9.4-bin.zip 下载，如图 1-1-15 所示。

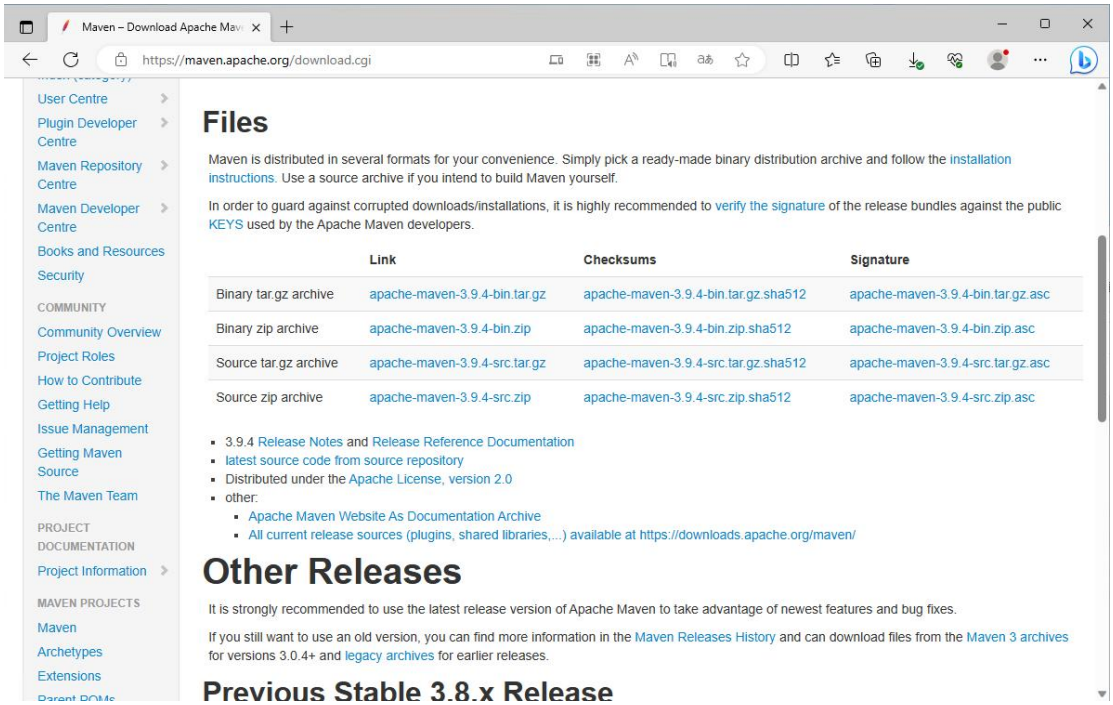


图 1-1-15 Maven 下载页面

2. 安装 Maven

把下载到的文件 apache-maven-3.9.4-bin.zip 解压到路径 D:\Program Files\apache-maven-3.9.4，如图 1-1-16 所示。

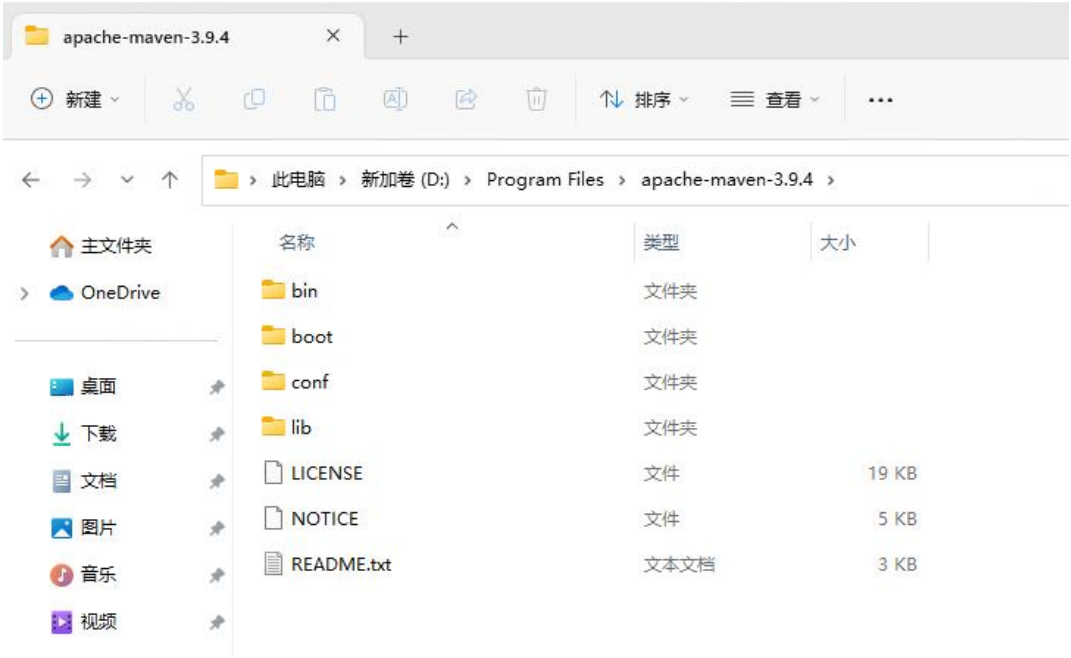


图 1-1-16 Maven 解压路径

3. 环境变量配置

在电脑上找到我的电脑，右击【此电脑】→【属性】→【高级系统设置】→【环境变量】→【系统变量(s)】→【新建】出现新建环境变量窗口，如图 1-1-17 所示。

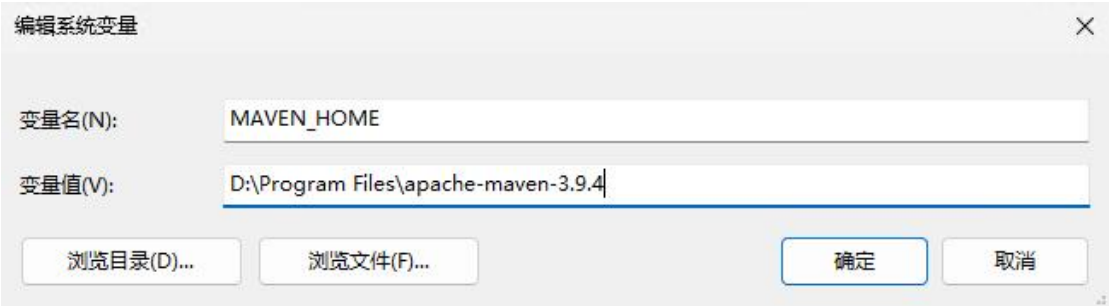


图 1-1-17 新建环境变量窗口

在【变量名】和【变量值】中分别输入 MAVEN_HOME 和 D:\Program Files\apache-maven-3.9.4，然后点击【确定】。

在变量名为 path 的一项中加入%MAVEN_HOME%\bin，如图 1-1-18 所示。

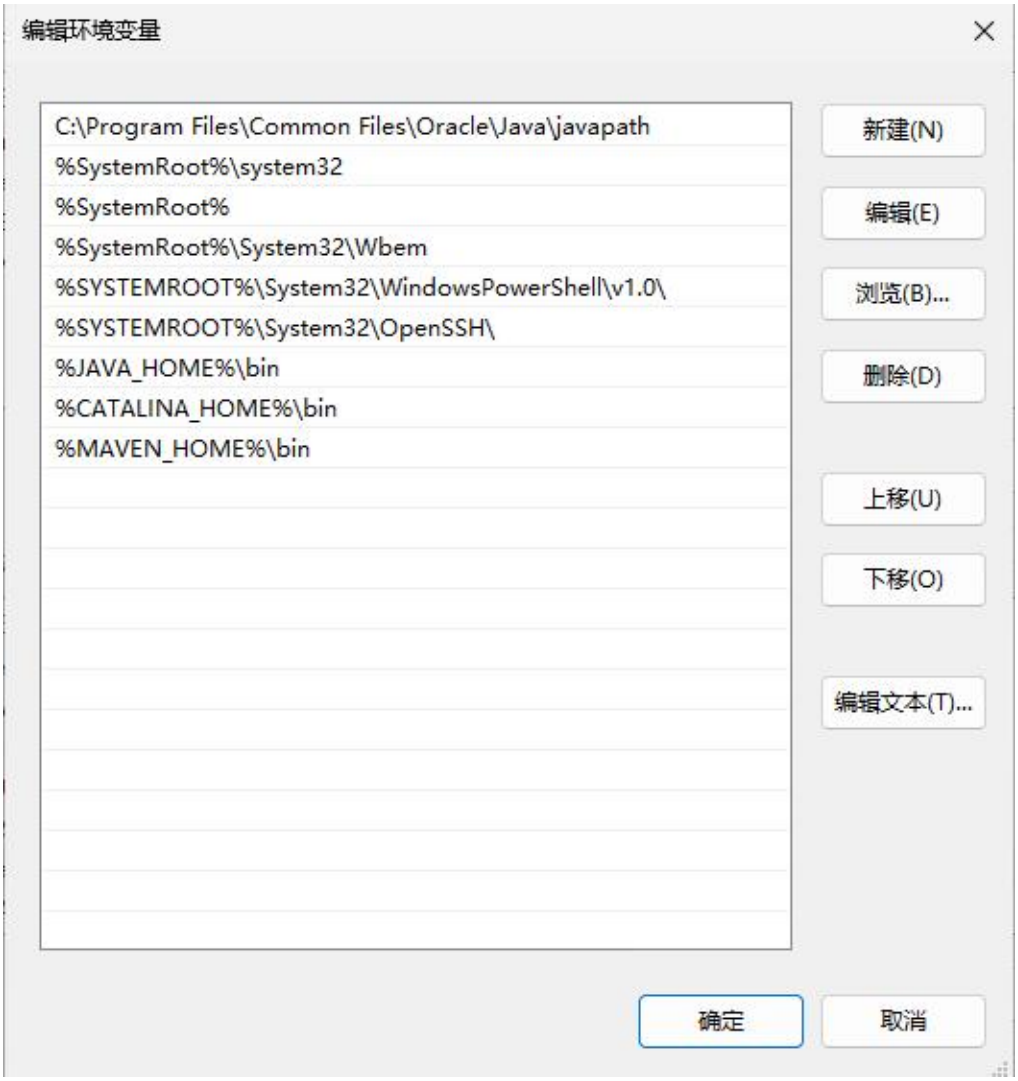


图 1-1-18 编辑环境变量 path 的值

4. 查看是否安装成功

打开命令提示符窗口，输入 mvn -v 命令然后回车，如出现图 1-1-19 所示，即表示安装成功。

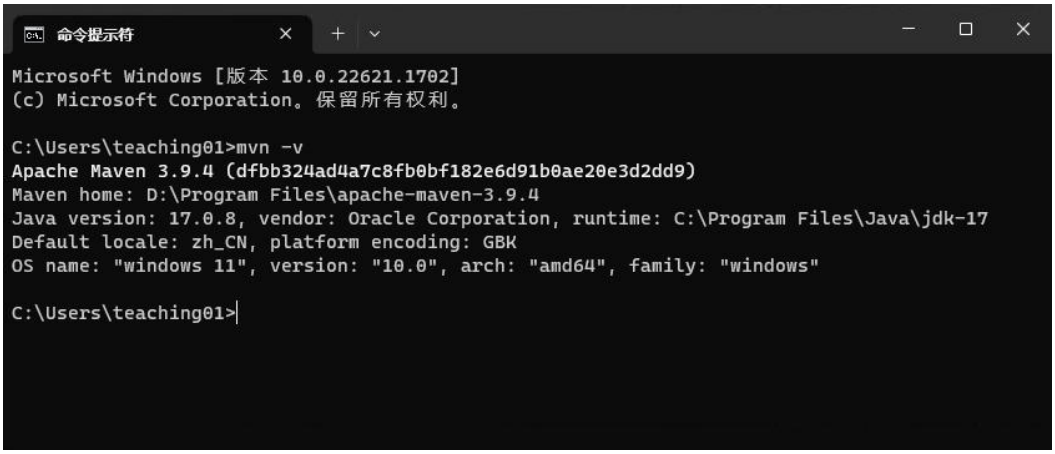


图 1-1-19 安装成功命令行窗口

5. 修改 Maven 配置文件

打开 Maven 的安装目录，进入 conf 文件夹，用记事本打开 setting.xml 文件。在里面找到<localRepository></localRepository>标签，这里是指定 jar 包下载的路径，我们把这里改成<localRepository>D:/mavenRepository</localRepository>，如图 1-1-20 所示。

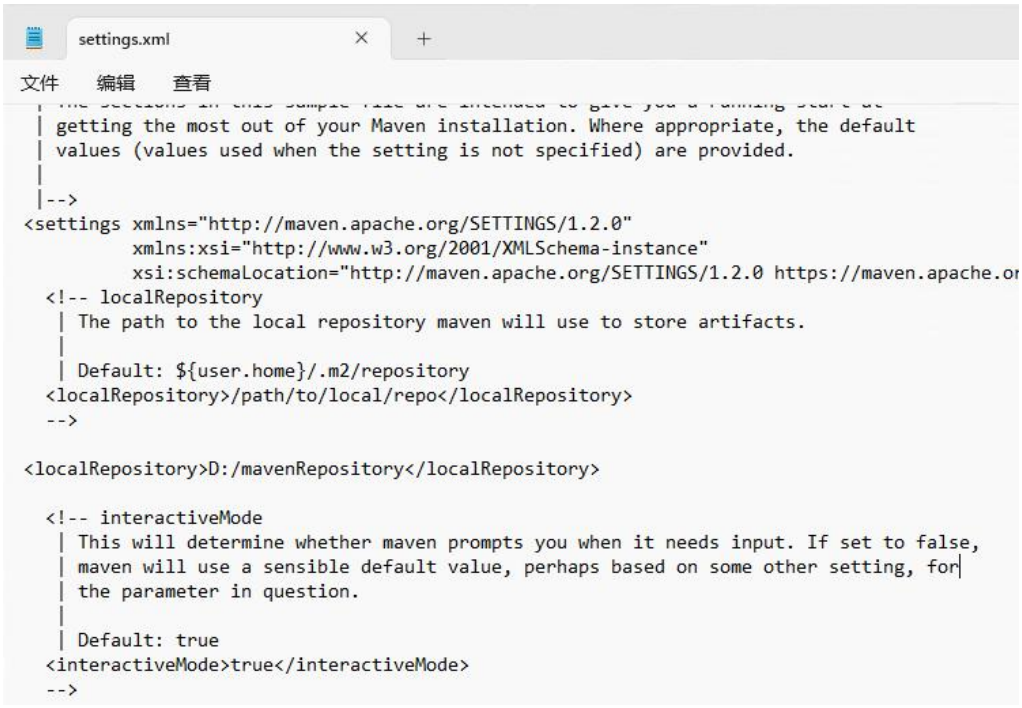


图 1-1-20 Maven 配置文件修改指定 jar 包下载的路径

找到<mirrors></mirrors>标签，这里配置的是 Maven 下载资源时用的镜像库，默认的太慢了，我们可以修改为阿里的镜像库。首先把原来的注释掉，然后加入阿里云的镜像配置代码，如图 1-1-21 所示。

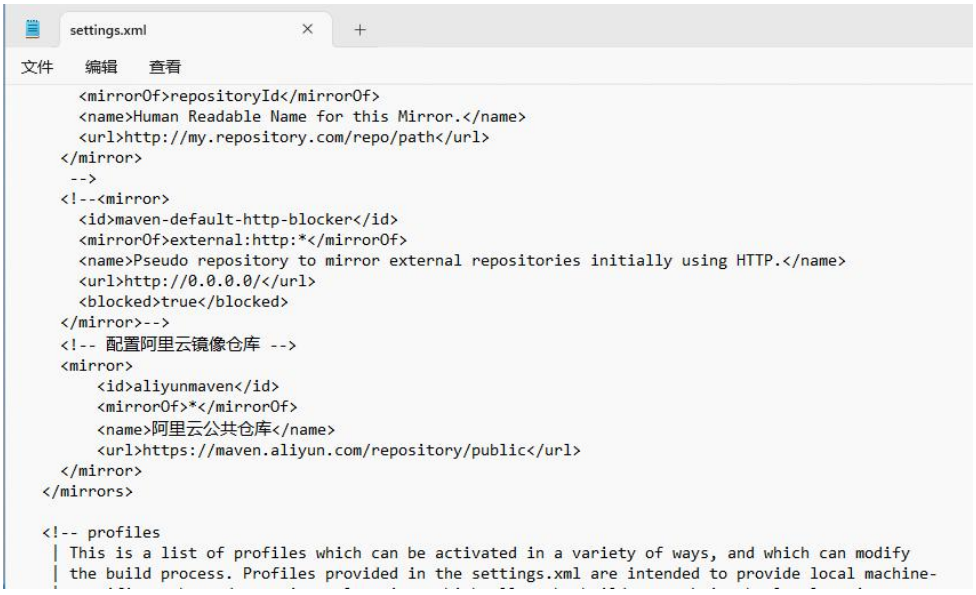


图 1-1-21 Maven 配置文件修改镜像库地址

命令提示符中输入 mvn help:system 命令，该命令展示平台详细信息，例如环境变量和系统变量，最后显示如图 1-1-22 所示。

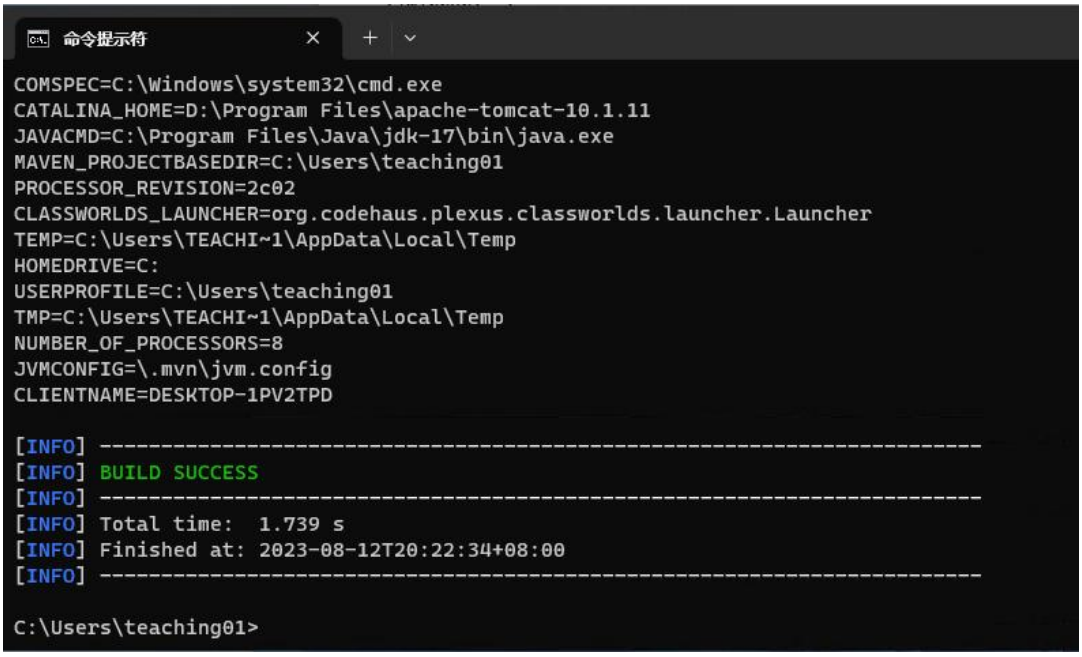


图 1-1-22 mvn help:system 执行成功信息

五、Eclipse 集成开发环境（IDE）的安装及配置

1. 下载 Eclipse

在 eclipse 的官网上直接进行下载，地址如下：https://www.eclipse.org/downloads，如图 1-1-23 所示。

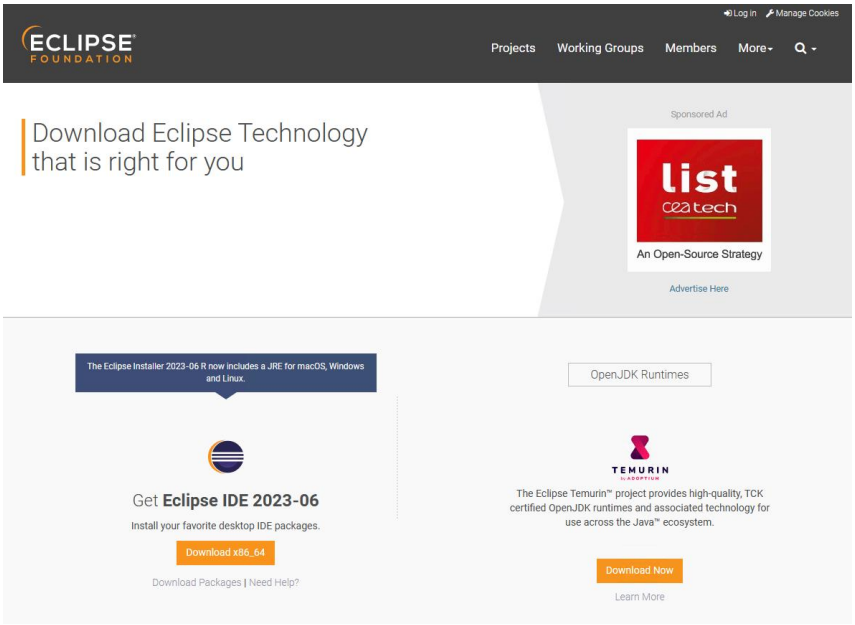


图 1-1-23 eclipse 下载页面

2. 安装 Eclipse

执行下载好的安装文件 eclipse-inst-jre-win64.exe，打开的安装器界面如图 1-1-24 所示，这里我们选择“Eclipse IDE for Enterprise Java and Web Developers”进行安装，点击后进入下一步安装。

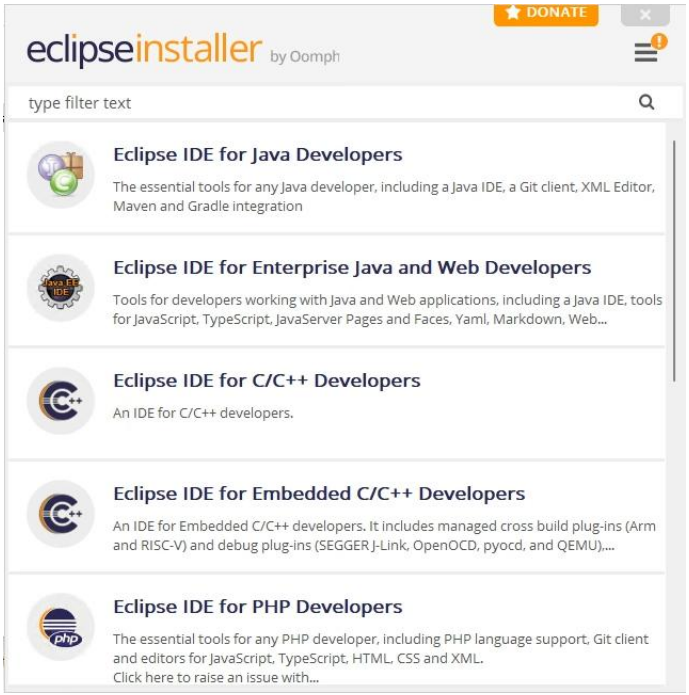


图 1-1-24 eclipse 安装器界面

这一步我们选择更改 eclipse 的安装目录，点击最右侧按钮选择安装路径 D:\Program Files，如图 1-1-25 所示。然后点击“INSTALL”按钮进行安装，此时会有一个安装协议窗口弹出点击“Accept”按钮同意即可，最后进入安装等待页面，可能需要的时间有些长，耐心等待即可。

图 1-1-25 选择 eclipse 安装目录

安装完成如图 1-1-26 所示，点击“LAUNCH”按钮启动 eclipse。

图 1-1-26 eclipse 安装完成

Eclipse 启动会有一个“选择工作空间目录”窗口弹出，选择我们的工作目录：D:\eclipse-workspace 如图 1-1-27 所示。把“Use this as the default and do not ask again”勾选上，这样下次启动 eclipse 就不会再弹出提示窗口了。启动后界面如图 1-1-28 所示。

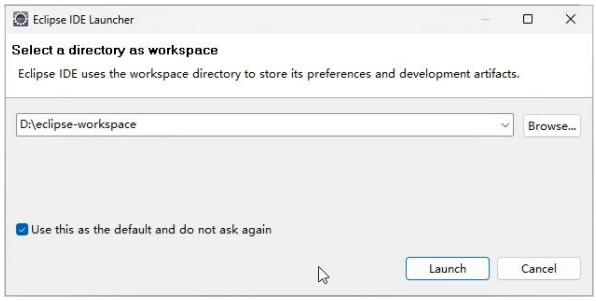


图 1-1-27 eclipse 工作目录选择窗口

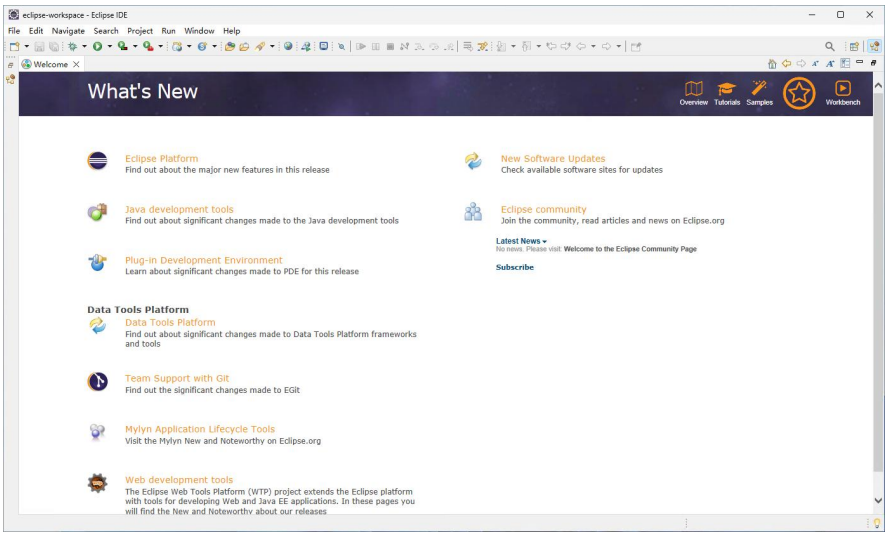


图 1-1-28 eclipse 初次启动后界面

3. Eclipse 设置

点击 eclipse 的菜单栏【Window】→【Preferences】弹出设置窗口，如图 1-1-29 所示。

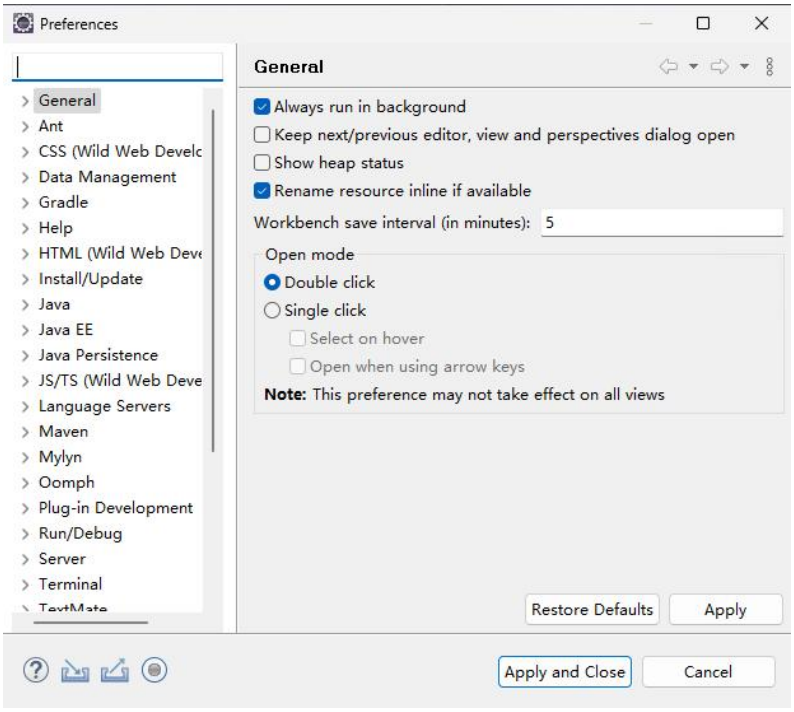


图 1-1-29 eclipse 设置窗口

(1) 皮肤样式设置

选择【General】→【Appearance】在右侧的 Theme 项中可以选择不同的皮肤样式，如图 1-1-30 所示。最后点击“Apply”或者“Apply and Close”按钮保存设置。

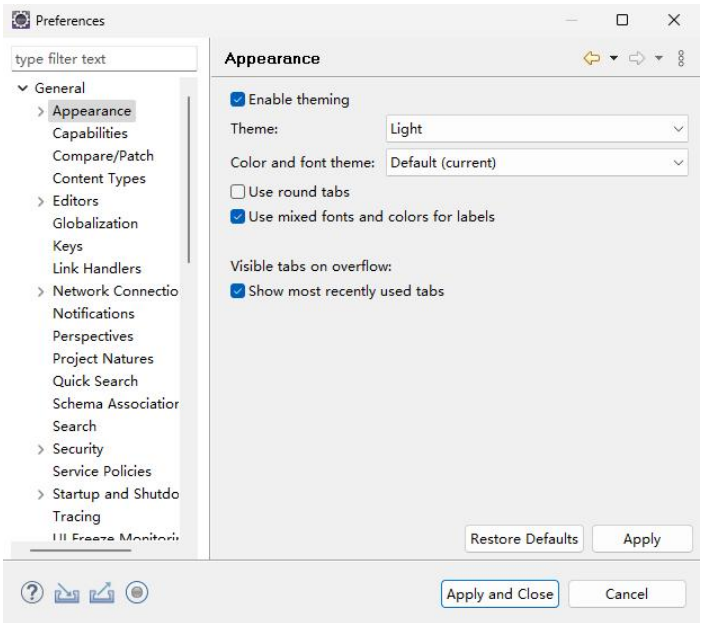


图 1-1-30 皮肤样式设置

(2) 字体设置

选择【General】→【Appearance】→【Colors and Fonts】在右侧的 Basic 项中双击展开，然后选中“Text Font”，然后点击最右侧的“Edit...”按钮进行设置，如图 1-1-31 所示。最后点击“Apply”或者“Apply and Close”按钮保存设置。

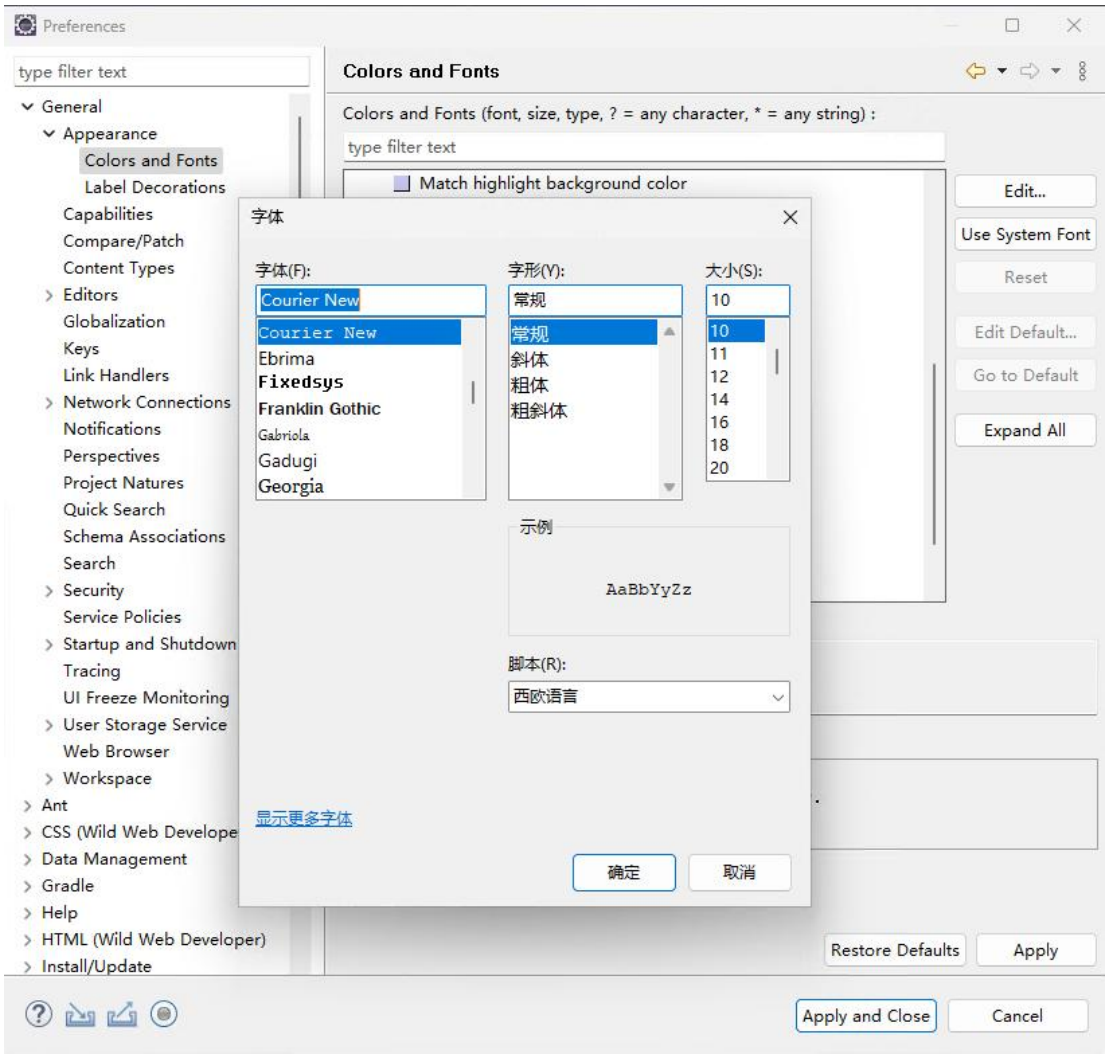


图 1-1-31 字体设置窗口

(3) 编码格式设置

大多数情况下，我们都是采用 UTF-8 格式，这是国际通用的编码格式。如果编码格式与别人的不一样，在代码中存在中文时，就可能会出现乱码。

选择【General】→【Appearance】→【Workspace】在右侧的“Text file encoding”项中选择“Other”并在下拉列表中选择“UTF-8”，如图 1-1-32 所示。最后点击“Apply”或者“Apply and Close”按钮保存设置。

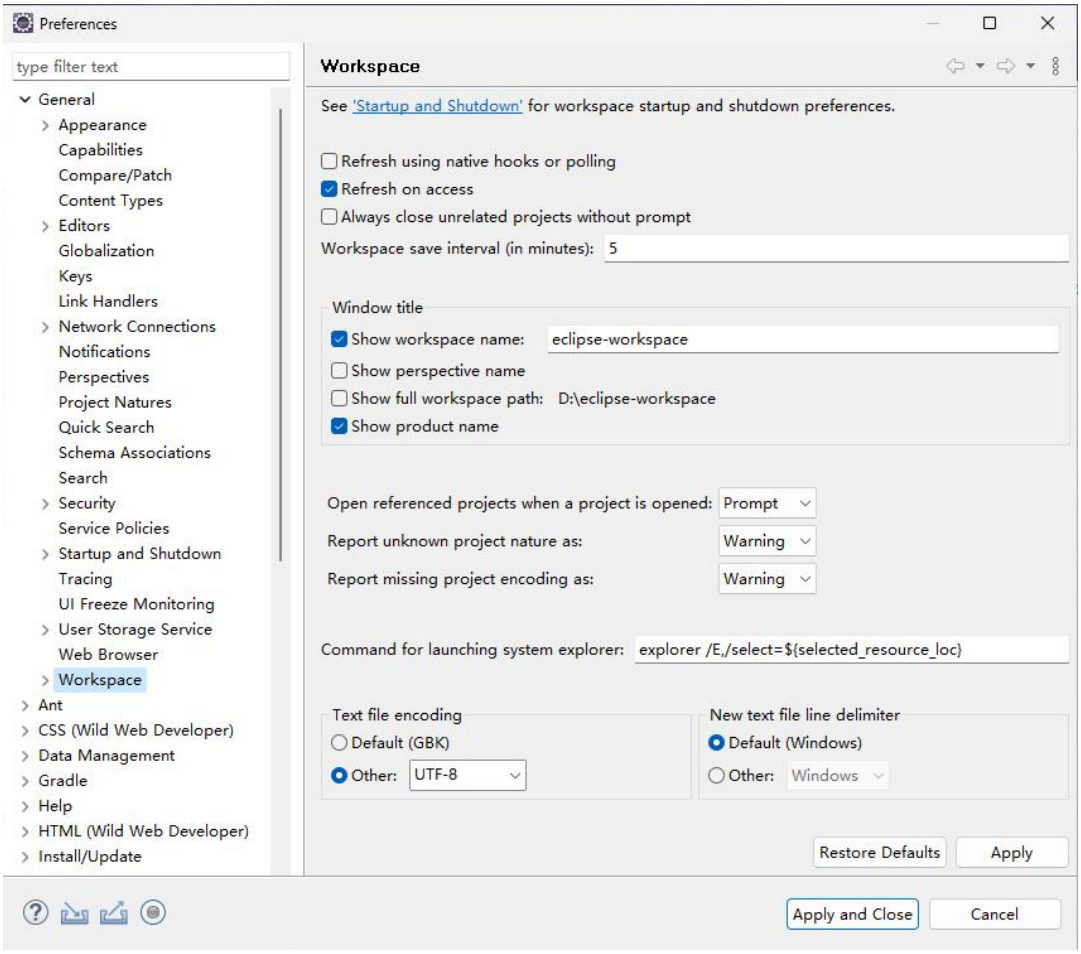


图 1-1-32 编码格式设置窗口

(4) JDK 编译版本设置

已经安装的 JRES 设置，选择【Java】→【Installed JREs】在右侧点击“Add...”按钮在弹出的窗口中选中“Standard VM”然后点击“Next>”按钮，最后可以进行 JDK 的不同版本添加，如图 1-1-33 所示。添加后点击“Finish”按钮结束。

编译版本设置，选择【Java】→【Compiler】在右侧的“Compiler compliance level”对应的下拉列表中可以设置编译的版本，本课程使用的是 17。如图 1-1-34 所示。

最后点击“Apply”或者“Apply and Close”按钮保存设置。

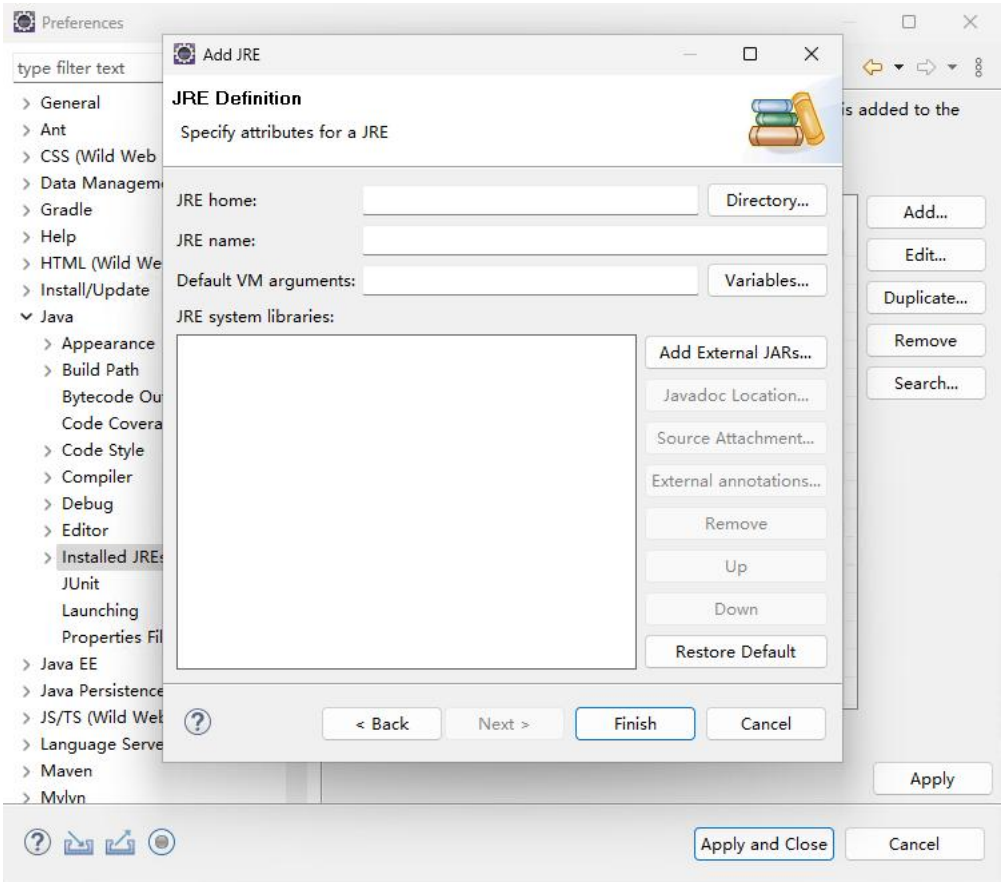


图 1-1-33 添加 JRE 窗口

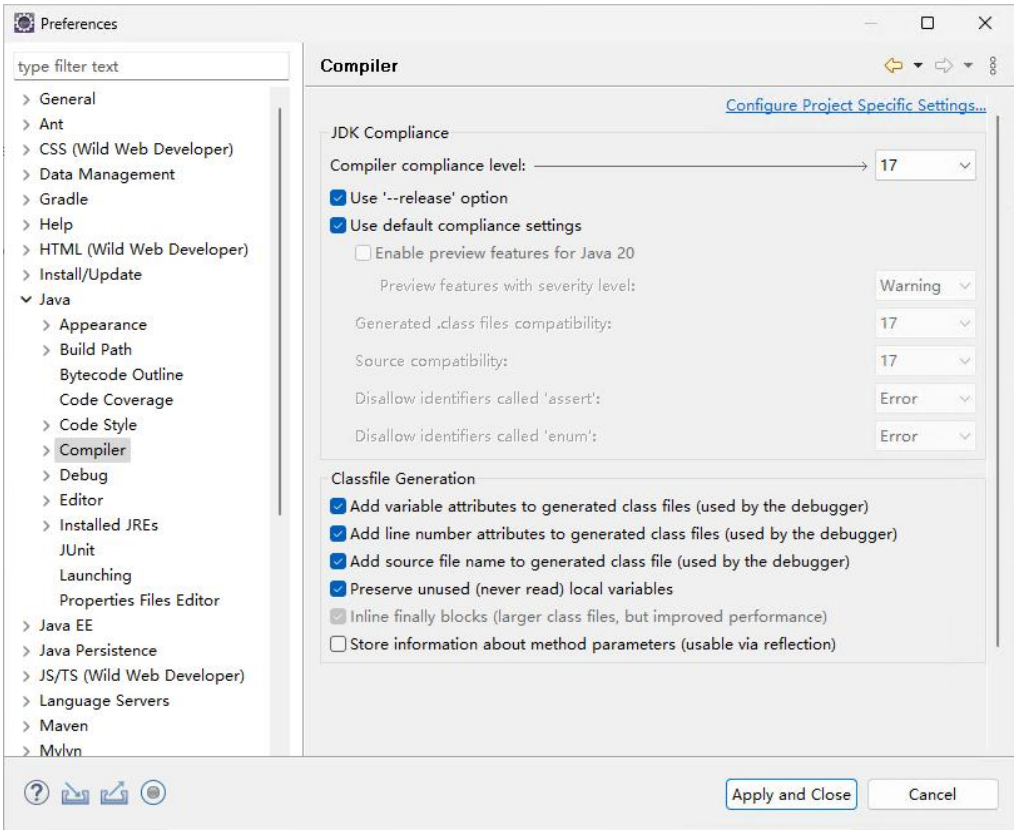


图 1-1-34 编译的版本选择

(5) Tomcat 服务器运行时设置

选择【Server】→【Runtime Environments】在最右侧点击“Add…”按钮，然后弹出“New Server Runtime Environment”窗口如图 1-1-35 所示。展开“Apache”选择“Apache Tomcat v10.1”，然后点击“Next>”按钮。

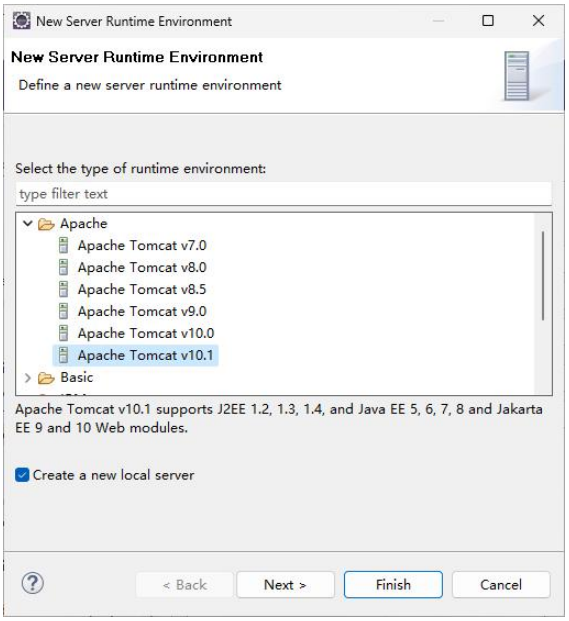


图 1-1-35 选择运行环境类型

这一步我们需要选择 tomcat 的相关信息即可，步骤如下：

“Tomcat installation directory:”（Tomcat 的安装路径）这一项，在右侧点击“Browse…”按钮然后选择 tomcat 的安装路径，本教程的安装路径是：D:\Program Files\apache-tomcat-10.1.11。

“JRE”这一项是我们已经安装的 JDK 版本，本教程选择的是“jdk-17”，如其他版本的 JDK 可以点击右侧的“Installed JREs…”按钮进行配置，这个配置参考前面讲的“JDK 编译版本设置”。如图 1-1-36 所示，点击“Finish”按钮。

最后点击“Apply and Close”按钮保存设置。

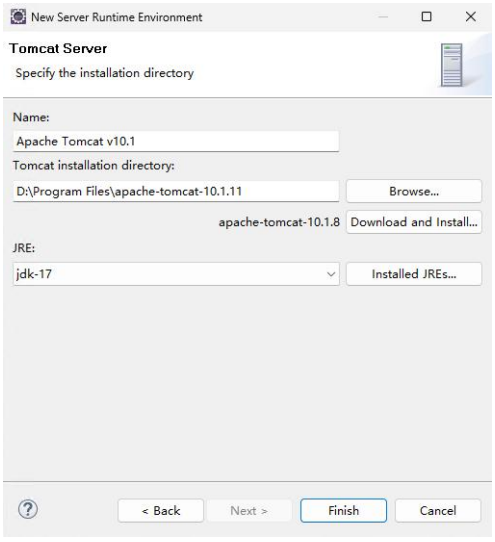


图 1-1-36 添加新的服务器

（6）Maven 配置设置

Eclipse 中默认是有自带的 Maven 插件的，但是自带的 Maven 插件不能修改本地仓库，所以通常我们不使用自带的 Maven，而是使用自己安装的，在 eclipse 中配置 maven 的步骤如下：

选择【Maven】→【Installations】在最右侧点击“Add…”按钮，然后弹出“New Maven Runtime”窗口如图 1-1-37 所示。

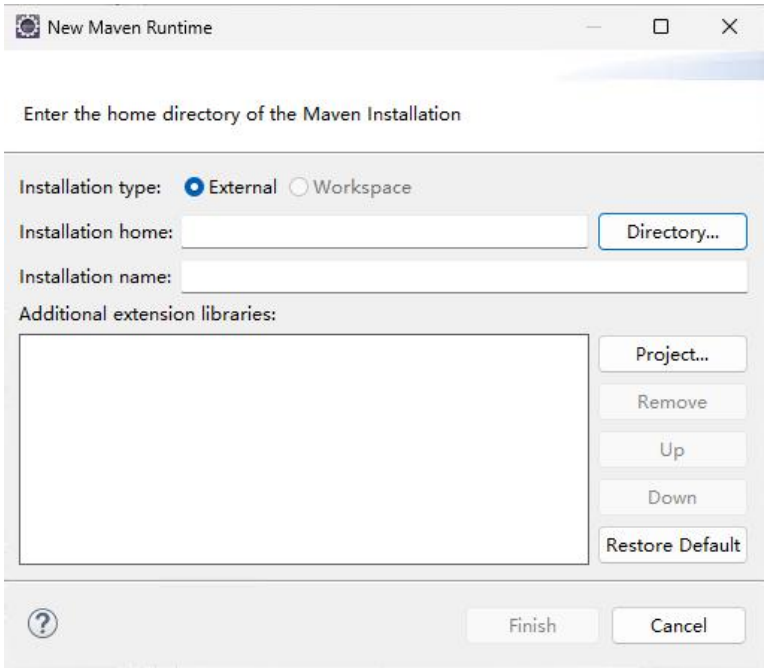


图 1-1-37 添加新的 Maven 环境

点击“Directory…”按钮，选择安装路径“D:\Program Files\apache-maven-3.9.4”，如图 1-1-38 所示，点击“Finish”按钮。

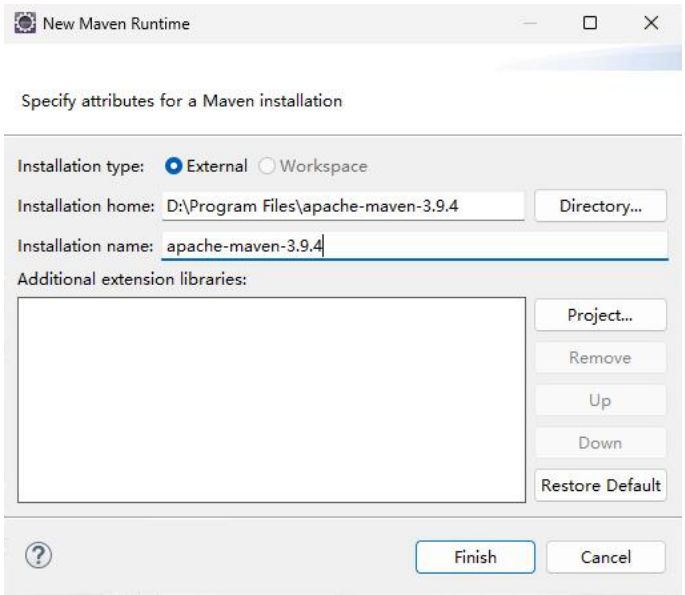


图 1-1-38 选择 Maven 的安装路径

把我们添加的 maven 勾选上，然后点击“Apply”按钮保存设置，如图 1-1-39 所示。

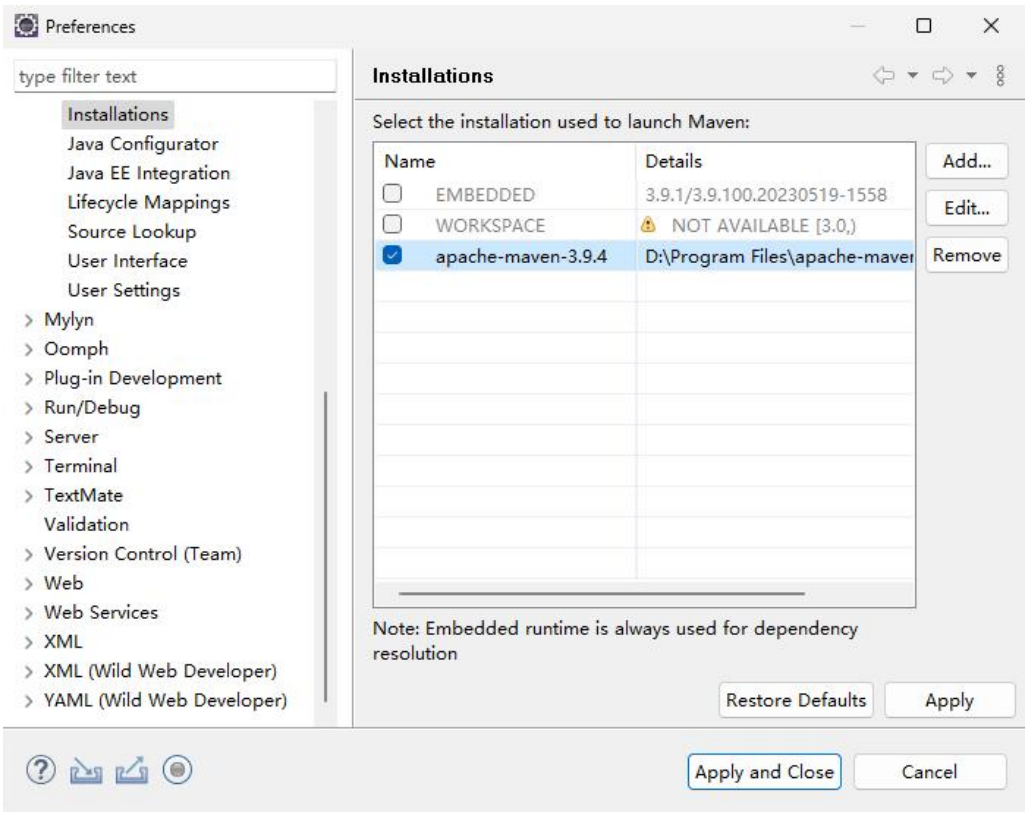


图 1-1-39 勾选 Maven

最后还需要选择【Maven】→【User Settings】来选择 Maven 的用户设置，在“User Settings”这一项里点击“Browse…”按钮，选择对应 Maven 版本安装目录下的 conf 文件夹里的 settings.xml 文件，把本地仓库改成我们 settings.xml 里的设置，如图 1-1-40 所示。

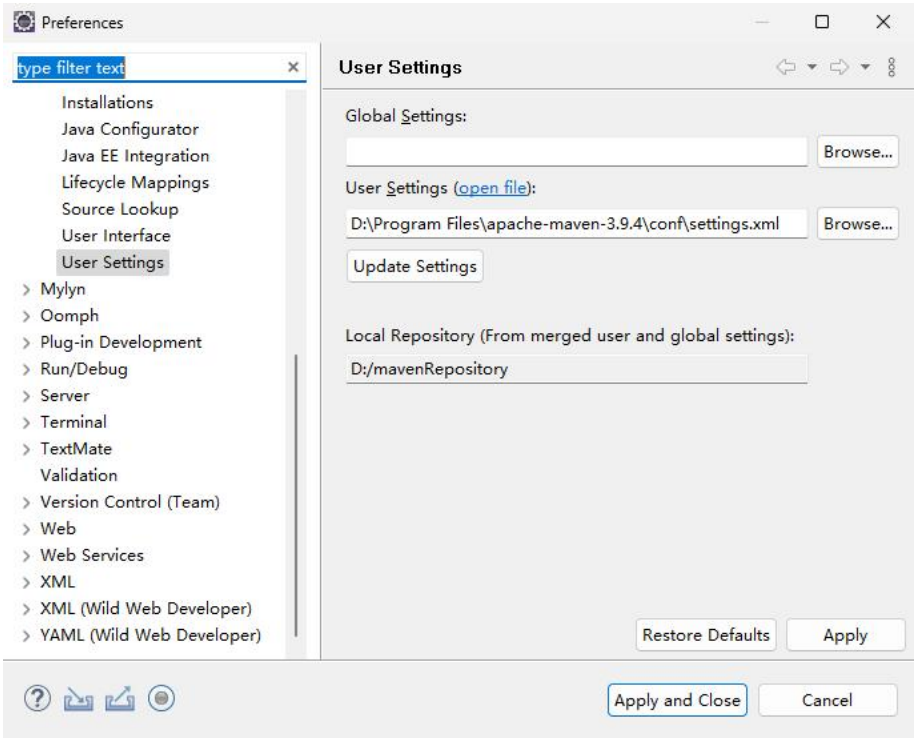


图 1-1-40 设置 Maven 配置文件 settings.xml

六、创建工程项目与调试运行

1. 创建新工程项目

在 eclipse 菜单栏中选择【File】→【New】→【Maven Project】来创建一个 Maven 项目，弹出“New Maven Project”窗口，如图 1-1-41 所示，这里默认直接点击“Next>”按钮进入下一步。

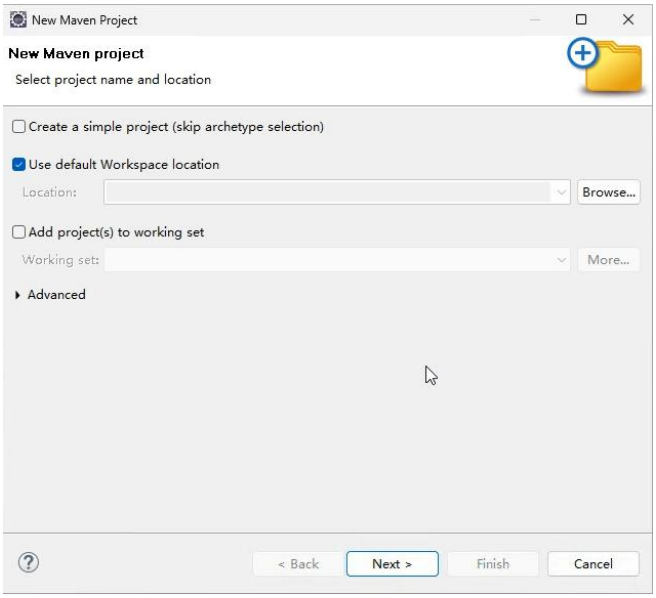


图 1-1-41 新建 Maven 项目

如图 1-1-42 所示，选择“maven-archetype-webapp”原型来创建一个 Java web 项目，然后点击“Next>”按钮进入下一步。

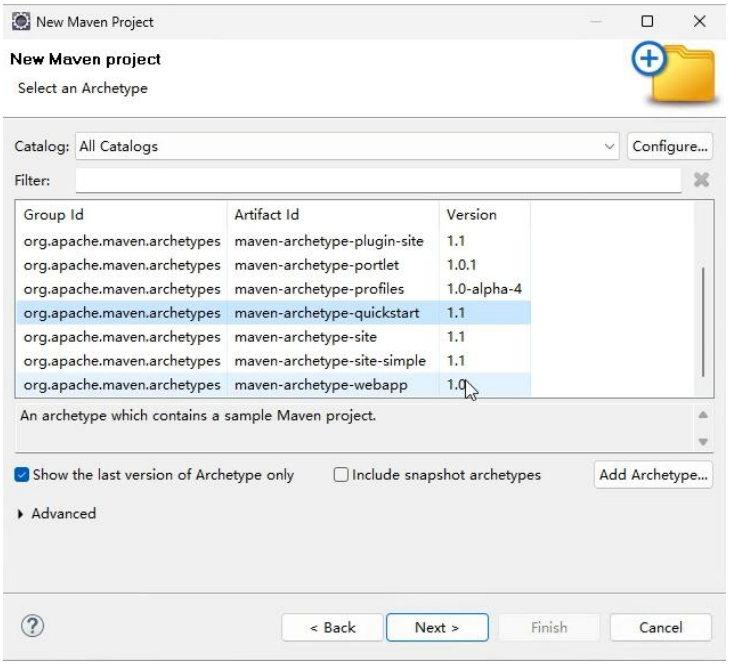


图 1-1-42 选择创建项目的原型

如图 1-1-43 所示，在“Group Id:”中输入“com.teaching”，“Artifact Id:”中输入“helloworld”，最后点击“Finish”按钮完成创建。

Group Id 和 Artifact Id 被统称为“坐标”，是为了保证项目唯一性而提出的，如果你要把你的项目弄到 Maven 本地仓库去，你想要找到你的项目就必须根据这两个 id 去查找。

Group Id 一般分为多个段，第一段为域，第二段为公司名称。域又分为 org、com、cn 等等许多，其中 org 为非营利组织，com 为商业组织。

Artifact Id 表示项目名。

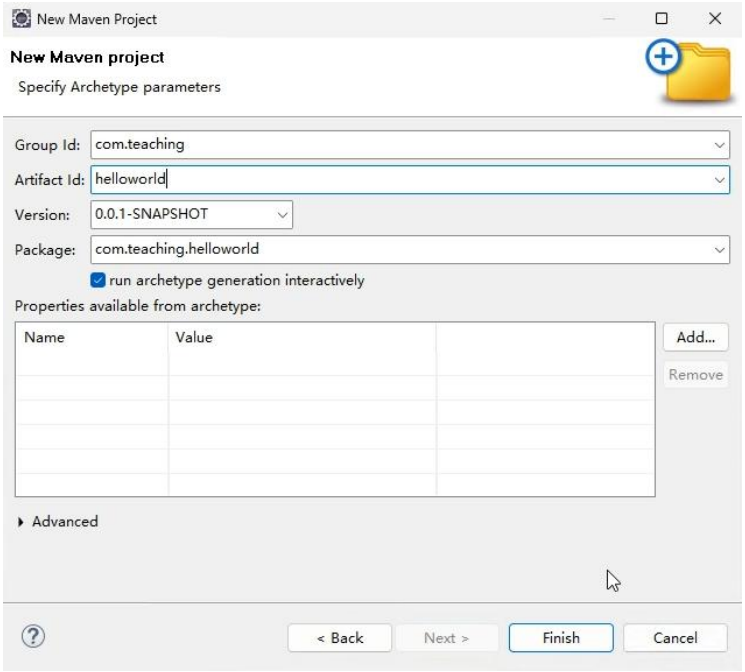


图 1-1-43 填写项目参数

在“Project Explorer”项目浏览器中可以看到新建的项目，如图 1-1-44 所示。

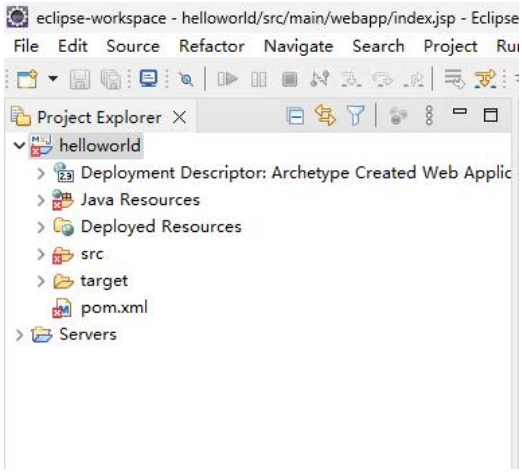


图 1-1-44 项目浏览

2. 工程项目进行 JDK 配置

新建的项目还需要进行 JDK 配置，有些时候新导入的项目也需要进程 JDK 配置，具体配置步骤如下：

选中项目，然后在菜单栏选择【Project】→【Properties】或对着项目鼠标右击在弹出的菜单中选择【properties】进行设置。

弹出窗口如图 1-1-45 所示，在窗口的左侧菜单中选择“Java Build Path”，然后右侧选择“Libraries”选项卡，选中“JRE System Library[JavaSE-1.7]”然后点击“Edit…”按钮。

弹出“Edit Library”窗口，如图 1-1-46 所示，在“System library”里选择“Workspace default JRE(jdk-17)”然后点击“Finish”按钮。

最后点击“Apply”或者“Apply and Close”按钮保存设置。

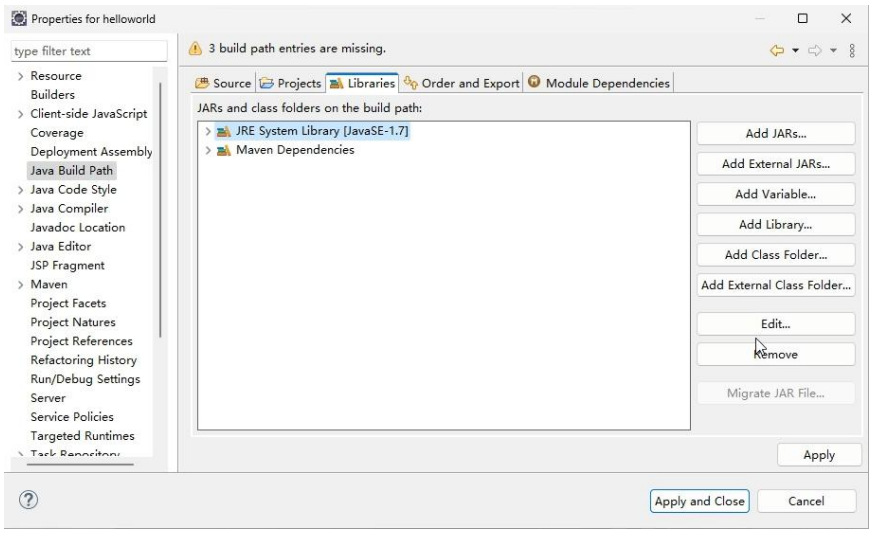


图 1-1-45 Java Build Path 窗口

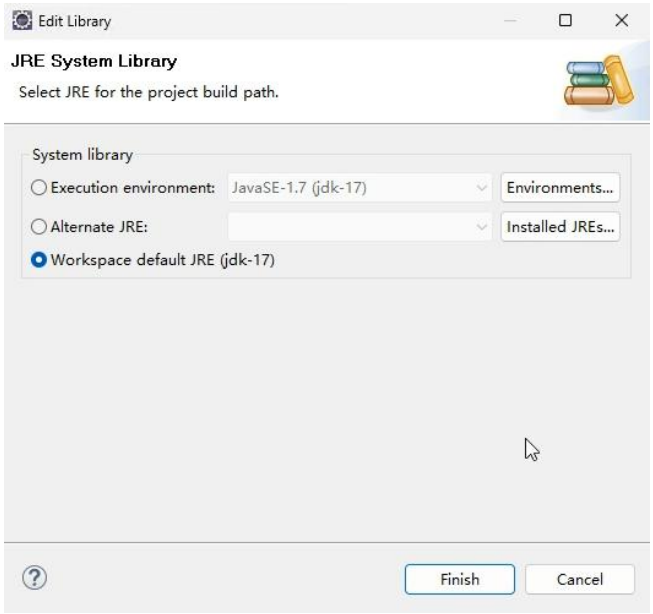


图 1-1-46 编辑 JDK 配置

3. 工程项目调试运行

在“Project Explorer”浏览器中选择需要运行的项目，鼠标右击在弹出菜单中我们可以看到两项“Run As”和“Debug As”，前者是正常运行，后者是调试运行，在项目开发中我们需要调试我们的程序时使用“Debug As”，这里我们只看运行结果就可以了，所以选择“Run As”，然后选择“Run on Server”，如图 1-1-47 所示。

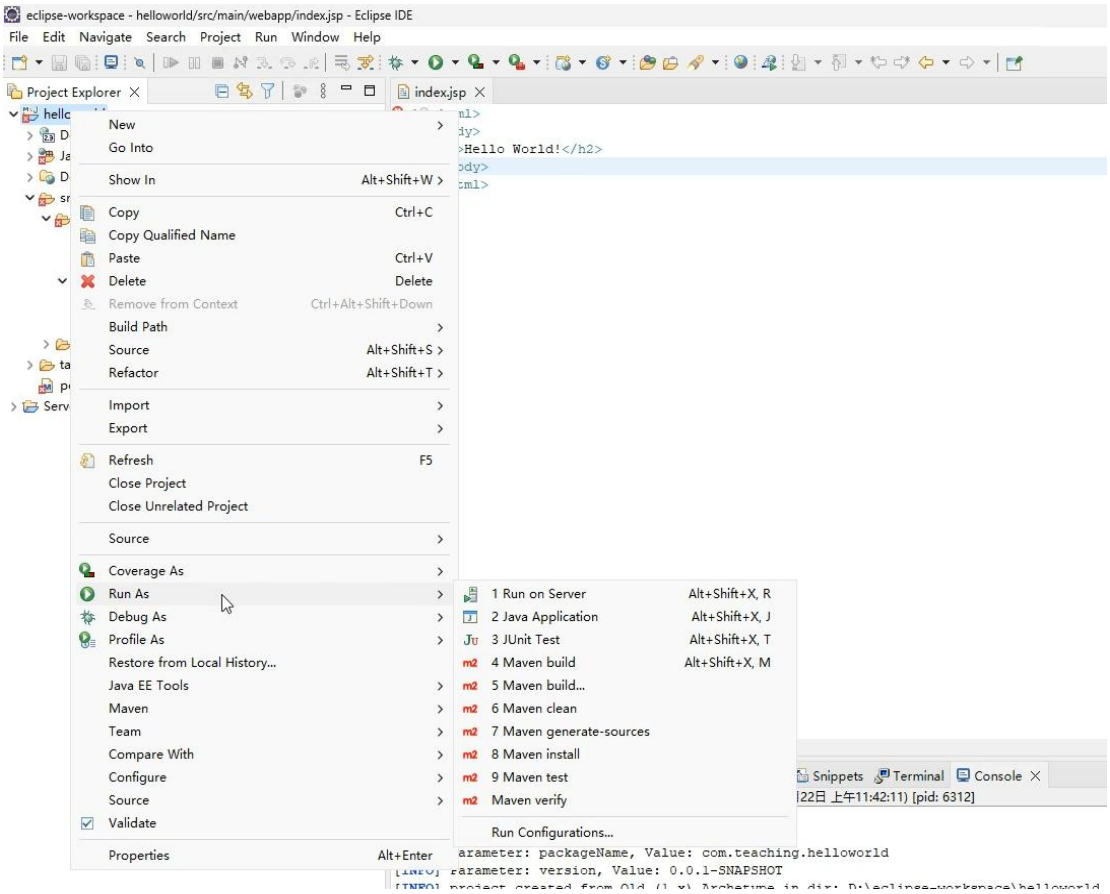


图 1-1-47 项目运行

弹出“Run On Server”窗口选择运行的服务器，如图 1-1-48 所示。这里可以根据实际需要选择“Manually define a new server”新建服务器，选择新建的可以参考前面的“Tomcat 服务器运行时设置”。我们不需要新建所以选择“Choose an existing server”，在服务器列表中选中“Tomcat v10.1 Server at localhost”然后点击“Finish”按钮。运行结果如图 1-1-49 所示。

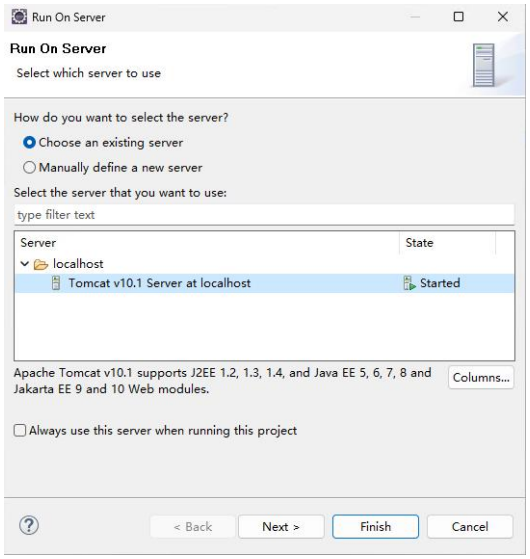


图 1-1-48 选择运行的服务器

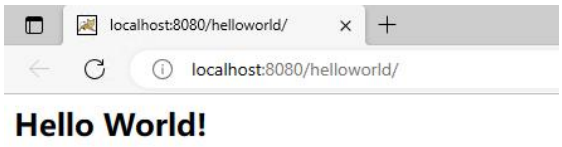


图 1-1-49 运行结果

任务实施

Step1 JDK 安装及系统环境配置

Step2 Tomcat 安装及系统环境配置

Step3 Maven 安装及系统环境配置

Step4 Eclipse 安装及系统环境配置

Step5 Eclipse 创建工程项目并运行

活动二 MariaDB 数据的安装与使用

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。项目使用 Java 语言进行开发，要求我们管理系统的数据存储数据库使用 MariaDB。项目开发前让我们安装配置 MariaDB 并且了解它的使用！

任务目标

1. 能正确安装数据库并且能启动数据库。
2. 能使用第三方工具操作数据库。
3. 能通过 SQL 对数据表进行增、删、查、改数据。

任务分析

1. 有哪些类型的数据库？如何安装 MariaDB 数据库？
2. 如何使用第三方工具操作数据库？
3. 如何通过 SQL 对数据表进行增、删、查、改数据？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、数据库类型

数据库有 2 种类型：关系型数据库、非关系型数据库。

关系型数据库有：MySQL、MariaDB、Oracle、SQL Server、SQLite、DB2、Microsoft Access 等等。

非关系型数据库有：redis、MongoDB、Memcache、HBase 等等。

MariaDB 和 MySQL 非常相似，但是 MariaDB 为用户提供了更好的功能和性能，所以本课程使用 MariaDB。

二、MariaDB 数据库安装与运行

1. 下载 MariaDB

在 MariaDB 的官网上直接进行下载，地址如下：https://mariadb.org/download，如图 1-2-1 所示。在这里选择具体的下载参数，版本选择 MariaDB Server 11.2.0Alpha、操作系统选择 Windows、计算机架构选择 64 位、安装包类型选择 MSI Package、下载镜像选择 Alibaba Cloud，点击“Download”按钮下载。

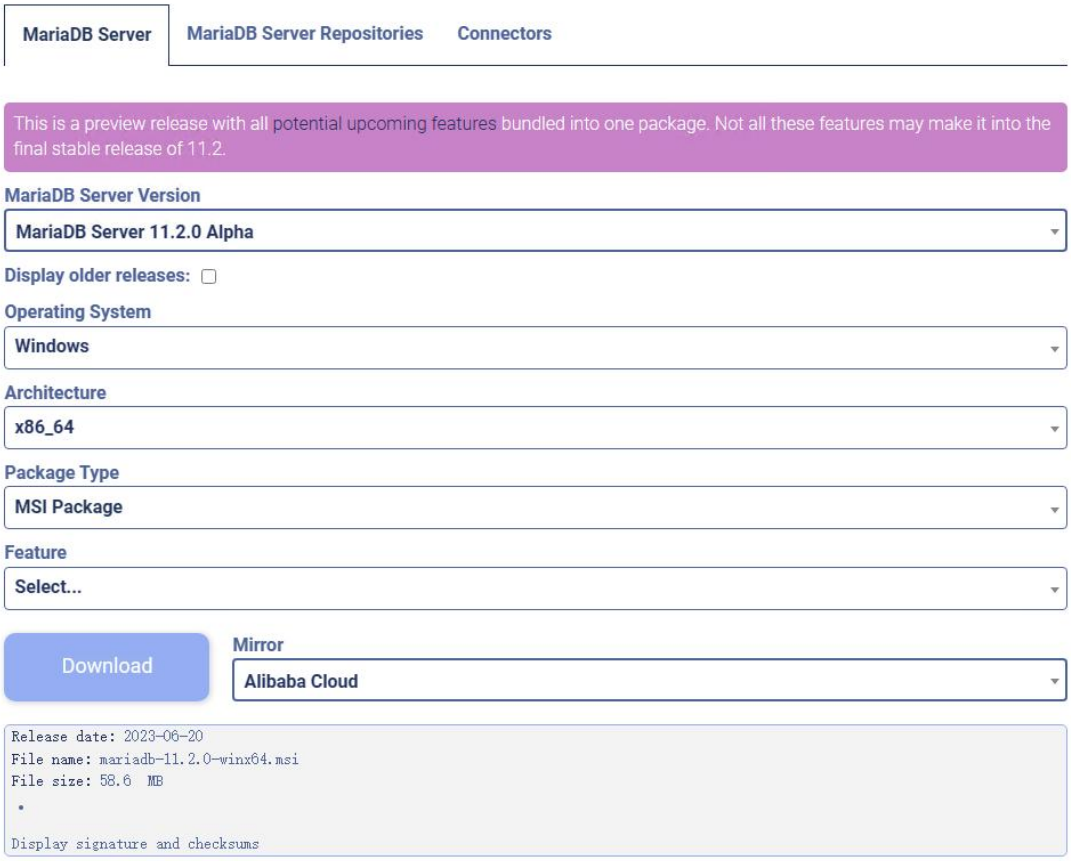


图 1-2-1 MariaDB 官方下载页面

2. 安装 MariaDB

双击下载到的文件“mariadb-11.2.0-winx64.msi”启动安装，如图 1-2-2 所示。

图 1-2-2 启动安装

点击勾选“I accept the terms in the License Agreement”接受许可协议，然后点击“Next”下一步按钮，如图 1-2-3 所示。

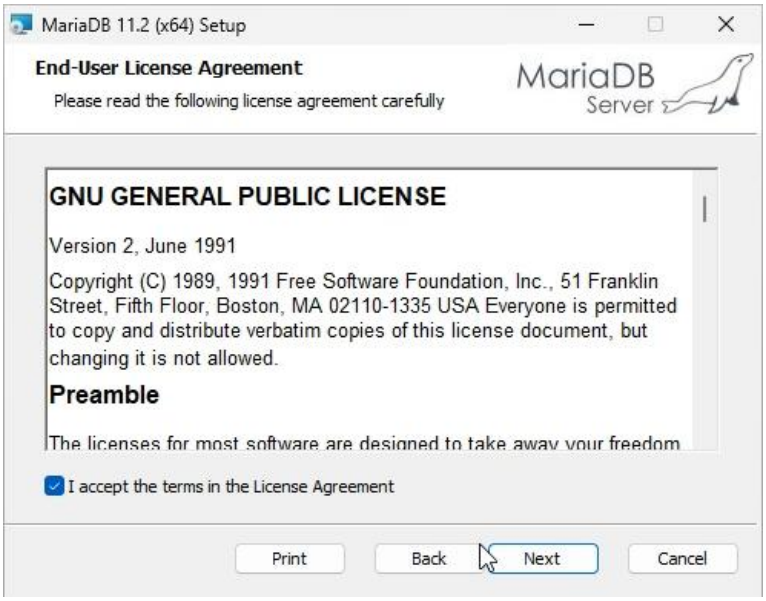


图 1-2-3 接受 MariaDB 许可协议

点击“Browse…”按钮更改 MariaDB 安装路径为：C:\Program Files\MariaDB 11.2，如图 1-2-4 所示。

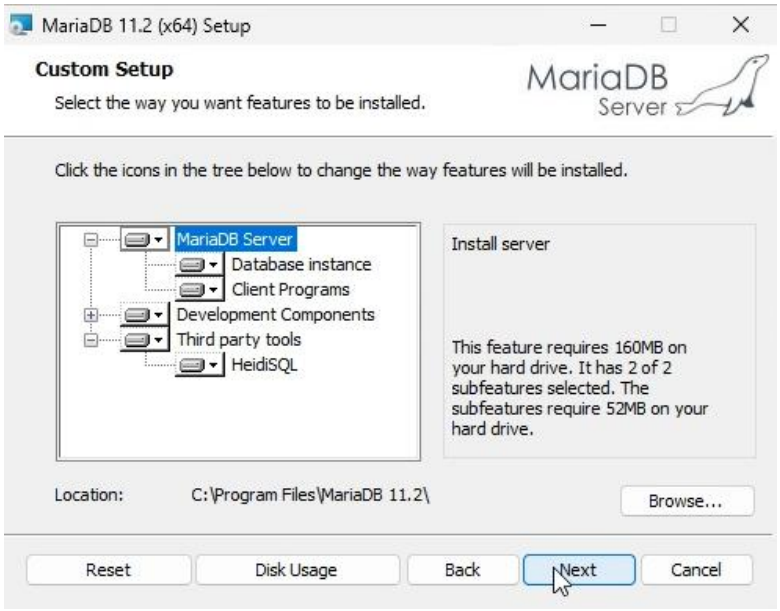


图 1-2-4 设置 MariaDB 安装路径

“Modify password for database user ‘root’”输入“root”账号的密码，勾选“Use UTF8 as default server’s character set”字符默认使用 UTF-8 的编码方式，其他的默认不管，点击“Next”按钮进入下一步，如图 1-2-5 所示。

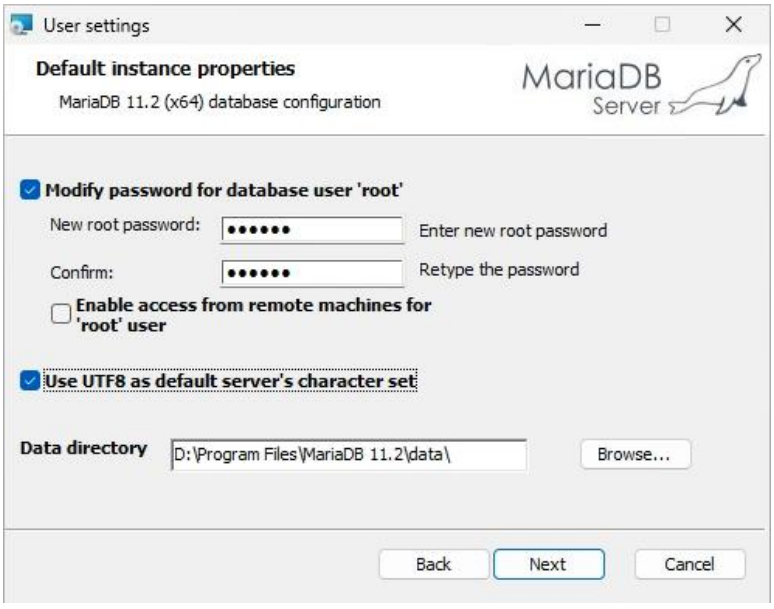


图 1-2-5 设置 MariaDB 密码及编码方式

MariaDB 的服务名称和端口等都使用默认的即可，点击“Next”进入下一步，如图 1-2-6 所示。

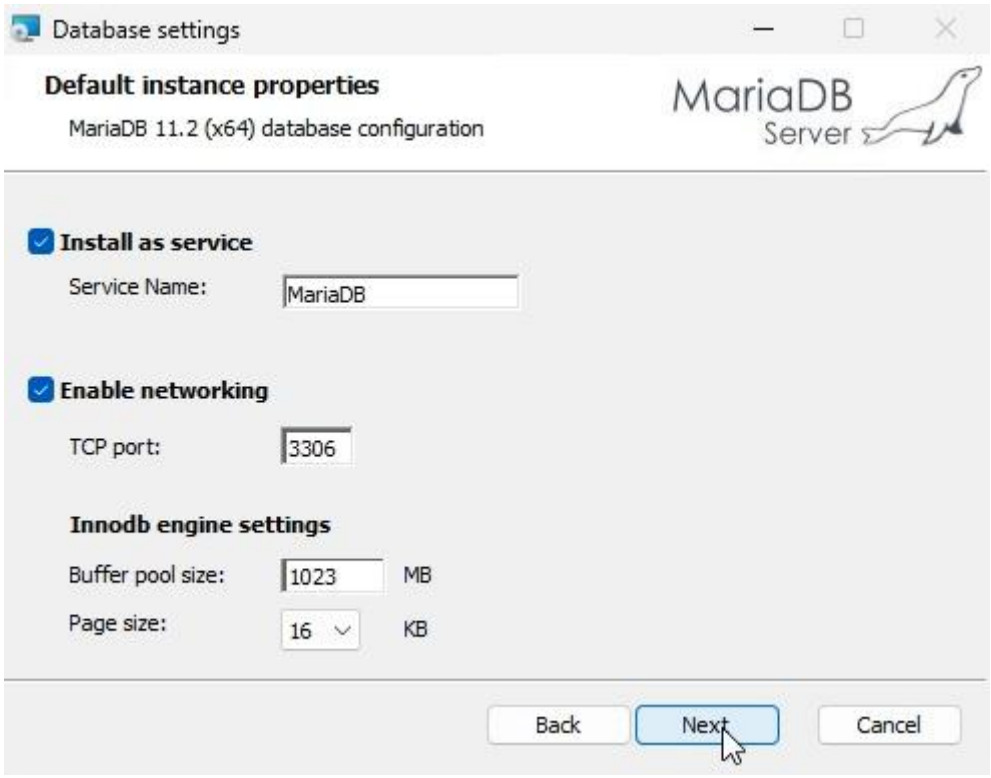


图 1-2-6 设置 MariaDB 的服务名称和端口

点击“Install”按钮，等待完成安装。如图 1-2-7 所示。

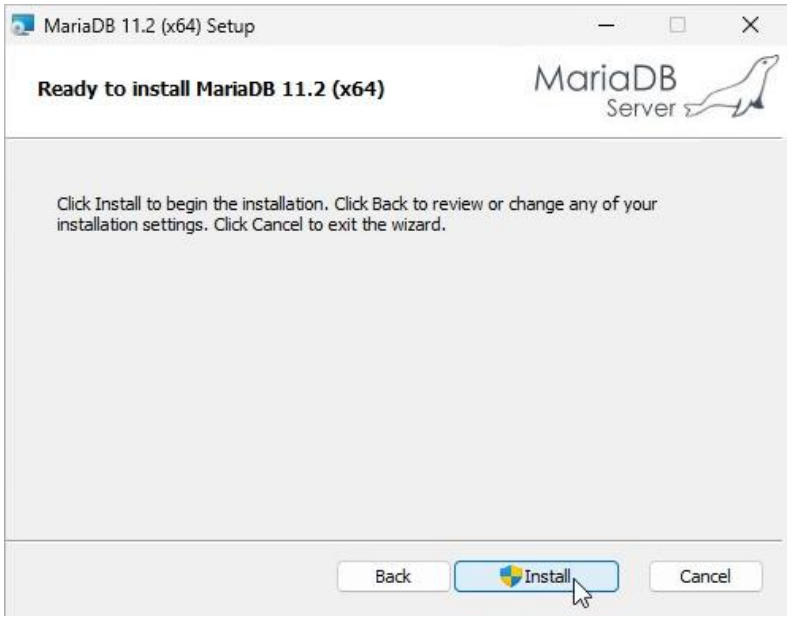


图 1-2-7 确认安装 MariaDB

安装完成后在桌面生成一个“HeidiSQL”图标，这个是一个客户端工具。如图 1-2-8 所示。

图 1-2-8 HeidiSQL 工具图标

3. 启动和关闭 MariaDB

安装好后在 Windows 的开始菜单中会生成快捷菜单，方便我们对 MariaDB 的使用，如图 1-2-9 所示。

鼠标右击“Command Prompt (MariaDB 11.2 (x64))”选择“以管理员身份运行”打开设置了 MariaDB 环境的命令提示符窗口。

安装包安装好后的 MariaDB 是已经启动的了，输入命令：net stop MariaDB 然后回车，执行成功此时 MariaDB 已经关闭了。输入命令：net start MariaDB 然后回车，执行成功此时 MariaDB 是启动了的。如图 1-2-10 所示为两条命令行的执行结果。

注意此处的服务名“MariaDB”是我们安装时输入的服务名称。

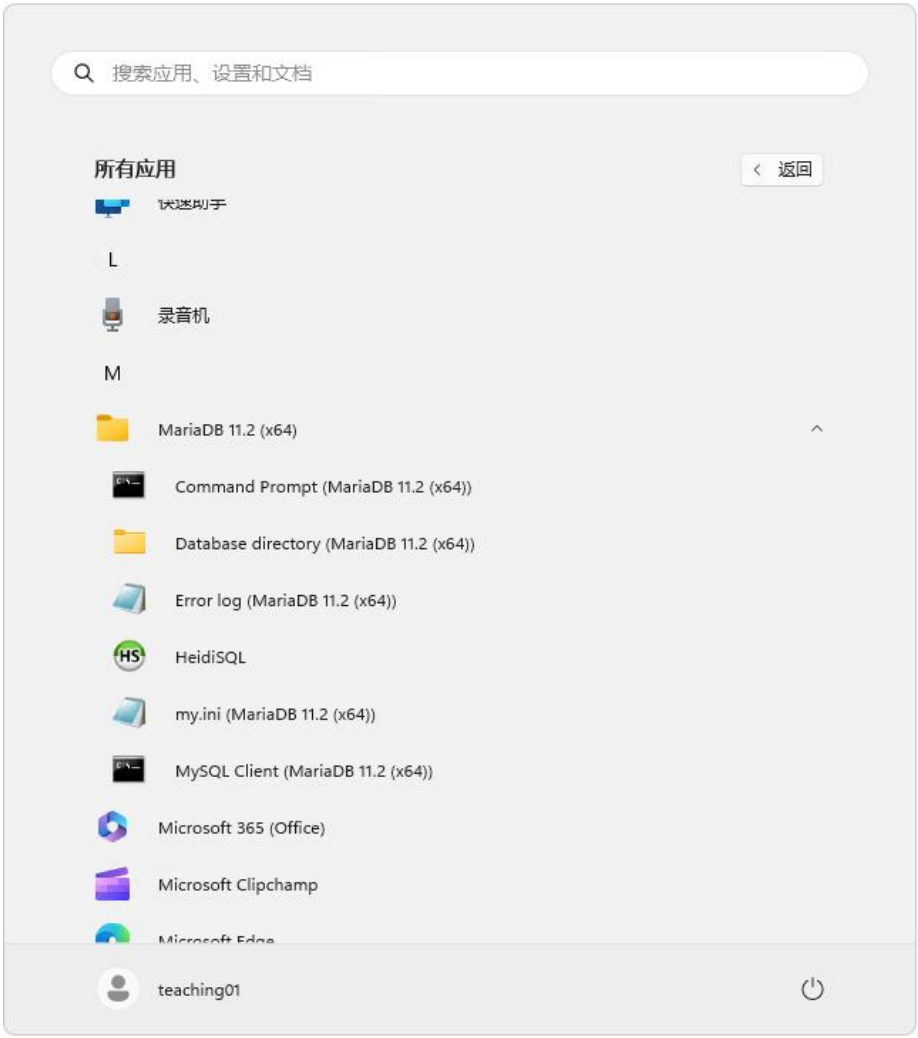


图 1-2-9 MariaDB 在 Windows 开始中的快捷菜单

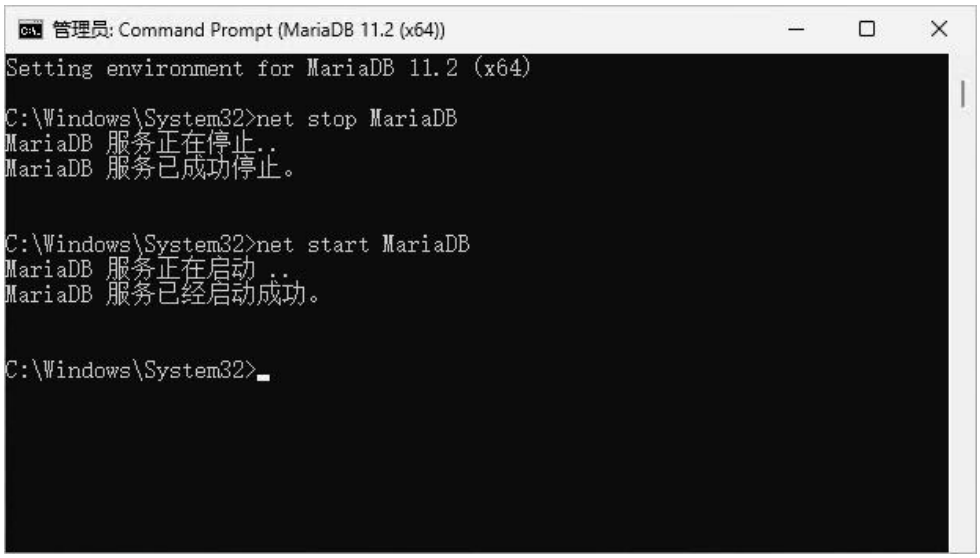


图 1-2-10 MariaDB 关闭启动结果

三、命令行登录操作数据库

点击“MySQL Client(MariaDB 11.2(x64))”直接可以打开并执行 MariaDB 登录命令，只需要输入密码即可登录，如图 1-2-11 所示。

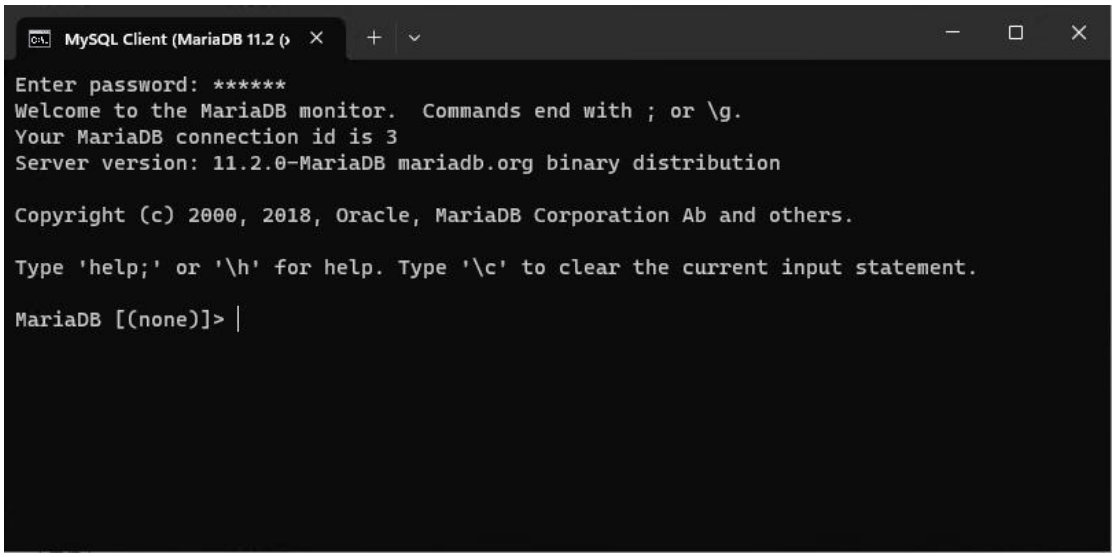


图 1-2-11 登录 MariaDB

输入：show databases; 然后回车，查看 MariaDB 中所有数据库，注意必须要以英文逗号“;”结束，如图 1-2-12 所示。

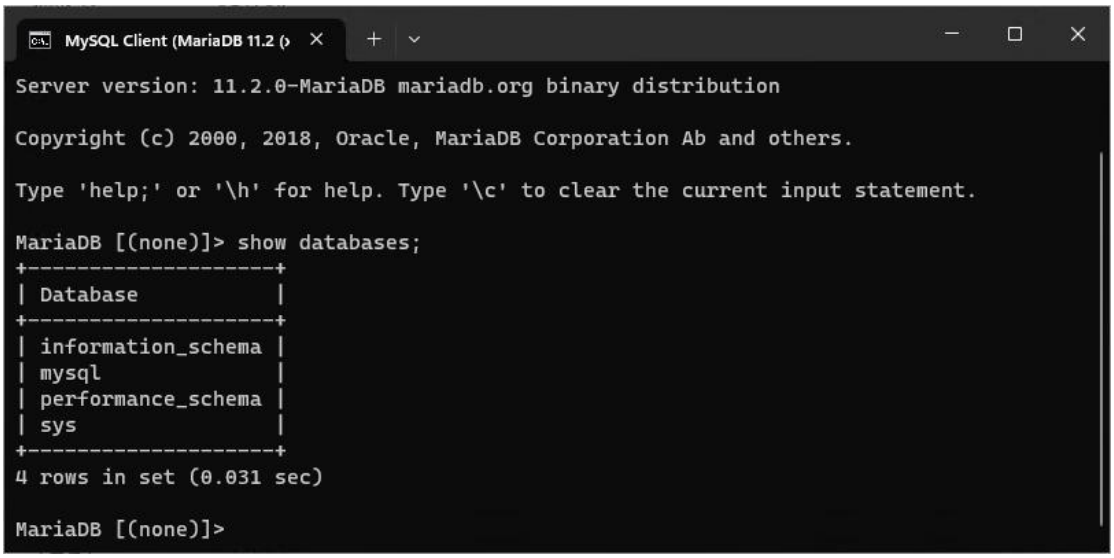


图 1-2-12 查看 MariaDB 的所有数据库

输入：use mysql; 然后回车，切换当前使用的数据库为 mysql，输入：show tables; 然后回车，查看该数据库的所有表，如图 1-2-13 所示。

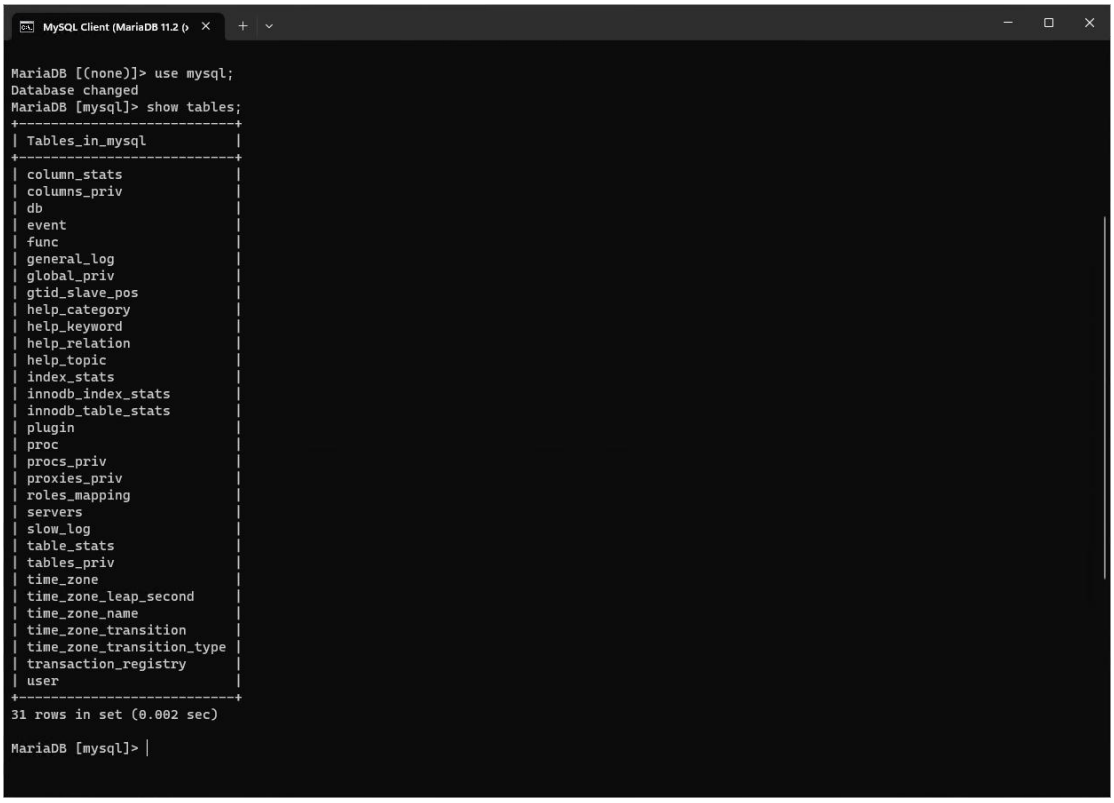


图 1-2-13 查看 mysql 数据库里的所有表

输入：exit 然后回车，可以退出客户端操作，如图 1-2-14 所示。

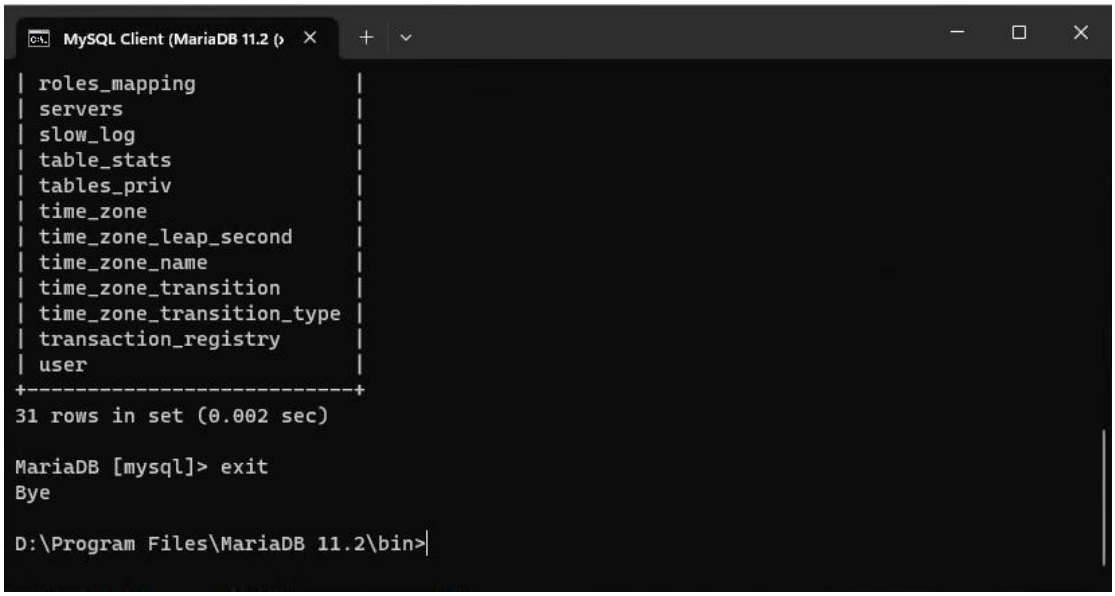


图 1-2-14 退出 MariaDB 客户端操作

四、使用客户端工具操作数据库

1. 连接 MariaDB 数据库服务器

打开“HeidiSQL”客户端工具，左侧是我们已经创建的数据库连接会话列表，首次使用需要点击左下角“新建”按钮新建一个连接会话，先把会话名称改自己想要的，然后在右侧对连接信息进行设置，如图 1-2-15 所示。

目前除了“密码”一项之外其他的所有设置项都无需进行修改，输入数据库 root 账号的密码之后在左下角点击“保存”按钮保存我们的设置。点击左下方的“打开”按钮打开连接，如图 1-2-16 所示。

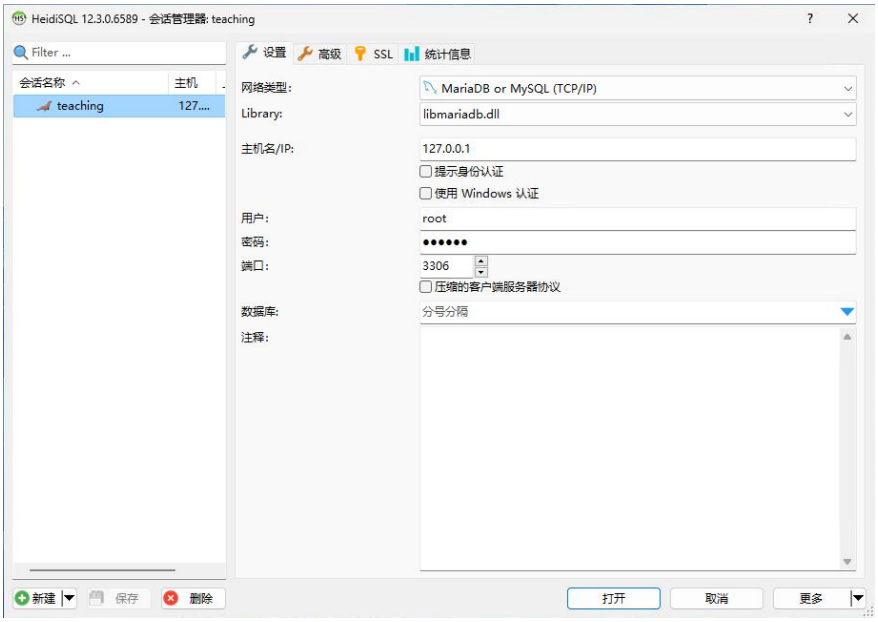


图 1-2-15 新建数据库连接

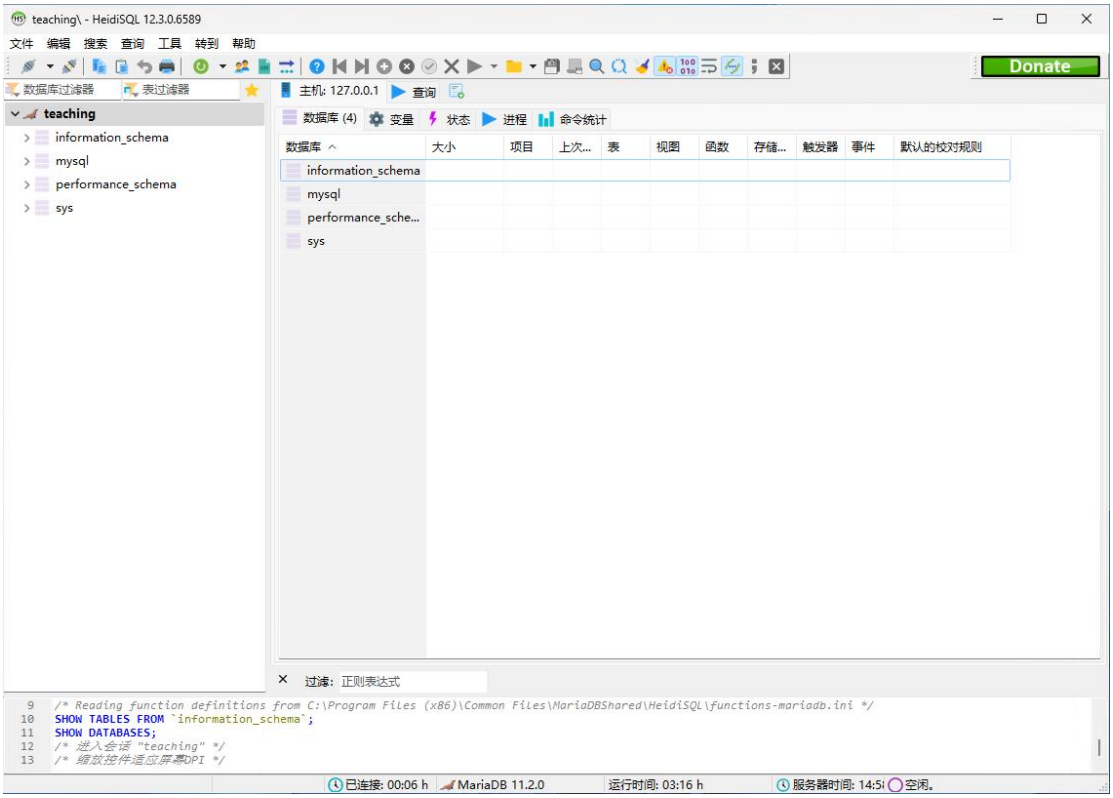


图 1-2-16 打开数据库连接界面

2. 创建数据库

右击打开菜单选择“创建新的(O)”，然后选择“数据库”，如图 1-2-17 所示。

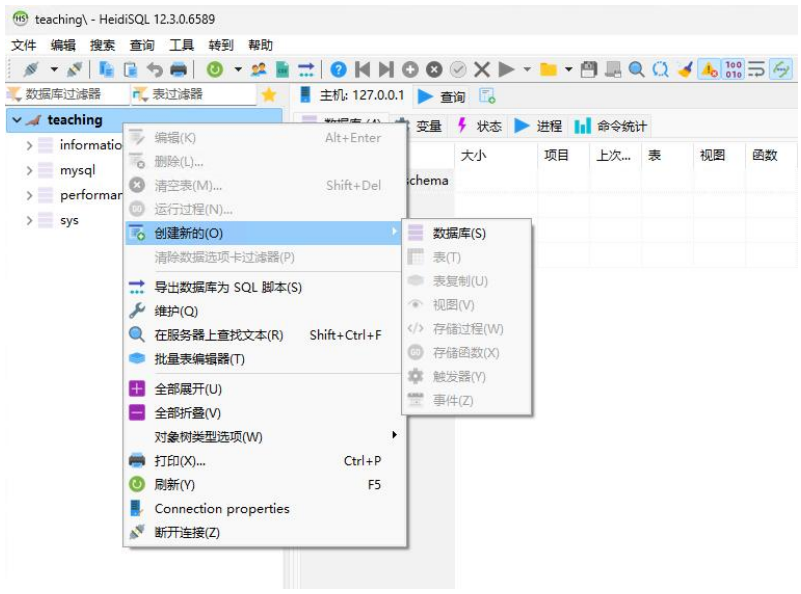


图 1-2-17 新建数据库操作

在弹出的新建窗口中“名称”一项输入“bio_seq_manage”，“字符校对”一项选择“utf8mb4_general_ci”，如图 1-2-18 所示。

图 1-2-18 数据库新建窗口

3. 创建数据库表

右击“boi_seq_manage”数据库在打开菜单选择“创建新的(O)”，然后选择“表(T)”，如图 1-2-19 所示。

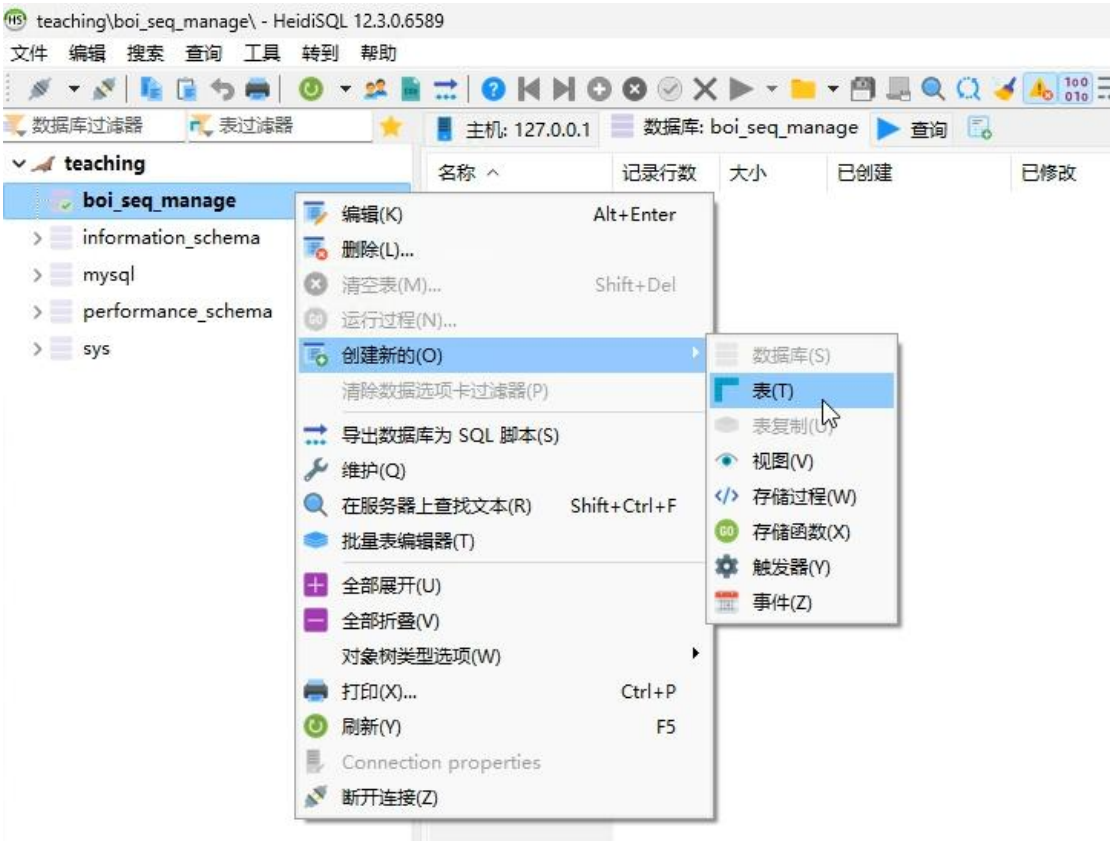


图 1-2-19 新建数据库表操作

在右侧出现一个“表:[Untitled]”，需要我们进行编辑创建数据表字段等信息，首先输入名称和注释分别是“users”和“用户表”，然后在字段那里点击“添加”按钮添加字段，第一个字段名称“user_id”，数据类型选择“VARCHAR”，字段长度默认 50，是否无符号默认无，是否允许 Null 把勾去掉表示不允许，默认值为“无默认值”，注释中输入“用户 ID”，校对规则默认不选（保存后会默认成 utf8mb4_general_ci）。

建好“user_id”用户 ID 字段需要把这个字段设置成关键字段，在上面的选项卡中选择“索引(1)”，左侧点击“添加”按钮后列表中出现一行信息，点击“类型/长度”一列将其改成“PRIMARY”关键字段，在“名称”一列右击“PRIMARY KEY”在出现的菜单中选择“添加字段”会把字段“user_id”添加为关键字段。最后点击左下角“保存”按钮进行保存，“users”用户表创建成功。

依次加入“user_name”用户名和“password”密码两个字段，再次点击左下角“保存”按钮进行保存。最终最基础的用户表建好，如图 1-2-20 所示。

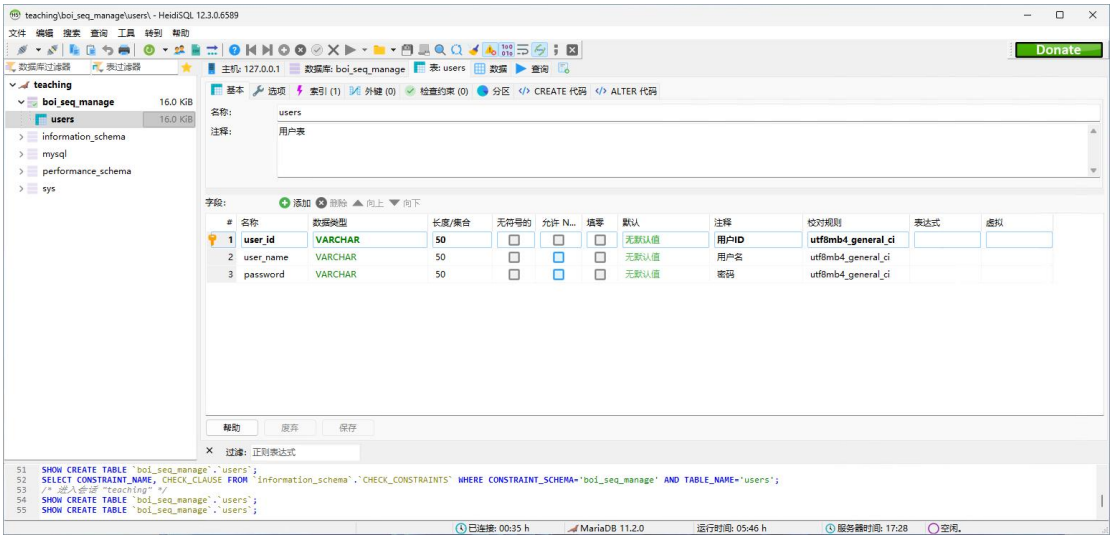


图 1-2-20 初建用户表

4. 插入、更改和删除数据

在左侧选中需要操作的数据表，然后在右侧的选项卡里选中“数据”，下方出现数据表的数据列表，如图 1-2-21 所示。

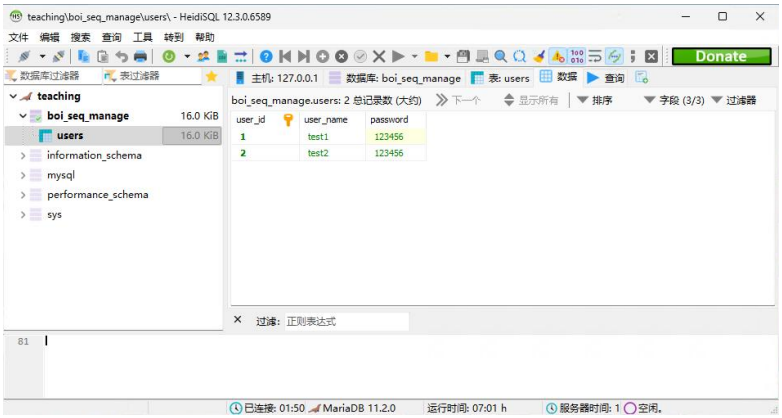


图 1-2-21 数据库表的数据列表

插入数据操作方法，在工具条中点击  按钮或鼠标右击数据列表区域出现的菜单中点击“插入记录行”，然后直接点击进入内容输入即可，如图 1-2-22 所示。

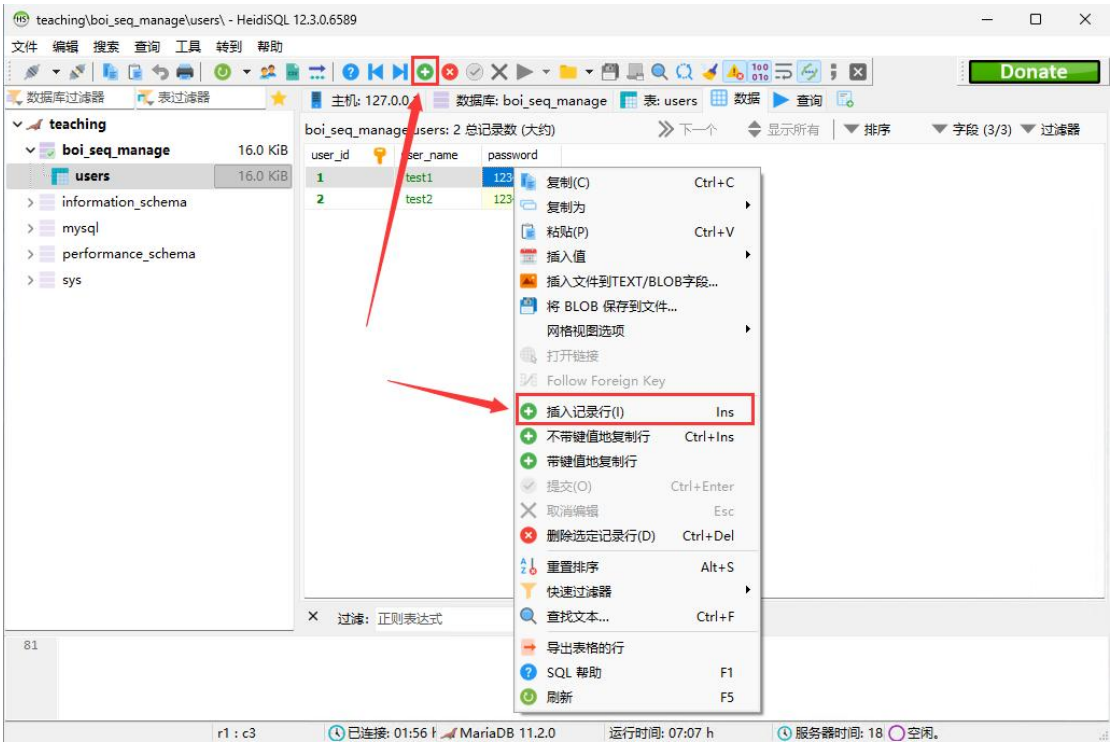


图 1-2-22 向表插入数据

更新数据的操作方法，直接点击需要更改的数据，然后进行更改和插入数据行类似，如图 1-2-23 所示。

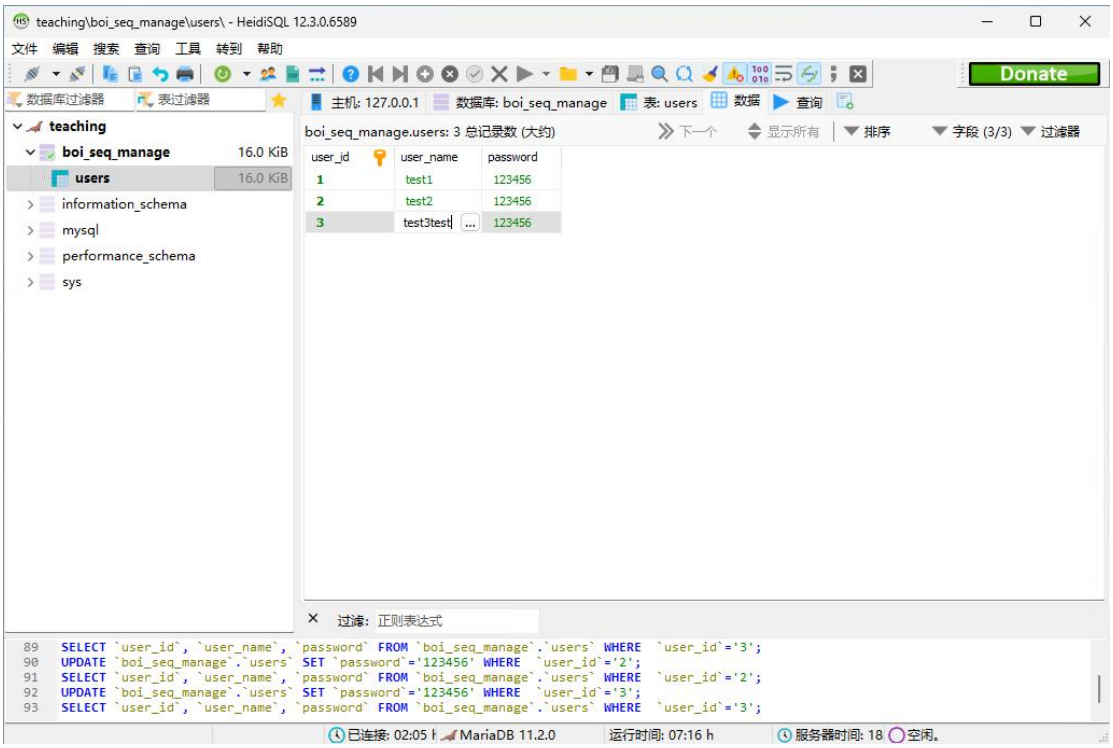


图 1-2-23 更改数据

删除数据行的操作方法，选中需要删除的数据行然后点击工具条上的  删除按钮或者

鼠标对着要删除的数据行右击在菜单中选择“删除选定记录行(D)”，如图 1-2-24 所示。

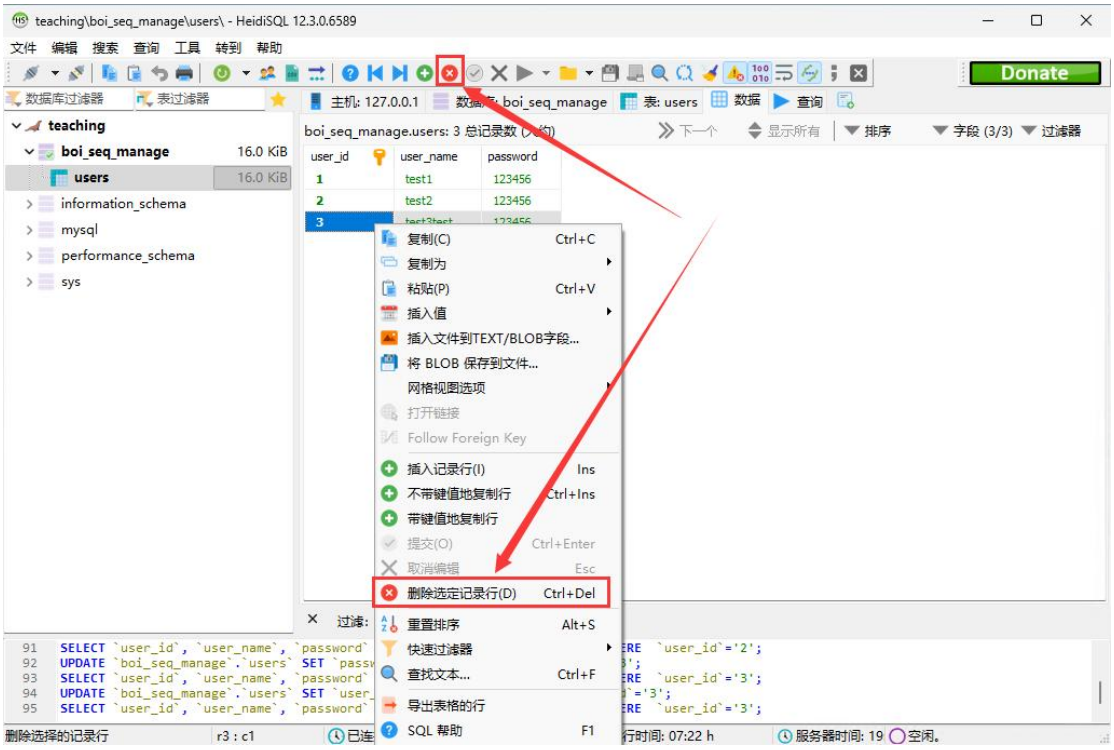


图 1-2-24 删除数据行

五、认识 SQL 语句

1. SQL 简介

结构化查询语言(Structured Query Language)简称 SQL，是对数据库进行操作的一种语言，用于对关系数据库系统存取数据、更新数据和管理数据库。

2. 插入语句

insert 语句可以用来将数据插到数据库表中，使用的一般形式如下：insert [into] 表名[(列名 1, 列名 2, 列名 3, ...)] values (值 1, 值 2, 值 3, ...); 其中 [] 内的内容是可选的，例如，要给“boi_seq_manage”数据库中的 users 表插入一条记录，执行语句：insert into users values('3', 'test3', '123456'); 为了建议类名写上顺序一一对应，执行语句：insert into users(user_id, user_name, password) values('3', 'test3', '123456'); 打开 HeidiSQL 客户端，选中我们需要操作的数据库，左侧点击“查询”选项卡，把 SQL 输入到下方代码编辑区域，完成输入后点击工具条的 执行按钮，如图所示 1-2-25 所示。

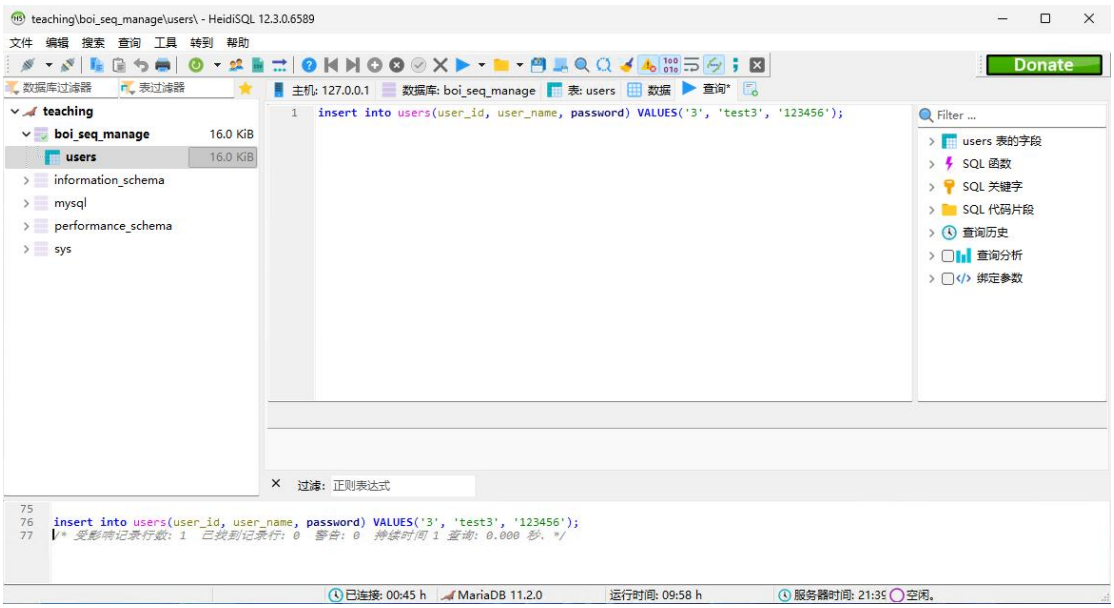


图 1-2-25 执行 SQL 插入语句

3. 查询语句

select 语句常用来根据一定的查询规则到数据库中获取数据，其基本的用法为：select 列名称 from 表名称[查询条件]；例如：要查询 users 表中所有用户，输入语句 select user_id, user_name, password from users; 也可以使用通配符*查询表中所有的字段，语句：select * from users;

在 HeidiSQL 客户端的查询中执行上述 SQL 语句查看结果，如图 1-2-26 所示。

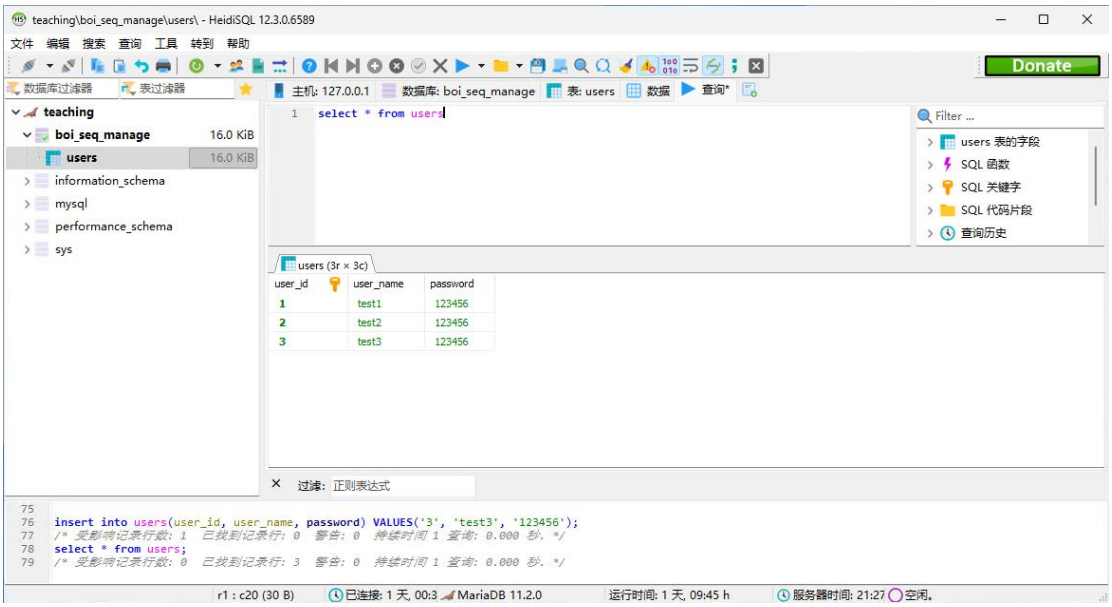


图 1-2-26 执行 SQL 查询语句

条件查询：where 关键词用于指定查询条件，用法形式为：select 列名称 from 表名称

where 条件；以查询符合条件的信息。

语句：select * from users where user_id='1'；查询用户 ID 等于“1”的用户信息。

语句：select * from users where user_name like '%test%'；查询用户名中带有“test”字的所有用户信息。

4. 更新语句

更新表中的数据 update 语句可用来修改表中的数据，基本的使用形式为：update 表名称 set 列名称=新值 where 更新条件。

语句：update users set password='888888' where user_id='1'；将用户 ID 为“1”的用户密码更改成“888888”。同时输入一条查询语句可以查看更新后的数据，如图 1-2-27 所示。

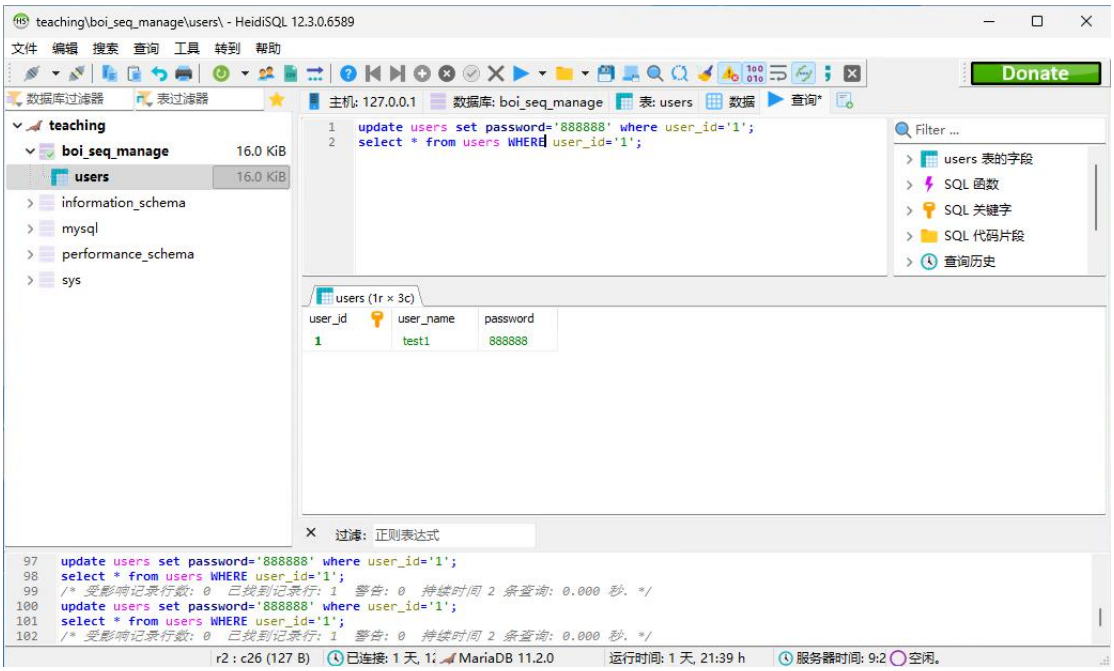


图 1-2-27 执行 SQL 更新语句

5. 删除语句

删除表中的数据 delete 语句用于删除表中的数据，基本用法为：delete from 表名称 where 删除条件；HeidiSQL 中输入语句：delete from users where user_id='1'；删除表中用户 ID 为“1”的用户，注意一定要写条件否则将删除所有数据：delete from users，如图 1-2-28 所示。

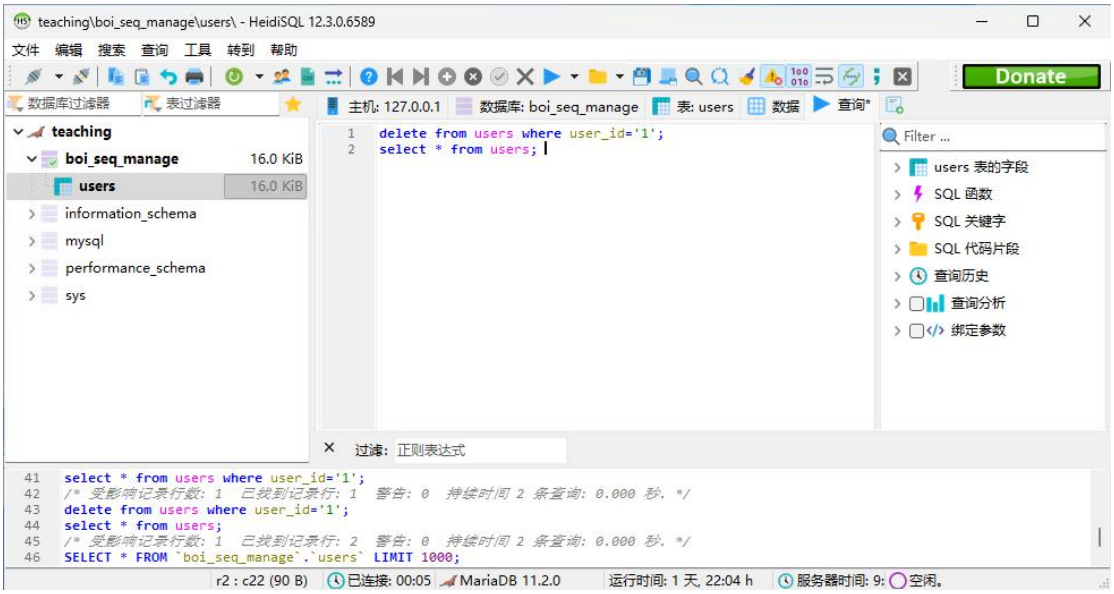


图 1-2-28 执行 SQL 删除语句

任务实施

Step1 MariaDB 安装及运行

Step2 命令行登录操作数据库

Step3 使用客户端工具连接数据库并登录，然后创建数据库、数据表及插入、更改和删除数据

Step4 使用 SQL 语句对数据表进行插入、查询、更新和删除数据

活动三 Git 的安装配置与 Git 对代码版本管理使用

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。项目使用 Java 语言进行开发，要求我们管理系统开发的代码使用 Git 进行版本管理。项目开发前让我们安装配置 Git 并且了解它的使用吧！

任务目标

1. 能安装和配置 Git 插件。
2. 掌握 Git 的使用方法。

任务分析

1. 目前能进行代码版本管理的有哪些？如何在 Eclipse 安装和配置 Git 插件？
2. Git 有哪些代码管理功能以及怎么使用？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、Git 插件安装和配置

1. 安装 Git 插件

在 Eclipse 中安装 Git 插件即可使用 Git 而不需要安装 Git 核心程序也可以，在 Eclipse 中可以通过以下方式安装：

方式一：从 Eclipse 插件市场下载安装。

打开Eclipse的菜单中选择Help→Eclipse Marketplace...，在“Find:”输入EGit，找到EGit，点击Install即可，如图1-3-1所示。

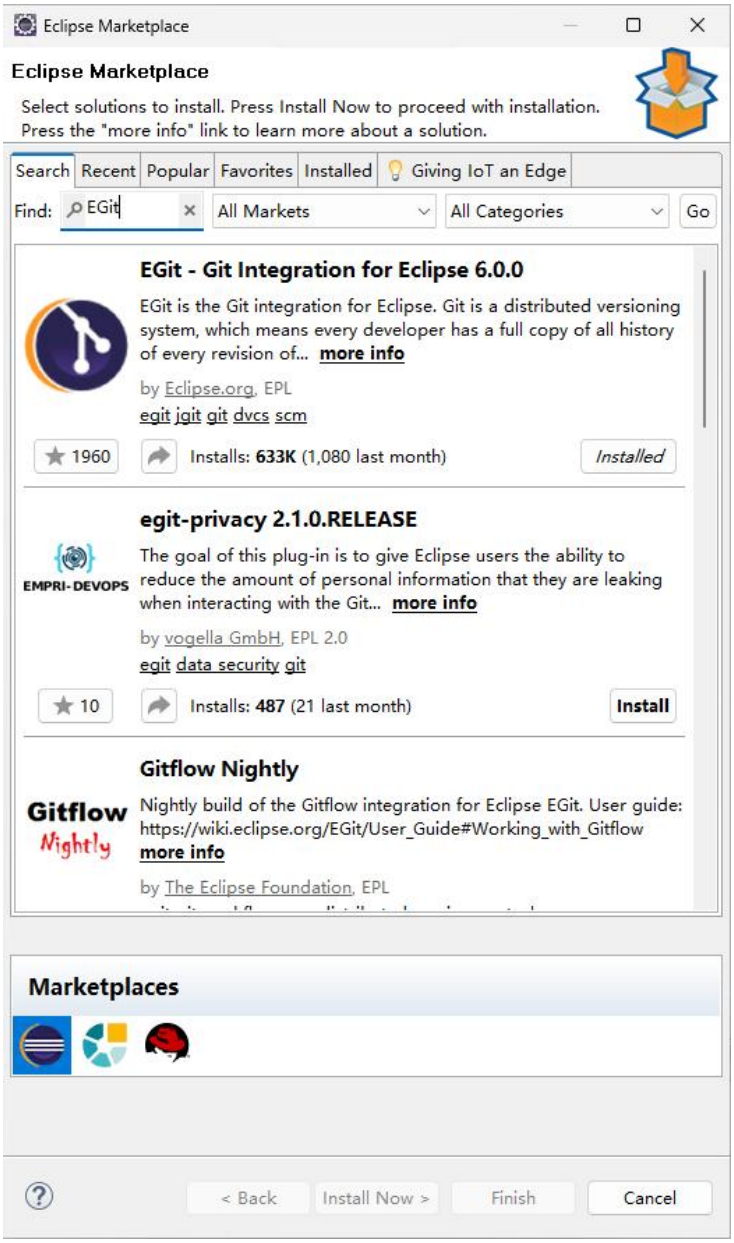


图 1-3-1 Eclipse 插件市场下载安装 Git

方式二：指定插件 URL 下载安装。

打开Eclipse，Help—>Install New Software…

如图 1-3-2 所示，点击 Add，在 Name 中输入 EGit，Location 中输入：https://download.eclipse.org/egit/updates-6.6，点击“Add”按钮。

（注：关于 url→在浏览器地址栏输入 http://wiki.eclipse.org/EGit/FAQ#Where_can_I_find_older_releases_of_EGit.3F，在页面上寻找与自己 Eclipse 版本对应的 EGit 插件版本，然后找到对应 EGit 版本的 url）

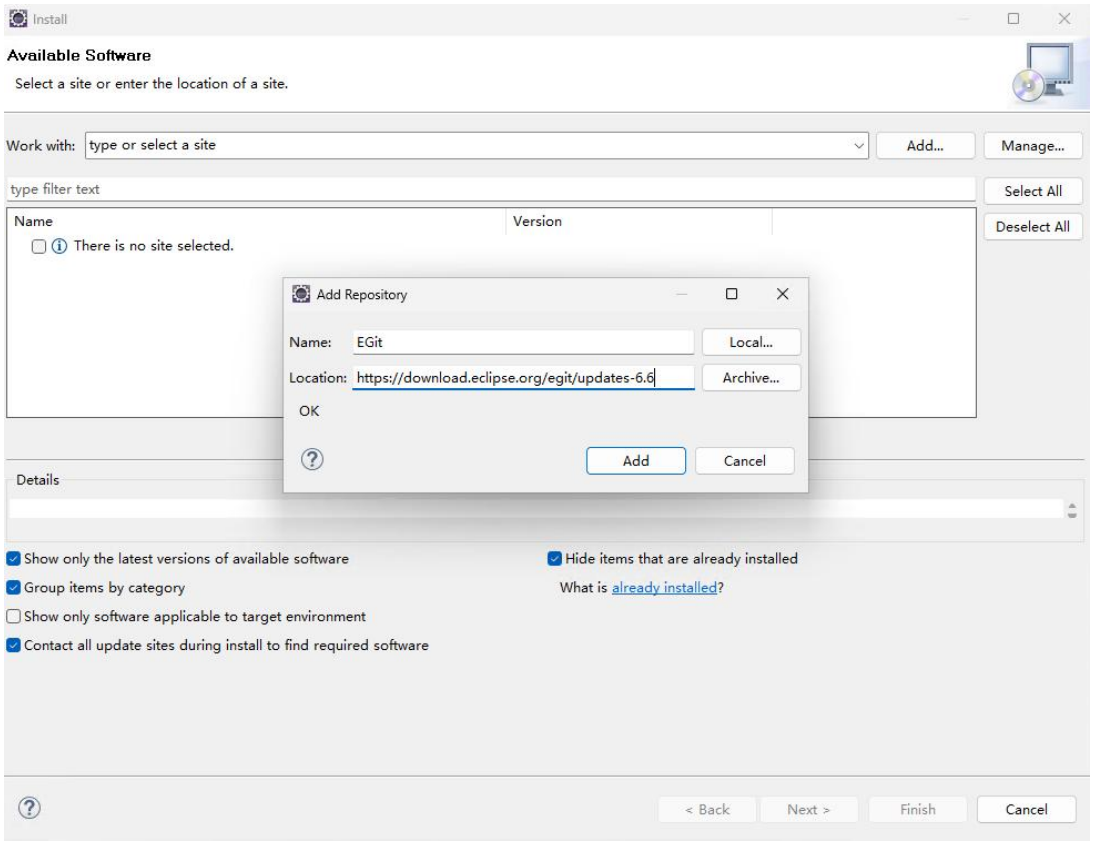


图 1-3-2 eclipse URL 下载安装

如图 1-3-3 所示，勾选所有项目，然后点击“Next>”按钮，直到 Finish。安装完成后需要重启 Eclipse。

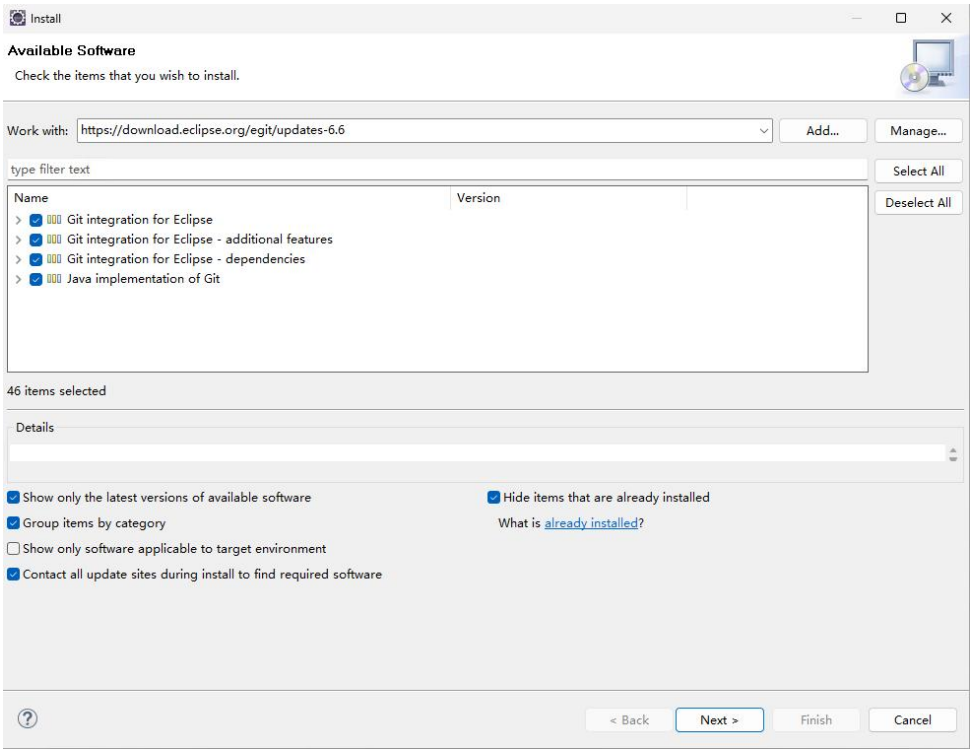


图 1-3-3 eclipse 勾选安装项目

2. Git 配置

（1）用户配置

安装好 Git 插件后，先进行 Git 全局设置（用户名和邮箱）

Eclipse 菜单中选择 Window→Preferences，弹出窗口中选择 Version Control(Team)→Git→Configuration，如图 1-3-4 所示。

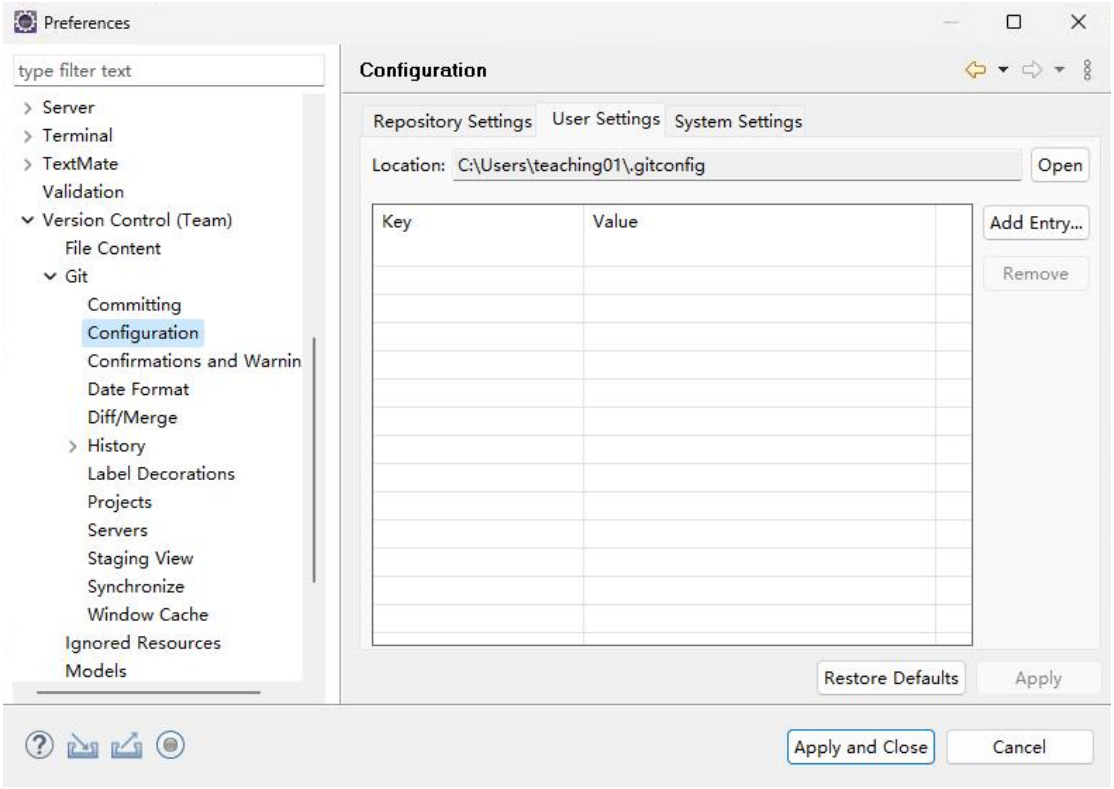


图 1-3-4 Git 环境配置窗口

点击 Add Entry，在弹出框里面输入 key 和 value 的值

①名字的 key：user.name；value：是你的 Github 用户名。

②邮箱的 key：user.email；value：你的登陆 GitHub 邮箱账号。

以后你使用 Git 提交代码时候，Eclipse 会自动提取这些信息，和代码一起发送到 Git 远程仓库。

（2）修改 Git 地址、用户、密码

Window→Preferences→Version Control(Team)→Git→Configuration→Repository Settings，注意还没项目创建仓库之前这里是没有任何信息的。如图 1-3-5 所示。

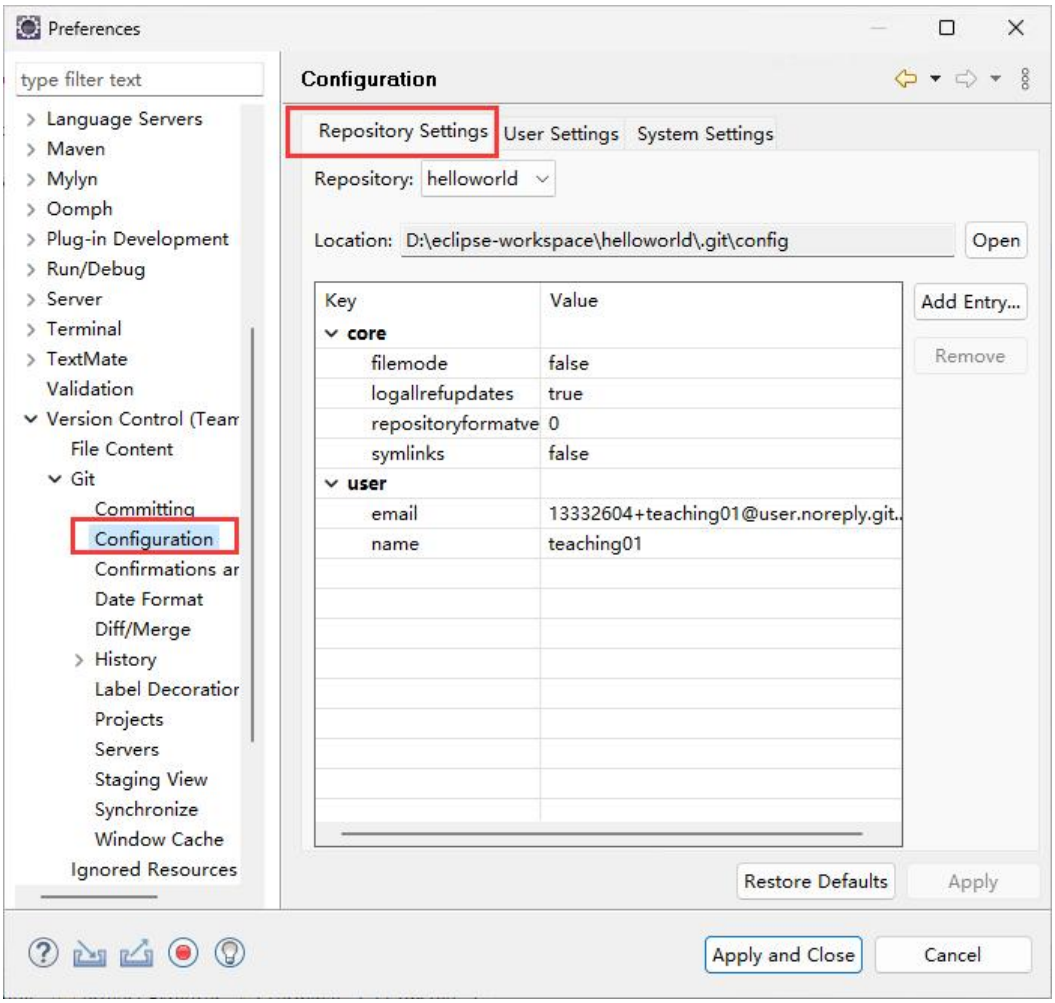


图 1-3-5 Git 的配置

（3）修改用户名和密码

Window→Preferences→General→Security→Secure Storage→Contents，注意在提交远程 git 仓库时保存了账号密码后这里才会有，如图 1-3-6 所示。

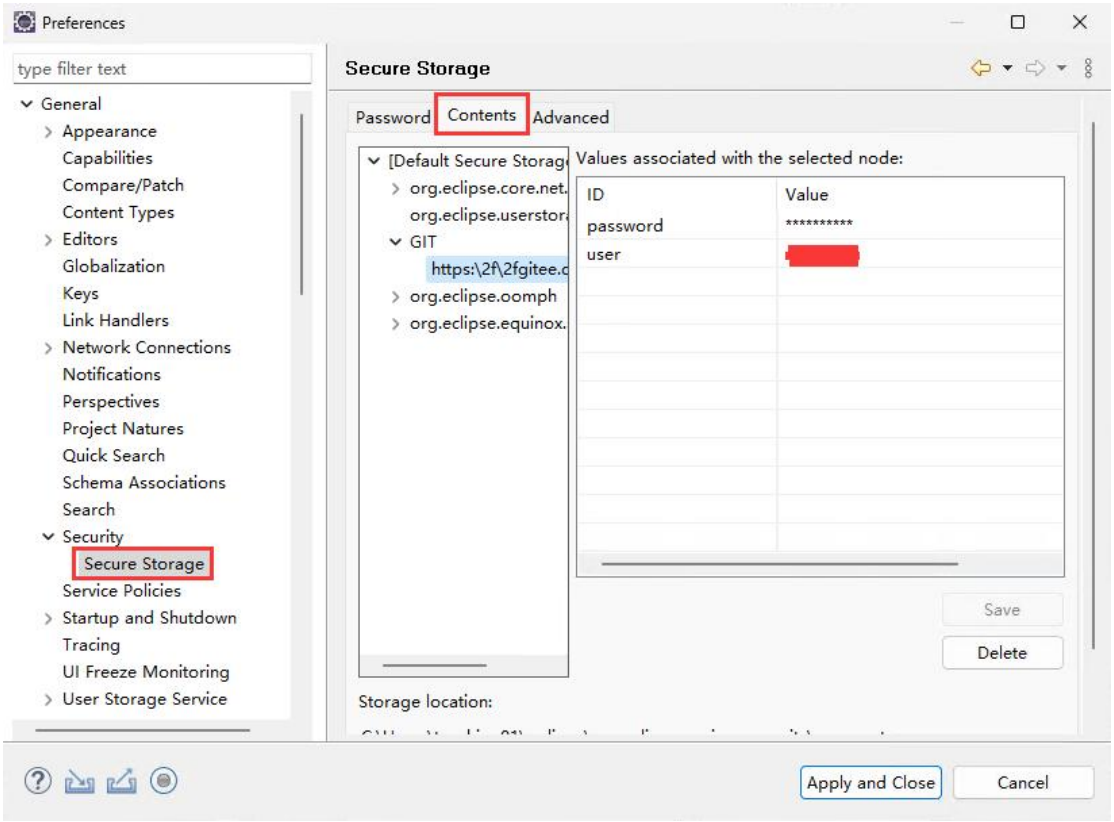


图 1-3-6 设置账号密码

二、初始化本地库

在 Eclipse 项目上右键→Team→Share Project，在弹出的配置 git 仓库窗口，勾选选项：Use or create repository in parent folder of project，如图 1-3-7 所示。

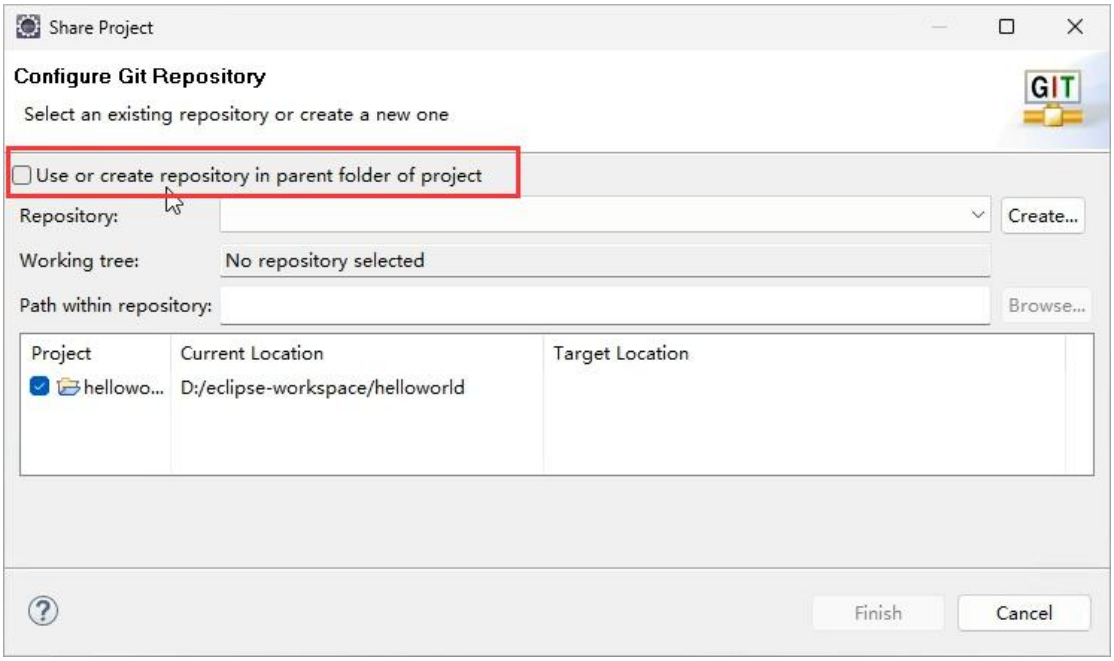


图 1-3-7 共享项目

选择要创建 git 仓库的项目，点击 Create Repository 按钮，点击 Finish 按钮创建完成，如图 1-3-8 所示。

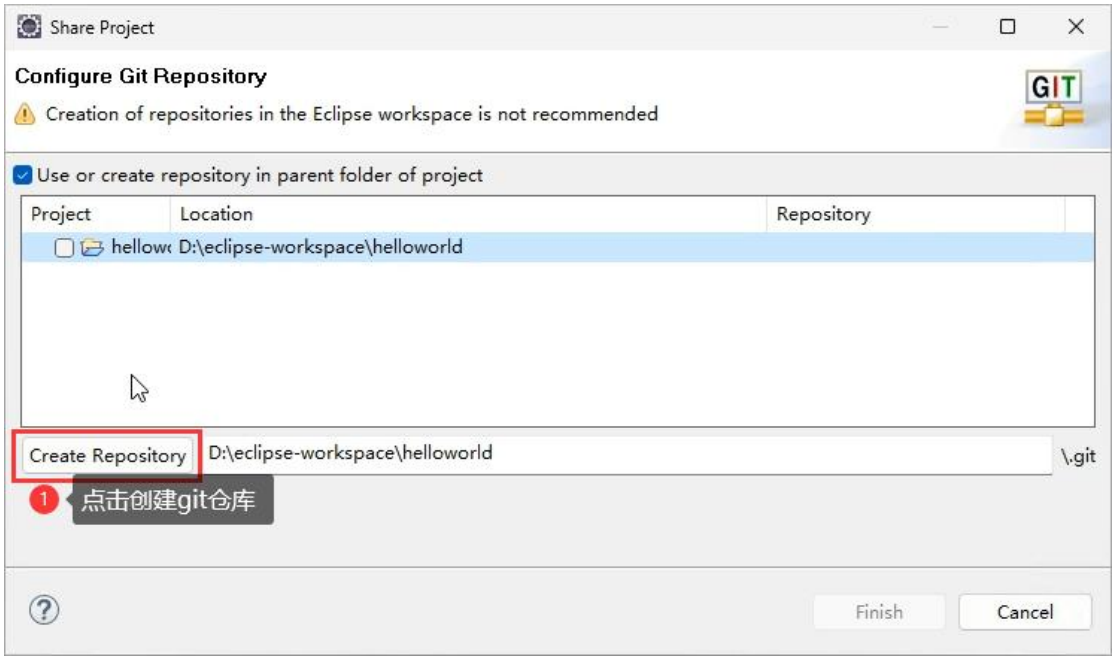


图 1-3-8 创建仓库

创建本地仓库后，如图 1-3-9 所示。

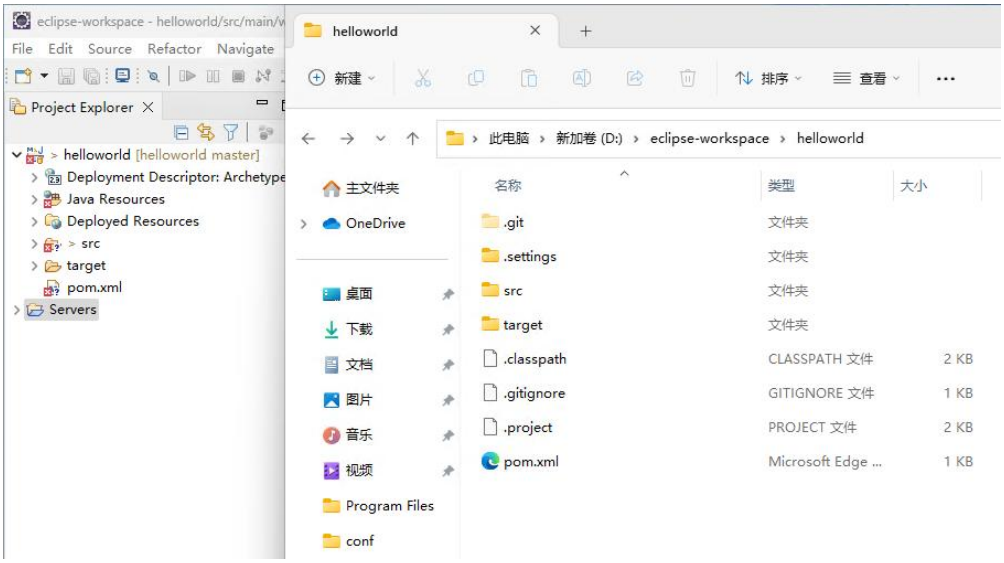


图 1-3-9 创建的本地仓库

三、设置忽略提交的文件

对于和代码本身无关的，只是 Eclipse 本身的配置文件，以及编译的 class 文件，不需要提交到 git 仓库来进行 git 管理和追踪(因为同一个团队中很难保证大家使用相同的 IDE 工具，

也就不能保证项目工程的特定文件一致，若要把这些文件也纳入到 git 管理中，势必会造成冲突），因此可以在 git 中设置忽略。

通常需要进行忽略的文件有：

```
.classpath 文件
.project 文件
.settings 目录下的所有文件
```

若要设置忽略某文件，只需在要忽略的文件上鼠标右键 Team->→Ignore，将文件加入忽略，如图 1-3-10 所示。

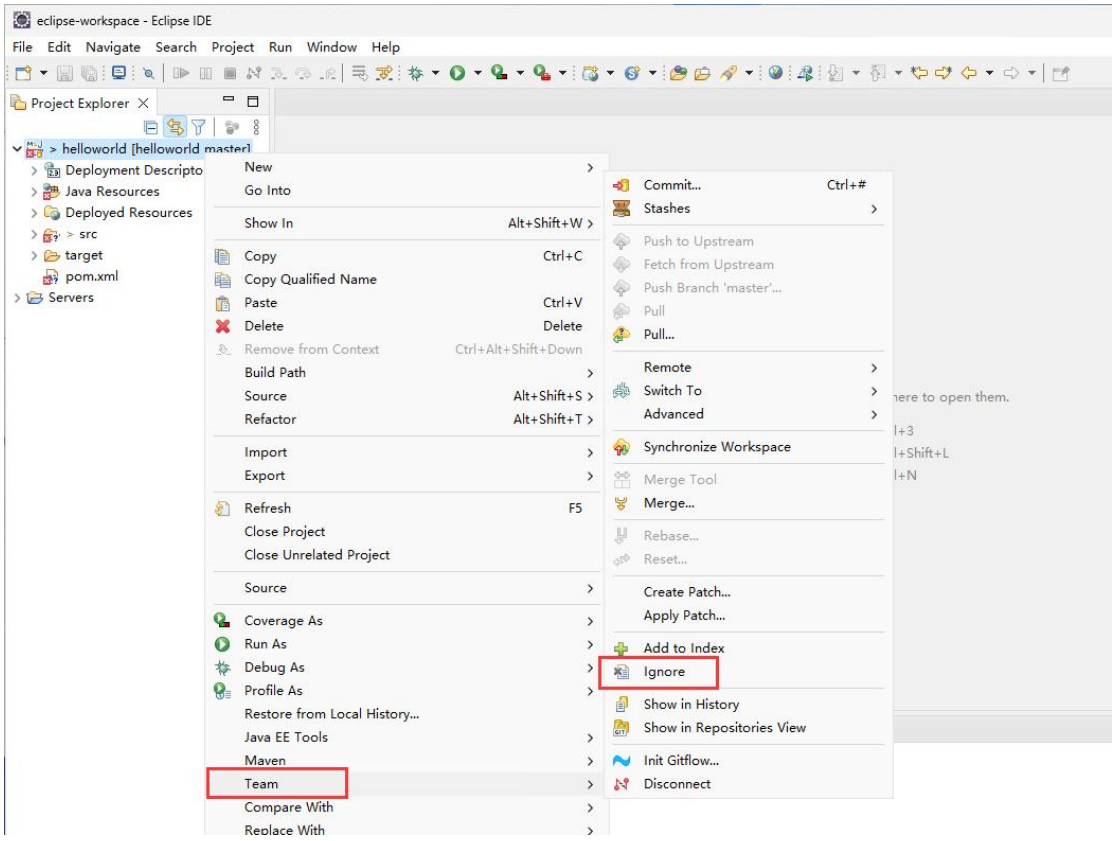


图 1-3-10 进入忽略设置

设置了忽略文件之后，会在项目根目录生成一个.gitignore 文件，如图 1-3-11 所示。

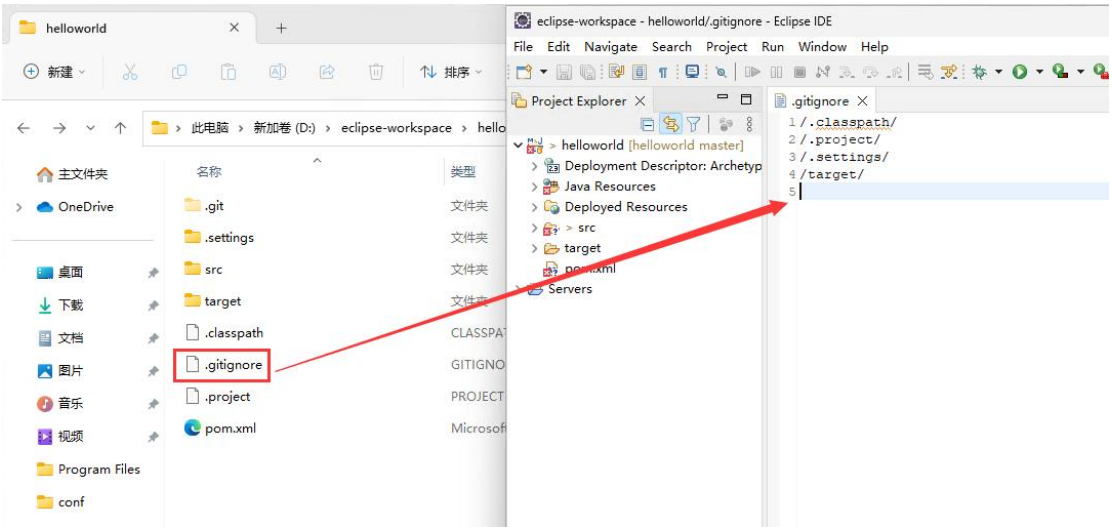


图 1-3-11 .gitignore 文件

GitHub 官网给出了 Java 开发忽略哪些文件的模板：https://github.com/github/gitignore/blob/master/Java.gitignore

可以在 Eclipse 中手动指定哪些文件忽略，也可以配置 git 选项，引入忽略文件配置模板。

模版内容如下（官网最新内容可能有所改动）：

```
# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
```

```
*.rar

# virtual machine crash logs, see
http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
replay_pid*

.classpath
.project
.settings
target
```

将 GitHub 官网给出的模板文件内容放到本地（如：D:\git\Java.gitignore），可以在此基础之上编辑内容增加或更改要忽略的文件和目录，如图 1-3-12 所示。

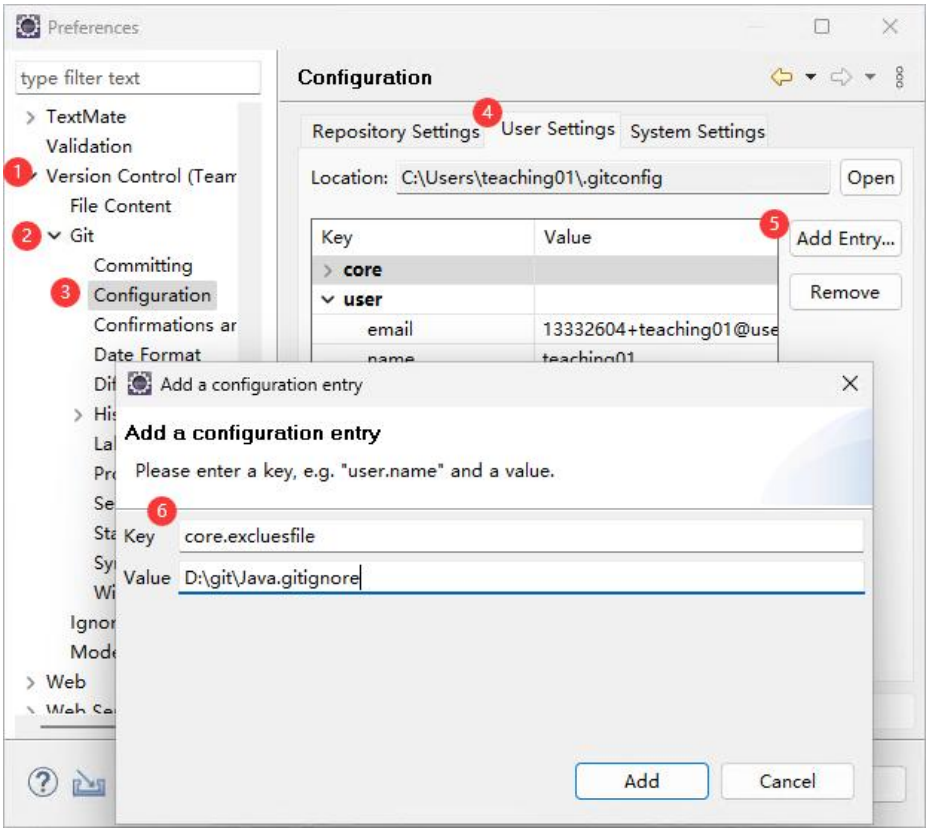


图 1-3-12 设置 git 的忽略文件

设置好之后重启 Eclipse。

四、本地库 Git 签名

Eclipse 设置 Window→Preferences→Git→Configuration，Eclipse 会自动搜索到本机的 Git 仓库地址，选择所需配置项目的 Git 仓库，选择好后，点击 Add Entry...，如图 1-3-13 所示。

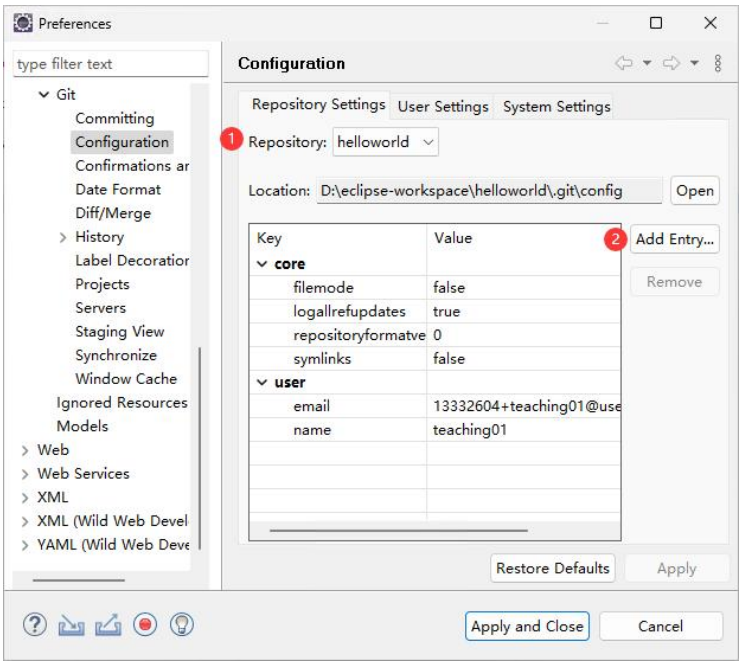


图 1-3-13 配置 git 签名

在弹出框中增加 user.name 和值，如图 1-3-14 所示。

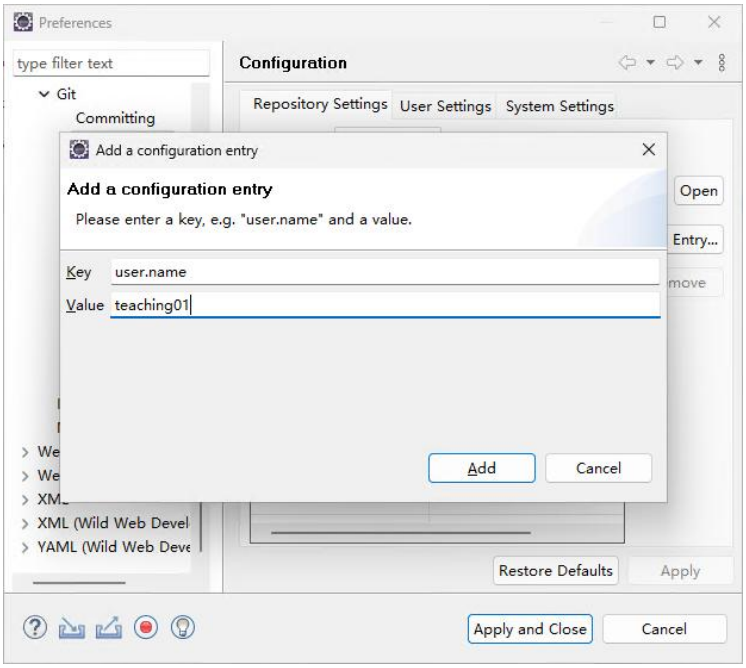


图 1-3-14 输入 key 和 value

重复上一步，添加 user.email。

五、Eclipse 中 Git 图标含义

Git 管理的示例项目，如图 1-3-15 所示。

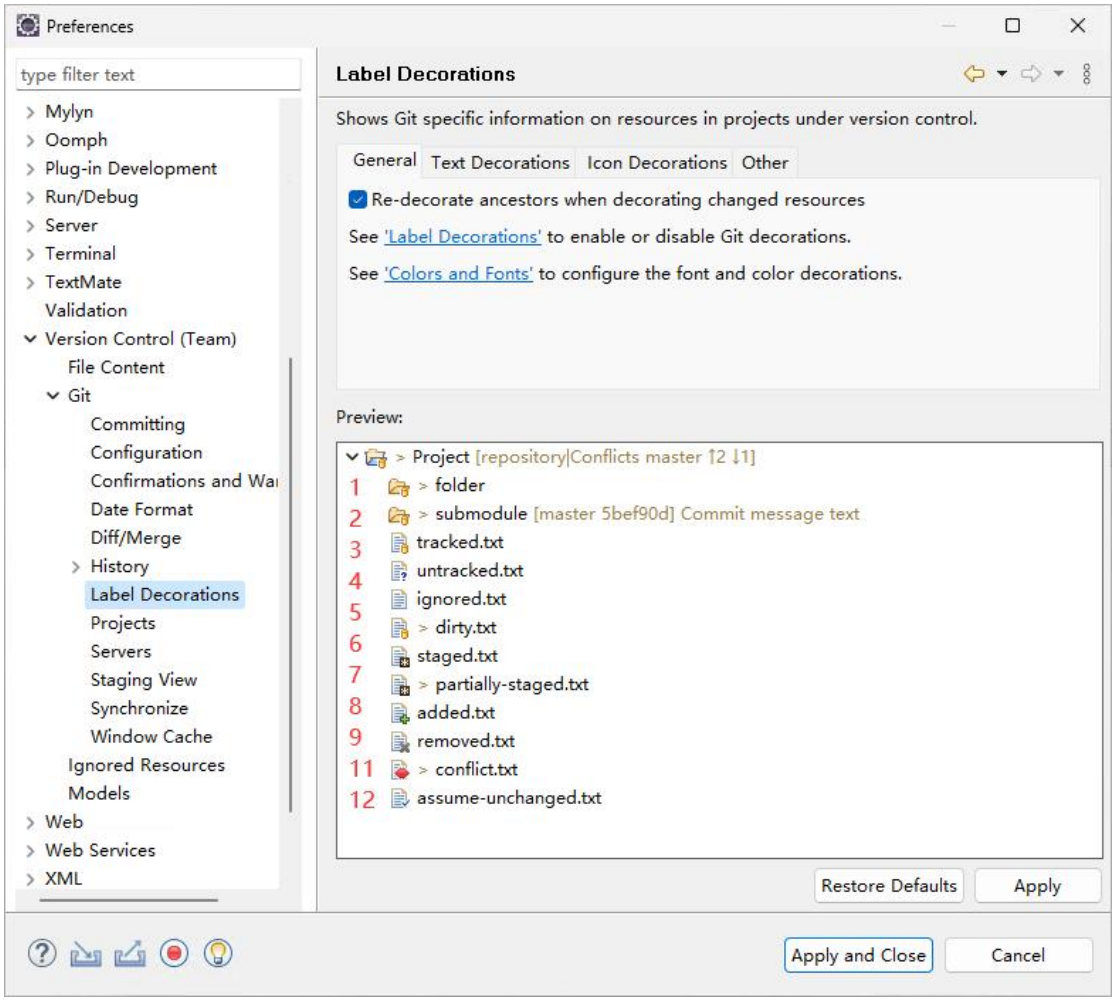


图 1-3-15 git 图标含义

图标含义说明：

1	已纳入版本控制，目录下有未提交的修改
2	已纳入版本控制，目录下有未提交的修改
3	文件已提交到本地库，且工作区文件与本地库文件内容一致
4	新建的文件，未纳入版本控制

5	被版本控制忽略的文件
6	已纳入版本控制，文件有未提交的修改
7	已纳入版本控制，提交过本地库，现在有更改，并提交到了暂存区
8	在 7 的基础上，在工作区又对文件进行了修改
9	未曾提交到过版本库，但已提交到了暂存区，由状态 4 提交暂存区而来
10	当意图把已经被 git 管理的文件取消其管理状态时，该文件会在暂存区处于这种状态
11	在试图合并两个分支的时候，发生冲突，文件会进入这种状态
12	无变化的文件

六、clone 项目到本地

1. clone 项目

在 Git Repositories 视图里面点击 Clone a Git repository 选项，如图 1-3-16 所示。

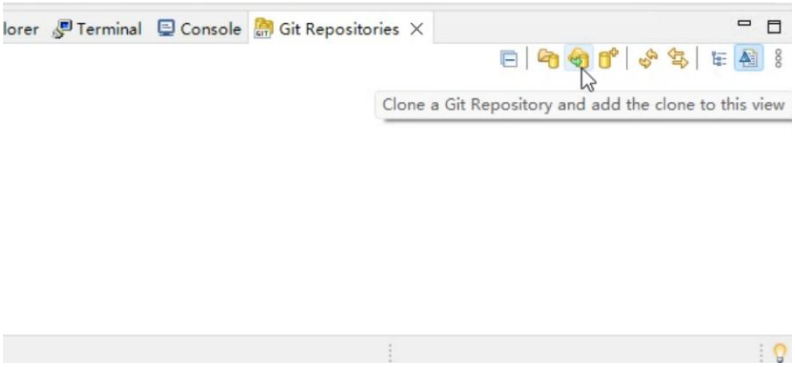


图 1-3-16 克隆项目

选择 Clone URI，下一步, 如图 1-3-17 所示。

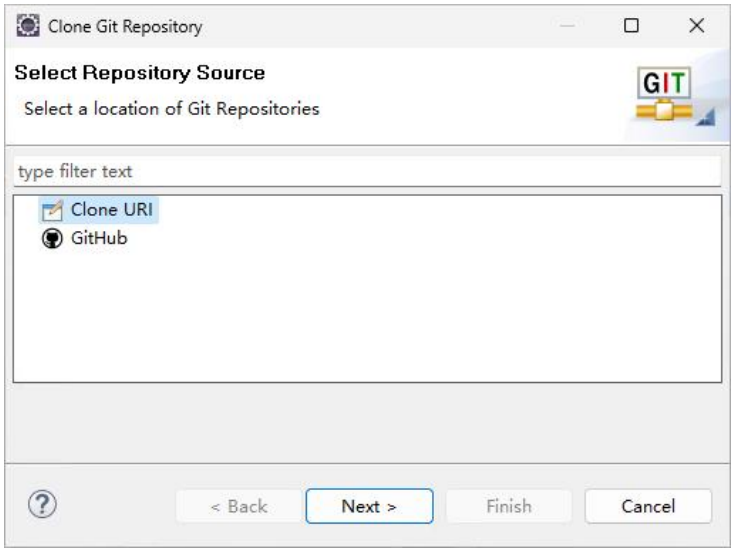


图 1-3-17 选择仓库地址

输入配置信息，点击 Next>, 如图 1-3-18 所示。

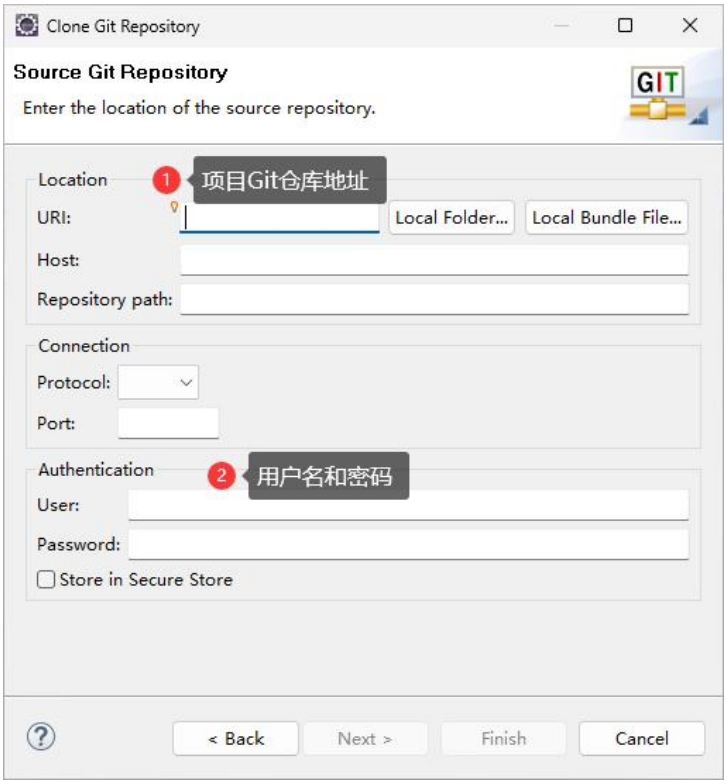


图 1-3-18 输入 git 的地址、用户名和密码

选择分支，我这里选择 master（主干分支），然后点击 Next>, 如图 1-3-19 所示。

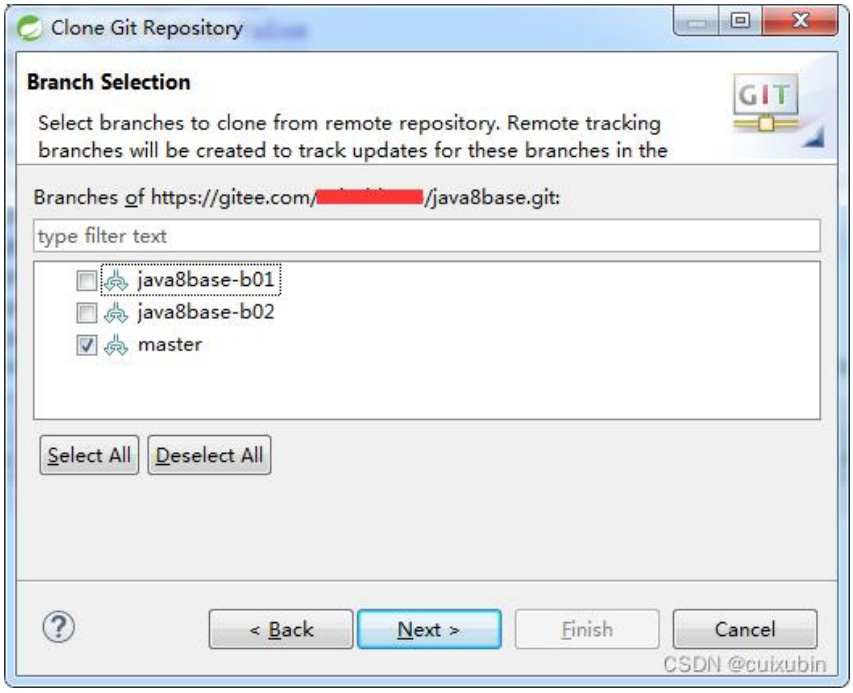


图 1-3-19 选择分支

选择拉取的代码在本地存放的位置，然后下一步，如图 1-3-20 所示。

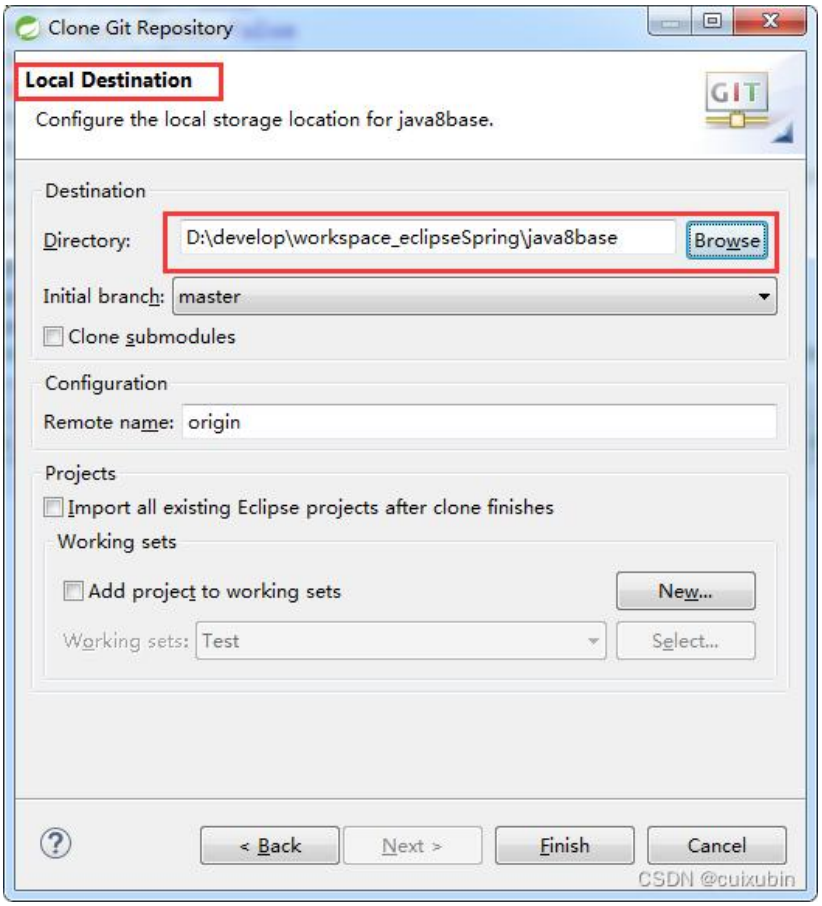


图 1-3-20 选择存放位置

完成之后，可以看到 Eclipse 的 Git Repositories 视图里多了一个 recommend-all 项目，本地存放目录也有了该项目，如图 1-3-21 所示。

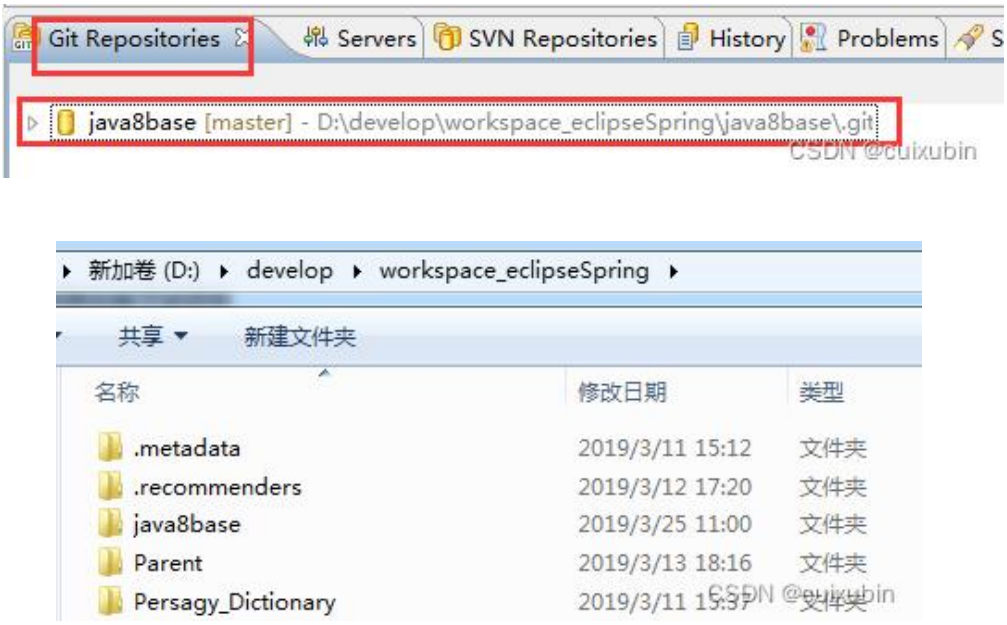


图 1-3-21 克隆成功

2. 导入项目

将该项目导入工作空间：import→Git→Existing local repository，找到克隆到本地的项目目录导入。如图 1-3-22 所示。

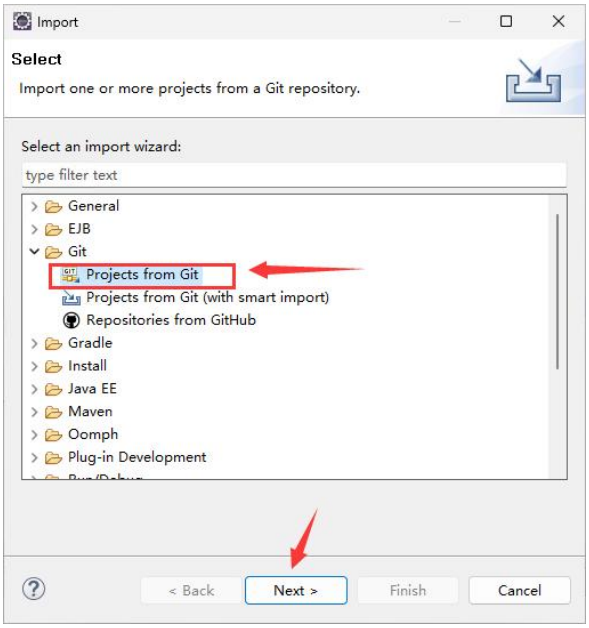


图 1-3-22 选择从 Git 导入项目

选择本地仓库，如图 1-3-23 所示。

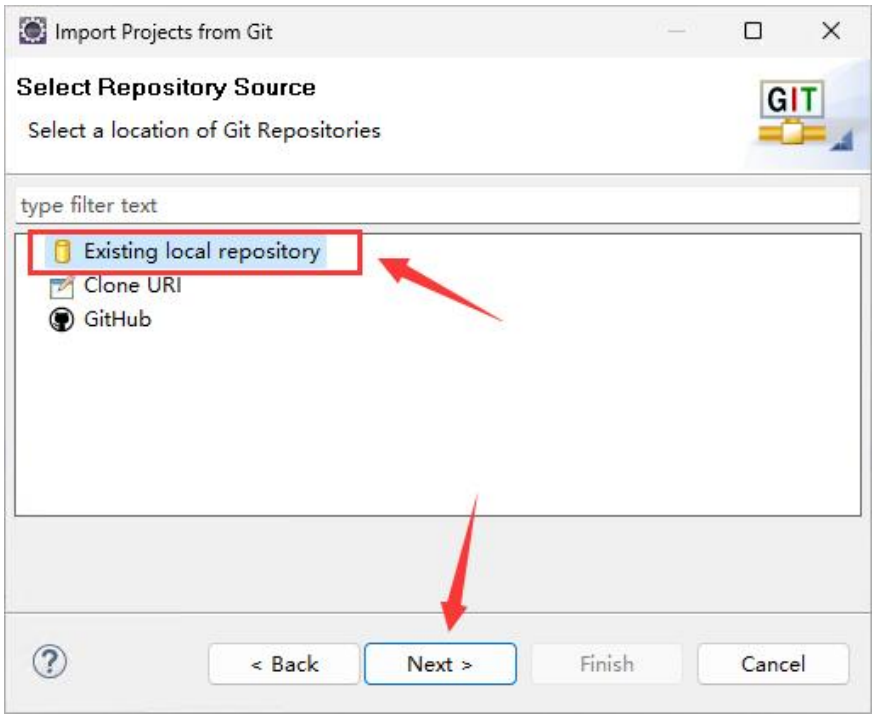


图 1-3-22 选择仓库类型

选择对应的仓库导入，如图 1-3-23 所示。

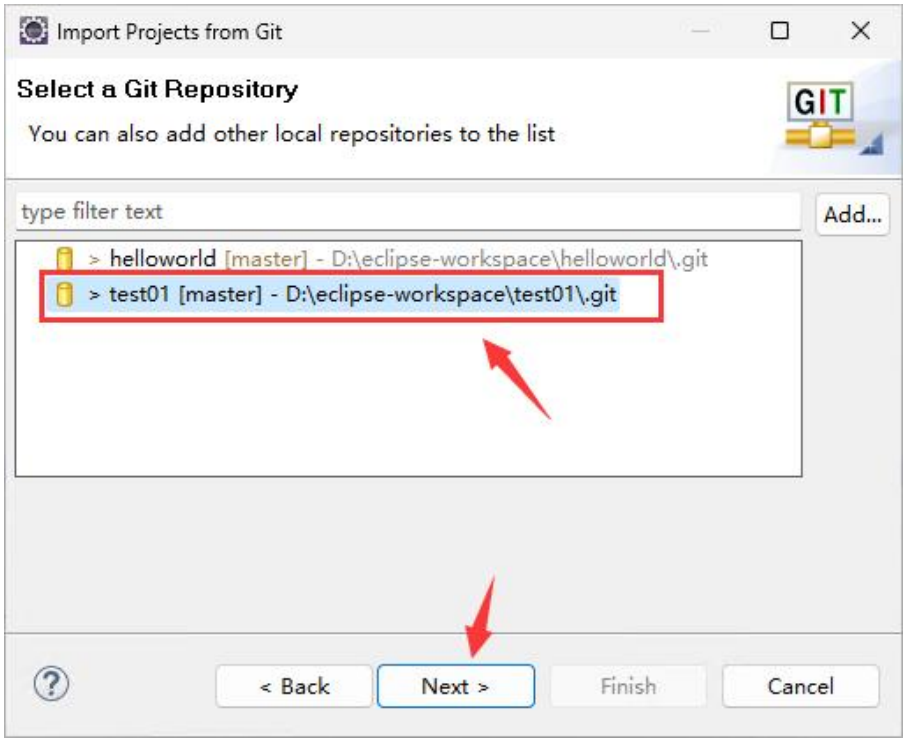


图 1-3-23 选择具体的仓库

选择导入向导，这里现在普通导入，如图 1-3-24 所示。

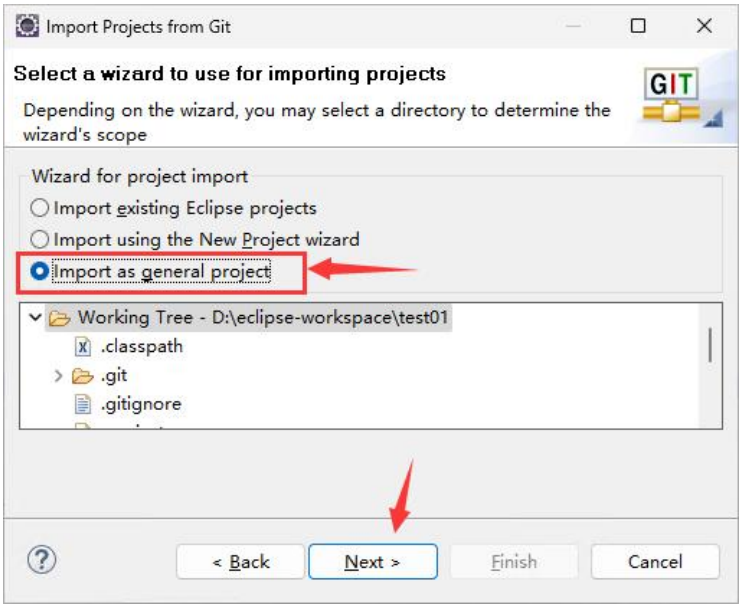


图 1-3-24 选择导入向导

如果该工作空间中目录已经存在，可以选择创建性的项目向导，如图 1-3-25 所示。

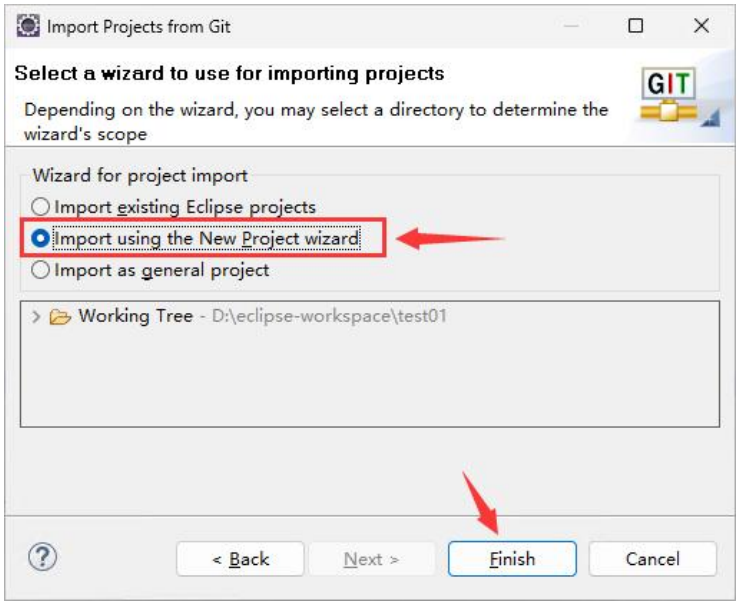


图 1-3-25 选择创建信的项目向导

也可以选择导入已经项目，如图 1-3-26 所示。

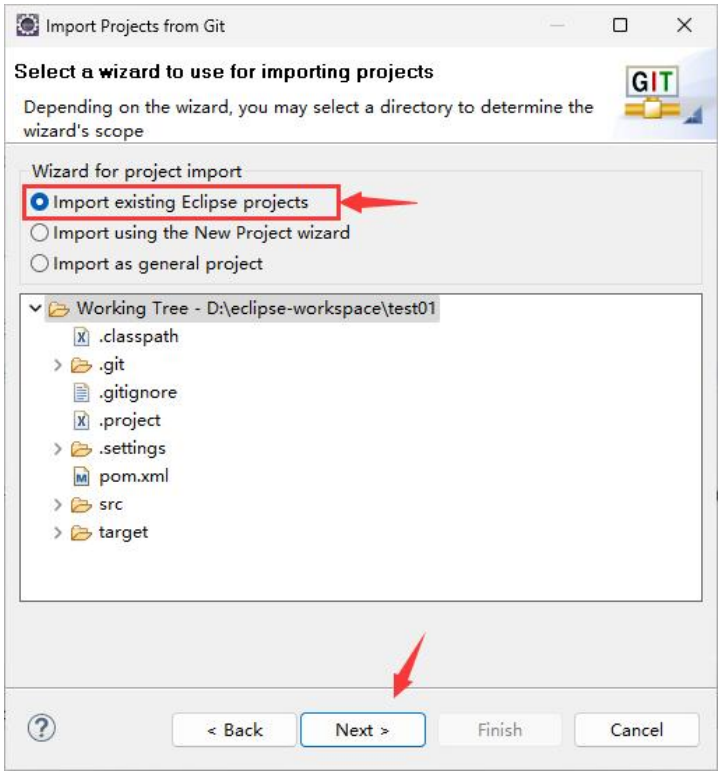


图 1-3-26 选择导入已存在项目向导

七、代码添加到暂存区

在文件上鼠标右键然后选择 Team→Add to Index，如图 1-3-27 所示。

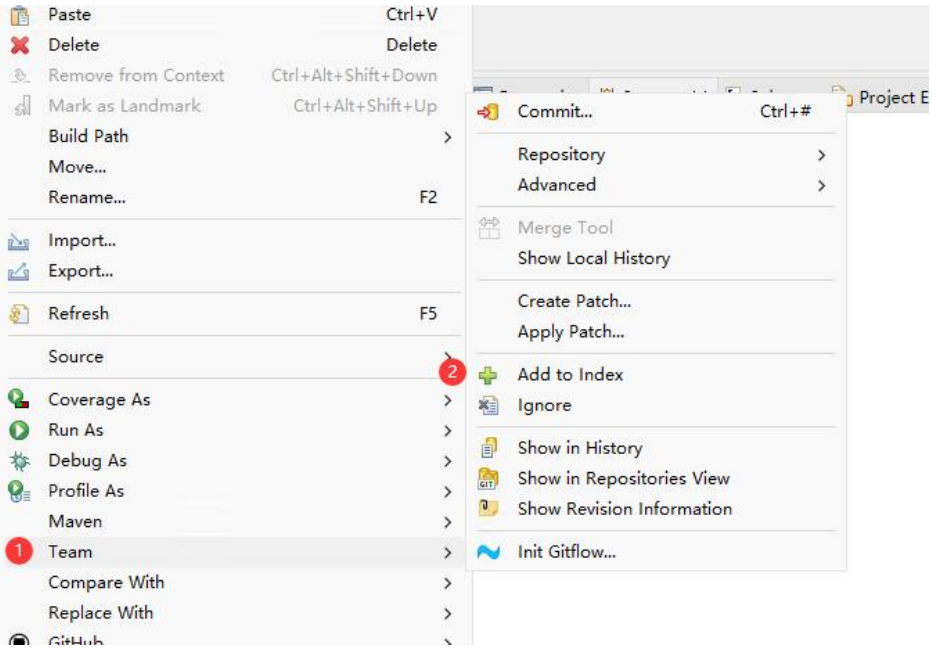


图 1-3-27 添加暂存区

八、代码提交到本地库

在要提交本地库的文件上鼠标右键选择 Team→Commit...，如图 1-3-28 所示。

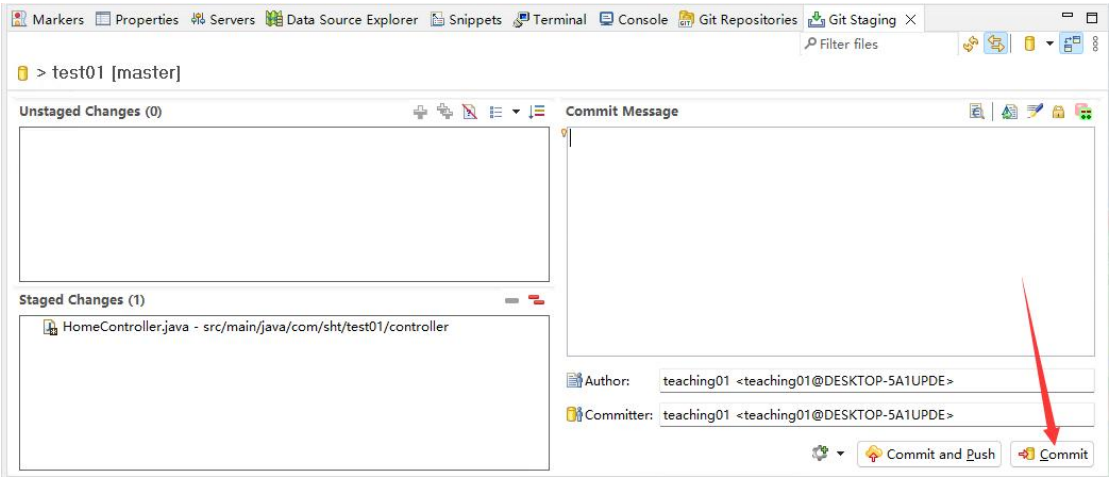


图 1-3-28 添加代码到本地仓库

也可以直接跳过（实际没跳过）加入暂存区的动作直接进行提交。

首先设置一下提交视图，选择菜单 Window→Preferences→Team→Git→Committing，不选择第一项复选框，如图 1-3-29 所示。

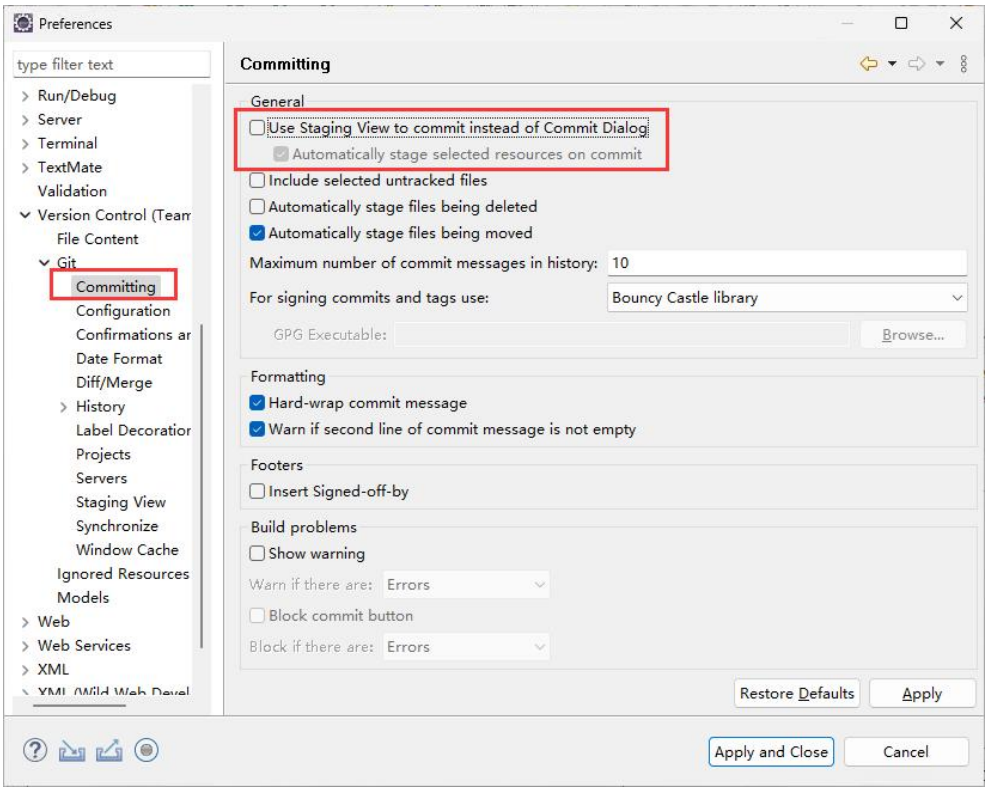


图 1-3-29 提交跳过设置

右键项目 Team→Commit…

在 Git Staging 窗口选择要提交的文件，填写注释，点击 Commit 按钮提交到本地库，如图 1-3-29 所示。

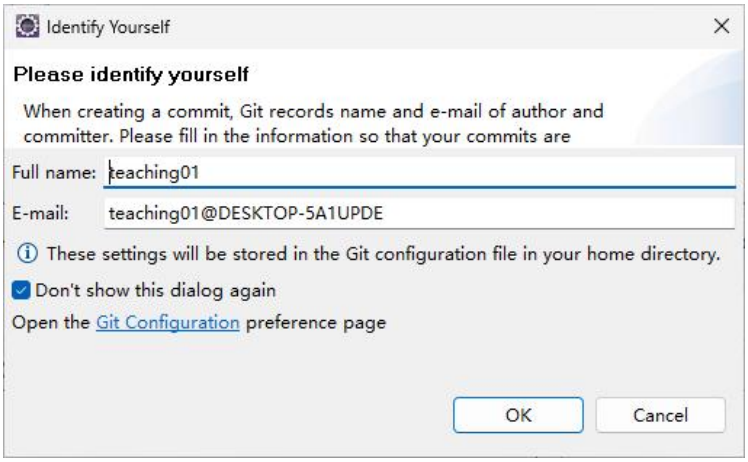


图 1-3-29 提交确认

首次提交后，就会自动生成 master 分支，此刻项目文件上没有了问号，而是有了一个仓库图标, 如图 1-3-30 所示。

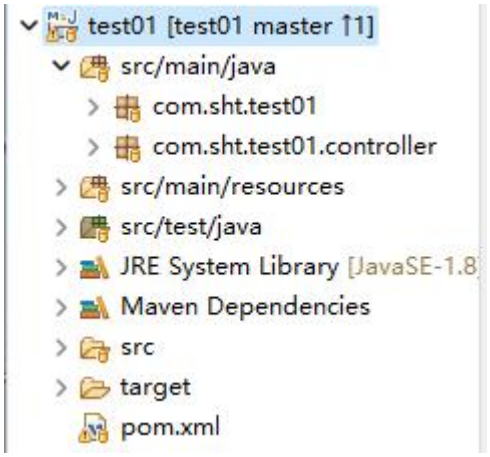


图 1-3-30 仓库图标

九、代码提交到远程 Git 仓库

首先在远程 Git 服务（比如 GitHub 或码云）中创建一个仓库，用于存放项目代码。以 GitHub 为例，登陆 Git，创建一个仓库，如图 1-3-31 所示。

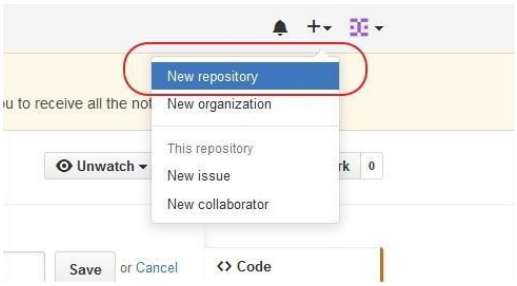


图 1-3-31 创建远程仓库

仓库名自定，创建完成后，点击进入可看到仓库信息，其中最重要的就是地址，如图 1-3-32 所示。

图 1-3-32 仓库信息

Eclipse 中项目右键选择 Team→Remote→Push…，弹出窗口如图 1-3-33 所示。

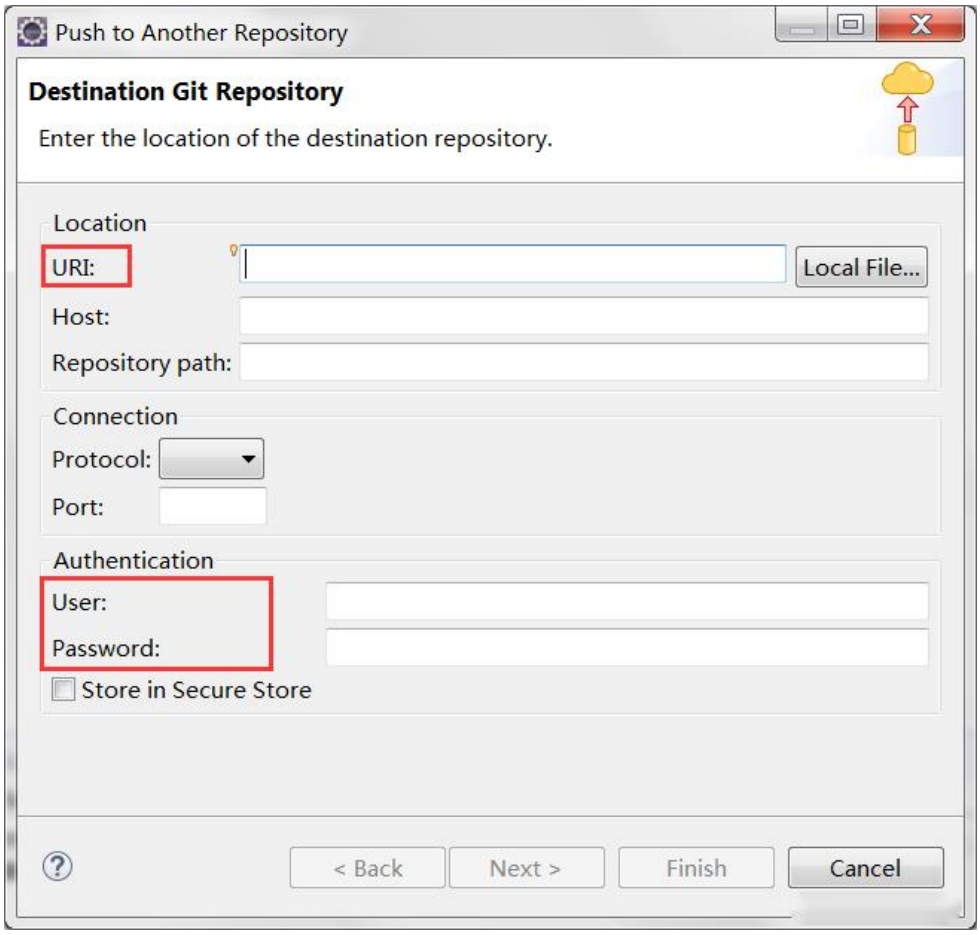


图 1-3-33 Eclipse 项目选择

URI 填写 Git 仓库地址；Protocol 为遵循的协议（Git 支持 Https 和 ssh）；User 和 Password 分别为 GitHub 登录账号和密码，如图 1-3-34 所示。

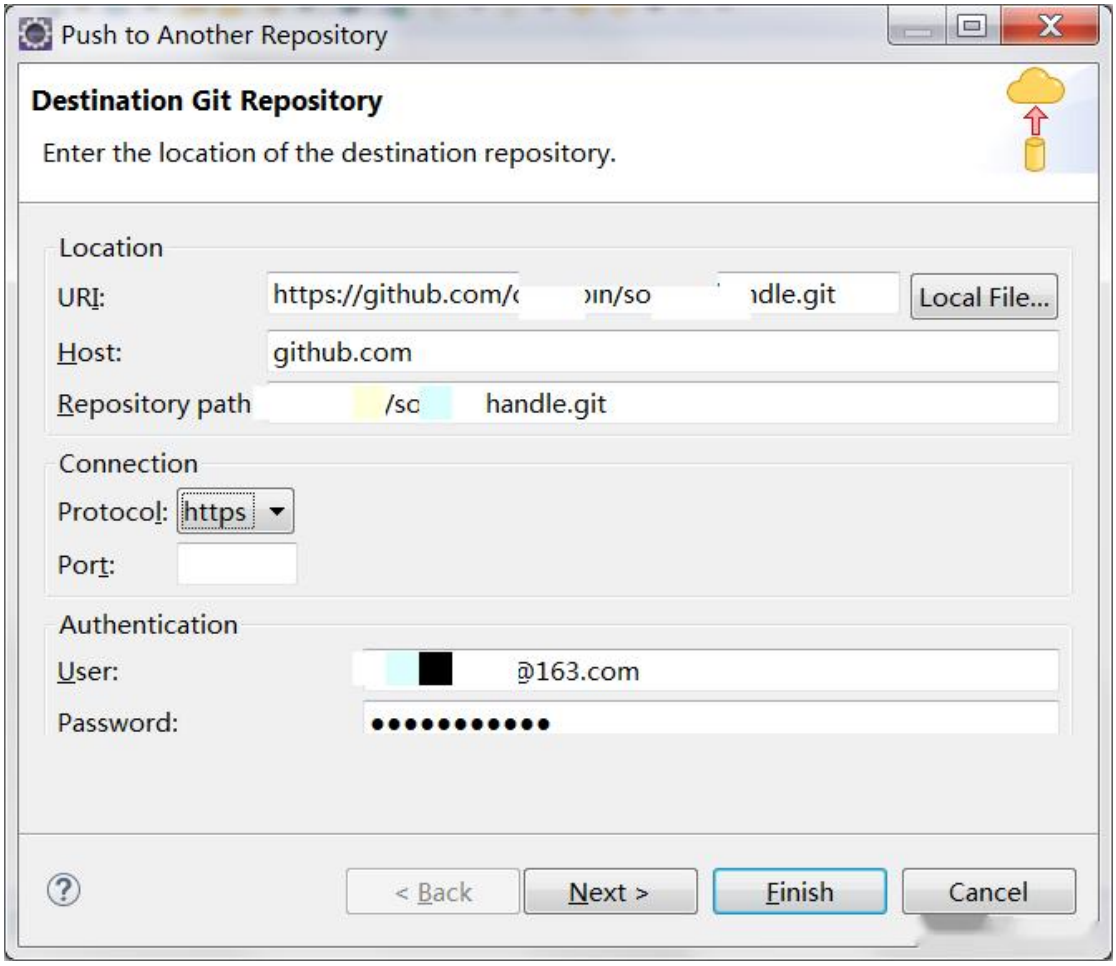


图 1-3-34 git 信息填写

点击 Next>，选择 Source ref 为 master，然后点击“Add Spec”按钮。如图 1-3-35 所示。

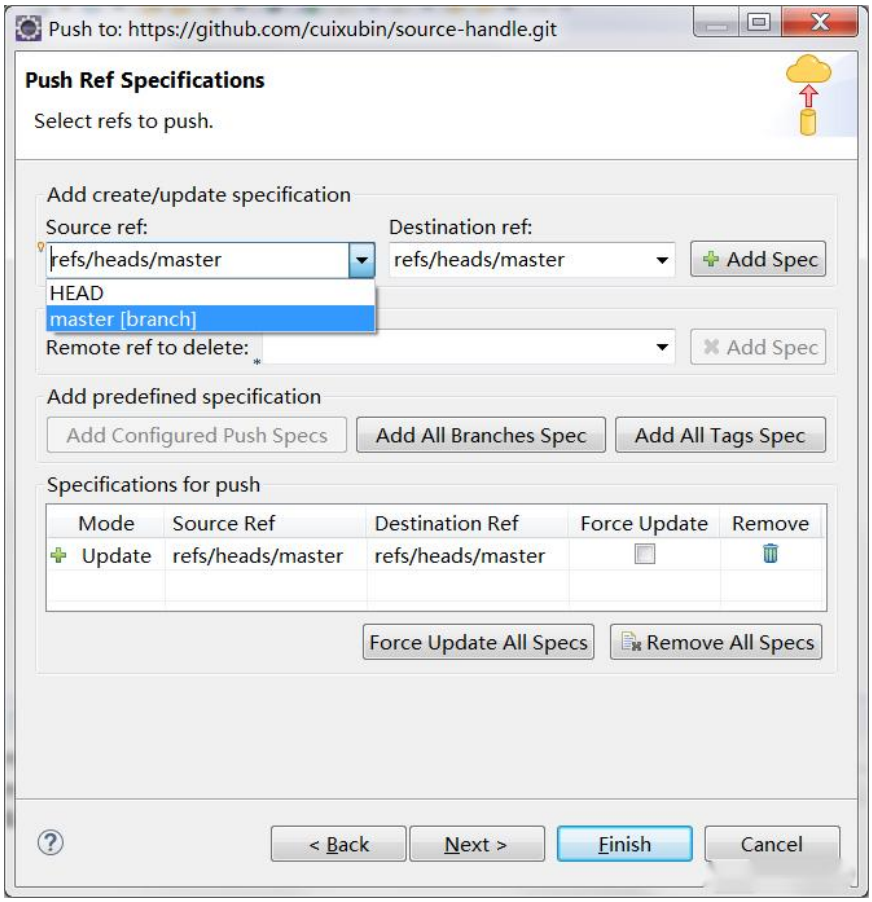


图 1-3-35 git 信息确认

401 Authorization Required 报错处理

在进行 Git commit 的时候，报出了 401 未授权的错误，可能为 eclipse 的 bug，可尝试按如下步骤操作解决：

a. Eclipse 选择 Help→ Install New Software...；

b. 在 Work with:中填上 http://download.eclipse.org/mpc/releases/1.5.1a，回车；

c. 勾选″EPP Marketplace Client″，Next>直至 Finish；

d. 重启后，再进行提交，问题不再出现，可以成功提交。

十、更新远程 Git 代码到本地

选中项目右击：选择 team→fetch from Upstream。

如果有版本改动的话，项目是有箭头的，如果没有什么版本改动的话，项目是没有箭头。如图 1-3-36 所示。

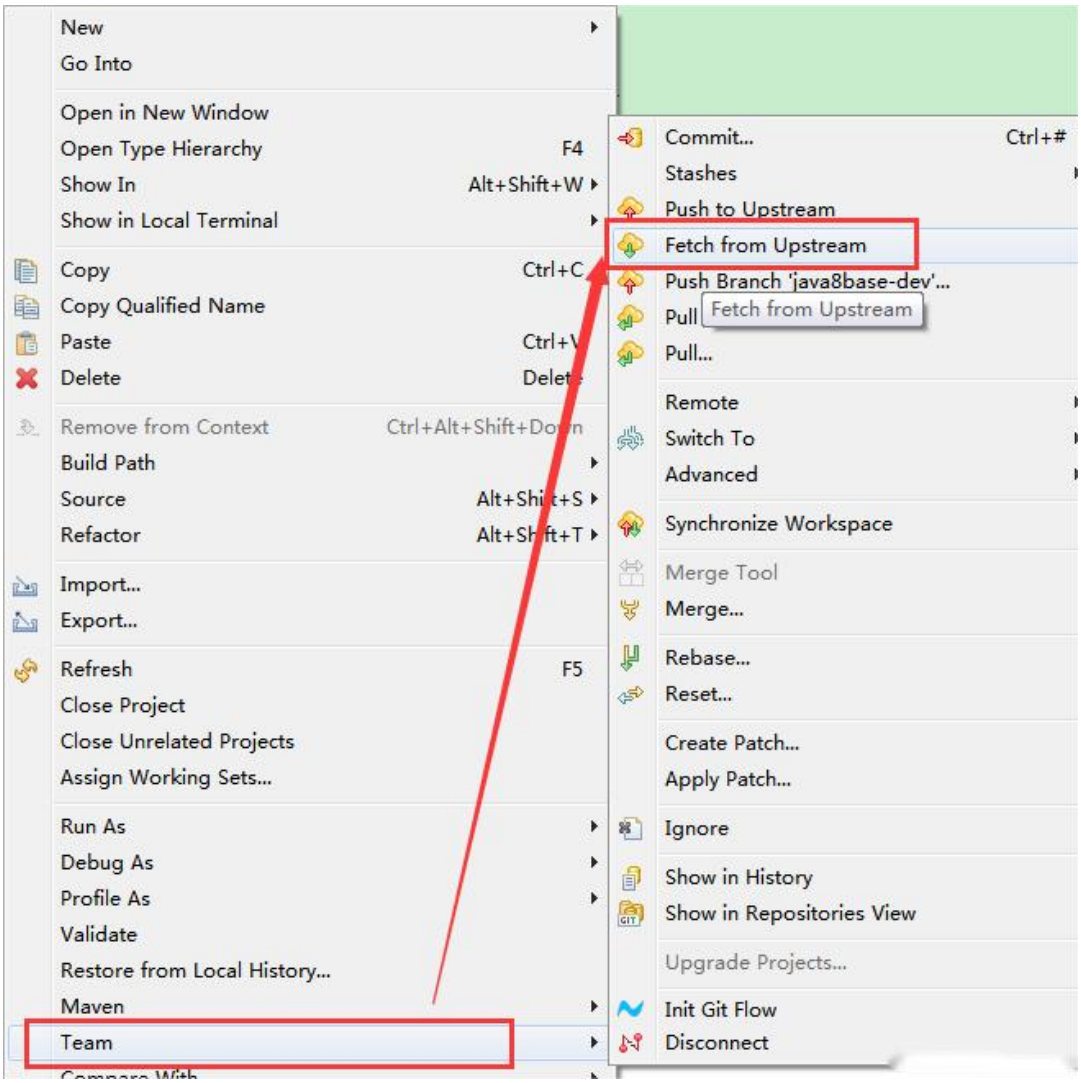


图 1-3-36 远程 git

如果箭头向下表明你落后了远程仓库几个版本。

通过项目右击：选择 team→Synchronize WorkSpace 可以查看当前工作空间内容的差异情况。

处理方式：选中项目右击：选择 team→pull，表示从远程仓库“同步”代码，pull 完成后，箭头向下图标消失了。

如果箭头向上则表明你比远程仓库的版本提前了几个版本，就是说你有提交到本地仓库，但是没有 push 到远程仓库。

处理方式：选中项目右击：选择team→Push Branch…，表示是向远端仓库提交代码。

十一、分支创建和切换

项目右键→Team→Switch To，如图 1-3-37 所示。

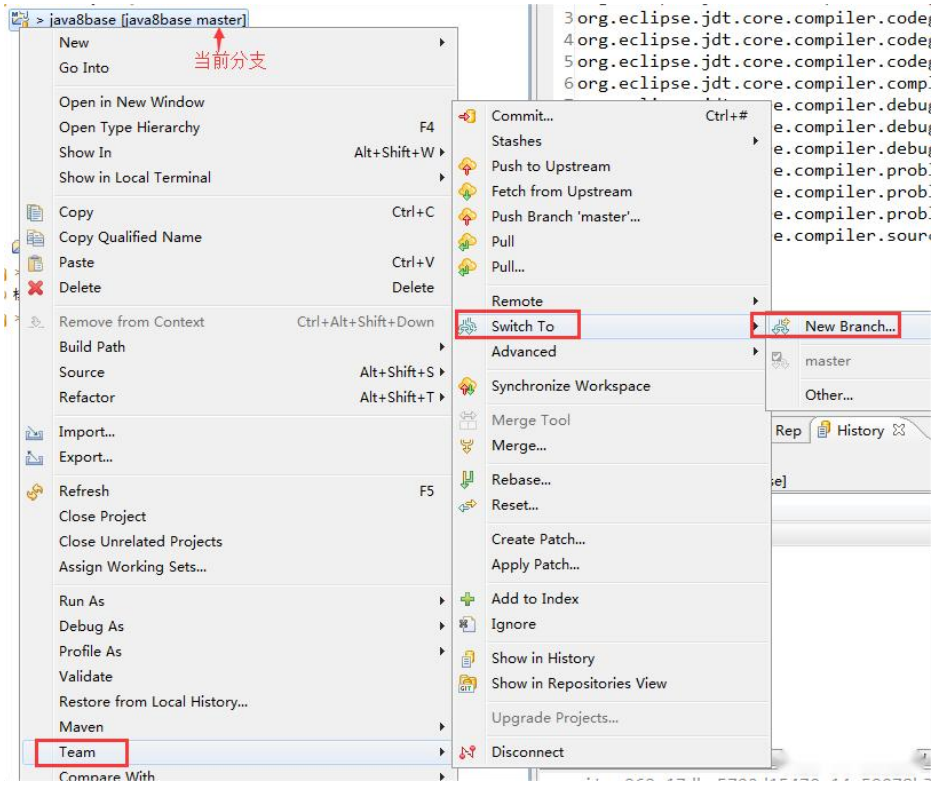


图 1-3-37 项目切换

选择New Branch...，如图1-3-38所示。

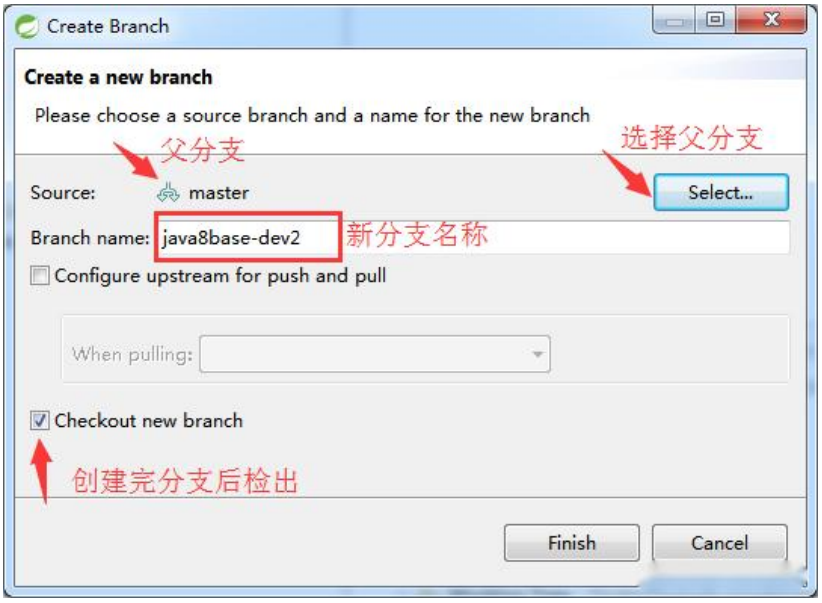


图 1-3-38 创建分支

当前分支切换到了新创建的分支，如图 1-3-39 所示。

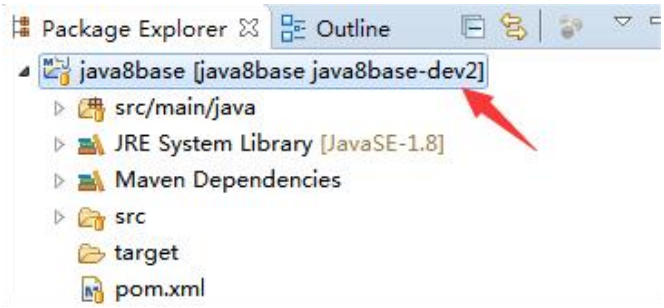


图 1-3-39 切换新分支

项目右键→Team→Switch To，选择 other...， 如图 1-3-40 所示。

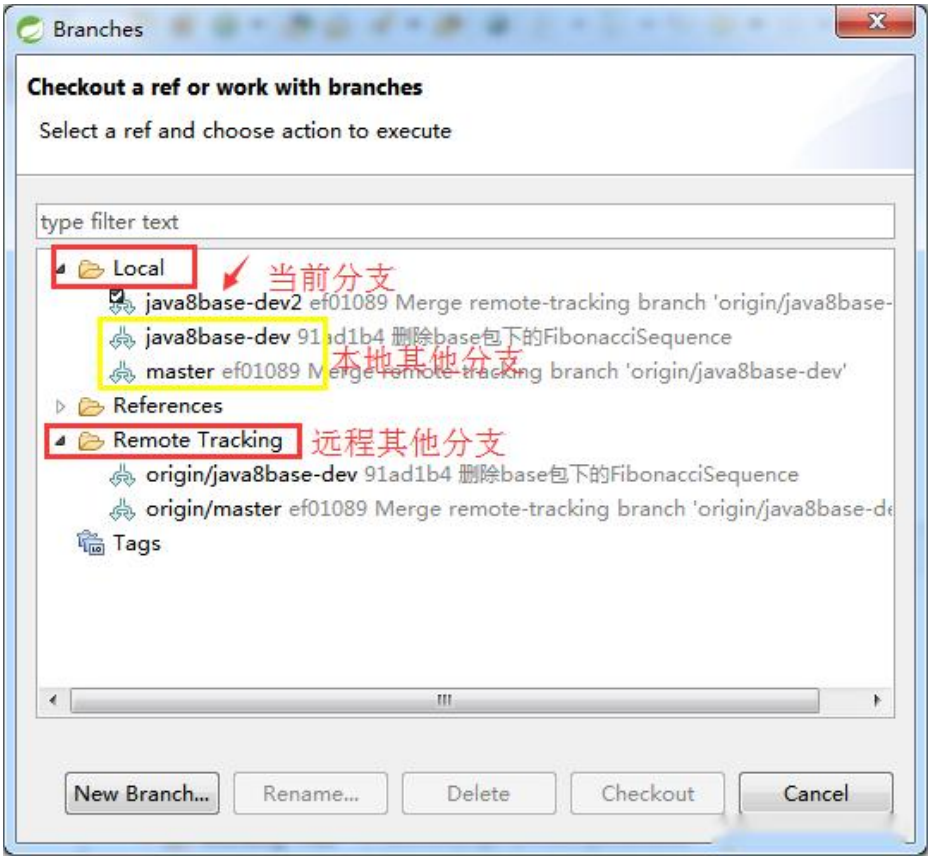


图 1-3-40 远程切换分支

此时 Remote Tracking 有 master 和 java8base-dev 分支，当遇到 Remote Tracking 里面没有自己或者团队成员新创建的分支时处理方式：选中项目右击：选择 team→fetch from Upstream 就能从远程仓库获取最新版本到本地（包括代码、分支）。

Fetch from Upstream 后，能看到 Remote Tracking 里面有很多分支了，选择自己想切换的分支继续确定。如图 1-3-41 所示。

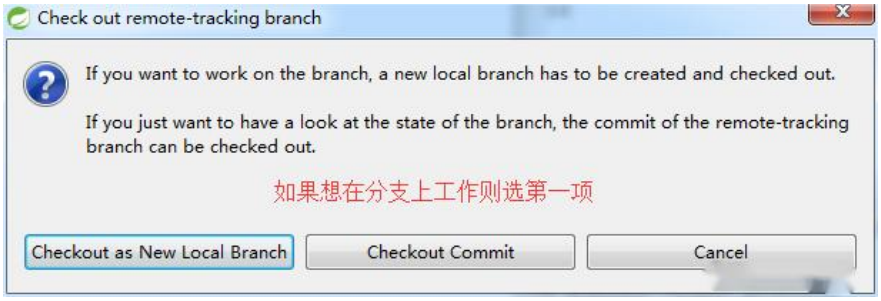


图 1-3-41 远程切换分支

十二、分支合并

在 Eclipse 工作空间的项目下右键 team 选择 merge 然后选择你要合并的分支；点击 ok。如图 1-3-42 所示。

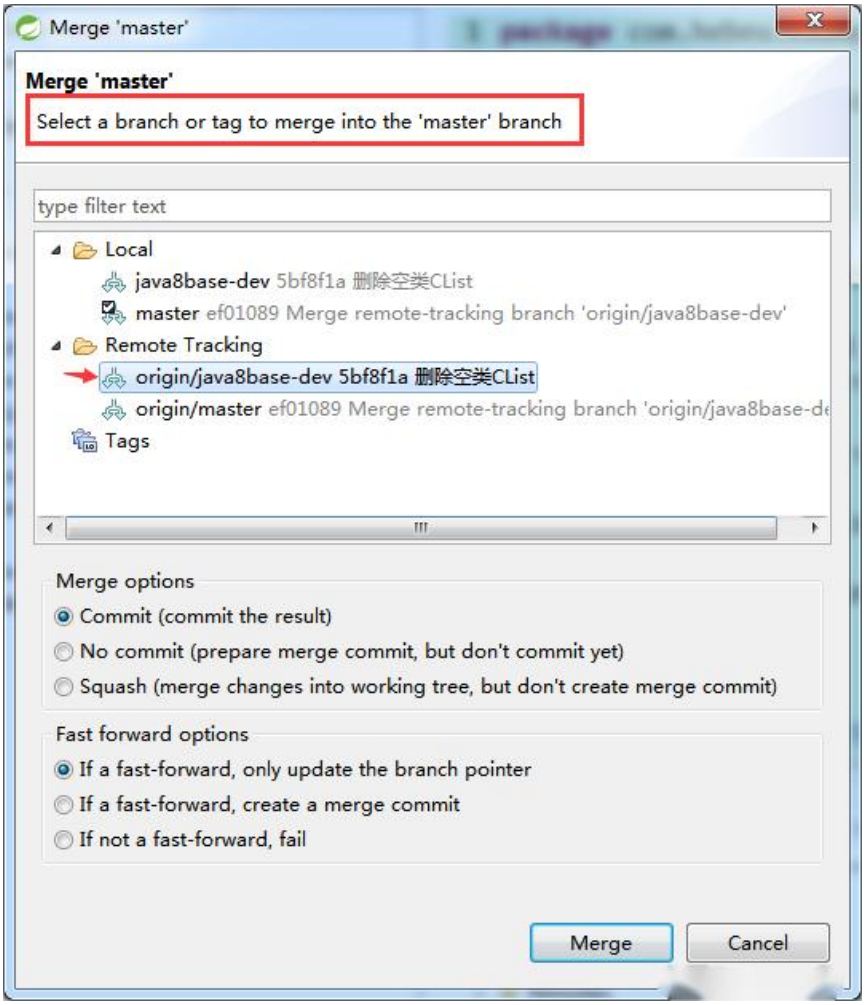


图 1-3-42 分支合并

十三、提交时冲突的解决

Git 远程提交数据时若有冲突，则远程服务会拒绝提交，如图 1-3-43 所示。

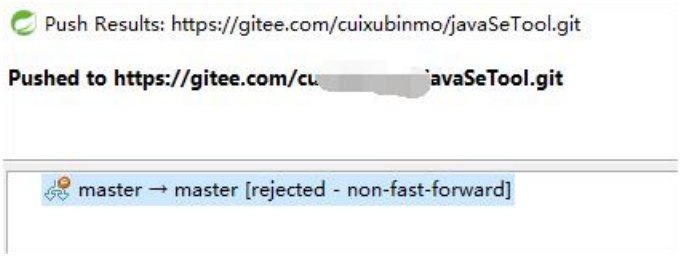


图 1-3-43 远程服务拒绝

此时需要先拉取远程库的代码，解决完冲突，再提交。

拉取远程库代码后可以看到冲突文件的信息，如图 1-3-44 所示。

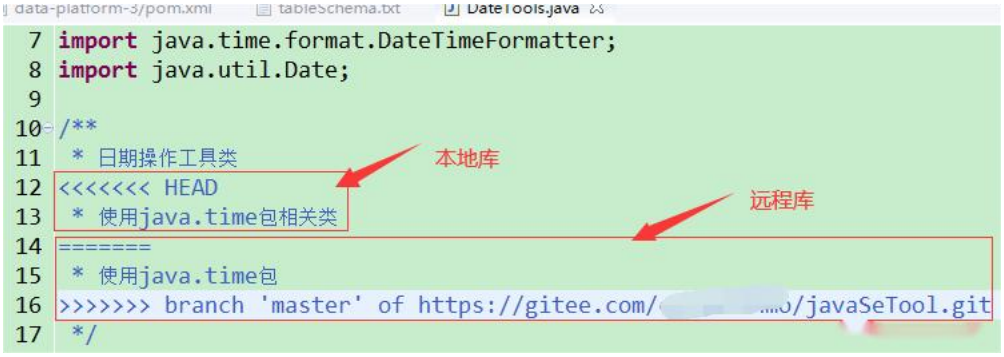


图 1-3-44 远程冲突文件

也可以在冲突文件上右键选择，Team→Merge Tool，以图形化对比的方式查看冲突信息，如图 1-3-45 所示，1-3-46 所示。

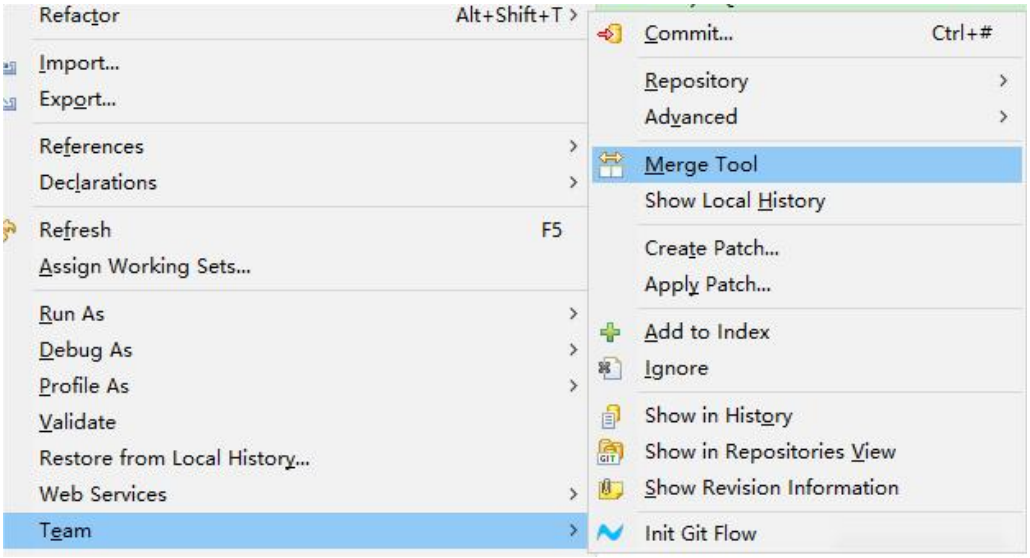


图 1-3-45 冲突文件对比

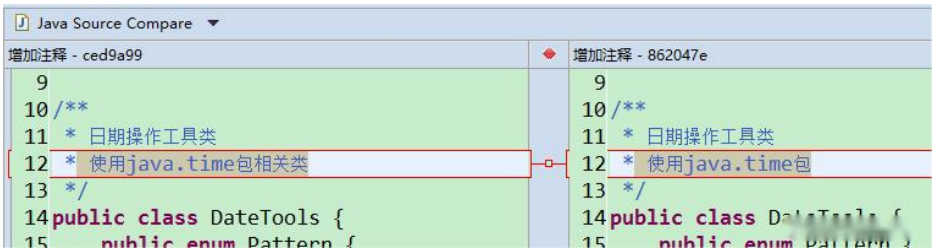


图 1-3-46 对比视图

左侧的为本地库代码，可以修改，右侧的为远程库代码不能修改（可以进行复制操作）。

修改完成后打开 Git Staging 视图，将冲突文件从 Unstaged Changes 窗口拉取到 Staged Changes 窗口里，再选择 Commit and Push 按钮进行提交即可，如图 1-3-47 所示。

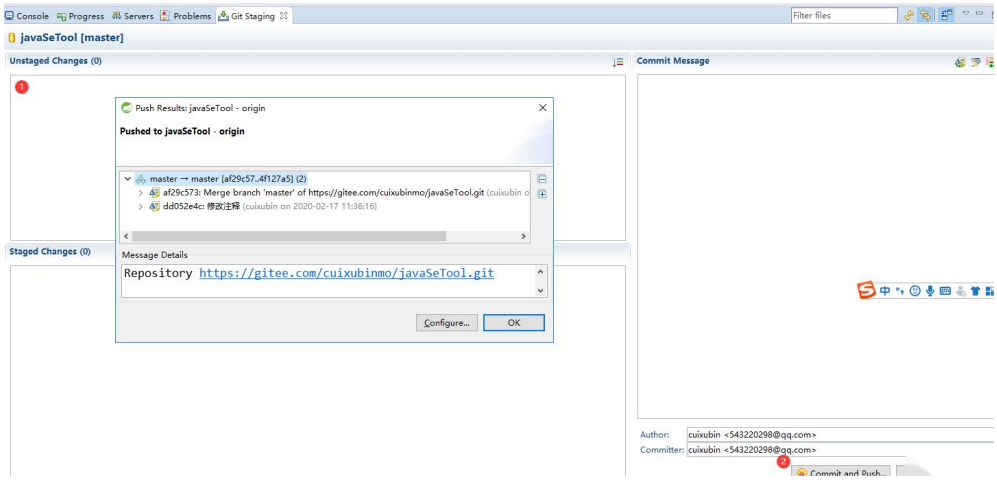


图 1-3-47 修改冲突文件

任务实施

Step1 Git 插件安装和配置

Step2 初始化本地库及本地库 Git 签名

Step3 Clone 项目到本地

Step4 代码提交到本地库以及远程 Git 仓库

Step5 更新远程代码到本地

Step6 分支创建和切换

Step7 分支合并

Step8 提交时冲突的解决

任务 2 生物测序实验管理系统分析与设计

项目描述

接收到开发生物测序使用管理系统的任务后，我们需要进行系统分析与设计。要成为一名合格的软件开发者，系统分析和设计也是我们成长的必经之路，无论开发什么样的应用程序都必须遵循系统分析和设计的基本原则。系统分析是一个复杂的过程，它需要考虑到每一个方面，包括其功能、性能、安全等。而系统设计的目的是为了实现这些分析结果，即为了满足系统的需求而设计出符合标准的系统。

项目目标

1. 能对生物测序实验管理系统进行系统分析。
2. 能对生物测序实验管理系统进行系统设计。

活动一 生物测序实验管理系统功能分析

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。在项目开发之前我们先要进行功能分析。做好了需求分析我们才能开发出符合要求的系统，让我们一起进行系统分析吧！

任务目标

1. 能完成系统功能分析。
2. 能完成系统功能结构分析。
3. 能完成系统业务流程图分析。

任务分析

1. 生物测序实验管理系统有哪些功能模块？
2. 用户管理需要实现哪些功能？
3. 样本管理需要实现哪些功能？
4. 任务管理需要实现哪些功能？
5. 系统的业务流程是怎样的？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、系统需求分析

需求分析是以正确、可行、必要等标准对生物测序实验管理系统做一个完整的功能需求说明，分析要求每个需求的功能必须描述清楚，确保在当前的开发能力和系统环境下可以实现。并且每个需求的功能是否必须交付、是否可以推迟实现、是否可以在削减开支情况发生时进行删减也必须描述清楚。

生物测序实验管理系统运行各组成要素提供综合管理功能，主要有：用户登录管理、样本导入、样本导出、样本数据管理、创建分析任务、任务搜索等功能。

1. 用户登录管理

生物测序实验管理系统包括两类用户，普通用户和系统管理员。普通用户可以使用样本导入、样本导出、样本数据管理、创建分析任务、任务搜索等功能，系统管理员主要对系统信息进行维护，对管理用户进行维护，可以添加、删除管理用户。用户管理模块主要提供以下功能：

a. 用户登录，系统需要登录才能使用，用户系统里的每个操作都会判断是否有登录，如果未登录，提示未登录并且跳转到登录界面。用户在登录时，如果会员名、密码错误，系统会提示错误。超级管理员创建普通用户，该用户才可以登录系统，进行生物测序实验管理的操作。

b. 用户信息创建，系统管理员在系统后台页面上可以创建用户信息，包括用户的登录账号密码信息。

c. 用户信息的删除，系统管理员在系统后台页面上可以查看用户的信息，并能将会员注册的信息删除。

d. 用户信息的修改，用户在登录后可以修改自己基本信息，在修改页面显示该用户目前的信息，提供信息的修改输入。

e. 用户信息的查询，系统管理员在系统后台页面上可以查看用户的信息。

f. 系统管理员操作，系统管理员专门用来维护管理用户的信息，具体包括管理用户基本信息增加、删除、修改和查询等操作。

（1）样本管理

生物测序实验管理系统重要的一个功能样本数据管理。用户可以使用样本导入、样本导出、样本数据管理等功能对数据进行管理，用户对样本数据进行分析，主要功能描述如下:

①样本导入，用户下载样本导入模板文件，然后完成模板文件数据的录入，完成数据录入后通过导入功能把输入导入到系统中。

②样本导出，导出功能是导入的相反的过程，系统从数据库获取到数据写入文件后，向前端输入下载到用户电脑上的。

③样本数据列表管理，主要包括样本数据的修改、删除、查询等等功能。

（2）任务管理

生物测序实验管理系统任务管理是对样本数据创建的分析任务。用户创建分析任务功能对样本进行数据分析。主要有以下功能：

①创建分析任务，用户对样本创建分析任务，每个任务分析只对应一个样本，并且产生一条任务数据。注意本教程的创建任务只是在系统里产生一天任务数的操作，没有真正意义上的生物信息分析。

②任务数据管理，主要包括任务数据的修改、删除、查询等功能。

（3）多条件模糊搜索

生物测序实验管理系统涉及数据列表的都需要能进行搜索，目前需要多样本管理和任务管理的数据进行搜索，搜索需要能做到以下两点功能:

①多条件搜索，即根据实际需要能多个条件输入对数据库进行查询。

②模糊搜索，即能对某些字段能进行模糊的查询。

二、系统目标

根据需求分析以及用户对系统的使用体验，生物测序实验管理系统需要达到以下目标：

a. 界面设计友好、美观。

b. 具有易维护性和易操作性。

c. 用户登录后才能使用系统内的功能。

d. 超级管理员能对用户进行管理。

e. 实验室测序得到的样本数据能导入系统内。

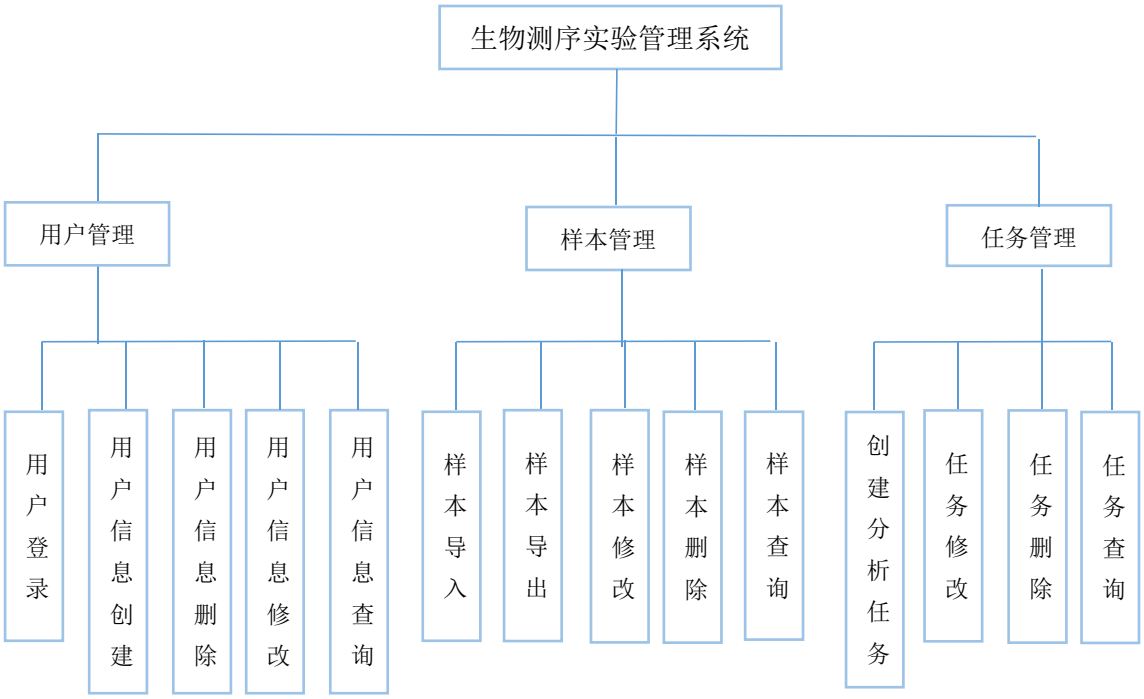
f. 用户能把样本数据从系统中导出。

g. 用户能到系统内的样本数据进行管理。

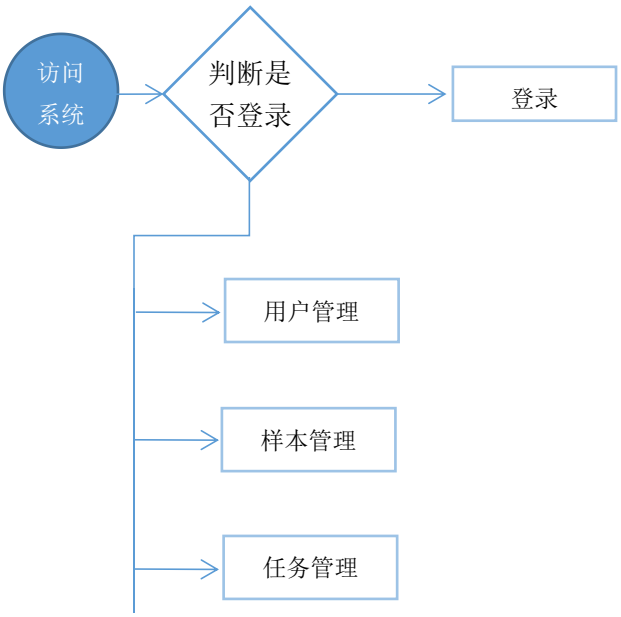
h. 用户能创建分析任务。

I. 用户能对已有的分析任务进行搜索。

三、系统功能结构



四、系统业务流程图



五、系统预览

任务实施

Step1 系统需求分析

Step2 系统目标

Step3 系统功能结构

Step4 使用系统业务流程图

Step5 系统预览

活动二 生物测序实验管理系统架构设计与搭建

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。在项目开发之前我们先要进行功能分析，做好了需求分析我们才能开发符合要求的系统，让我们一起进行系统分析吧！

任务目标

1. 能完成系统架构设计。
2. 能完成业务实体设计。
3. 能完成业务逻辑设计。
4. 能说明开发环境。

任务分析

1. 生物测序实验管理系统使用什么系统架构？
2. 生物测序实验管理系统中涉及哪些业务实体？
3. 生物测序实验管理使用什么业务逻辑实现？
4. 生物测序实验管理系统开发需要什么环境？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、系统架构设计

1. MVC 设计模式

模型－视图－控制器（MVC）是 Xerox PARC 在二十世纪八十年代为编程语言 Smalltalk-80 发明的一种软件设计模式，该设计模式在 Java 项目中被大量使用，甚至被很多前端框架应用。

M 即 model 模型，是指模型表示业务规则，在 Java EE 项目 model 被命名为 Service。

V 即 view 视图，是指用户看到并与之交互的界面。

C 即 controller 控制器，是指控制器接收用户的输入并调用模型和视图去完成用户的需求。

控制器本身不处理逻辑。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

在 Java 早期项目中，JSP+Servlet+JavaBean 这种模式即就是典型的 MVC 模式：

1）JSP 负责人机交互（view）；

2）Servlet 负责流程控制（controller）；

3）JavaBean 负责业务逻辑（model）。

编写 Servlet 比较麻烦，随后开始出现 MVC 框架，比如 Struts1，Struts2，Spring MVC 等。

MVC 模式对架构设计影响巨大，不但在 Java EE 项目中大量使用，比如 Swing 中也有对应实现。此外，前端目前流行的 MVVM 双向绑定也可以看作是一种 MVC 模式。

通常我们将控制器单独分包作为一层，命名为 controller。此外该包的控制器类命名也建议加上 Controller。

2. DAO 设计模式

DAO (Data Access Object) 数据访问对象是一个面向对象的数据库接口，以下是标准开发的架构：

客户层：目前使用 B/S 开发架构居多，客户可以通过浏览器访问；

显示层：使用 Vue 框架、React 框架、JSP/Servlet 等进行页面展示；

业务层：负责将 DAO 层的操作进行组合，形成一个完整的业务逻辑；

数据层：提供原子性操作，比如增删查改。

DAO 的设计流程包括六个部分，如下：

a. DataBaseConnection

设计一个专门负责打开连接数据库和关闭数据库操作的类。

命名规则：xxx.dbc.DataBaseConnection.

b. VO

设计 VO(值对象)，其主要由属性，setter 和 getter 组成与数据库中的字段进行对应。

命名规则：xxx.vo.ttt，其中 ttt 要和数据库中的表的名字一致。

c. DAO

定义一系列原子性操作，比如增删查改，和实现业务的接口。

命名规则：xxx.dao.I.xxx.DAO.

d.. Impl

设计 DAO 接口真正的实现类，完成具体的操作，但是不负责数据库的开关。

命名规则：xxx.dao.imp.xxxDAOImpl.

e. Proxy

Proxy 代理类的实现，主要将以上四个部分组合起来，完成整个操作过程。

命名规则：xxx.dao.Proxy.xxx.Proxy.

f. Factory

Factory 类主要用于获得 DAO 类的实例对象。

命名规则：xxx.factory.DAOFactory.

在项目中开发人员使用 DAO 设计模式把底层的数据访问逻辑和高层的业务逻辑分开，这样可以使业务逻辑和数据库操作都保持高内聚度，提高可重用性和扩展性。

3. Spring Boot 微服务架构

在当今的企业级应用开发中，微服务架构成为了一个备受瞩目的技术选型。而 Spring Boot 作为一个快速开发的框架，可以帮助开发者更高效地构建微服务架构的应用。

微服务架构：解耦、可扩展、易维护，微服务架构将应用拆分成小的、独立的服务，每个服务专注于一个特定的业务功能。这种架构解耦了不同业务模块，使得开发、测试、部署和维护更加灵活。同时，微服务架构也支持水平扩展，使得应对高并发和大规模访问变得更容易。

因此本教程将使用 Spring Boot 架构来开发生物测序实验管理系统。

Spring Boot 框架一般分为 View 层、Controller 层、Service 层、Mapper 层、Pojo（entity）层，如图 2-2-1 所示。

A. View 层：视图层，根据接到的数据展示页面给用户

B. Controller 层：响应用户需求，决定用什么视图，需要准备什么数据来显示。Controller 层负责前后端交互，接收前端请求，调用 Service 层，接收 Service 层返回的数据，最后返回具体的数据和页面到客户端

C. Service 层：Service 层也可以分为三个方面：

a. 接口：用来声明方法；

b. 继承实现接口；

c. Impl：接口的实现（将 mapper 和 service 进行整合的文件）。

Service 层存放业务逻辑处理，有一些关于数据库处理的操作，但是不是直接和数据库打交道，有接口，也有接口的实现方法，在 impl 实现接口类中需要导入 Mapper 类，Mapper 层是直接与数据库进行操作的。

D. Mapper 层：也可以称为 DAO 层，是数据库访问的接口，只有方法名，具体实现在 mapper.xml 文件中，对数据库进行数据持久化操作。

E. src/main/resource 文件夹中的 mapper.xml 文件，里面存储的是真正的数据库 SQL 语句。

F. Pojo（entity）层：存放实体类，与数据库中的属性基本保持一致，一般包括 getter、setter 方法。

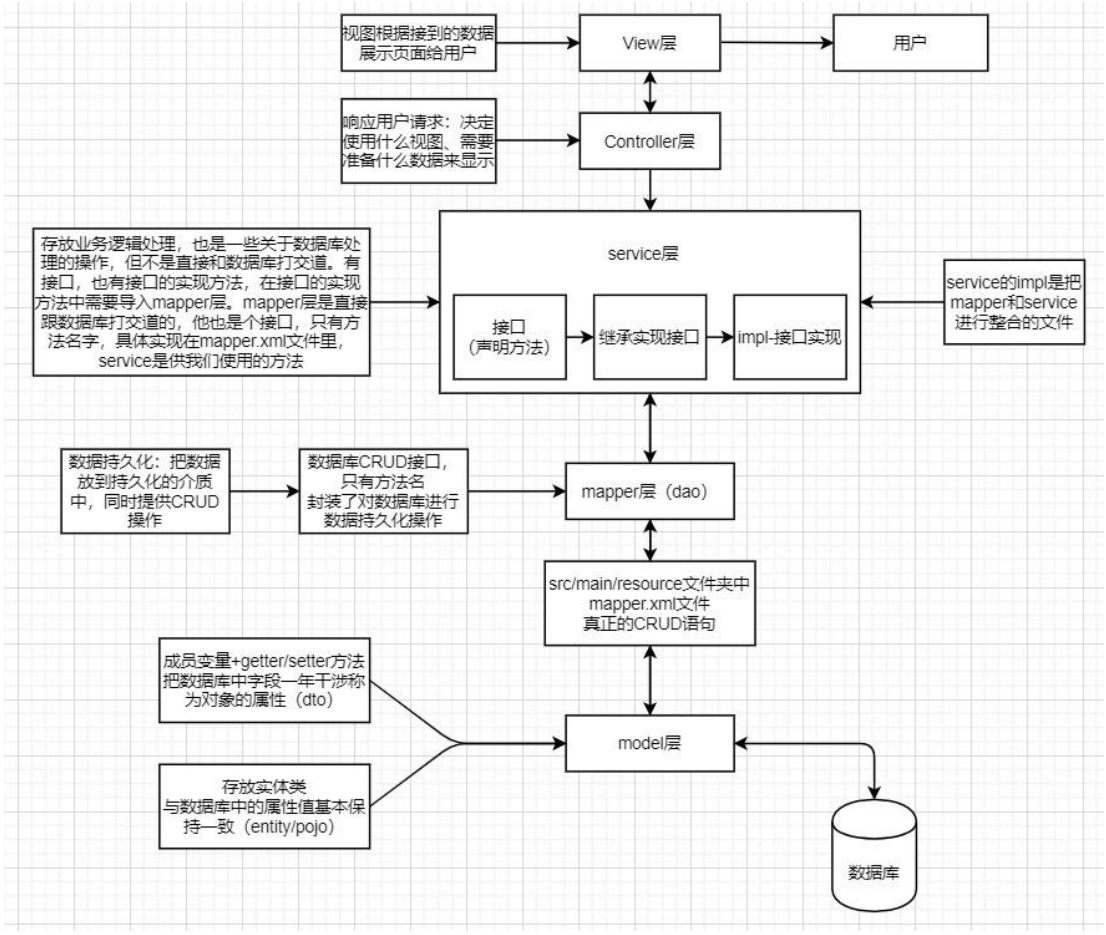


图 2-2-1 Spring Boot 架构

二、业务实体设计

生物测序实验管理系统的业务实体在内存中的表现为实体域对象，在数据库中表现为关系数据，业务实体主要包括关系数据模型设计和与之对应的实体对象模型设计两个方面。在生物测序实验管理系统中主要有以下的业务实体：用户、样本数据、任务数据等等。下面对这些业务实体做一个简单的介绍，在后面的开发中会有详细的设计和实现。

1. 实体列举

用户（Users）：代表一个用户实体，主要包括用户的详细信息，如用户 ID、用户名、登录密码、用户创建时间等。

样本数据：代表一个测序数据的信息，主要包括样本类型、样本编号、测序方法、数据文件路径等等。

任务数据：代表一个分析任务的信息，主要包括任务名称、任务类型、样本个数等等。

2. 实体之间的对应关系

这些实体之间的关系如图 2-2-2 所示。

用户和样本数据：一个用户可以拥有多个样本数据，而一个样本数据只能属于一个用户的。用户和样本数据之间的关系是一对多的关系，在数据库表中表现为样本数据表中有一个用户表的外键。

用户和任务数据：一个用户可以拥有多个任务数据，而一个任务数据只能属于一个用户的。用户和任务数据之间的关系是一对多的关系，在数据库表中表现为任务数据表中有一个用户表的外键。

样本数据和任务数据：一个任务可以拥有多个样本数据，同时一个样本数据可以属于多个任务，样本数据和任务数据之间的关系是多对多的关系，在数据库表中表现为任务有多个样本数据。

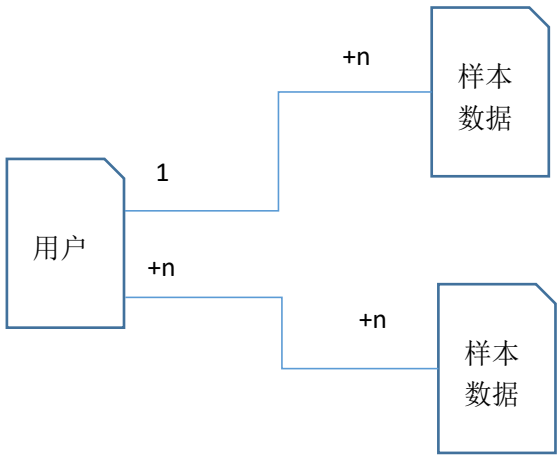


图 2-2-2 实体对应关系

三、业务逻辑设计

在生物测序实验管理系统中，使用 DAO 设计模式实现对数据层的访问，DAO 设计模式是 Java Web 应用开发中的一种常用模式，其主要思想是在业务处理部分和数据库之间再增加一层数据库操作层，用这层来连接业务处理和数据源，这样就实现了业务处理核心逻辑和具体数据源之间的功能对立和分离。

因为具体的数据库或数据源可能是多种多样的，可能是关系数据库或者是 XML。在具体的关系数据库中，也可能是不同的产品，如 SQL Server、Oracle、MySQL 或者 MariaDB。通过使用 DAO 模式，业务处理部分就不用关心数据库操作层是如何实现对数据库的操作的，而只关系自己的业务逻辑，对数据库的操作全部留给了 DAO，有 DAO 执行具体的数据库操作，并将结果返回给业务逻辑，如图 2-2-3 所示。

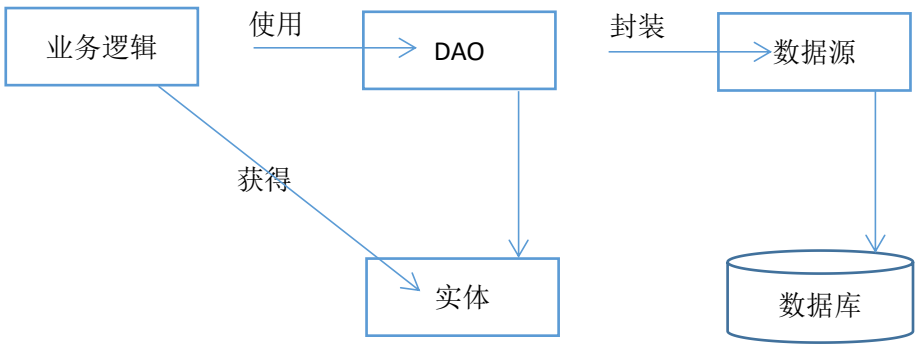


图 2-2-3 DAO 模式

四、开发环境

在正式开发之前，要构建好系统的开发环境。目前主流的 Java Web 应用集成开发使用 Eclipse，是一个开放源代码、基于 Java 的可扩展 IDE。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 附带了一个标准的插件集，包括 Java 开发工具(Java Development Tools, JDT)。Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下代 IDE 开发环境，2001 年 11 月 IBM 公司将价值 4000 万美元的源代码贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理，并由该联盟负责这种工具的后续开发。2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构，2007 年 6 月，稳定版 3.3 发布，2008 年 6 月发布代号为 Ganymede 的 3.4 版。Eclipse 最初主要用来开发 Java 语言，但是目前也有人通过插件使其作为其他计算机语言，比如 C++ 的开发工具。Eclipse 本身只是一个框架平台，但是众多插件的支持使得 Eclipse 拥有其他功能相对固定的 IDE 软件很难具有的灵活性。许多软件开发商以 Eclipse 为框架开发自己的 IDE。

在生物测序实验管理系统时，需要具备以下开发环境：

- 操作系统：Windows 11
- Java 开发包：jdk-17
- Web 服务器：apache-tomcat-10.1.11
- 开发框架：Spring Boot 3
- 数据库：MariaDB 11.2
- 集成开发环境：Eclipse

任务实施

Step1 系统架构设计

Step2 业务实体设计

Step3 业务逻辑设计

Step4 开发环境

任务 3 生物测序实验管理系统项目 Spring Boot 3 开发框架认知与搭建

项目描述

开发项目要能事半功倍，一个好的开发框架少不了，跟系统设计使用集成开发工具 Eclipse 创建项目，在创建的项目中引入开发框架 Spring Boot 3，并且在项目中创建 View 层、Controller 层、Service 层、Dao 层、Pojo（entity）层。

项目目标

1. 通过 Eclipse 工具创建项目并引入 Spring Boot 框架。
2. 能在配置文件中完成 Spring Boot 配置。
3. 能在启动类中写入 Spring Boot 的启动代码。
4. 能集成 Thymeleaf 模板引擎。
5. 掌握 Spring Boot Web 创建 Controller 类输出模板到浏览器。
6. 能集成 Spring Data JPA 及完成对数据库访问的配置。
7. 能完成实体层和数据访问层代码编写并且读取数据库中的数据。

活动一 创建系统项目并引入 Spring Boot 框架并启动运行

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。在项目开发之前我们先要创建工程项目，并且引入 Spring Boot，让我们一起实现吧！

任务目标

1. 通过 Eclipse 工具创建项目并引入 Spring Boot 框架。
2. 能在配置文件中完成 Spring Boot 配置。
3. 能在启动类中写入 Spring Boot 的启动代码。

任务分析

1. 如何使用项目构建工具 Maven 快速创建开发项目并且引入 Spring Boot 框架？
2. 如何完成 Spring Boot 的配置？
3. 如何启动 Spring Boot？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、认识 Spring Boot 框架

1. Spring Boot 是什么

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化 Spring 应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。

简单的理解就是 Spring Boot 其实不是什么新的框架，它默认配置了很多框架的使用方式，就像 Maven 整合了所有的 jar 包，Spring Boot 整合了所有的框架。

Spring Boot 统一江湖后，大家都会按照 Spring Boot 约定的方式来走，这样都会有统一的编程体验，介于 Spring 处于江湖一哥的位置，各大相关软件、开源项目都进行主动适配。

2. Spring Boot 的优缺点

（1）优点

①可快速构建独立的 Spring 应用

Spring Boot 是一个依靠大量注解实现自动化配置的全新框架。在构建 Spring 应用时，我们只需要添加相应的场景依赖，Spring Boot 就会根据添加的场景依赖自动进行配置，在无须额外手动添加配置的情况下快速构建出一个独立的 Spring 应用。

②直接嵌入 Tomcat、Jetty 和 Undertow 服务器

传统的 Spring 应用部署时，通常会将应用打成 WAR 包形式并部署到 Tomcat、Jetty 或 Undertow 服务器中。Spring Boot 框架内嵌了 Tomcat、Jetty 和 Undertow 服务器，而且可以自动将项目打包，并在项目运行时部署到服务器中。

③通过依赖启动器简化构建配置

在 Spring Boot 项目构建过程中，无须准备各种独立的 JAR 文件，只需在构建项目时根据开发场景需求选择对应的依赖启动器“starter”，在引入的依赖启动器“starter”内部已经包含了对应开发场景所需的依赖，并会自动下载和拉取相关 JAR 包。

④自动化配置 Spring 和第三方库

Spring Boot 充分考虑到与传统 Spring 框架以及其他第三方库融合的场景，在提供了各种场景依赖启动器的基础上，内部默认提供了各种自动化配置类（例如 Redis Auto Configuration）。使用 Spring Boot 开发项目时，一旦引入了某个场景的依赖启动器，Spring Boot 内部提供的默认自动化配置类就会生效，开发者无须手动在配置文件中进行相关配置（除非开发者需要更改默认配置），从而极大减少了开发人员的工作量，提高了程序的开发效率。

⑤提供生产就绪功能

Spring Boot 提供了一些用于生产环境运行时的特性，例如指标、监控检查和外部化配置。其中，指标和监控检查可以帮助运维人员在运维期间监控项目运行情况；外部化配置可以使运维人员快速、方便地进行外部化配置和部署工作。

⑥极少的代码生成和 XML 配置

Spring Boot 框架内部已经实现了与 Spring 以及其他常用第三方库的整合连接，并提供了默认最优化的整合配置，使用时基本上不需要额外生成配置代码和 XML 配置文件。在需要自定义配置的情况下，Spring Boot 更加提倡使用 Java config (Java 配置类)替换传统的 XML 配置方式，这样更加方便查看和管理。

（2）缺点

Spring Boot 也有一些明显的缺点，如 Spring Boot 入门较为简单，但是深入理解和学习却有一定的难度，这是因为 Spring Boot 是在 Spring 框架的基础上推出的，所以开发人员想要弄明白 Spring Boot 的底层运行机制，有必要对 Spring 框架有一定的了解。

3. Spring Boot 的四大核心

（1）自动配置

针对很多 Spring 应用程序和常见的应用功能，Spring Boot 相关配置可自动提供，通过简单的配置，甚至零配置，可以构建一套完整的框架。

（2）起步依赖

告诉 Spring Boot 它可以引入所需的依赖库；通过启动依赖机制(Starter)，简化 jar 包的引用，解决 jar 版本的冲突。

（3）Actuator

Actuator 是 Spring Boot 的程序监控器，可以监控 Spring 应用程序上下文中的 Bean、查看自动配置决策、Controller 映射、线程活动、应用程序健康状况等，能深入运行的 Spring Boot 应用程序，探索 Spring boot 程序内部信息。

（4）命令界面

这是 Spring Boot 的可选特性主要用于 Groovy 语言。

4. Spring Boot 的应用场景

（1）快速构建 RESTful API 服务

Spring Boot 提供了一系列的自动配置和基础组件，可以帮助你快速构建基于 Spring MVC 的 RESTful API 服务。

（2）快速构建微服务架构

Spring Boot 提供了一系列的微服务工具和组件，包括服务注册与发现、负载均衡、断路器等，可以帮助你快速构建微服务架构。

（3）快速构建企业级应用

Spring Boot 提供了丰富的企业级应用组件，包括数据持久化、消息中间件、安全认证、任务调度等，可以帮助你快速构建企业级应用。

（4）快速构建云原生应用

Spring Boot 提供了对云原生应用的支持，包括对云服务的集成、对容器化应用的支持等，可以帮助你快速构建云原生应用。

二、使用 Maven 快速引入 Spring Boot 框架

1. 使用 Maven 创建项目

在 Eclipse 菜单栏中选择【File】→【New】→【Other】打开向导窗口如图 3-1-1 所示，在向导窗口中选择【Maven】→【Maven Project】然后点击“Next>”按钮进入下一步。

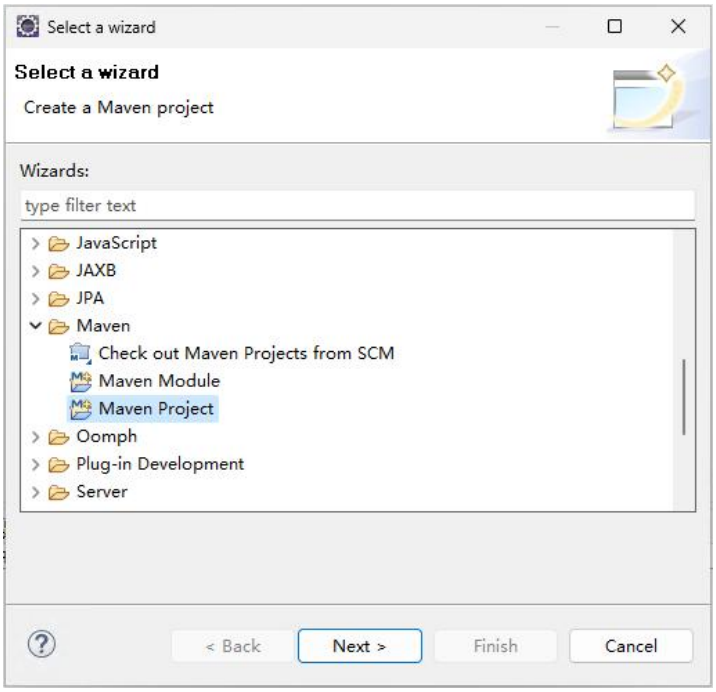


图 3-1-1 选择一个创建导向

在创建新的 Maven 项目窗口中保留默认，如图 3-1-2 所示，点击“Next>”按钮直接下一步。

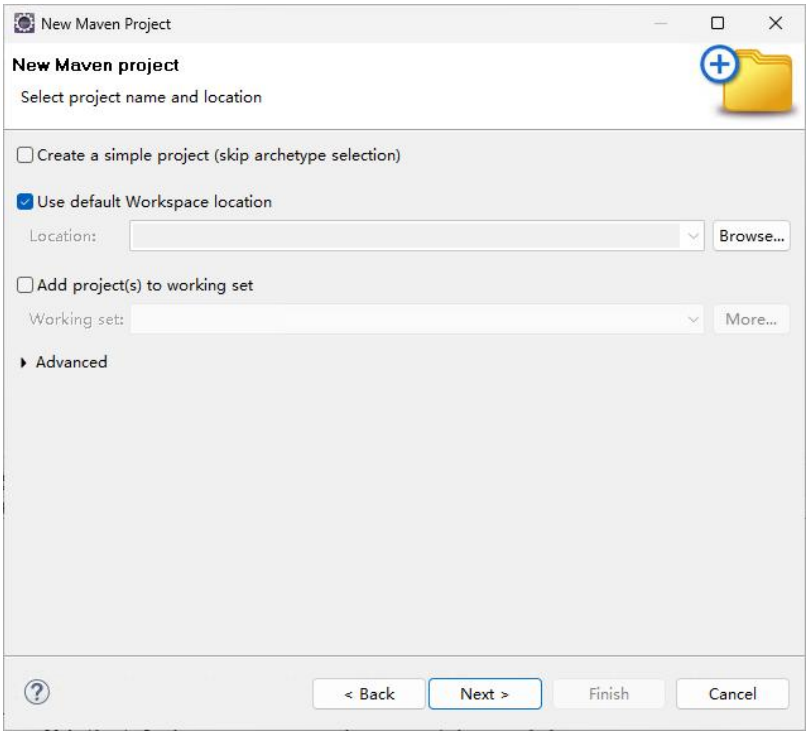


图 3-1-2 创建新的 Maven 项目

选择一个创建原型，如图 3-1-3 所示，这里选择【Artifact Id】列的“maven-arachetype-quickstart”，点击“Next>”按钮进入下一步。

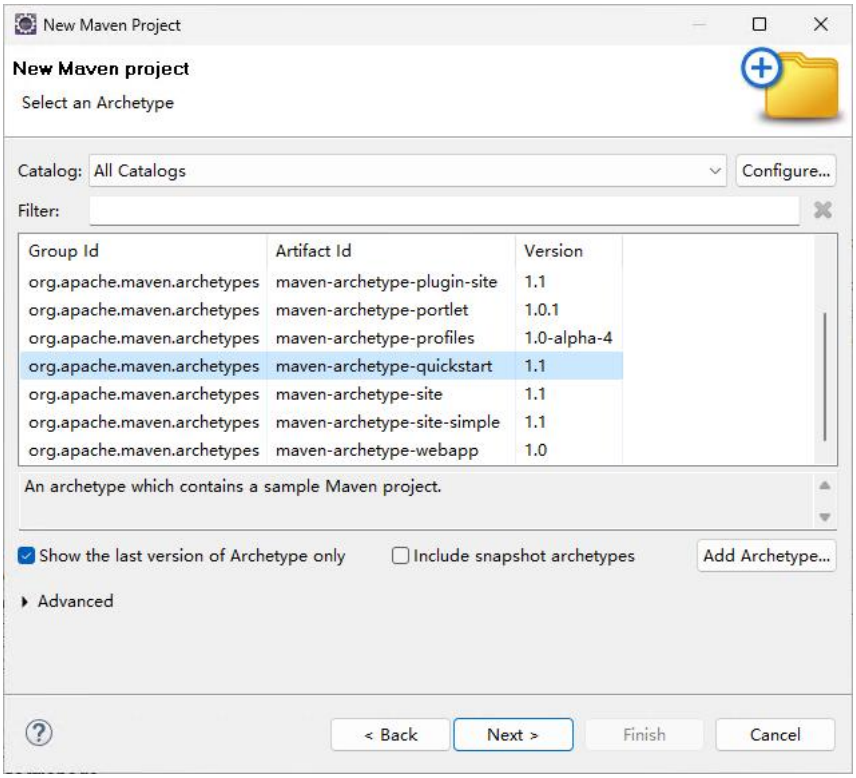


图 3-1-3 选择原型

输入创建项目需要的参数，“Group Id”一般填写的是“com”，“Artifact Id”填写项目的名称，“Version”默认即可，“Package”会自动生成，我们创建项目参数如图 3-1-4 所示。

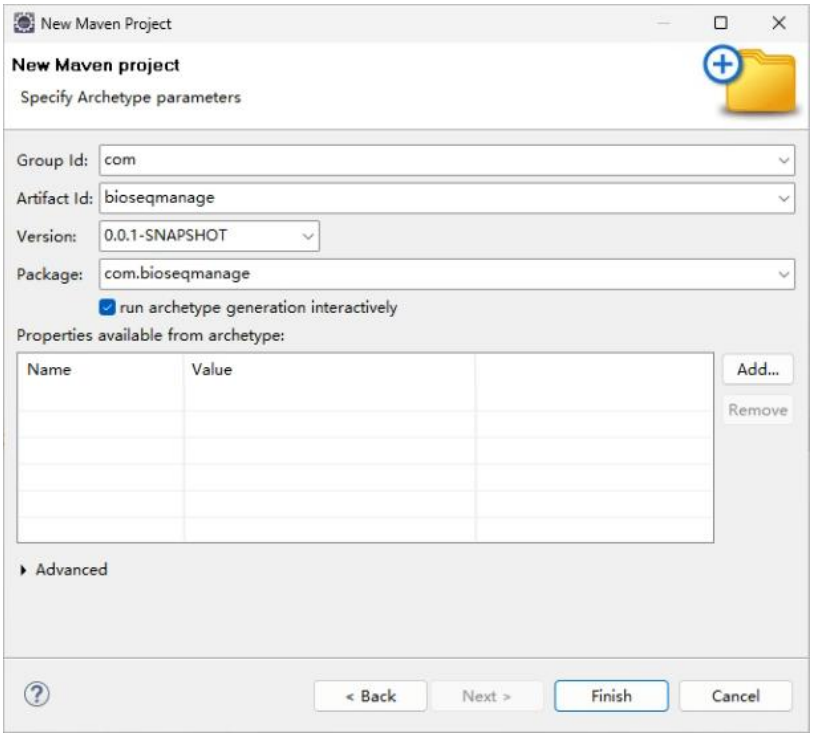


图 3-1-4 填写创建项目参数

在项目生成的过程中需要在控制台中输入“y”然后回车，如图 3-1-5 所示。

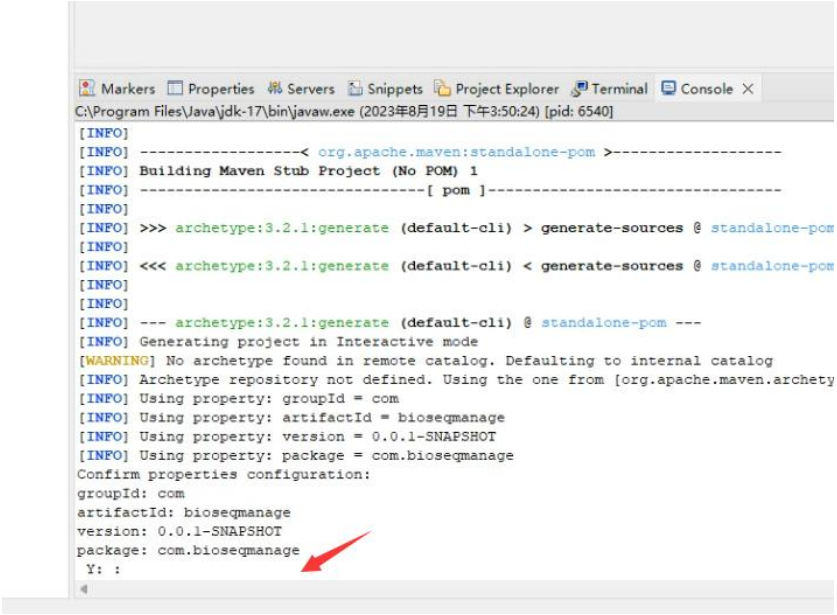


图 3-1-5 输入确认信息

最后创建成功如图 3-1-6 所示。

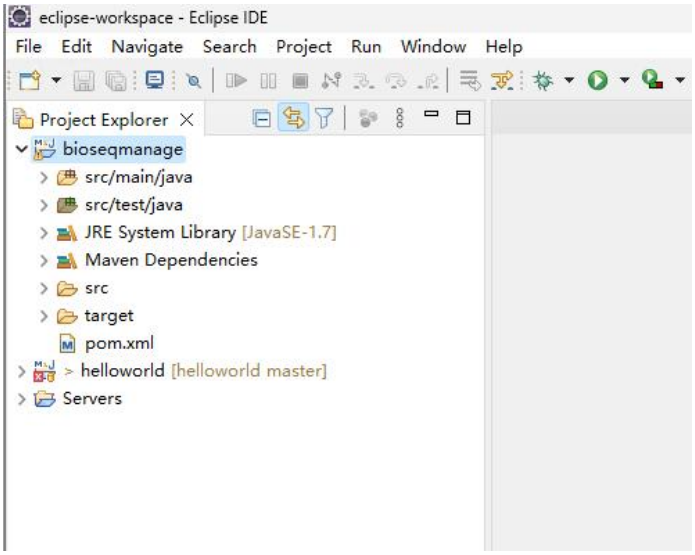


图 3-1-6 项目创建成功

2. 引入 Spring Boot 框架

加入 SpringBoot 框架，打开 pom.xml 文件在根节点加入如下代码，定义父类：springboot 所有 jar 包版本，我们使用 3.1.1。

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.1</version>
```

```
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
```

三、引入 Spring Boot Starter 与自动配置

在 pom.xml 文件的 dependencies 节点加入 springboot 核心包

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
```

四、引入 Spring Boot Web

在 pom.xml 文件的 dependencies 节点加入以下代码：

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

五、引入 Spring Boot 热部署

在 pom.xml 文件的 dependencies 节点加入以下代码：

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
    </dependency>
```

六、Spring Boot 启动过程与拓展应用

Spring Boot 的启动需要创建一个类，我们习惯使用 App 这个类文件来作为启动类，在创建项目的时候已经生成有这个类了，只需要对这个类里加一个注解和一个程序启动方法即可，如图 3-1-7 所示。

注解：@SpringBootApplication

启动方法：SpringApplication.run(App.class, args);

App.java

```
1  package com.bioseqmanage;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class App
8  {
9      public static void main( String[] args )
10     {
11         SpringApplication.run(App.class, args);
12     }
13 }
14
```

图 3-1-7 启动类及启动方法

任务实施

Step1 认识 Spring Boot 框架

Step2 使用 Maven 快速引入 Spring Boot 框架

Step3 引入 Spring Boot Starter 与自动配置

Step4 引入 Spring Boot Web

Step5 引入 Spring Boot 热部署

Step6 Spring Boot 启动

活动二 系统项目集成模板引擎

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。要求在工程项目引入页面模板引擎，日后开发的系统界面用它来渲染，让我们一起实现吧！

任务目标

1. 能集成 Thymeleaf 模板引擎。
2. 掌握 Spring Boot Web 创建 Controller 类输出模板到浏览器。

任务分析

1. 如何集成 Thymeleaf 模板引擎？
2. 如何创建 Controller 类把模板内容渲染到浏览器？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、引入 Thymeleaf 模板引擎

打开 pom.xml 文件在 dependencies 节点加入如下代码，引入 Thymeleaf 相关 jar 包。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

二、模板引擎文件目录

创建资源目录“resources”，然后在这个目录里创建目录“templates”，这个目录用于

存放模板文件，如图 3-2-1 所示。

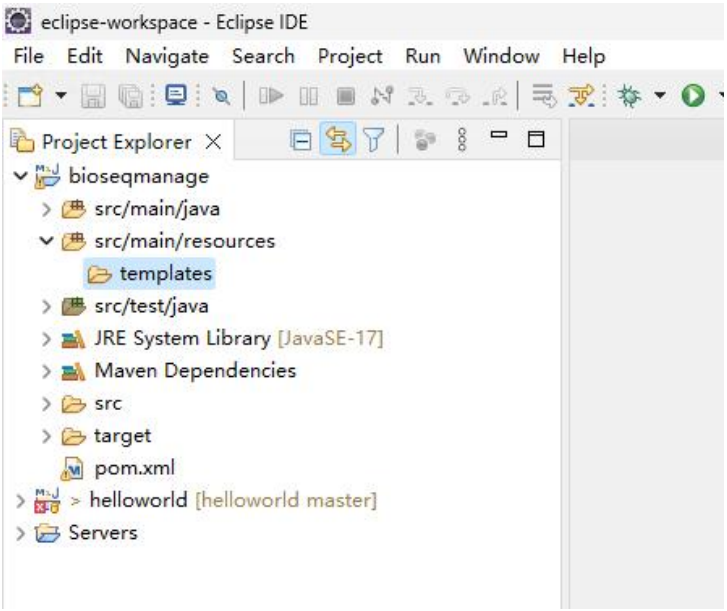


图 3-2-1 模板文件目录

三、模板引擎配置

在资源目录“resources”创建 application.properties 文件，如果已经存在直接打开即可，如图 3-2-2 所示。

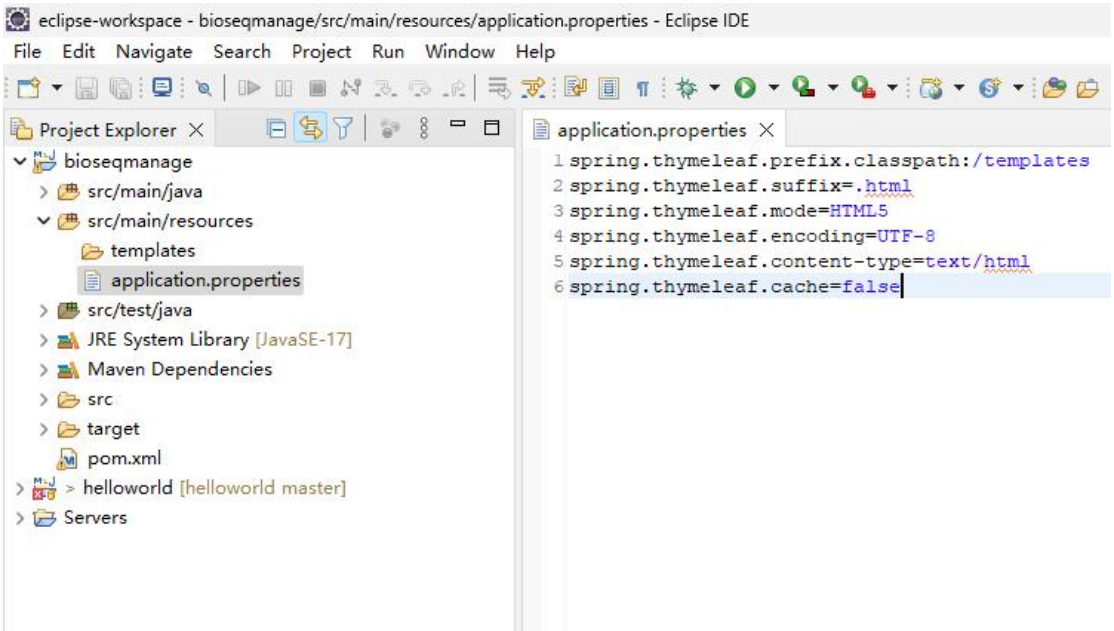


图 3-2-2 application.properties 文件

在 application.properties 文件里加入以下代码：

```
spring.thymeleaf.prefix.classpath:/templates
```

```
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML5
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.content-type=text/html
spring.thymeleaf.cache=false
```

四、Controller 控制层认知

在项目中右击在菜单中选择【New】→【Package】然后在弹出窗口中找到“Name:”一项输入名称“com.bioseqmanage.controller”，最后点击“Finish”创建 controller 包。在该包中创建 IndexController.java 文件做为系统的默认启动页面，如图 3-2-3 所示。

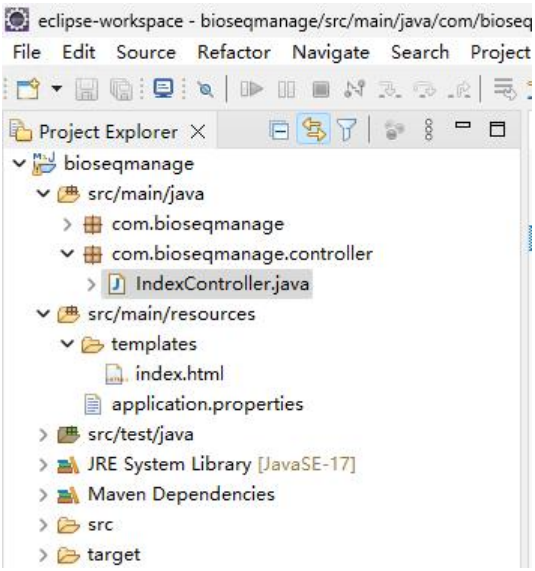


图 3-2-3 controller 控制层及 IndexController.java 文件

类文件中代码如下：

```
@Controller
public class IndexController {
    @RequestMapping(value={"/","/index"})
    public String index() {
        return "index";
    }
}
```

在模板目录创建模板文件 index.html，代码输入如下：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
```

```
<title>Index</title>
</head>
<body>
    我是默认启动页面
</body>
</html>
```

右击项目在快捷菜单中选择【Run As】→【Java Application】启动运行应用程序，打开浏览器输入 http://localhost:8080 查看输出结果，如图 3-2-4 所示。

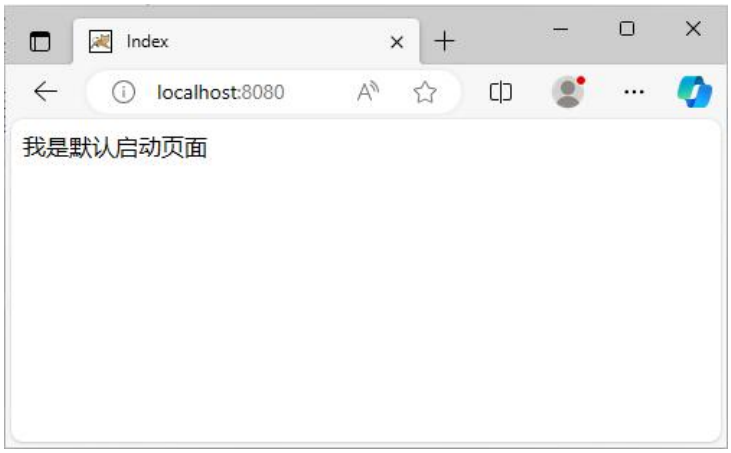


图 3-2-4 输入结果

任务实施

Step1 引入 Thymeleaf 模板引擎

Step2 模板引擎文件目录

Step3 模板引擎配置

Step4 Controller 控制层认知

活动三 项目系统集成数据库访问

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。要求在工程项目中做数据库访问集成，让我们一起实现吧！

任务目标

1. 能集成 Spring Data JPA 及完成对数据库访问的配置。
2. 能完成实体层和数据访问层代码编写并且读取数据库中的数据。

任务分析

1. 如何使用 Spring Data JPA 访问数据库？
2. 如何访问 MariaDB 数据库？
3. 什么是实体类以及在实体类中使用@Entity 的作用是什么？
4. 数据访问接口继承 JpaRepository 有什么作用？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、集成 Spring Data JPA

Spring Data 是 Spring 提供的操作数据的框架，Spring Data JPA 是 Spring Data 的一个模块，通过 Spring data 基于 jpa 标准操作数据的模块。

Spring Data 的核心能力，就是基于 JPA 操作数据，并且可以简化操作持久层的代码。

它使用一个叫作 Repository 的接口类为基础，它被定义为访问底层数据模型的超级接口。而对于某种具体的数据访问操作，则在其子接口中定义。

Spring Data 可以让我们只定义接口，只要遵循 spring data 的规范，就无需写实现类，不

用写 sql 语句直接查询数据。

引入 Spring Boot JPA 框架，打开 pom.xml 文件在 dependencies 节点加入如下代码。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

二、引入 MariaDB 以及配置

使用 MariaDB 数据库 Java 应用程序在读取数据前需要进行连接，这里我们使用 MariaDB JDBC 进行连接，打开 pom.xml 文件在 dependencies 节点加入如下代码。

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
</dependency>
```

连接配置需要在 application.properties 文件里进行，打开该文件加入如下代码。

```
spring.datasource.url=jdbc:mariadb://localhost:3306/bio_seq_manage?autoReconnect=true&useUnicode=true&characterEncoding=utf-8
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.hikari.maxLifeTime=600000
```

spring.datasource.url 是数据库连接字符串，localhost 是数据库的地址，因为是本地所以使用 localhost。冒号后面的是数据库端口，默认是 3306。紧接着的 bioseqmanage 是数据库名。问号后面的是一些参数。

spring.datasource.username 是登录数据库的用户名。

spring.datasource.password 是数据库登录的密码。

三、实体类 Entity 的编写

实体类其实就是俗称的 POJO，这种类一般不实现特殊框架下的接口，在程序中仅作为数据容器用来持久化存储数据用的。

在项目中右击在菜单中选择【New】→【Package】然后在弹出窗口中找到“Name:”一项输入名称“com.bioseqmanage.entity”，最后点击“Finish”创建 com.bioseqmanage.entity 包。在该包中创建 Config.java 文件，该类文件是系统的配置实体类，如图 3-3-1 所示。

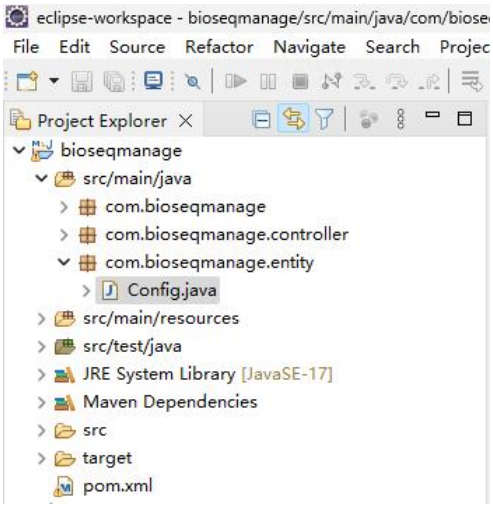


图 3-3-1 系统配置实体类文件

在 Config 类名上加入两个注解@Entity 和@Table(name = "config")，并且实现 Serializablejie 接口，如图 3-3-2 所示。

@Entity 表示该类问实体类

@Table 指定映射的数据库表名

Serializable 接口是启用其序列化功能的接口。实现 java.io.Serializable 接口的类是可序列化的。没有实现此接口的类将不能使它们的任意状态被序列化或逆序列化。

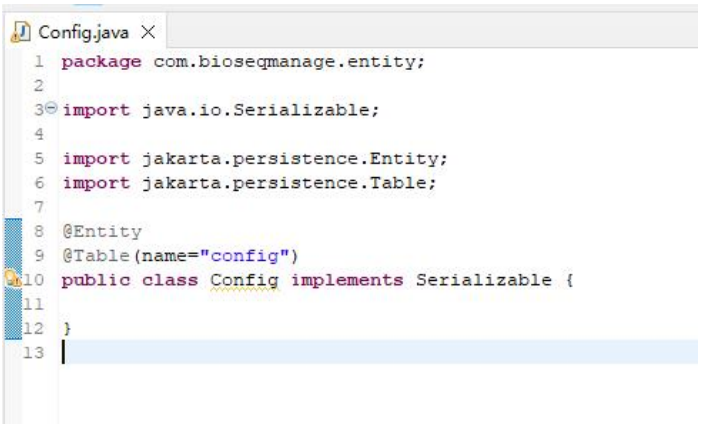


图 3-3-2 配置实体类注解与继承

此时在“Config”类名中有一条黄色的下划线，把鼠标放到上面会出现提示，如图 3-3-3 所示。点击“Add default serial version ID”完成添加后提示消失，如图 3-3-4 所示。

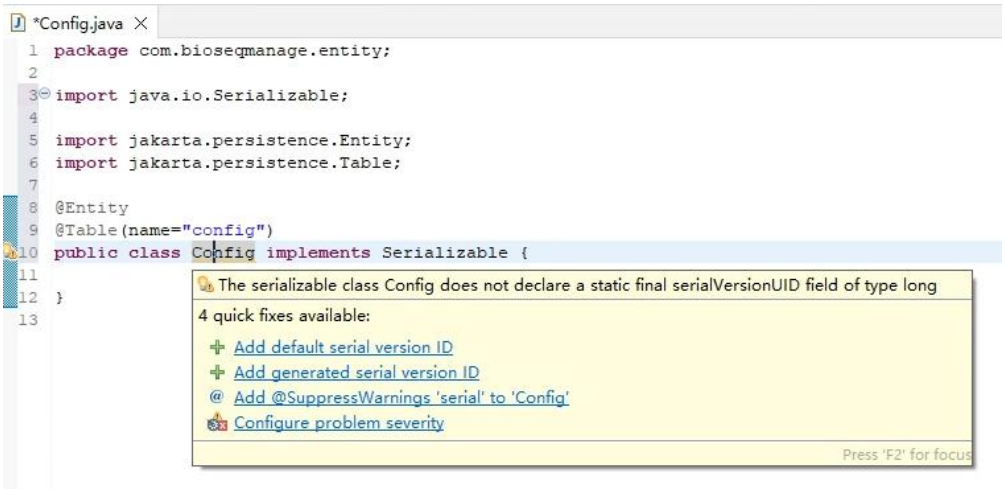


图 3-3-3 加入序列版本 ID 提示

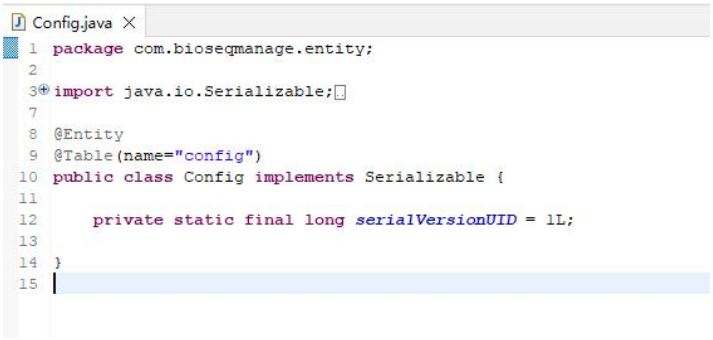


图 3-3-4 序列版本 ID

实体类有属性和方法，属性对应数据库中表的字段，主要有 getter 和 setter 方法。config 数据表如图 3-3-5 所示。

图 3-3-5 config 表

为了方便学习提供 config 数据库表的 SQL 创建语句以及数据插入语句如下：

```
CREATE TABLE `config` (
    `config_id` INT(11) NOT NULL,
    `name` VARCHAR(50) NOT NULL DEFAULT '' COLLATE 'utf8mb4_general_ci',
    `logo` VARCHAR(255) NOT NULL DEFAULT '' COLLATE 'utf8mb4_general_ci',
    PRIMARY KEY (`config_id`) USING BTREE
)
```

```
COMMENT='系统配置表'
COLLATE='utf8mb4_general_ci'
ENGINE=InnoDB
;

INSERT INTO `bio_seq_manage`.`config` (`config_id`, `name`,`logo`) VALUES (1, '生物测序实验管理系统', '/static/images/syslogo.jpg');
```

实体类最后完成的代码如下：

```
package com.bioseqmanage.entity;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Table;

@Entity
@Table(name="config")
public class Config implements Serializable {

	private static final long serialVersionUID = 1L;

	private int configId;
	private String name;
	private String logo;

	/**
	 * 配置 ID
	 * @return
	 */
	public int getConfigId() {
		return this.configId;
	}
	/**
	 * 配置 ID
	 * @param configId
	 */
	public void setConfigId(int configId) {
		this.configId = configId;
	}
	/**
	 * 系统名称
	 * @return
	 */
```

```
    public String getName() {
        return this.name;
    }
    /**
     * 系统名称
     * @param name
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * 系统 logo
     * @return
     */
    public String getLogo() {
        return this.logo;
    }
    /**
     * 系统 logo
     * @param logo
     */
    public void setLogo(String logo) {
        this.logo = logo;
    }
}
```

四、数据访问接口的实现

在项目中右击在菜单中选择【New】→【Package】然后在弹出窗口中找到“Name:”一项输入名称“com.bioseqmanage.dao”，最后点击“Finish”创建 com.bioseqmanage.dao 包。在该包中创建 ConfigDao.java 文件，该类文件是系统配置的数据访问接口，如图 3-3-6 所示。

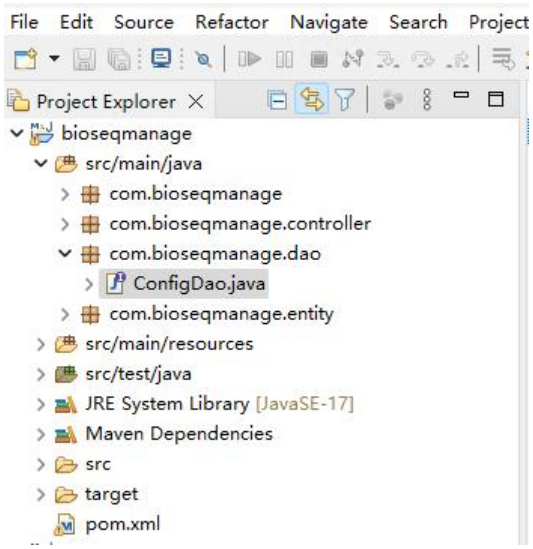


图 3-3-6 Dao 层以及配置数据访问接口

在 Config 类名上加入一个注解@Repository，并且继承 JpaRepository 接口，JpaRepository 需要输入实体类类型以及 ID 类型。

JpaRepository 继承 PagingAndSortingRepository，添加了一组 JPA 规范相关的方法。对继承父接口中方法的返回值进行了适配，因为在父类接口中通常都返回迭代器，需要我们自己进行强制类型转化。而在 JpaRepository 中，直接返回了 List 或响应的实体。

ConfigDao 接口的代码如下：

```
package com.bioseqmanage.dao;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.bioseqmanage.entity.Config;

@Repository
public interface ConfigDao extends JpaRepository<Config, Integer> {

    Optional<Config> findById(int configId);

}
```

五、业务逻辑层 service 接口以及 service 实现类

在项目中右击在菜单中选择【New】→【Package】然后在弹出窗口中找到“Name:”一项输入名称“com.bioseqmanage.service”，最后点击“Finish”创建 com.bioseqmanage.service

包。在该包中创建 ConfigService.java 文件，该类文件是系统配置的业务逻辑服务接口。

在项目中右击在菜单中选择【New】→【Package】然后在弹出窗口中找到“Name:”一项输入名称“com.bioseqmanage.service.impl”，最后点击“Finish”创建 com.bioseqmanage.service.impl 包。在该包中创建 ConfigServiceImpl.java 文件，该类文件是系统配置的业务逻辑服务实现，如图 3-3-7 所示。

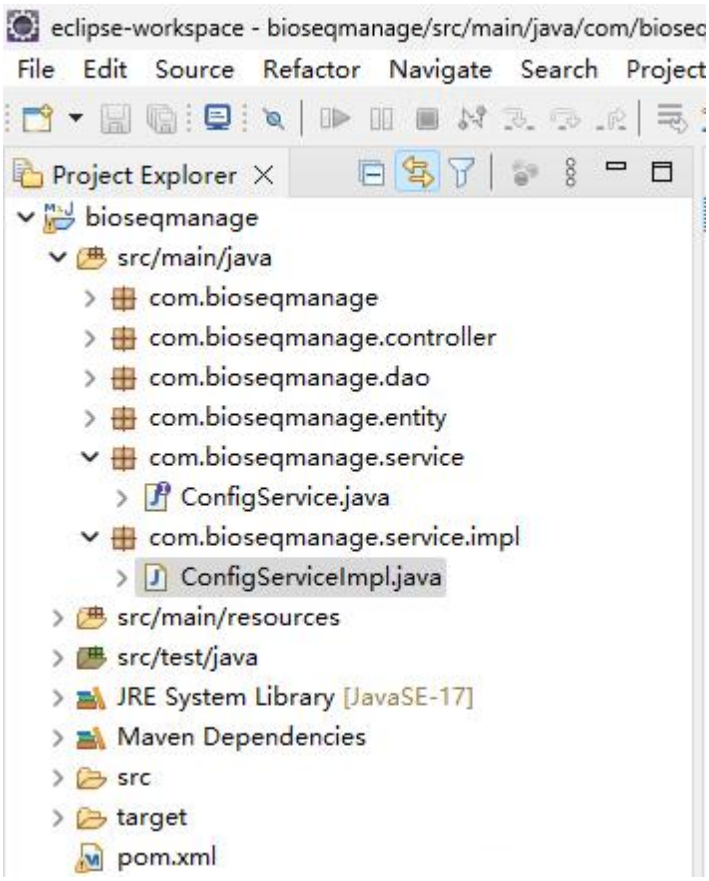


图 3-3-7 业务逻辑层 service 接口以及 service 实现类

最后运行应用程序，打开浏览器输入 http://localhost:8080 查看结果，如图 3-3-8 所示。

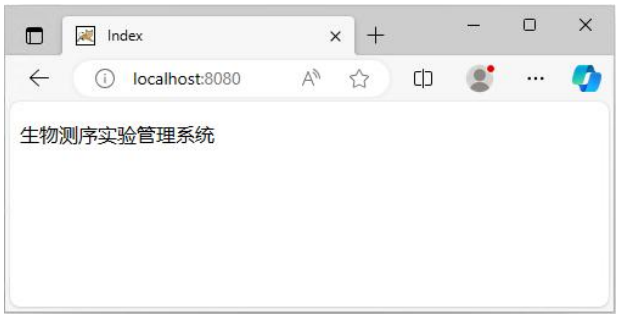


图 3-3-8 读取配置信息并且输出

任务实施

Step1 引入 Spring Boot JPA 框架

Step2 引入 MariaDB 需要的 JAR 包

Step3 对 MariaDB 进行配置

Step4 实体类 Entity 的编写

Step5 数据访问接口的实现

Step6 业务逻辑层 service 接口以及 service 实现类

任务 4 生物测序实验管理系统开发

项目描述

项目开发框架搭建完成后接下来就是进行代码编程，根据系统功能分析我们需要完成：用户登录管理、样本导入、样本导出、样本数据管理、创建分析任务、任务搜索等功能。系统最终是给用户去用的，所以我们需要注意用户体验，系统必须有一个良好的操作界面，然后是一个健壮的逻辑代码。

项目目标

1. 掌握 HTML 和 CSS 完成登录页面的制作。
2. 能 Thymeleaf 布局标签完成系统登录页面制作。
3. 通过引入 jQuery 库并且获取到标签元素对象，然后通过 jQuery 的方法操作元素对象。
4. 能对 MySql 数据库进行 user 和 user_role 表结构及表之间关联关系的设计。
5. 对实体类的认知能完成实体类的代码编写。
6. 对数据访问层以及 service 层的认知能完成用户数据的集成封装。
7. 能通过读数据库的用户表进行查询并且判断用户的账号密码是否正确。
8. 能对查询出来的数据进行判断是否登录成功并且做出页面跳转处理。
9. 能通过过滤器 Filter 和 session 会话判断当前访问是否已经登录。
10. 对文件的下载功能的实现认知然后完成样本模板文件的下载。
11. 能对导入的 Excel 文件里的数据进行读取并且能插入数据库保存。
12. 能读出数据库数据通过 Java 语言编程实现对 Excel 文件的数据写入。
13. 能完成数据列表的展示。
14. 能完成数据的分页功能。
15. 能完成数据的删除功能。
16. 能通过 URL 传递参数和服务器端接收传递的数据来完成不同的分析任务的创建。
17. 能完整表单数据提交前的校验和进行服务器校验，并能输出错误提示。
18. 能通过前端表单向服务器提交搜索数据。
19. 服务器段接收数据完成数据库数据搜索。

活动一 用户登录页面与系统首页制作

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。要求完成用户登录页面与系统首页制作！

任务目标

1. 掌握 HTML 和 CSS 完成登录页面的制作。
2. 能 Thymeleaf 布局标签完成系统登录页面制作。

任务分析

1. 什么是 HTML？
2. HTML 的文档结果是怎样的？
3. HTML 都有些什么标签？
4. 什么是 CSS？
5. CSS 如何使用到网页中？
6. Thymeleaf 如果输出数据以及都有哪些基本语法？
7. 如何使用 HTML 和 CSS 完成系统登录和首页的制作？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、HTML 超文本标记语言

1. 什么是 HTML

HTML，全名“Hyper Text Markup Language”，即超文本标记语言，是一种用于创建网页的标准标记语言。HTML 是用于描述网页的一种语言，HTML 文本是由 HTML 命令组成的描述性文本，HTML 命令可以说明文字、图形、动画、声音、表格、链接等。

超文本是一种组织信息的方式，它通过超级链接方法将文本中的文字、图表与其他信息媒体相关联。这些相互关联的信息媒体可能在同一文本中，也可能是其他文件，或是地理位置相距遥远的某台计算机上的文件。这种组织信息方式将分布在不同位置的信息资源用随机方式进行链接，为人们查找，检索信息提供方便。

2. HTML 的文档结构

HTML 文档结构由以下几个标签组成：

<!DOCTYPE html>标签：声明定义文档类型为 HTML5。

<html>标签：包含整个 HTML 文档的内容。

<head>标签：网页的头部是所有头部元素的容器，如字符集、标题、样式表和脚本等。

<meta>标签：位于文档的头部，不包含任何内容，它的属性定义了与文档相关联的名称/值对

<title>标签：网页的标题，定义文档的标题，显示在浏览器的标题栏或标签页上。

<body>标签：网页的身体，包含网页的实际内容，如文本、图片、链接等。

HTML 文档代码如下：

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>页面的标题</title>
 </head>
 <body>
        网页的主体内容
 </body>
</html>
```

3. HTML 的标签

（1）排版标签

超文本是一种组织信息的方式，它通过超级链接方法将文本中的文字、图表与其他信息媒体相关联。这些相互关联的信息媒体可能在同一文本中，也可能是其他文件，或是地理位置相距遥远的某台计算机上的文件。这种组织信息方式将分布在不同位置的信息资源用随机方式进行链接，为人们查找，检索信息提供方便。

1） 标题标签

场景：在新闻和文章的页面中，都离不开标题，用来突出显示文章主题。

h 系列标签：

```
<h1>1 级标题</h1>
<h2>2 级标题</h2>
<h3>3 级标题</h3>
<h4>4 级标题</h4>
<h5>5 级标题</h5>
<h6>6 级标题</h6>
```

语义：1~6 级标题，重要程度依次递减

特点：文字都有加粗、文字都有变大，并且从 h1→h6 文字逐渐减小，独占一行。

注意点：h1 标签对于网页尤为重要，开发中有特定的使用场景，如新闻的标题、网页的 logo 部分。

2) 段落标签

场景：在新闻和文章的页面中，用于分段显示。
p 标签：
我是一段文字
语义：段落
特点：段落之间存在间隙，独占一行。

3) 换行标签

场景：让文字强制换行显示

换行标签：
语义：换行
特点：单标签，让文字强制换行。

4) 水平线标签

场景：分割不同主题内容的水平线
代码：<hr>
语义：主题的分割转换
特点：单标签，在页面中显示一条水平线。

(2) 文本格式化标签

场景：需要让文字加粗、下划线、倾斜、删除线等效果
代码：

标签		说明	标签	说明
b		加粗	strong	加粗
u		下划线	ins	下划线
i		倾斜	em	倾斜
s		删除线	del	删除线

语义：突出重要性的强调语境（强调语境用右边的）

(3) 媒体标签

1) 图片标签

①图片标签的介绍
场景：在网页中显示图片
代码：

```
<img src="" alt="">
```

特点：单标签，img 标签需要展示对应的效果，需要借助标签的属性进行设置！

标签的完整结构：

```
<!-- src="" :标签属性  src：属性名 "":属性值 →-->
<img src="" alt="">
```

属性注意点：

- ●标签的属性写在开始标签内部
- ●标签上可以同时存在多个属性
- ●属性之间以空格隔开
- ●标签名与属性之间必须以空格隔开
- ●属性之间没有顺序之分

②图片标签 src 属性

属性名：src

属性值：目标图片的路径

注意点：

当前网页和目标图片在同一个文件夹中，路径直接写目标图片名字即可（包括后缀名）

路径的情况有很多，详细介绍在后面

```
<img src="./1.jpg" alt="" title="">
```

③图片标签 alt 属性

属性名：alt

属性值：替换文本

当前图片加载失败时，才显示 alt 的文本

当前图片加载成功时，不会显示 alt 的文本

```
<img src="./错的图片路径会加载失败" alt="我是替换文本（失败时显示）">
```

④图片标签 title 属性

属性名：title

属性值：提示文本

当鼠标悬停时，才显示的文本

注意点：title 属性不仅仅可以用于图片标签，还可以用于其他标签

```
<img src="./1.jpg" alt="我是替换文本" title="我是 title 的效果">
```

⑤图片标签的 width 和 height 属性

属性名：width 和 height

属性值：宽度和高度（数字）

注意点：

如果只设置 width 或 height 中的一个，另一个没设置的就会自动等比例缩放（此时图片不会变形）

```
<img src="./1.jpg" alt="我是替换文本" title="我是 title 的效果" width="200">
```

如果同时设置了 width 和 height 两个，若设置不当此时图片可能会变形

```
<img src="./1.jpg" alt="我是替换文本" title="我是 title 的效果" width="200"
 height="100">
```

2）路径

①路径的介绍
场景：页面需要加载图片，需要先找到对应的图片
路径可分为：绝对路径、相对路径
②绝对路径
绝对路径以 Web 站点根目录为参考基础的目录路径。之所以称为绝对，意指当所有网页引用同一个文件时，所使用的路径都是一样的。
分为两种情况：物理绝对路径、网络绝对路径
物理绝对路径：指电脑磁盘的真实位置，如：“D:/wwwroot/images/pic.jpg”，一般在现实项目开发中不会使用此方法。

```
<img src="D:/wwwroot/images/pic.jpg" />
```

网络（URL）绝对路径：指网络中的完整的地址，以协议（http 或 https）开头，例如：“http://www.xxx.cn/images/pic.jpg”，表示网站根目录下的 images 文件夹中的 pic.jpg 文件，可以通过完整的 URL 引用图片或文件。

```
<img src="http://www.xxx.cn/images/pic.jpg" />
```

③相对路径
相对路径是以当前页面为参照物，指向目标文件的路径。相对路径适合于在同一个根目录下的文件引用，无需完整的 URL，可以省略部分路径。

相对路径示例：

相对于当前目录的路径：例如，./表示当前目录，../表示父目录，../../表示父目录的父目录，以此类推。
相对于根目录的路径：以“/”开头，例如，/images/pic.jpg 表示根目录下的 images 文件夹中的 pic.jpg 文件。
相对于同级目录下的路径：没有“/”开头，例如，images/test.jpg 表示和当前页面同级的目录下的 images 文件夹中的 pic.jpg 文件。

3）音频标签

场景：在页面中插入音频
代码：

```
<audio src="./音频文件名称.mp3" controls></audio>
```

常见属性：

属性名	功能
src	音频的路径
controls	显示播放的控件
autoplay	自动播放（部分浏览器不支持）
loop	循环播放

注意点：音频标签目前支持三种格式：MP3、Wav、Ogg

4）视频标签

场景：在页面中插入视频
代码：

```
<video src="./视频文件名称.mp4" controls></video>
```

常见属性：

属性名	功能
src	视频的路径
controls	显示播放的控件
autoplay	自动播放（谷歌浏览器中需要配合 muted 实现静音播放）

注意点：视频标签目前支持三种格式：MP4、WebM、Ogg

5）链接标签

①链接标签的介绍
场景：点击之后，从一个页面跳转到另一个页面
称呼：a 标签、超链接、锚链接
代码：

```
<a href="./目标网页.html">超链接</a>
```

特点：
双标签，内部可以包裹内容
如果需要 a 标签点击之后去指定页面，需要设置 a 标签的 href 属性
②链接标签的 href 属性
属性名：href
属性值：点击之后跳转去哪一个网页（目标网页的路径）
外部链接：

```
<a href="https://www.baidu.com/">百度一下</a>
```

内部链接：

```
<a href="./目标网页.html">超链接</a>
```

③链接标签的 target 属性
属性名：target
属性值：目标网页的打开方式

取值	效果
_self	默认值，在当前窗口中跳转（覆盖原网页）
_blank	在新窗口中跳转（保留原网页）

```
<a href="https://www.baidu.com/" target="_blank">百度一下</a>
```

6）锚链接

代码：

```
<!-- 创建命名锚点，给该标签一个 id -->
<a name="锚点 ID">我是锚点</a>

<!-- 链接到命名锚点，其 href 属性的值为#命名锚点的 id 值 -->
<a href="#锚点 ID">显示的内容</a>

<!-- #后面没有锚点 ID 的，为一个空链接 -->
<a href="#">显示的内容</a>
```

功能：

锚链接点击后会跳转到锚点的位置。
在开发中不确定该链的，用空链接，点击之后回到网页顶部。

7）注释标签

代码：

```
<!-- 我是注释内容 -->
<!--[if IE]>这里是正常的 html 代码<![endif]-->
```

功能：
普通注释就是对代码的解释和说明，其目的是让人们能够更加轻松地了解代码；注释是编写程序时，写程序的人给一个语句、程序段、函数等的解释或提示，能提高程序代码的可读性。
if 条件注释，让不同浏览器读取所能识别条件内容。

8）语义标签

语义 Semantics 指的是一段代码的含义。优秀的代码不是为了实现什么样子，而是为了达到什么作用。比如<b>和标签，两者都有加粗效果。前者只为了加粗，而后者有强调文本重要性。以下是一些常见的语义元素：

```
<head> HTML 的基本结构，网页头部，不可见内容
<body> HTML 的基本结构，网页主体，可见内容
<header> 网页主体的头部，比如标题、Logo、搜索
<footer> 网页主体的页脚，比如版权、作者、链接
<nav> 导航栏，菜单位置
<main>是<body>的主体部分，内容应该独一无二
<article> 独立结构，如文章、帖子，可重复使用
<aside> 附加内容，如相关文章、侧边栏
<section> 一个通用独立章节，通常会包含一个标题
<div> 只是分段，没有语义，使用 div+css 做前端开发
```

9）标签罗列

按功能类别排列 HTML 标签
基础标签

标签	描述
<!DOCTYPE>	定义文档类型。
<html>	定义 HTML 文档。

<head>	定义关于文档的信息。
<title>	定义文档的标题。
<body>	定义文档的主体。
<h1> to <h6>	定义 HTML 标题。
<p>	定义段落。
 	定义简单的折行。
<hr>	定义水平线。
<!--...-->	定义注释。

格式化标签

标签	描述
<acronym>	定义只取首字母的缩写。HTML5 中不支持。请使用<abbr>代替。
<abbr>	定义缩写。
<address>	定义文档作者或拥有者的联系信息。
<b>	定义粗体文本。
<bdi>	定义文本的文本方向，使其脱离其周围文本的方向设置。
<bdo>	定义文字方向。
<big>	定义大号文本。HTML5 中不支持。请使用 CSS 代替。
<blockquote>	定义长的引用。
	定义大号文本。HTML5 中不支持。请使用 CSS 代替。
<cite>	定义引用(citation)。
<code>	定义计算机代码文本。
<del>	定义被删除文本。
<dfn>	定义定义项目。
	定义强调文本。
<font>	定义大号文本。HTML5 中不支持。请使用 CSS 代替。
<i>	定义斜体文本。
<ins>	定义被插入文本。
<kbd>	定义键盘文本。
<mark>	定义有记号的文本。
<meter>	定义预定义范围内的度量。
<pre>	定义预格式文本。
<progress>	定义任何类型的任务的进度。
<q>	定义短的引用。
<rp>	定义若浏览器不支持 ruby 元素显示的内容。
<rt>	定义 ruby 注释的解释。
<ruby>	定义 ruby 注释。

<s>	定义加删除线的文本。
<samp>	定义计算机代码样本。
<small>	定义小号文本。
<strike>	定义加删除线文本。HTML5 中不支持。请使用<del>或<s>代替。
	定义语气更为强烈的强调文本。
<sup>	定义上标文本。
<sub>	定义下标文本。
<template>	定义用作容纳页面加载时隐藏内容的容器。
<time>	定义日期/时间。
<tt>	定义打字机文本。HTML5 中不支持。请使用 CSS 代替。
<u>	定义下划线文本。
<var>	定义文本的变量部分。
<wbr>	定义可能的换行符。

表单和输入标签

标签	描述
<form>	定义供用户输入的 HTML 表单。
<input>	定义输入控件。
<textarea>	定义多行的文本输入控件。
<button>	定义按钮。
<select>	定义选择列表（下拉列表）。
<optgroup>	定义选择列表中相关选项的组合。
<option>	定义选择列表中的选项。
<label>	定义 input 元素的标注。
<fieldset>	定义围绕表单中元素的边框。
<legend>	定义 fieldset 元素的标题。
<isindex>	定义与文档相关的可搜索索引。HTML5 中不支持。
<datalist>	定义下拉列表。
<keygen>	定义生成密钥。
<output>	定义输出的一些类型。

框架

标签	描述
<frame>	定义框架集的窗口或框架。HTML5 中不支持。
<frameset>	定义框架集。HTML5 中不支持。
<noframes>	定义针对不支持框架的用户的替代内容。HTML5 中不支持。
<iframe>	定义内联框架。

图像

标签	描述
<img>	定义图像。
<map>	定义图像映射。
<area>	定义图像地图内部的区域。
<canvas>	定义图形。
<figcaption>	定义 figure 元素的标题。
<figure>	定义媒介内容的分组，以及它们的标题。
<svg>	定义 SVG 图形的容器。

音频/视频

标签	描述
<audio>	定义声音内容。
<source>	定义媒介源。
<track>	定义用在媒体播放器中的文本轨道。
<video>	定义视频。

链接

标签	描述
<a>	定义锚。
<link>	定义文档与外部资源的关系。
<nav>	定义导航链接。

列表

标签	描述
	定义无序列表。
	定义有序列表。
	定义列表的项目。
<dir>	定义大号文本。HTML5 中不支持。请使用 CSS 代替。
<dl>	定义定义列表。
<dt>	定义定义列表中的项目。
<dd>	定义定义列表中项目的描述。
<menu>	定义命令的菜单/列表。
<menuitem>	定义用户可以从弹出菜单调用的命令/菜单项目。
<command>	定义命令按钮。

表格

标签	描述
<table>	定义表格

<caption>	定义表格标题。
<th>	定义表格中的表头单元格。
<tr>	定义表格中的行。
<td>	定义表格中的单元。
<thead>	定义表格中的表头内容。
<tbody>	定义表格中的主体内容。
<tfoot>	定义表格中的表注内容（脚注）。
<col>	定义表格中一个或多个列的属性值。
<colgroup>	定义表格中供格式化的列组。

样式和语义

标签	描述
<style>	定义文档的样式信息。
	定义文档中的节。
	定义文档中的节。
<header>	定义 section 或 page 的页眉。
<footer>	定义 section 或 page 的页脚。
<main>	定义文档的主要内容。
<section>	定义 section。
<article>	定义文章。
<aside>	定义页面内容之外的内容。
<details>	定义元素的细节。
<dialog>	定义对话框或窗口。
<summary>	为<details>元素定义可见的标题。
<data>	添加给定内容的机器可读翻译。

元信息

标签	描述
<head>	定义关于文档的信息。
<meta>	定义关于 HTML 文档的元信息。
<base>	定义页面中所有链接的默认地址或默认目标。
<basefont>	定义页面中文本的默认字体、颜色或尺寸。HTML5 中不支持。请使用 CSS 代替。

编程

标签	描述
<script>	定义客户端脚本。
<noscript>	定义针对不支持客户端脚本的用户的替代内容。
<applet>	定义嵌入的 applet。HTML5 中不支持。请使用<embed>和<object>代替。

<embed>	为外部应用程序（非 HTML）定义容器。
<object>	定义嵌入的对象。
<param>	定义对象的参数。

二、CSS 层叠样式表

1. 什么是 CSS

CSS，即层叠样式表 Cascading Style Sheets，和 HTML 一样，它也不是编程语言。

CSS 是一门样式表语言，为标记语言 HTML 元素添加样式。比如颜色、大小、位置、浮动方式等。

HTML 文件的扩展名是.html。主页默认名称为 index.html。

CSS 文件的扩展名是.css。名称没有限制，但通常命名为 style.css。

2. 如何添加 CSS 到网页中

在 HTML 添加 CSS 样式有三种方法：内嵌 Inline，内部 Internal，和外部 External。

（1）内嵌样式

内嵌样式 Inline CSS 直接在 HTML 代码，在标签里加入 style=""属性。

```
<h1 style="color:red;">内嵌样式</h1>
```

●优点：快速在 HTML 添加样式，适合测试或快速修复，不需要另外上传 CSS 文件。

●缺点：给每个元素添加 CSS 会花费很多时间，而且让 HTML 网页变得很臃肿，增加网页大小，也不能发挥 CSS 的优势，所以尽可能避免使用内嵌方式。

（2）内部样式

内部样式表 Internal CSS 在 HTML 的<head></head>之间添加<style></style>元素来放置样式代码。

```
<!-- 给 h1 标签字体颜色设置为红色 -->
<head>
   <style>
       h1{
```

```
            color: #ff0000;
        }
    </style>
</head>
```

●优点：适合单页面网站。不需要另外上传 CSS 文件。

●缺点：增加网页大小，影响网站加载速度。多页面网站还需要在每个页面添加 CSS 代码。

（3）外部样式

外部样式表 External CSS 全部 CSS 代码放在一个文件，然后在 HTML 的<head>里面使用代码把 CSS 文件引入 HTML 文件里。

```
<head>
    <link rel="stylesheet" href="style.css" />
</head>
```

●优点：可以在多页面使用一样的 CSS 文件。不影响 HTML 文件大小和内容。

●缺点：CSS 文件可能会增加网站加载速度需要加载完 CSS 文件后，网站才能显示出样式。

3. CSS 语法规则

CSS 规则由两个部分组成：选择器和声明，如图 4-1-1 所示。

图 4-1-1 CSS 语法规则组成

选择器：可以是 HTML 标签名，或标签的 ID，或类名 Class 等，类名可以自定义。

声明：是大括号{}里的是声明，声明由一条或多条“属性”和“值”组成，多个声明之

间使用英文分号隔开。

一个选择器或多个选择器可以共同使用一个声明，选择器之间使用英文逗号分开。

4. CSS 选择器

（1）基础选择器

1） 标签选择器

规则：标签名{ 属性: 值 ;}
作用：对所有使用该标签的元素设置样式

```
p{
    color: #000000;
}
```

2） 类名选择器

规则：类名{ 属性: 值 ;}
作用：对所有使用该类名的标签设置样式

```
.title{
    color: #000000;
}
```

说明：在 HTML 使用中类名可以重复使用，并且一个标签可以同时使用多个 class 选择器，多个 class 名之间需要用控股隔开。

```
<p class="title"></p>
<p class="title item"></p>
```

3） ID 选择器

规则：#ID{ 属性: 值 ;}

作用：对所有使用该 ID 的标签设置样式
说明：在一个 HTML 页面中元素的 ID 只能有一个不能重复。

```
#title{
    color: #000000;
}
```

4) 全局选择器

规则：*{ 属性: 值 ;}
作用：对页面所有标签设置样式
说明：各个浏览器都有默认的样式，使用全局选择器可以统一网站的样式。

```
*{
    margin: 0;
    Padding: 0;
}
```

5) 复合选择器

①群组选择器
规则：选择器 1,选择器 **2,**选择器 3{ 属性: 值 ;}
作用：对所有选择器设置样式
说明：使用群组选择器可以精简代码，多个选择器可以放在一行或多行。

```
h1,h2,h3,.class{
    color: #000000;
}
```

②属性选择器
规则：标签名[属性=值]{ 属性: 值 ;}
作用：对元素的属性和值设置样式

```
/* 链接中有“ex”这两个字母的都会有样式 */
a[href*=“ex”]{
```

```
    color: #ff0000;
}
```

6) 组合选择器

①后代选择器

规则：选择器 1 选择器 2{ 属性: 值 ;}
作用：对选择器 1 下的所有选择器 2 设置样式
说明：选择器之间用空格隔开

```
div p{
    color: #ff0000;
}
```

②子代选择器

规则：选择器 1>选择器 2{ 属性: 值 ;}
作用：对选择器 1 直接对应的选择器 2 设置样式

```
div > p{
    color: #ff0000;
}
```

③相邻兄弟组合器

规则：选择器 1 + 选择器 2{ 属性: 值 ;}
作用：对选择器 1 同级的下一个选择器 2 设置样式

```
div + p{
    color: #ff0000;
}
```

④通用兄弟组合器

规则：选择器 1 ~ 选择器 2{ 属性: 值 ;}
作用：对选择器 1 同级的全部选择器 2 设置样式

```
div ~ p{
    color: #ff0000;
}
```

7) 伪类选择器

①链接伪类选择器

规则：选择器:伪类{ 属性: 值 ;}
作用：对选择器的某个状态设置样式

```
a:link{ color: #000000; }          /* 为访问的链接样式 */
a:visited{ color: #7159bf; }       /* 已经访问的链接样式 */
a:hove{ color: #0000ff; }          /* 鼠标划过链接样式 */
a:active{ color: #b46262; }        /* 已经选中的链接样式 */
```

②结构伪类选择器

规则：选择器:伪类{ 属性: 值 ;}
作用：对选择器的第 x 个元素设置样式
说明：p 标签必须是父元素，它包含的一组标签元素，这一组元素之间不能有其他元素

```
p:first-child{ color: #ff0000; }          /* 一组元素的第一个元素 */
p:nth-child(3){ color: #7159bf; }        /* 一组元素的第 3 个元素 */
p:last-child{ color: #0000ff; }           /* 一组元素的最后一个元素 */
```

8) 伪元素选择器

①::before 伪元素

规则：选择器::before{ 属性: 值 ;}
作用：在元素的内容前面插入内容

```
p::before{
```

```
        content: "警告";
        background-color: #ff0000;
    }
```

②::first-line 伪元素

规则：选择器::first-line{ 属性: 值 ;}

作用：在文本的第一行设置样式

```
p::first-line{
    color: #ff0000;
}
```

9) 选择器的优先级

CSS 选择器优先级从高到低排列如下：

a. !important 声明。

b. 带有!important 声明的属性具有最高优先级。

c. 内联样式：在 HTML 元素的 style 属性中指定的样式具有第二高的优先级。

d. ID 选择器：ID 选择器具有比属性选择器更高的优先级。

e. 类选择器、属性选择器和伪类选择器：这些选择器的优先级相同。

f. 元素选择器和伪元素选择器：这些选择器的优先级相同。

g. 通配符选择器和子选择器：这些选择器的优先级相同，但通常通配符选择器不会被使用。

h. 相邻兄弟选择器和通用兄弟选择器：这些选择器的优先级相同。

当两个样式规则发生冲突时，优先级较高的规则将覆盖优先级较低的规则。在优先级相同时，使用就近原则，即找到最后出现的样式规则。但是，需要注意的是，继承得来的属性具有最低的优先级。

5. CSS 盒子模型 Box Model

（1）什么是盒子模型

盒子模型是一种思维模型，页面中的每一个标签，都可看做是一个“盒子”，通过盒子的视角更方便的进行布局。浏览器在渲染（显示）网页时，会将网页中的元素看做是一个个的矩形区域，我们也形象地称之为盒子。

(2) 盒子模型的组成

CSS 中规定每个盒子分别由：内容区域（content）、内边距区域（padding）、边框区域（border）、外边距区域（margin）构成，这就是盒子模型，如图 4-1-2 所示。

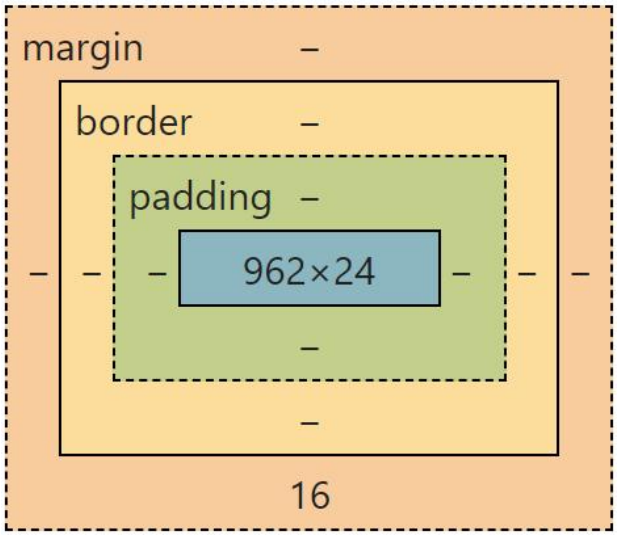


图 4-1-2 盒子模式组成

(3) 内容区域（content)

内容区是盒子模型的中心，它呈现了盒子的主要信息内容，这些内容可以是文本、图片等多种类型。

内容区有三个属性，width、height 和 overflow。使用 width 和 height 属性可以指定盒子内容区的高度和宽度，当内容信息太多，超出内容区所占范围时，可以使用 overflow 溢出属性来指定处理方法。当 overflow 属性值为 hidden 时，溢出部分将不可见；为 visible 时，溢出的内容信息可见，只是被呈现在盒子的外部；当为 scroll 时，滚动条将被自动添加到盒子中，用户可以通过拉动滚动条显示内容信息；当为 auto 时，将由浏览器决定如何处理溢出部分。

(4) 内边距区域（padding)

填充是内容区和边框之间的空间。填充的属性有五种，即 padding-top、padding-bottom、padding-left、padding-right 以及综合了以上四种方向的快捷填充属性 padding。

使用这五种属性可以指定内容区信息内容与各方向边框间的距离。设置盒子背景色属性时，可使背景色延伸到填充区域。

padding 属性举例如下：

取值	示例	含义
一个值	padding: 10px;	上右下左都设置为 10px

两个值	padding: 10px 20px;	上下设置为 10px、左右设置为 20px
三个值	padding: 10px 20px 30px;	上设置为 10px、左右设置为 20px、下设置为 30px
四个值	padding: 10px 20px 30px 40px;	上设置为 10px、右设置为 20px、下设置为 30px、左设置为 40px

(5) 边框区域（border)

边框是环绕内容区和填充的边界。边框的属性有 border-style、border-width 和 border-color 以及综合了以上三类属性的快捷边框属性 border。

border-style 属性是边框最重要的属性，如果没有指定边框样式，其他的边框属性都会被忽略，边框将不存在。CSS 规定了 dotted（点线）、dashed（虚线）、solid(实线)等九种边框样式。使用 border-width 属性可以指定边框的宽度，其属性值可以是长度计量值，也可以是 CSS 规定的 thin、medium 和 thick。使用 border-color 属性可以为边框指定相应的颜色，其属性值可以是 RGB 值，也可以是 CSS 规定的 17 个颜色名。在设定以上三种边框属性时，既可以进行边框四个方向整体的快捷设置，也可以进行四个方向的专向设置，如 border: 2px solid green 或 border-top-style: solid、border-left-color: red 等。设置盒子背景色属性时，在 IE 中背景不会延伸到边框区域，但在 FF 等标准浏览器中，背景颜色可以延伸到边框区域，特别是单边框设置为点线或虚线时能看到效果。

(6) 外边距区域（margin)

空白边位于盒子的最外围，是添加在边框外周围的空间。空白边使盒子之间不会紧凑地连接在一起，是 CSS 布局的一个重要手段。空白边的属性有五种，即 margin-top、margin-bottom、margin-left、margin-right 以及综合了以上四种方向的快捷空白边属性 margin，其具体的设置和使用与内边距属性类似。

对于两个相邻的（水平或垂直方向）且设置有空白边值的盒子，他们邻近部分的空白边将不是二者空白边的相加，而是二者的并集。若二者邻近的空白边值大小不等，则取二者中较大的值。同时，CSS 容许给空白边属性指定负数值，当指定负空白边值时，整个盒子将向指定负值方向的相反方向移动，以此可以产生盒子的重叠效果。采用指定空白边正负值的方法可以移动网页中的元素，这是 CSS 布局技术中的一个重要方法。

6. CSS 的 Position 定位属性

position 定位属性用来指定元素位置，一共有 5 种定位类型:static 默认位置，relative 相对定位，absolute 绝对定位，fixed 固定位置和 sticky 粘性定位。

(1) static 默认位置

static 其实是 position 的默认值，就是当没有写 position 属性的时候，浏览器默认的定位是 position: static，这个时候元素还没有脱离文档流，元素与元素还没有产生重叠，元素所在的位置就是按照布局默认的位置。

(2) relative 相对定位

relative 是相对定位，所谓的相对定位是相对于自身默认所在的位置进行偏移。

下面的代码.box 元素距离默认位置 30px，即向左偏移 30px。

```
.box{
    position:relative;
    left:30px;
}
```

(3) absolute 绝对定位

absolute 是绝对定位，是相对于已定位的父元素进行定位（即 position 属性不是 static 的元素），如果父元素都没有定位的话，会一直往上找，直到父元素定位属性不是 static 的元素，如果一直找不到，就相对于 body 定位。所以这就有了为什么我们一般给父元素写个 relative 的原因，其实父元素写别的定位属性也行，例如 fixed、sticky、absolute，但是如果父元素是 absolute，父元素就要继续找它已定位的父元素，不建议这样写，不太符合布局思想。

下面的代码就是相对父元素偏移的基础再偏移 30px。

```
<div class="parent">
    <div class="child"></div>
</div>

.parent{
    position: relative;
    left: 30px;
}
.child{
    position: absolute;
    left: 30px;
}
```

(4) fixed 固定位置

fixed 是固定定位，定位基点是浏览器窗口，这会导致元素的位置不随页面滚动而变化，好像固定在网页上一样。注意 position 也可以相对于 fixed 的父元素定的。

下面代码中，元素始终距离浏览器窗口顶部 50px，不随网页滚动而变化。

```
.box{
```

```
    position:fixed;
    top:50px;
}
```

(5) sticky 粘性定位

sticky 跟前面四个属性值都不一样，它会产生动态效果，很像 relative 和 fixed 的结合：一些时候是 relative 定位（定位基点是自身默认位置），另一些时候自动变成 fixed 定位（定位基点是浏览器窗口）。

一般浏览商城网页的时候，你经常看到搜索框离开浏览器窗口的就变成了固定定位，等到页面重新向上滚动回到原位，搜索框也会回到默认位置。

生效的具体规则是，当页面滚动，父元素开始脱离浏览器窗口时（即部分不可见），只要与 sticky 元素的距离达到生效门槛，relative 定位自动切换为 fixed 定位；等到父元素完全脱离浏览器窗口时（即完全不可见），fixed 定位自动切换回 relative 定位。

sticky 生效的前提是必须搭配 top、bottom、left、right，这四个属性其中只是一个一起使用，不能省略，否则等同于 relative 定位，不产生“动态固定”的效果。原因是这四个属性用来定义“偏移距离”，浏览器把它当作 sticky 的生效门槛。

目前的浏览器基本都已支持，但是，Safari 浏览器需要加上浏览器前缀-webkit-。

```
#toolbar {
  position: -webkit-sticky; /* safari 浏览器 */
  position: sticky; /* 其他浏览器 */
  top: 20px;
}
```

7. CSS 的 Float 浮动属性

float 属性只有两个值 left 和 right，所以 float 只会使元素向左浮动或向右浮动，其周围的元素也会重新排列。

(1) 使用 float 的元素会怎么浮动

元素的水平方向浮动，意味着元素只能左右移动而不能上下移动。

一个浮动元素会尽量向左或向右移动，直到它的外边缘碰到包含框或另一个浮动框的边框为止。

浮动元素之后的元素将围绕它，浮动元素之前的元素将不会受到影响。

如果图像是右浮动，下面的文本流将环绕在它左边：

```
```

```
img{
  float:right;
}
```

(2) 彼此相邻的浮动元素怎么浮动

如果你把几个浮动的元素放到一起，如果有空间的话，它们将彼此相邻。

在这里，我们对图片廊使用 float 属性：

```
.thumbnail{
    float:left;
    width:110px;
    height:90px;
    margin:5px;
}
```

(3) 清除浮动 clear

元素浮动之后，周围的元素会重新排列，为了避免这种情况，使用 clear 属性。

clear 属性指定元素两侧不能出现浮动元素。

使用 clear 属性往文本中添加图片廊：

```
.text_line{
    clear:both;
}
```

8. Flex 弹性布局

(1) 什么是 Flex

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称“容器”。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称“项目”。

容器默认存在两根轴，分别为水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置叫做 main start，结束位置叫做 main end；交叉轴的开始位置叫做 cross start，结束位置叫做 cross end。项目默认沿主轴排列。单个项目占据的主轴空间叫做 main size，占据的交叉轴空间叫做 cross size，如下图 4-1-3 所示。

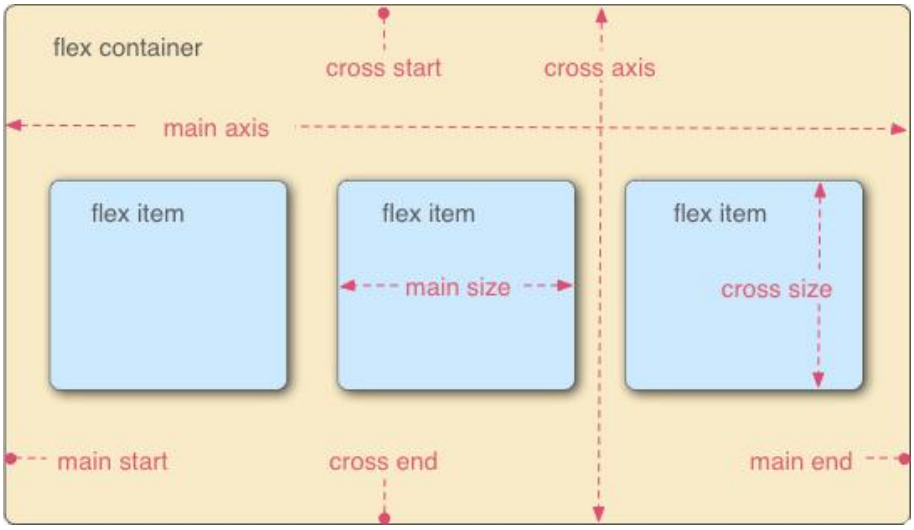


图 4-1-3 Flex 容器

提示：您可以通过将元素的 display 属性设置为 flex（生成块级 flex 容器）或 inline-flex（生成类似 inline-block 的行内块级 flex 容器）。当一个元素设置了 Flex 布局以后，其子元素的 float、clear 和 vertical-align 等属性将失效。

CSS 中提供了以下属性来实现 Flex 布局：

属性	描述
display	指定 HTML 元素的盒子类型
flex-direction	指定弹性盒子中子元素的排列方式
flex-wrap	设置当弹性盒子的子元素超出父容器时是否换行
flex-flow	flex-direction 和 flex-wrap 两个属性的简写
justify-content	设置弹性盒子中元素在主轴（横轴）方向上的对齐方式
align-items	设置弹性盒子中元素在侧轴（纵轴）方向上的对齐方式
align-content	修改 flex-wrap 属性的行为，类似 align-items，但不是设置子元素对齐，而是设置行对齐
order	设置弹性盒子中子元素的排列顺序
align-self	在弹性盒子的子元素上使用，用来覆盖容器的 align-items 属性
flex	设置弹性盒子中子元素如何分配空间
flex-grow	设置弹性盒子的扩展比率
flex-shrink	设置弹性盒子的收缩比率
flex-basis	设置弹性盒子伸缩基准值

按照作用范围的不同，这些属性可以分为两类，分别为容器属性（flex-direction、flex-wrap、

flex-flow、justify-content、align-items、align-content）和项目属性（order、align-self、flex、flex-grow、flex-shrink、flex-basis）。

(2) 容器属性

1) flex-direction

flex-direction 属性用来决定主轴的方向（即项目的排列方向），属性的可选值如下：

值	描述
row	默认值，主轴沿水平方向从左到右
row-reverse	主轴沿水平方向从右到左
column	主轴沿垂直方向从上到下
column-reverse	主轴沿垂直方向从下到上
initial	将此属性设置为属性的默认值
inherit	从父元素继承此属性的值

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        #main {
            border: 1px solid #CCC;
            padding: 5px;
            position: relative;
        }
        .row, .row_reverse, .column, .column_reverse{
            display: flex;
            margin-bottom: 5px;
        }
        .row {
            flex-direction: row;
        }
        .row_reverse {
```

```
            flex-direction: row-reverse;
        }
        .column {
            flex-direction: column;
        }
        .column_reverse {
            flex-direction: column-reverse;
            position: absolute;
            top: 120px;
            left: 400px;
        }
        .row div, .row_reverse div, .column div, .column_reverse div {
            width: 50px;
            height: 50px;
            border: 1px solid black;
        }
    </style>
</head>
<body>
    <div id="main">
        <div class="row">
            <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
        </div>
        <div class="row_reverse">
            <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
        </div>
        <div class="column">
            <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
        </div>
        <div class="column_reverse">
            <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
        </div>
    </div>
</body>
</html>
```

运行结果如图 4-1-4 所示。

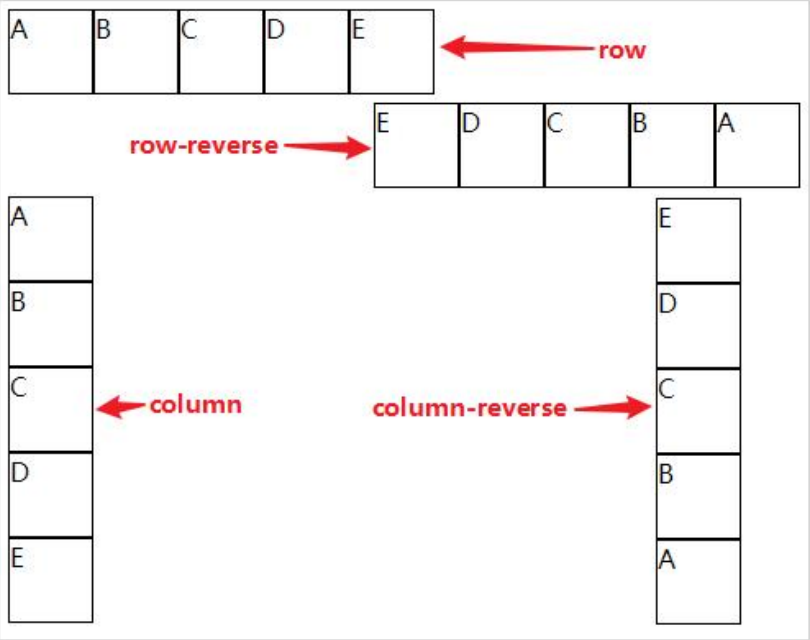


图 4-1-4 flex-direction 属性运行效果

2) flex-wrap

flex-wrap 属性用来设置当弹性盒子的子元素（项目）超出父容器时是否换行，属性的可选值如下：

值	描述
nowrap	默认值，表示项目不会换行
wrap	表示项目会在需要时换行
wrap-reverse	表示项目会在需要时换行，但会以相反的顺序
initial	将此属性设置为属性的默认值
inherit	从父元素继承属性的值

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        #main {
            border: 1px solid #CCC;
            padding: 5px;
        }
```

```
            .nowrap, .wrap, .wrap_reverse {
                display: flex;
                flex-direction: row;
                margin-bottom: 15px;
            }
            .nowrap {
                flex-wrap: nowrap;
            }
            .wrap {
                flex-wrap: wrap;
            }
            .wrap_reverse {
                flex-wrap: wrap-reverse;
            }
            .nowrap div, .wrap div, .wrap_reverse div {
                width: 70px;
                height: 50px;
                border: 1px solid black;
            }
        </style>
    </head>
    <body>
        <div id="main">
            <div class="nowrap">

<div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><di
v>8</div><div>9</div><div>10</div>
            </div>
            <div class="wrap">

<div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><di
v>8</div><div>9</div><div>10</div>
            </div>
            <div class="wrap_reverse">

<div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><di
v>8</div><div>9</div><div>10</div>
            </div>
        </div>
    </body>
    </html>
```

运行结果如图 4-1-5 所示：

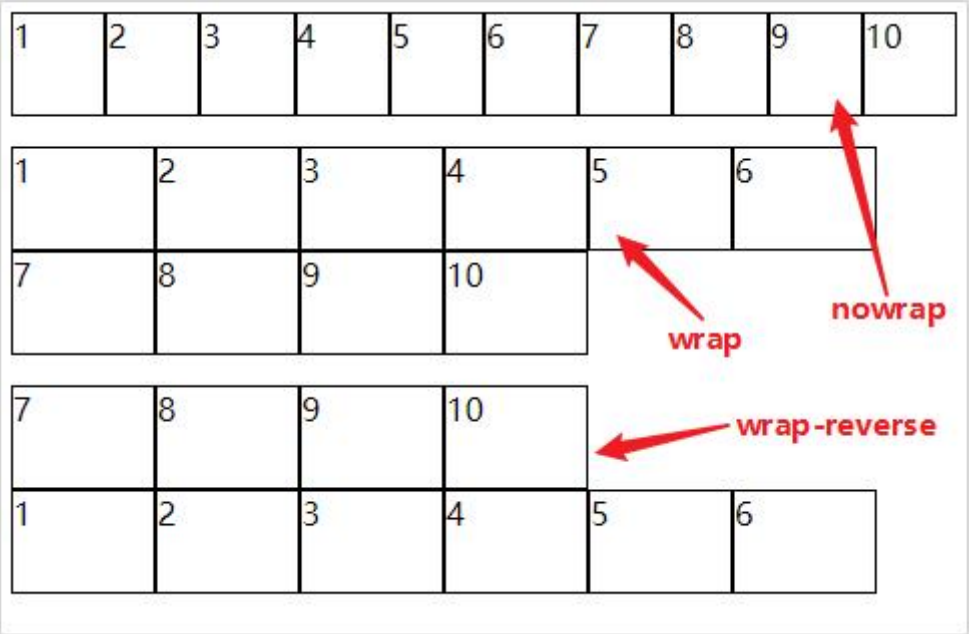


图 4-1-5 flex-wrap 属性运行效果

3) flex-flow

flex-flow 属性是 flex-direction 和 flex-wrap 两个属性的简写，语法格式如下：

```
flex-flow: flex-direction flex-wrap;
```

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .flex_flow {
            display: flex;
            flex-flow: row-reverse wrap;
        }
        .flex_flow div {
            width: 60px;
            height: 60px;
            margin-bottom: 5px;
            border: 1px solid black;
        }
    </style>
</head>
```

```
    <body>
        <div class="flex_flow">

<div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><di
v>8</div><div>9</div><div>10</div>
        </div>
    </body>
    </html>
```

运行结果如图 4-1-6 所示：

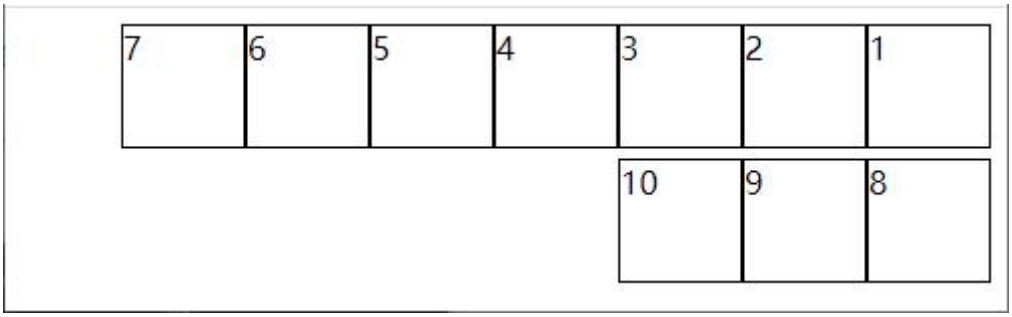


图 4-1-6 flex-flow 属性运行效果

4) justify-content

justify-content 属性用于设置弹性盒子中元素在主轴（横轴）方向上的对齐方式，属性的可选值如下：

值	描述
flex-start	默认值，左对齐
flex-end	右对齐
center	居中
space-between	两端对齐，项目之间的间隔是相等的
space-around	每个项目两侧的间隔相等
initial	将此属性设置为属性的默认值
inherit	从父元素继承属性的值

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .flex {
            display: flex;
            flex-flow: row wrap;
            margin-top: 10px;
        }
        .flex div {
            width: 60px;
            height: 60px;
            margin-bottom: 5px;
            border: 1px solid black;
        }
        .flex-start {
            justify-content: flex-start;
        }
        .flex-end {
            justify-content: flex-end;
        }
        .center {
            justify-content: center;
        }
        .space-between {
            justify-content: space-between;
        }
        .space-around  {
            justify-content: space-around;
        }
    </style>
</head>
<body>
    <div class="flex flex-start">
        <div>A</div><div>B</div><div>C</div><div>D</div>
    </div>
    <div class="flex flex-end">
        <div>A</div><div>B</div><div>C</div><div>D</div>
    </div>
    <div class="flex center">
        <div>A</div><div>B</div><div>C</div><div>D</div>
    </div>
    <div class="flex space-between">
        <div>A</div><div>B</div><div>C</div><div>D</div>
```

```
        </div>
        <div class="flex space-around">
            <div>A</div><div>B</div><div>C</div><div>D</div>
        </div>
    </body>
    </html>
```

运行结果如图 4-1-7 所示：

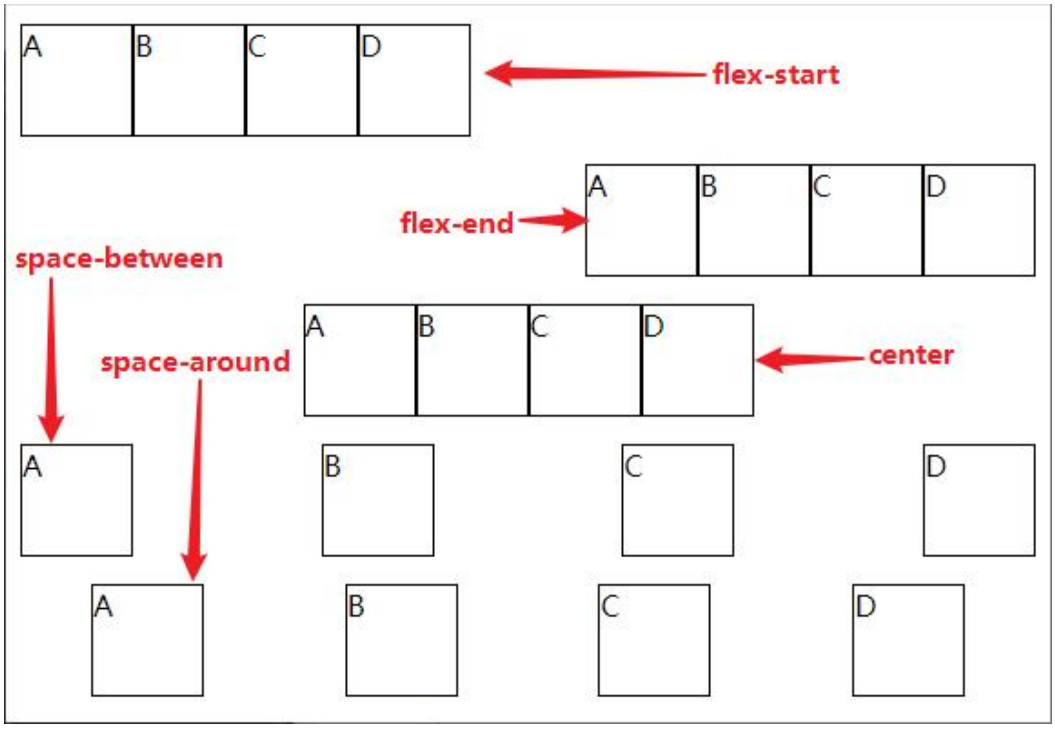


图 4-1-7 justify-content 属性运行效果

5) align-items

align-items 属性用来设置弹性盒子中元素在侧轴（纵轴）方向上的对齐方式，属性的可选值如下：

值	描述
stretch	默认值，项目将被拉伸以适合容器
center	项目位于容器的中央
flex-start	项目位于容器的顶部
flex-end	项目位于容器的底部
baseline	项目与容器的基线对齐

initial	将此属性设置为属性的默认值
inherit	从父元素继承属性的值

以上属性值效果如图 4-1-8 所示。

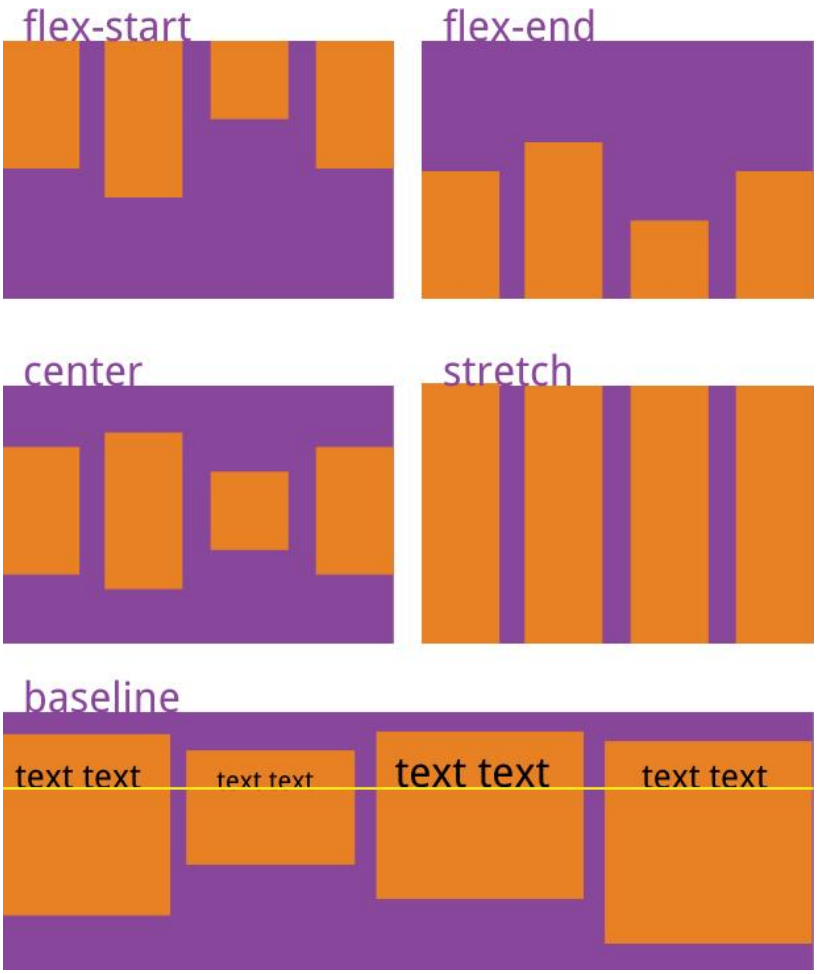


图 4-1-8 align-items 属性运行效果

6) align-content

align-content 属性与 justify-content 属性类似，可以在弹性盒子的侧轴还有多余空间时调整容器内项目的对齐方式，属性的可选值如下：

值	描述
stretch	默认值，将项目拉伸以占据剩余空间
center	项目在容器内居中排布
flex-start	项目在容器的顶部排列

flex-end	项目在容器的底部排列
space-between	多行项目均匀分布在容器中，其中第一行分布在容器的顶部，最后一行分布在容器的底部
space-around	多行项目均匀分布在容器中，并且每行的间距（包括离容器边缘的间距）都相等
initial	将此属性设置为属性的默认值
inherit	从父元素继承该属性的值

以上属性值效果如图 4-1-9 所示。

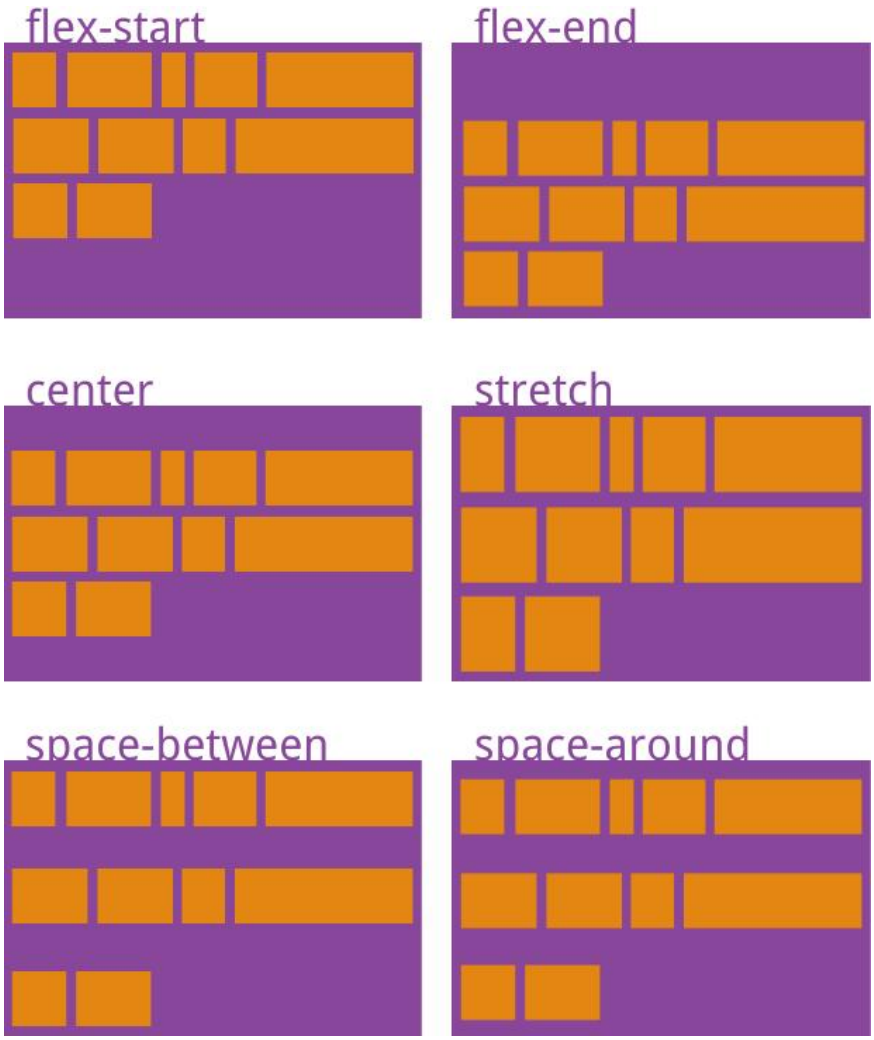


图 4-1-9 align-content 属性运行效果

(3) 项目属性

1) order

order 属性用来设置项目在容器中出现的顺序，您可以通过具体的数值来定义项目在容器中的位置，属性的语法格式如下：

```
order: number;
```

其中 number 就是项目在容器中的位置，默认值为 0。

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .flex {
            display: flex;
            flex-flow: row wrap;
            margin-top: 10px;
        }
        .flex div {
            width: 60px;
            height: 60px;
            margin-bottom: 5px;
            border: 1px solid black;
        }
        .flex div:nth-child(1) {
            order: 5;
        }
        .flex div:nth-child(2) {
            order: 3;
        }
        .flex div:nth-child(3) {
            order: 1;
        }
        .flex div:nth-child(4) {
            order: 2;
        }
        .flex div:nth-child(5) {
```

```
            order: 4;
        }
    </style>
</head>
<body>
    <div class="flex">
        <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
    </div>
</body>
</html>
```

运行结果如图 4-1-10 所示。

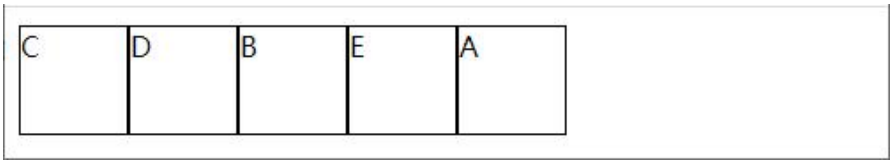


图 4-1-10 order 属性运行效果

2) align-self

align-self 属性允许您为某个项目设置不同于其他项目的对齐方式，该属性可以覆盖 align-items 属性的值。align-self 属性的可选值如下：

值	描述
auto	默认值，表示元素将继承其父容器的 align-items 属性值，如果没有父容器，则为“stretch”
stretch	项目将被拉伸以适合容器
center	项目位于容器的中央
flex-start	项目位于容器的顶部
flex-end	项目位于容器的底部
baseline	项目与容器的基线对齐
initial	将此属性设置为属性的默认值
inherit	从父元素继承属性的值

示例代码如下：

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <style>
        .flex {
            display: flex;
            flex-flow: row wrap;
            align-items: flex-end;
            border: 1px solid #CCC;
            height: 150px;
        }
        .flex div {
            width: 60px;
            height: 60px;
            border: 1px solid black;
        }
        .flex div:nth-child(4) {
            align-self: flex-start;
        }
    </style>
</head>
<body>
    <div class="flex">
        <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
    </div>
</body>
</html>
```

运行结果如图 4-1-11 所示：

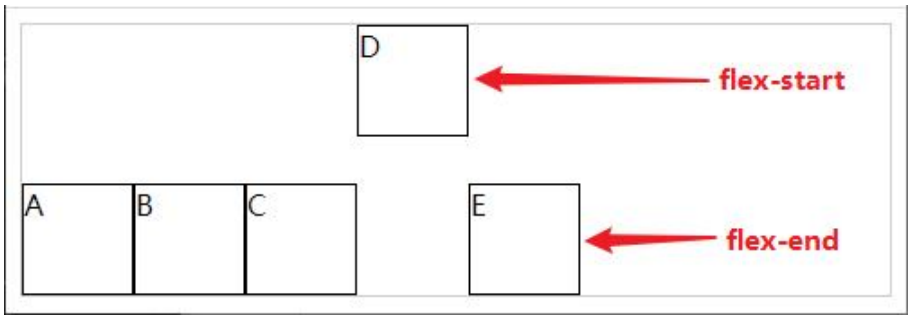


图 4-1-11 align-self 属性运行效果

3) flex

flex 属性是 flex-grow、flex-shrink 和 flex-basis 三个属性的简写，语法格式如下：

```
flex: flex-grow flex-shrink flex-basis;
```

参数说明如下：

flex-grow：（必填参数）一个数字，用来设置项目相对于其他项目的增长量，默认值为0；

flex-shrink：（选填参数）一个数字，用来设置项目相对于其他项目的收缩量，默认值为 1；

flex-basis：（选填参数）项目的长度，合法值为 auto（默认值，表示自动）、inherit（表示从父元素继承该属性的值） 或者以具体的值加“%”、“px”、“em”等单位的形式。

另外，flex 属性还有两个快捷值，分别为 auto（1 1 auto）和 none（0 0 auto）。

示例代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .flex {
            display: flex;
            flex-flow: row wrap;
            align-items: flex-end;
            border: 1px solid #CCC;
        }
        .flex div {
            width: 60px;
            height: 60px;
            border: 1px solid black;
        }
        .flex div:nth-child(2) {
            flex:0;
        }
        .flex div:nth-child(4) {
            flex:1 1 auto;
        }
    </style>
</head>
<body>
    <div class="flex">
        <div>A</div><div>B</div><div>C</div><div>D</div><div>E</div>
    </div>
</body>
</html>
```

运行结果如图 4-1-12 所示。

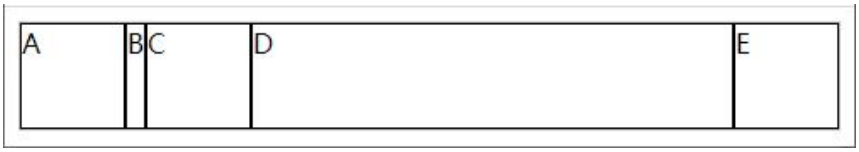


图 4-1-12 flex 属性运行效果

9. CSS 的水平/垂直居中方法

在日常开发中居中是经常需要使用的，下面说一下的几种方法。

（1）水平居中

1) text-align 使用该属性可以让文本水平居中

该属性只能让文本在水平方向居中。

```
div.title{
    text-align:center;
};
```

2) margin 使用该属性可以让块级元素水平居中

该属性能让块级元素在水平方向居中，但是需要注意的是要对块级元素指定宽度。

```
div.title{
    width:200px;
    margin:0 atuo;
};
```

3) flexbox 使用该属性可以让文本以及块级元素水平居中

```
div.title{
    display:flex;
    justify-content:center;
};
```

4) position+transform 这两个属性同时使用可以让块级元素水平居中

```
div.container{
    height: 100px;
    position: relative;
    background-color: #e7e7e7;
}
div.title{
    position: absolute;
    left:50%;
    transform: translate(-50%);
};
```

(2) 垂直居中

1) padding 该属性可以垂直居中

padding 该属性是控制元素内边距让元素内的内容垂直居中的效果，前提是不能指定元素的高度。

```
div.title{
    padding: 30px 0;
};
```

2) line-height 该属性可以文本垂直居中

line-height 该属性设置行高，文本该行内居中，该方法不好用，当文本过长产生换行时两行之间的间距很大。如果给元素设置同样的高度，此时只能显示一行内容。

```
div.title{
    height: 100px
    line-height: 100px 0;
};
```

3) flexbox 使用该属性可以让文本以及块级元素垂直居中

```
div.title{
    height: 100px;
    display:flex;
    align-items:center;
};
```

4) position+transform 这两个属性同时使用可以让块级元素垂直居中

```
div.container{
    height: 100px;
    position: relative;
    background-color: #e7e7e7;
}
div.title{
    position: absolute;
    top:50%;
    transform: translate(0, -50%);
};
```

(3) 水平和垂直同时居中

1) flexbox 使用该属性可以让文本以及块级元素水平垂直居中

```
div.container{
    height: 100px;
    background-color: #e7e7e7;
    display:flex;
    justify-content: center;
    align-items:center;
};
```

2) position+transform 这两个属性同时使用可以让块级元素水平垂直居中

```
div.container{
    height: 100px;
    position: relative;
    background-color: #e7e7e7;
}
```

```
div.title{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
};
```

三、Thymeleaf 模板引擎

1. 什么是 Thymeleaf

Thymeleaf 是面向 Web 和独立环境的现代服务器端 Java 模板引擎，能够处理 HTML，XML，JavaScript，CSS 甚至纯文本。

Thymeleaf 旨在提供一个优雅的、高度可维护的创建模板的方式。为了实现这一目标，Thymeleaf 建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板设计原型。这改善了设计的沟通，弥合了设计和开发团队之间的差距。

Thymeleaf 从设计之初就遵循 Web 标准，特别是 HTML5 标准，如果需要，Thymeleaf 允许您创建完全符合 HTML5 验证标准的模板。

Spring Boot 体系内推荐使用 Thymeleaf 作为前端页面模板，并且 Spring Boot 2.0 中默认使用 Thymeleaf 3.0，性能提升幅度很大。

2. 变量输出

Thymeleaf 通过在 html 标签中增加额外属性来达到“模板+数据”的展示方式，输出属性有：

属性	说明
th:text	在页面中输出值
th:value	可以将一个值放入到 input 标签的 value 中

表达式	说明	用途
变量取值	${...}	获取请求域、session 域、对象等值
选择变量	*{...}	获取上下文对象值
消息	#{...}	获取国际化等值
链接	@{...}	生成链接

片段表达式	~{...}	jsp:include 作用，引入公共页面片段

模板中代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Thymeleaf 的使用</title>
    </head>
    <body>
        <h1>变量内容输出</h1>
        <h2>th:text 输出</h2>
        <span th:text="${msg}"></span><br>
        <h2>th:value 输出</h2>
        <input type="text" th:value="${msg}"><br>
    </body>
    </html>
```

Controller 文件中代码如下：

```
        @RequestMapping("/t1")
        public String t1(Model model){
            model.addAttribute("msg","th:text 使用");
            return "thymeleafTest/t1";
        }
```

运行结果如图 4-1-13 所示。

图 4-1-13 运行效果

3. 文本处理

Thymeleaf 提供内置对象来对字符串进行处理、判断等，我们可以使用这些内置对象得到我们需要的处理结果。

调用内置对象一定要使用“#”，同时大部分内置对象都是以“s”结尾，比如：strings、numbers、dates 等等。

表达式	说明
${#strings.isEmpty(key)}	判断字符串是否为空，如果为空返回 true，否则返回 false
${#strings.contains(msg,'T')}	判断字符串是否包含指定的子串，如果包含返回 true，否则返回 false
${#strings.startsWith(msg,'a')}	判断当前字符串是否以子串开头，如果是返回 true，否则返回 false
${#strings.endsWith(msg,'a')}	判断当前字符串是否以子串结尾，如果是返回 true，否则返回 false
${#strings.length(msg)}	返回字符串的长度
${#strings.indexOf(msg,'h')}	查找子串的位置，并返回该子串的下标，如果没找到则返回-1
${#strings.substring(msg,13)}	截取字符串从指定索引开始到末尾的子串，功能与 JDK 中 String 类的 substring(int beginIndex) 方法相同
${#strings.substring(msg,13,15)}	截取子串，用户与 jdk String 类下 subString 方法相同
${#strings.toUpperCase(msg)}	字符串转大写
${#strings.toLowerCase(msg)}	字符串转小写

模板中代码如下：

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Thymeleaf 的使用</title>
</head>
<body>
    <h1>文本处理</h1>
    <h2>string 类型处理</h2>
    <span th:text="${#strings.isEmpty(msg)}"></span>
    <hr/>
    <span th:text="${#strings.contains(msg,'9')}"></span>
    <span th:text="${#strings.contains(msg,'t')}"></span>
    <hr/>
    <span th:text="${#strings.startsWith(s1,'a')}"></span>
    <span th:text="${#strings.startsWith(s1,'T')}"></span>
    <hr/>
    <span th:text="${#strings.endsWith(s1,'a')}"></span>
    <span th:text="${#strings.endsWith(s1,'g')}"></span>
    <hr/>
    <span th:text="${#strings.length(s1)}"></span>
    <hr/>
    <span th:text="${#strings.indexOf(s1,'b')}"></span>
    <hr/>
    <span th:text="${#strings.substring(s1,4)}"></span>
    <span th:text="${#strings.substring(s1,4,6)}"></span>
    <hr/>
    <span th:text="${#strings.toUpperCase(s1)}"></span>
    <span th:text="${#strings.toLowerCase(s2)}"></span>
    <hr/>
</body>
</html>
```

Controller 文件中代码如下：

```
@RequestMapping("/t2")
 public String t2(Model model){
     model.addAttribute("msg","th:text 使用");
     model.addAttribute("s1","abcdefg");
```

```
        model.addAttribute("s2","AbCdEfG");
        model.addAttribute("s3","abc123");
        return "thymeleafTest/t2";
    }
```

运行结果如图 4-1-14 所示。

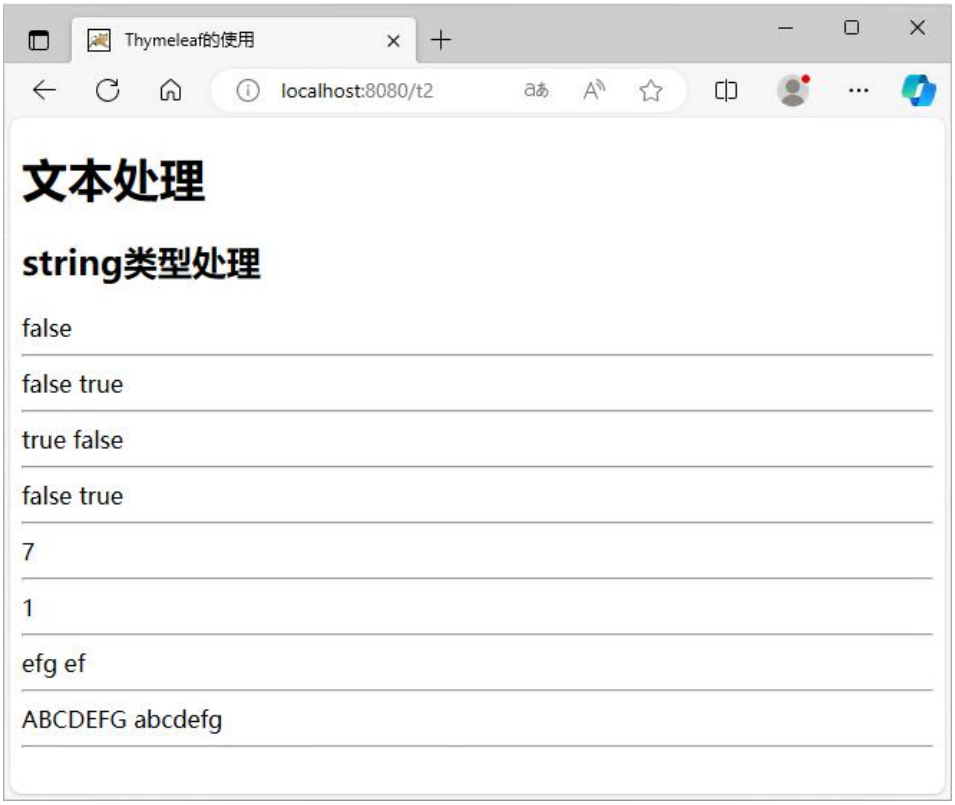


图 4-1-14 运行效果

4. 日期格式处理

表达式	说明
${#dates.format(key)}	格式化日期，默认的以浏览器默认语言为格式化标准
${#dates.format(key,‘yyy/MM/dd’)}	按照自定义的格式做日期转换
${#dates.year(key)}	取年
${#dates.month(key)}	取月
${#dates.day(key)}	取日

模板中代码如下：

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Thymeleaf 的使用</title>
</head>
<body>
    <h1>日期格式处理</h1>
    <h2>Date 使用</h2>
        <span th:text="${#dates.format(now)}"></span>
    <hr>
    <span th:text="${#dates.format(now,'yyyy-MM-dd')}"></span>
    <hr>
    <span th:text="${#dates.format(now,'yyyy-MM-dd hh:ss:mm')}"></span> <hr>
    <span th:text="${#dates.format(now,'yyyy-MM-dd HH:ss:mm')}"></span> <hr>
    <span th:text="${#dates.year(now)}"></span> <hr>
    <span th:text="${#dates.month(now)}"></span> <hr>
    <span th:text="${#dates.day(now)}"></span> <hr>
    <span th:text="${#dates.dayOfWeek(now)}"></span> <hr>
    <span th:text="${#dates.hour(now)}"></span> <hr>
</body>
</html>
```

Controller 文件中代码如下：

```
@RequestMapping("/t3")
public String t3(Model model){
    model.addAttribute("now",new Date());
    return "thymeleafTest/t3";
}
```

运行结果如图 4-1-15 所示。

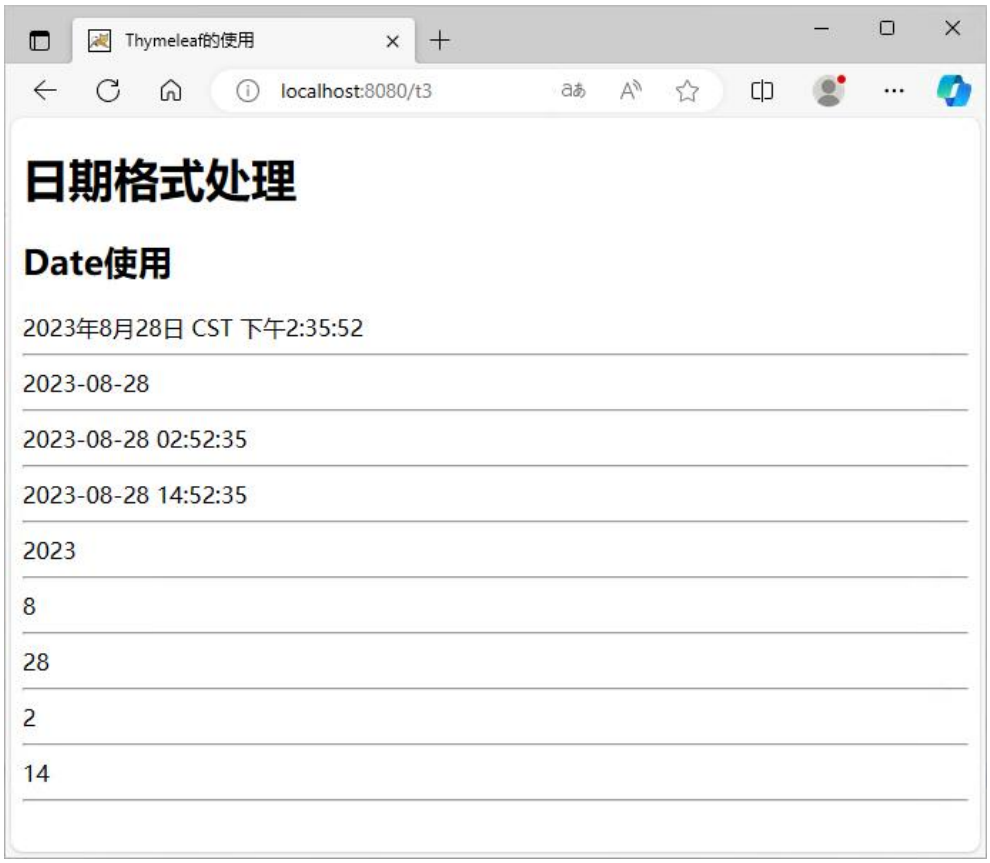


图 4-1-15 运行效果

5. 条件判断

表达式	说明
th:if	if 语句
th:switch	switch 语句

模板中代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Thymeleaf 的使用</title>
    </head>
    <body>
        <h1>条件判断</h1>
        <h2>if 语句</h2>
```

```
    <span th:if="${sex} == '男'" >男</span>
    <span th:if="${sex} == '女'" >男</span> <hr>
    <h2>switch 语句</h2>

    <span th:switch="${id}">
        <span th:case="1">ID 为 1</span>
        <span th:case="2">ID 为 2</span>
        <span th:case="3">ID 为 3</span>
    </span>
</body>
</html>
```

Controller 文件中代码如下：

```
@RequestMapping("/t4")
public String t4(Model model){
    model.addAttribute("sex","男");
    model.addAttribute("id",2);
    return "thymeleafTest/t4";
}
```

运行结果如图 4-1-16 所示。

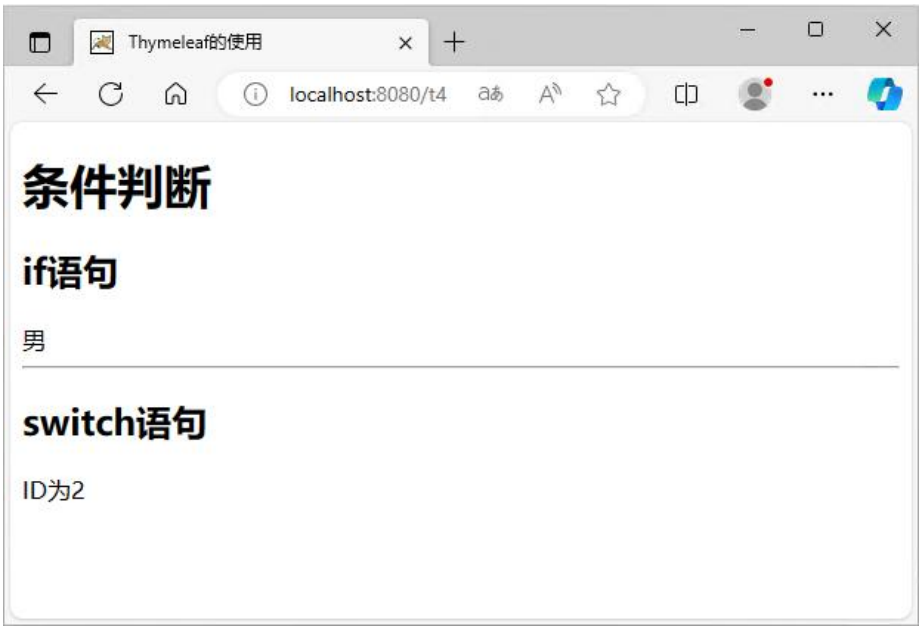


图 4-1-16 运行效果

6. 迭代遍历

表达式	说明
th:each	each 循环

模板中代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Thymeleaf 的使用</title>
    </head>
    <body>
        <h1>迭代遍历</h1>
        <h2>each 语句</h2>
        <table border="1"  style="border-collapse: collapse">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Age</th>
            </tr>
            <tr th:each="u : ${list}">
                <td th:text="${u.id}"></td>
                <td th:text="${u.username}"></td>
                <td th:text="${u.age}"></td>
            </tr>
        </table>
        <hr>
        <h2>状态变量</h2>
        <table border="1" style="border-collapse: collapse">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Age</th>
                <th>Index</th>
                <th>Count</th>
                <th>Size</th>
                <th>Even</th>
                <th>Odd</th>
                <th>First</th>
```

```
            <th>lase</th>
        </tr>
        <tr th:each="u,var : ${list}">
            <td th:text="${u.id}"></td>
            <td th:text="${u.username}"></td>
            <td th:text="${u.age}"></td>
            <td th:text="${var.index}"></td>
            <td th:text="${var.count}"></td>
            <td th:text="${var.size}"></td>
            <td th:text="${var.even}"></td>
            <td th:text="${var.odd}"></td>
            <td th:text="${var.first}"></td>
            <td th:text="${var.last}"></td>
        </tr>
    </table>
    <h2>th:each 迭代 Map</h2>
    <table border="1"  style="border-collapse: collapse">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Age</th>
        </tr>
        <tr th:each="maps : ${map}">
            <td th:text="${maps.getKey()+':'+maps.getValue().id}"></td>
            <td th:text="${maps.getKey()+':'+maps.getValue().username}"></td>
            <td th:text="${maps.getKey()+':'+maps.getValue().age}"></td>
        </tr>
    </table>
    <th/>

</body>
</html>
```

Controller 文件中代码如下：

```
class User{
   private int id;
   private String username;
   private int age;

   public void setId(int id) {
     this.id = id;
```

```
    }
    public int getId() {
      return this.id;
    }

    public void setUsername(String username) {
      this.username = username;
    }
    public String getUsername() {
      return this.username;
    }

    public void setAge(int age) {
        this.age = age;
    }
    public int getAge() {
      return this.age;
    }

    public User(int id, String username, int age) {
      this.id = id;
      this.username = username;
      this.age = age;
    }
}
@RequestMapping("/t5")
public String t5(Model model){
    List<User> list = new ArrayList<>();
    list.add(new User(1,"张三",18));
    list.add(new User(2,"李四",19));
    list.add(new User(3,"王五",20));
    Map<String, User> map = new HashMap<>();
    map.put("u1", new User(1,"张三",20));
    map.put("u2", new User(2,"李四",22));
    map.put("u3", new User(3,"王五",24));
    model.addAttribute("map", map);
    model.addAttribute("list",list);
    return "thymeleafTest/t5";
}
```

运行结果如图 4-1-17 所示。

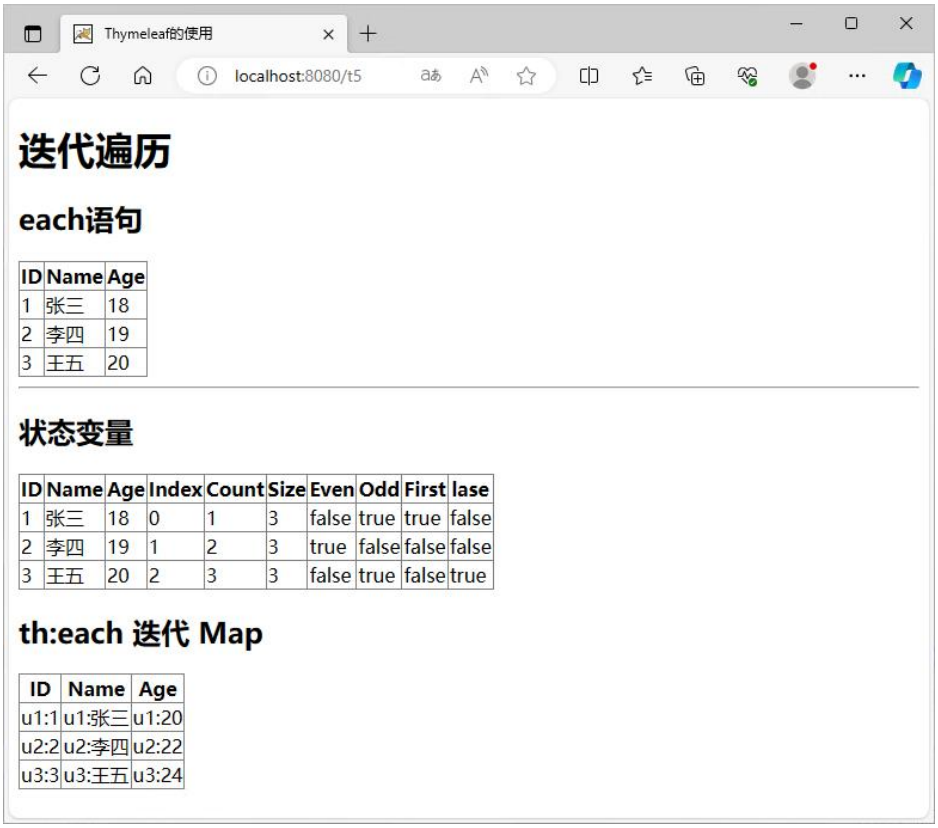


图 4-1-17 运行效果

7. URL 表达式

模板中代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Thymeleaf 基本使用</title>
    </head>
    <body>
        <h1>基本使用</h1>
        <h2>URL 使用</h2>

        <a th:href="@{http://www.baidu.com}">绝对路径</a><br/>
        <a href="http://www.baidu.com">绝对路径 2</a>

        <hr/>
        <a th:href="@{show}">相对路径（相对于当前目录）</a><br/>
```

```
        <a th:href="@{./show}">相对路径（相对于当前目录）</a><br/>
        <a th:href="@{../show}">相对路径（相对于上级目录）</a><br/>

        <hr/>
        <a th:href="@{~/project2/resourcename}">绝对路径（从于服务器的根）</a>

        <hr/>
        <a th:href="@{/path/{id}/show(id=1,name=zhagnsan)}">绝对路径（从于服务器的根）-传参</a>

    </body>
    </html>
```

Controller 文件中代码如下：

```
    @RequestMapping("/t6")
    public String t5(Model model){
        return "thymeleafTest/t6";
    }
```

运行结果如图 4-1-18 所示。

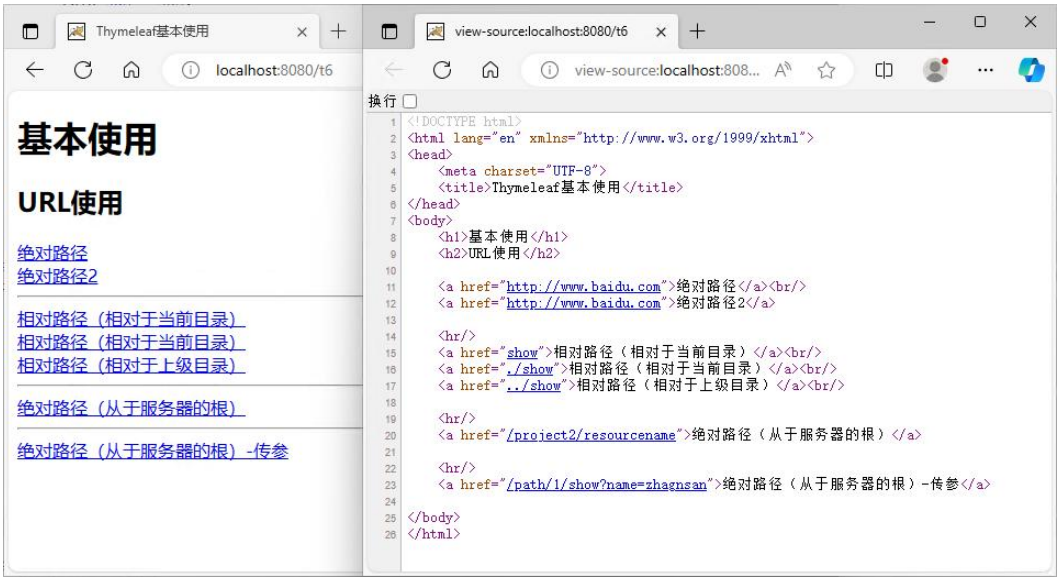


图 4-1-18 运行效果

四、用户登录界面制作

1. 创建 login.html 模板文件

在模板文件夹“templates”创建 login.html 模板文件，打开 login.html 写入 HTML 代码，具体代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>生物测序实验管理系统</title>
        <meta http-equiv="Pragma" content="no-cache">
        <meta http-equiv="Cache-Control" content="no-cache">
        <meta http-equiv="Expires" content="0">
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=0, minimal-ui">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <link rel="stylesheet" href="css/style.css" />
     <link rel="stylesheet" href="css/login.css" />
     <link rel="stylesheet" href="css/dialog.css" />
     <script type="text/javascript" src="js/jquery-3.7.1.min.js"></script>
     <script type="text/javascript" src="js/dialog.js"></script>
    </head>
    <body class="body-login">

     <div class="top">
               <div class="logo">
                         <img src="images/syslogo.jpg" />
               </div>
               <div class="title">
                         生物测序实验管理系统
               </div>
     </div>

     <div class="login-form">
               <h3 class="login-title">登录</h3>
               <div class="form-group">
                         <input type="text" name="userName" id="userName" class="input
user-name" placeholder="用户名" autocomplete="off" />
               </div>
               <div class="form-group">
```

```
                        <input type="password" name="password" id="password"
class="input password" placeholder="密码" autocomplete="off" />
                </div>
                <div class="form-group">
                        <button class="btn-login">登录</button>
                </div>
     </div>

    </body>
    </html>
```

给登录页面创建控制类，具体代码如下：

```
    package com.bioseqmanage.controller;
    import org.springframework.stereotype.Controller;
    import org.springframework.ui.Model;
    import org.springframework.web.bind.annotation.RequestMapping;

    @Controller
    public class LoginController {

     @RequestMapping(value={"/login"})
     public String index(Model model) {
                return "login";
     }

    }
```

2. 创建 static 静态资源文件夹

在“src/main/resource”创建 static 文件，该文件夹存放网页需要的静态资源。这里我们还需要分别创建“css”、“images”两个文件夹，存放样式表和图片，如图 4-1-19 所示。

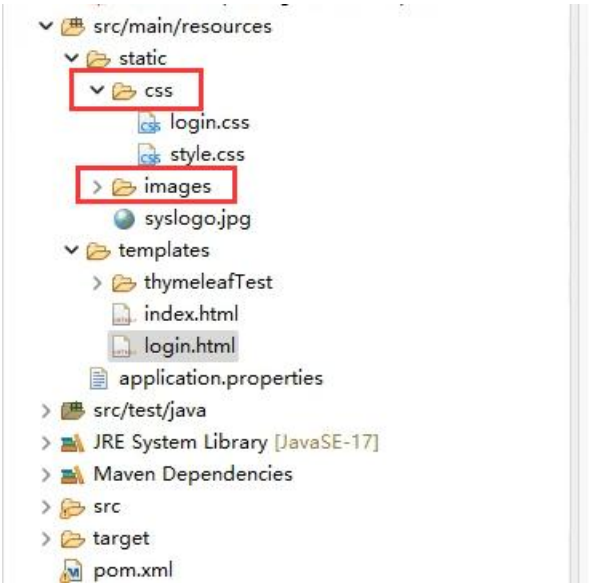


图 4-1-19 静态文件夹

3. 静态资源文件夹访问配置

创建配置包“com.bioseqmanage.config”，然后创建 WebMVConfig.java 类文件，给创建的“css”和“images”自定义映射，如果在以后还有创建的文件夹，记得在这里进行映射配置，否则会出现无法访问到资源的情况。具体代码如下：

```
package com.bioseqmanage.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport;

@Configuration
public class WebMVCConfig extends WebMvcConfigurationSupport {

/**
 * 自定义资源映射
 */
@Override
protected void addResourceHandlers(ResourceHandlerRegistry registry) {
registry.addResourceHandler("/css/**").addResourceLocations("file:src/main/resourc
es/static/css/");
registry.addResourceHandler("/images/**").addResourceLocations("file:src/main/reso
urces/static/images/");
}
}
```

4. css 样式表

在 css 文件里分别创建 style.css 和 login.css 两个样式表文件，style.css 是整个系统页面公用的，login.css 值在登录界面使用到。

style.css 代码如下：

```
    section, article, aside, header, footer, nav, figure { display: block; }
    body, h1, h2, h3, h4, h5, h6, blockquote, div, dl, dt, dd, ul, ol, li, p, pre, form, fieldset, legend, button, input, textarea, th, td, tr, table, tbody, thead, tfoot, em, strong, span, figure { margin: 0; padding: 0; transition: background 0.3s;}
    body, button, input, select, textarea { font: 12px/1.8 Microsoft YaHei,Simsun,sans-serif; outline: none; color: #666; }
    h1, h2, h3, h4, h5, h6, button, input, select, textarea { font-size: 100%; font-style: normal; font-weight: normal; }
    address, cite, dfn, em, var { font-style: normal; }
    code, kbd, pre, samp { font-family: courier new,courier,monospace; }
    p, ul, li, dl, dd, dt, form, h1, h2, h3, h4, h5, h6 { list-style: none; }
    a { color: #666; text-decoration: none; }
    a:hover { text-decoration: none; }
    fieldset, img { border: 0; }
    img { display: block; border: 0 none; }
    em { font-weight: normal; font-style: normal; }
    table { border-collapse: collapse; border-spacing: 0; }
    input::-moz-focus-inner, button::-moz-focus-inner { border: 0; padding: 0; outline: 0; }
    body{background-color: #f3f5fb;}
```

login.css 代码如下：

```
    html{
     height: 100%;
    }

    .body-login{
     height: 100%;
     background: url(../images/bg-login.png) no-repeat center center;
     background-size: 100% 100%;
     padding-top: 40px;
     box-sizing: border-box;
     position: relative;
    }
```

```
.top{
 width: 100%;
 height: 40px;
 padding: 0 50px;
 box-sizing: border-box;
}
.top .logo{
 width: 40px;
 height: 40px;
 border-radius: 20px;
 overflow: hidden;
 float: left;
}
.top .logo img{
 width: 100%;
 height: 100%;
}
.top .title{
 float: left;
 color: #ffffff;
 height: 40px;
 line-height: 40px;
 margin-left: 10px;
 font-size: 20px;
}

.login-form{
 width: 360px;
 position: absolute;
 right: 120px;
 top: 50%;
 transform: translateY(-50%);
 background-color: rgba(255, 255, 255, 0.15);
 border-radius: 4px;
 padding: 50px 25px;
}
.login-form .login-title{
 width: 100%;
 color: #ffffff;
 text-align: center;
 font-size: 22px;
}
```

```
.form-group{
 padding: 20px 0;
}

.form-group .input{
 width: 100%;
 height: 40px;
 background: rgba(2, 26, 67, 0.2);
 border: 1px solid #ffffff;
 border-radius: 2px;
 padding: 0 10px;
 box-sizing: border-box;
 font-size: 16px;
 font-weight: bold;
 color: #e6e6e6;
}

.form-group .btn-login{
 width: 100%;
 height: 40px;
 border: 1px solid #3e9ca3;
    background-color: #3e9ca3;
    color: #e6e6e6;
    border-radius: 2px;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
}
.form-group .btn-login:hover{
 background-color: rgb(55, 170, 167);
}
```

运行项目，打开浏览器输入 http://localhost:8080/login 查看效果，如图 4-1-20 所示。

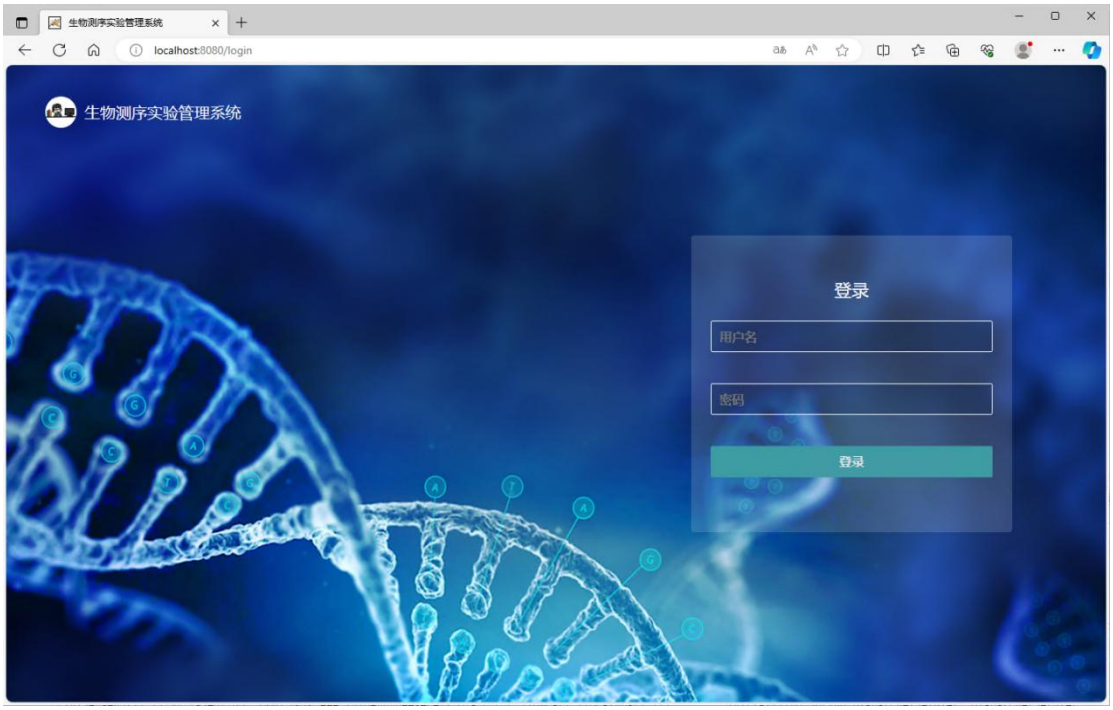


图 4-1-20 登录界面

五、系统首页界面制作

1. 创建 index.html 模板文件

在模板文件夹“templates”创建 index.html 模板文件，打开 index.html 写入 HTML 代码，具体代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>生物测序实验管理系统</title>
        <meta http-equiv="Pragma" content="no-cache">
        <meta http-equiv="Cache-Control" content="no-cache">
        <meta http-equiv="Expires" content="0">
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=0, minimal-ui">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <link rel="stylesheet" href="css/style.css" />
     <link rel="stylesheet" href="css/main.css" />
    </head>
```

```
<body>

    <div class="left-nav">
            <div class="left-nav-top">
                    <div class="logo">
                            <img src="images/syslogo.jpg" />
                    </div>
                    <div class="title">
                            生物测序实验管理系统
                    </div>
            </div>
            <div class="nav-title">
                    导航
            </div>
            <div class="nav-content">
                    <div class="nav-item">
                            <a href="#">任务管理</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">样本管理</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">生产管理 Lite</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">仪器监控</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">报告管理</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">其他</a>
                    </div>
                    <div class="nav-item">
                            <a href="#">基础配置</a>
                    </div>
            </div>
    </div>

    <div class="top-nav">
            <div class="top-nav-content">
                    <div class="top-nav-item">
                            <div class="item-icon icon0"></div>
                            <div class="">首页</div>
```

```
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon1"></div>
                              <div class="item-title">实验室管理</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon2"></div>
                              <div class="item-title">自动化分析</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon3"></div>
                              <div class="item-title">数据治理</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon4"></div>
                              <div class="item-title">应用市场</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon5"></div>
                              <div class="item-title">样本管理</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon6"></div>
                              <div class="item-title">科研管理</div>
                    </div>
                    <div class="top-nav-item">
                              <div class="item-icon icon7"></div>
                              <div class="item-title">系统管理</div>
                    </div>
          </div>
    </div>

    <div class="container">

    </div>

   </body>
   </html>
```

给首页创建控制类，具体代码如下：

```
    @RequestMapping(value={"/","/index"})
    public String index(Model model) {
```

```
            return "index";
    }
```

2. 创建 main.css 样式表文件

main.css 主要是首页的布局样式，代码如下：

```
.left-nav{
 position: fixed;
 top: 0;
 width: 264px;
 background: #3f4d67;
 height: 100vh;
 z-index: 2;
}
.left-nav .left-nav-top{
 height: 70px;
 padding: 5px 10px;;
}
.left-nav .left-nav-top .logo{
 width: 40px;
 height: 40px;
 border-radius: 25px;
 overflow: hidden;
 float: left;
 margin-top: 5px;
}
.left-nav .left-nav-top .logo img{
 width: 100%;
 height: 100%;
}

.left-nav .left-nav-top .title{
 font-size: 16px;
 font-weight: bold;
 color: #ffffff;
 line-height: 50px;
 float: left;
 padding: 0 0 0 10px;
}

.left-nav .nav-title{
```

```
 color: #ffffff;
 height: 50px;
 line-height: 30px;
 padding: 0 20px;
 font-size: 16px;
 font-weight: bold;
}

.left-nav .nav-content{
 padding: 0;
}
.left-nav .nav-content .nav-item{
 height: 50px;
 line-height: 50px;
 color: #ffffff;
 font-size: 14px;
 font-weight: bold;
 cursor: pointer;
 padding: 0 20px;
}

.left-nav .nav-content .nav-item:hover{
 background: rgb(40, 100, 125);
}

.left-nav .nav-content .nav-item a{
 color: #ffffff;
 display: block;
 width: 100%;
}

.top-nav{
 position: fixed;
 top: 0px;
 height: 70px;
 background-color: #ffffff;
 width: calc(100% - 264px);
 margin-left: 264px;
 z-index: 1;
}

.top-nav .top-nav-content{
 display: flex;
 justify-content: center;
```

```
 align-items: center;
 padding: 10px 0;
}
.top-nav .top-nav-content .top-nav-item{
 display: flex;
 flex-direction: column;
 align-items: center;
 cursor: pointer;
 width: 90px;
 height: 70px;
}
.top-nav .top-nav-content .top-nav-item:hover{
 color: #2aacd6;
}

.top-nav .top-nav-content .top-nav-item .item-icon{
 width: 30px;
 height: 30px;
}

.top-nav .top-nav-content .top-nav-item .item-icon.icon0{
 background: url(../images/nav-icon-00.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon0{
 background: url(../images/nav-icon-00-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon1{
 background: url(../images/nav-icon-01.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon1{
 background: url(../images/nav-icon-01-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon2{
 background: url(../images/nav-icon-02.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon2{
 background: url(../images/nav-icon-02-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon3{
 background: url(../images/nav-icon-03.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon3{
 background: url(../images/nav-icon-03-hover.png) no-repeat center center;
}
```

```
.top-nav .top-nav-content .top-nav-item .item-icon.icon4{
 background: url(../images/nav-icon-04.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon4{
 background: url(../images/nav-icon-04-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon5{
 background: url(../images/nav-icon-05.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon5{
 background: url(../images/nav-icon-05-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon6{
 background: url(../images/nav-icon-06.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon6{
 background: url(../images/nav-icon-06-hover.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item .item-icon.icon7{
 background: url(../images/nav-icon-07.png) no-repeat center center;
}
.top-nav .top-nav-content .top-nav-item:hover .item-icon.icon7{
 background: url(../images/nav-icon-07-hover.png) no-repeat center center;
}

.item-title{
 font-size: 14px;
}


.container{
 background: url(../images/main.jpg) no-repeat center center ;
 background-size: 100% 100%;
 min-height: calc(100vh - 60px);
 margin-left: 264px;
 margin-top: 60px;
}
```

运行项目，打开浏览器输入 http://localhost:8080 查看效果，如图 4-1-21 所示。

图 4-1-21 系统首页

任务实施

Step1 HTML 超文本标记语言认知

Step2 CSS 层叠样式表认知

Step3 Thymeleaf 的基本语法认知

Step4 用户登录界面制作

Step5 系统首页界面制作

活动二 系统中集成统一的前端操作提示窗口

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。用户在操作的过程中少不了一些交互提示，比如：警告、确认和加载等等。然而一些浏览器默认的对话框样式、交互体验等不能完全符合我们的需要。现要求我们在系统中集成预设一套操作提示窗口。

任务目标

1. 通过引入 JQuery 库并且获取到标签元素对象，然后通过 JQuery 的方法操作元素对象。
2. 能集成确认窗口和加载提示等交互提示。

任务分析

1. 什么是 JQuery，怎样在 HMTL 中使用？
2. JQuery 如何获取元素对象？
3. JQuery 能对元素进行哪些操作？
4. 什么是 JSON？它的书写格式是怎样的？
5. 如何制作确认提示对话窗口？
6. 如何制作加载提示？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、JQuery 库

1. 认识 JQuery

(1) 什么是 JQuery

JQuery 是一个快速、简洁的 JavaScript 框架，是继 Prototype 之后又一个优秀的 JavaScript 代码库（框架），于 2006 年 1 月由 John Resig 发布。JQuery 设计的宗旨是“write Less，Do More”，

即倡导写更少的代码，做更多的事情。它封装 JavaScript 常用的功能代码，提供一种简便的 JavaScript 设计模式，优化 HTML 文档操作、事件处理、动画设计和 Ajax 交互。

JQuery 的核心特性可以总结为：具有独特的链式语法和短小清晰的多功能接口；具有高效灵活的 CSS 选择器，并且可对 CSS 选择器进行扩展；拥有便捷的插件扩展机制和丰富的插件。JQuery 兼容各种主流浏览器，如 IE 6.0+、FF 1.5+、Safari 2.0+、Opera 9.0+等。

(2) 添加 JQuery 库

在 HTML 页面中添加 JQuery 库，像其他 JavaScript 文件一样通过<script>标签添加，代码如下：

```
<head>
   <script type="text/javascript" src="jquery.js"></script>
</head>
```

(3) 下载 JQuery

共有两个版本的 JQuery 可供下载：一份是精简过的，另一份是未压缩的（供调试或阅读），这两个版本都可从 https://jquery.com 下载。

(4) JQuery 运行

JQuery 的运行需要在文档加载完成之后运行，如果在文档没有完全加载之前就运行 JQuery 脚本，操作可能会失败。所以我们需要防止这种情况的方式是我们需要把所有的 JQuery 脚本都位于 document ready 函数中，具体代码如下：

```
<head>
   <script type="text/javascript" src="jquery.js"></script>
       $(document).ready(function(){
           //JQuery 脚本
       }
       //或者使用以下简写
       $(function(){
           //JQuery 脚本
       });
</head>
```

2. JQuery 的语法

(1) JQuery 语法

jQuery 语法是为 HTML 元素的选取编制的，可以对元素执行某些操作。
基础语法是：$(selector).action()

- $符号就是 jQuery
- selector 是选择器，用来“查找”HTML 元素，$(selector)返回获取到的 JQuery 元素对象
- jQuery 的 action() 执行对元素的操作

JQuery 使用$符号作为 JQuery 的简捷方式。某些其他 JavaScript 库中的函数（比如 Prototype）同样使用$符号，此时将会产生冲突，JQuery 使用名为 noConflict()的方法来解决该问题。var jq=jQuery.noConflict()帮助您使用自己的名称（比如 jq）来代替$符号。

(2) JQuery 选择器

如何获取 HTML 元素的实例是 JQuery 对元素进行操作的必要条件，JQuery 可以通过元素选择器、属性选择器和 CSS 选择器对 HTML 元素进行选择。

JQuery 同选择器获取到的元素可能是元素组或单个元素，于此同时 JQuery 允许您对 DOM 元素组或单个 DOM 节点进行操作。

1) JQuery 元素选择器

JQuery 使用元素标签来选取 HTML 元素。
$("p")选取<p>元素。
$("p.intro")选取所有 class="intro"的<p>元素。
$("p#demo")选取所有 id="demo"的<p>元素。

2) JQuery 属性选择器

JQuery 使用 XPath 表达式来选择带有给定属性的元素。
$("[href]") 选取所有带有 href 属性的元素。
$("[href='#']") 选取所有带有 href 值等于"#"的元素。
$("[href!='#']") 选取所有带有 href 值不等于"#"的元素。
$("[href$='.jpg']") 选取所有 href 值以".jpg"结尾的元素。

3) JQuery CSS 选择器

JQuery CSS 的 class 名来获取 HTML 元素。
$(".intro")选取所有 class 为"intro"的元素。

4) JQuery 选择器实例

选择器	实例	说明
*	$("*")	所有元素
#id	$("#lastname")	id="lastname" 的元素
.class	$(".intro")	所有 class="intro" 的元素
element	$("p")	所有 <p> 元素
.class.class	$(".intro.demo")	所有 class="intro" 且 class="demo" 的元素
:first	$("p:first")	第一个 <p> 元素
:last	$("p:last")	最后一个 <p> 元素
:even	$("tr:even")	所有偶数 <tr> 元素
:odd	$("tr:odd")	所有奇数 <tr> 元素
:eq(index)	$("ul li:eq(3)")	列表中的第四个元素（index 从 0 开始）
:gt(no)	$("ul li:gt(3)")	列出 index 大于 3 的元素
:lt(no)	$("ul li:lt(3)")	列出 index 小于 3 的元素
:not(selector)	$("input:not(:empty)")	所有不为空的 input 元素
:header	$(":header")	所有标题元素 <h1> - <h6>
:animated		所有动画元素
:contains(text)	$(":contains('W3School')")	包含指定字符串的所有元素
:empty	$(":empty")	无子（元素）节点的所有元素
:hidden	$("p:hidden")	所有隐藏的 <p> 元素
:visible	$("table:visible")	所有可见的表格
s1,s2,s3	$("th,td,.intro")	所有带有匹配选择的元素
[attribute]	$("[href]")	所有带有 href 属性的元素
[attribute=value]	$("[href='#']")	所有 href 属性的值等于 "#" 的元素
[attribute!=value]	$("[href!='#']")	所有 href 属性的值不等于 "#" 的元素
[attribute$=value]	$("[href$='.jpg']")	所有 href 属性的值包含以 ".jpg" 结尾的元素
:input	$(":input")	所有 <input> 元素
:text	$(":text")	所有 type="text" 的 <input> 元素
:password	$(":password")	所有 type="password" 的 <input> 元素
:radio	$(":radio")	所有 type="radio" 的 <input> 元素
:checkbox	$(":checkbox")	所有 type="checkbox" 的 <input> 元素
:submit	$(":submit")	所有 type="submit" 的 <input> 元素
:reset	$(":reset")	所有 type="reset" 的 <input> 元素
:button	$(":button")	所有 type="button" 的 <input> 元素
:image	$(":image")	所有 type="image" 的 <input> 元素
:file	$(":file")	所有 type="file" 的 <input> 元素

:enabled	$(":enabled")	所有激活的 input 元素
:disabled	$(":disabled")	所有禁用的 input 元素
:selected	$(":selected")	所有被选取的 input 元素
:checked	$(":checked")	所有被选中的 input 元素

3. JQuery 事件

JQuery 事件处理方法是 JQuery 中的核心函数。事件处理程序指的是当 HTML 中发生某些事件时所调用的方法。比如页面访问者对某个元素做了某个动作，比如：鼠标点击、鼠标经过等将会触发该元素绑定的事件方法。

比如对某个按钮点击了一下，然后对<p>里的内容进行隐藏，代码如下：

```
<html>
<head>
   <script type="text/javascript" src="jquery.js"></script>
   <script type="text/javascript">
   $(document).ready(function(){
      $("button").click(function(){
          $("p").hide();
      });
   });
   </script>
</head>

<body>
   <h2>这是一个标题</h2>
   <p>这是一段话。</p>
   <p>这是另一段话。</p>
   <button>点击我</button>
</body>
</html>
```

JQuery 事件列举

选择器	说明
bind()	向匹配元素附加一个或更多事件处理器
blur()	触发、或将函数绑定到指定元素的 blur 事件
change()	触发、或将函数绑定到指定元素的 change 事件
click()	触发、或将函数绑定到指定元素的 click 事件
dblclick()	触发、或将函数绑定到指定元素的 double click 事件

delegate()	向匹配元素的当前或未来的子元素附加一个或多个事件处理器
die()	移除所有通过 live() 函数添加的事件处理程序。
error()	触发、或将函数绑定到指定元素的 error 事件
event.isDefaultPrevented()	返回 event 对象上是否调用了 event.preventDefault()。
event.pageX	相对于文档左边缘的鼠标位置。
event.pageY	相对于文档上边缘的鼠标位置。
event.preventDefault()	阻止事件的默认动作。
event.result	包含由被指定事件触发的事件处理器返回的最后一个值。
event.target	触发该事件的 DOM 元素。
event.timeStamp	该属性返回从 1970 年 1 月 1 日到事件发生时的毫秒数。
event.type	描述事件的类型。
event.which	指示按了哪个键或按钮。
focus()	触发、或将函数绑定到指定元素的 focus 事件
keydown()	触发、或将函数绑定到指定元素的 key down 事件
keypress()	触发、或将函数绑定到指定元素的 key press 事件
keyup()	触发、或将函数绑定到指定元素的 key up 事件
live()	为当前或未来的匹配元素添加一个或多个事件处理器
load()	触发、或将函数绑定到指定元素的 load 事件
mousedown()	触发、或将函数绑定到指定元素的 mouse down 事件
mouseenter()	触发、或将函数绑定到指定元素的 mouse enter 事件
mouseleave()	触发、或将函数绑定到指定元素的 mouse leave 事件
mousemove()	触发、或将函数绑定到指定元素的 mouse move 事件
mouseout()	触发、或将函数绑定到指定元素的 mouse out 事件
mouseover()	触发、或将函数绑定到指定元素的 mouse over 事件
mouseup()	触发、或将函数绑定到指定元素的 mouse up 事件
one()	向匹配元素添加事件处理器。每个元素只能触发一次该处理器。
ready()	文档就绪事件（当 HTML 文档就绪可用时）
resize()	触发、或将函数绑定到指定元素的 resize 事件
scroll()	触发、或将函数绑定到指定元素的 scroll 事件
select()	触发、或将函数绑定到指定元素的 select 事件
submit()	触发、或将函数绑定到指定元素的 submit 事件
toggle()	绑定两个或多个事件处理器函数，当发生轮流的 click 事件时执行。
trigger()	所有匹配元素的指定事件
triggerHandler()	第一个被匹配元素的指定事件
unbind()	从匹配元素移除一个被添加的事件处理器
undelegate()	从匹配元素移除一个被添加的事件处理器，现在或将来
unload()	触发、或将函数绑定到指定元素的 unload 事件

二、JSON 数据库格式

1. JSON 概述

JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。它由 Douglas Crockford 在 2001 年首次提出，并逐渐成为一种广泛应用的数据格式。

JSON 采用简洁的文本形式表示结构化数据，可读性较高，易于理解和生成。它基于 JavaScript 语言的一个子集，但与具体编程语言无关，可以被多种编程语言解析和生成。这使得 JSON 成为跨平台和跨语言数据交换的理想选择。

JSON 的数据结构由键值对组成，键值对之间使用逗号分隔。键是字符串，值可以是字符串、数字、布尔值、数组、对象或 null。对象由花括号{}包围，数组由方括号[]包围。

JSON 的优势在于其简洁性和易解析性。与 XML 相比，JSON 的数据描述更加紧凑，使用的字符更少，易于传输和解析。同时，JSON 的层级结构以及简单的键值对形式使得数据的组织和访问更加直观和便捷。

JSON 的应用范围非常广泛。它常用于 Web 开发中的前后端数据传输、API 接口设计、配置文件、日志记录和存储、移动应用程序与服务器之间的数据交互等场景。许多编程语言都提供了内置的 JSON 处理库或函数，使得对 JSON 数据进行解析和生成变得简单和高效。

需要注意的是，JSON 只是一种数据格式，不包含逻辑运算和动态行为。它的设计初衷是用来描述和交换数据，而不是用于程序控制或函数调用。

总之，JSON 是一种轻量级的数据交换格式，具有简洁性、易读性和跨平台的特点。它在 Web 开发、数据交互和配置文件等领域得到广泛应用，并且成为编程语言间数据交换的标准格式之一。

2. JSON 语法

JSON 语法衍生于 JavaScript 对象标记法语法：

- ●数据在名称/值对中
- ●数据由逗号分隔
- ●花括号容纳对象
- ●方括号容纳数组

JSON 数据是一组键值对，既：名称和值。如下代码：

```
{ name : "张小明", age : 18, city : "深圳" }
```

3. JSON 数值类型

JSON 数据交换最终本质是文本内容，所以在 JSON 中，值必须是以下数据类型之一：

●字符串
●数字
●对象（JSON 对象）
●数组
●布尔
●null

如下代码所示：

```
{ key1: "字符串", key2: 18, key3: { }, key4: [], key5: true, key6: null }
```

在 JavaScript 中，以上所列均可为值，外加其他有效的 JavaScript 表达式，包括：

●函数
●日期
●undefined

4. JSON 转化

(一) JSON 解析

使用 JavaScript 函数 JSON.parse()把文本转换为 JavaScript 对象：

```
var json = '{ name : "张小明", age : 18, city : "深圳" }';
var obj = JSON.parse(json);
document.getElementById("demo").innerHTML = "姓名：" + obj.name + ", 年龄：" + obj.age;
```

(二) JSON 字符串转化

使用 JavaScript 函数 JSON.stringify()把 JavaScript 对象转化成字符串：

```
var obj = { name : "张小明", age : 18, city : "深圳" };
var json = JSON.stringify(obj);
```

```
    document.getElementById("demo").innerHTML = json ;
```

三、集成确认提示对话窗口

1. 确认提示对话窗口效果及代码

查看我们制作好的效果，如图 5-2-1 所示。

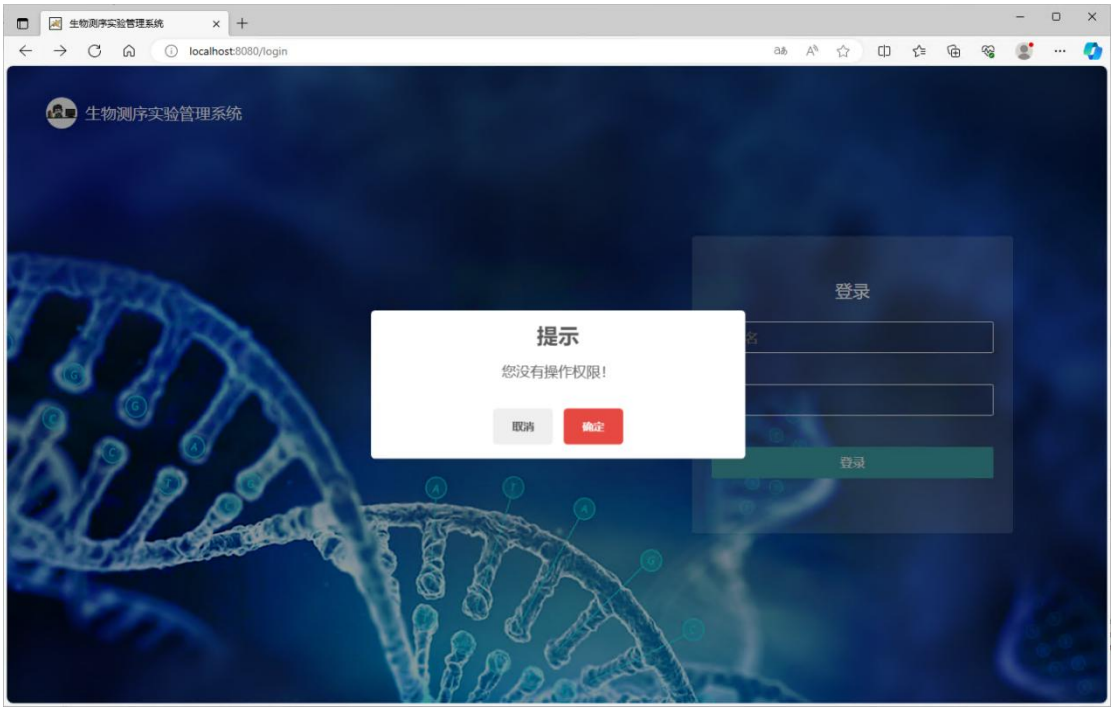


图 5-2-1 确认提示对话框效果

这个效果的 html 代码是写在登录界面的，封装后可以在系统的其他页面引入 js 文件来调用。该页面 HTML 代码如下：

```
    <!DOCTYPE html>
    <html lang="en" xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>生物测序实验管理系统</title>
        <meta http-equiv="Pragma" content="no-cache">
        <meta http-equiv="Cache-Control" content="no-cache">
        <meta http-equiv="Expires" content="0">
```

```
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=0, minimal-ui">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <link rel="stylesheet" href="css/style.css" />
     <link rel="stylesheet" href="css/login.css" />
     <link rel="stylesheet" href="css/dialog.css" />
     <script type="text/javascript" src="js/jquery-3.7.1.min.js"></script>
     <script type="text/javascript" src="js/dialog.js"></script>
    </head>
    <body class="body-login">

     <div class="top">
               <div class="logo">
                         <img src="images/syslogo.jpg" />
               </div>
               <div class="title">
                         生物测序实验管理系统
               </div>
     </div>

     <div class="login-form">
               <h3 class="login-title">登录</h3>
               <div class="form-group">
                         <input type="text" name="userName" id="userName" class="input
user-name" placeholder="用户名" autocomplete="off" />
               </div>
               <div class="form-group">
                         <input type="password" name="password" id="password"
class="input password" placeholder="密码" autocomplete="off" />
               </div>
               <div class="form-group">
                         <button class="btn-login">登录</button>
               </div>
     </div>

     <div class="dialog-mask" tabindex="-1" >
               <div class="dialog-modal" role="dialog" aria-modal="true">
                         <div class="swal-title" style="">提示</div>
                         <div class="swal-text" style="">您没有操作权限！</div>
                         <div class="swal-footer">
                                   <div class="swal-button-container">
                                             <button class="swal-button
swal-button--cancel" tabindex="0">取消</button>
                                   </div>
```

```
                                    <div class="swal-button-container">
                                            <button class="swal-button
swal-button--confirm swal-button--danger">确定</button>
                                    </div>
                            </div>
                    </div>
        </div>

        <div class="loading-mask" tabindex="-1" style="display: none;">
                    <div class="loading-modal">
                            <div class="spinner-border text-primary" role="status">
                            <span class="sr-only">Loading...</span>
                        </div>
                    </div>
        </div>

    </body>
    </html>
```

在 css 文件夹创建 dialog.css 文件，该文件包含了确认提示对话窗口和加载提示，具体代码如下：

```
    .dialog-mask {
     position: fixed;
        top: 0;
        bottom: 0;
        left: 0;
        right: 0;
        text-align: center;
        overflow-y: auto;
        background-color: rgba(0,0,0,.4);
        z-index: 10000;
        pointer-events: auto;
        transition: opacity .3s;
    }
    .loading-mask {
     position: fixed;
        top: 0;
        bottom: 0;
        left: 0;
        right: 0;
        text-align: center;
```

```
    overflow-y: auto;
    background-color: rgba(0,0,0,.4);
    z-index: 10009;
    pointer-events: auto;
    transition: opacity .3s;
}
.dialog-mask:before,.loading-mask:before {
    content: " ";
    display: inline-block;
    vertical-align: middle;
    height: 100%;
}

.dialog-modal {
    width: 478px;
    background-color: #fff;
    text-align: center;
    border-radius: 5px;
    position: static;
    margin: 20px auto;
    display: inline-block;
    vertical-align: middle;
    transform: scale(1);
    transform-origin: 50% 50%;
    z-index: 10001;
    transition: transform .3s,opacity .2s,-webkit-transform .3s;
    pointer-events: auto;
    box-sizing: border-box;
    animation: showSweetAlert .3s;
    will-change: transform;
}
.loading-modal {
    text-align: center;
    border-radius: 5px;
    position: static;
    margin: 20px auto;
    display: inline-block;
    vertical-align: middle;
    transform: scale(1);
    transform-origin: 50% 50%;
    z-index: 10010;
    transition: transform .3s,opacity .2s,-webkit-transform .3s;
    pointer-events: auto;
    box-sizing: border-box;
```

```
        animation: showSweetAlert .3s;
        will-change: transform;
    }


    .swal-icon {
        width: 80px;
        height: 80px;
        border-width: 4px;
        border-style: solid;
        border-radius: 50%;
        padding: 0;
        position: relative;
        box-sizing: content-box;
        margin: 20px auto;
    }
    .swal-title {
        color: rgba(0,0,0,.65);
        font-weight: 600;
        text-transform: none;
        position: relative;
        display: block;
        padding: 13px 16px;
        font-size: 27px;
        line-height: normal;
        text-align: center;
        margin-bottom: 0;
    }
    .swal-text {
        font-size: 18px;
        position: relative;
        float: none;
        vertical-align: top;
        text-align: left;
        display: inline-block;
        margin: 0;
        padding: 0 10px;
        font-weight: 400;
        max-width: calc(100% - 20px);
        overflow-wrap: break-word;
        box-sizing: border-box;
    }
    .swal-footer {
        text-align: center;
```

```
    padding-top: 13px;
    margin-top: 13px;
    padding: 13px 16px;
    border-radius: inherit;
    border-top-left-radius: 0;
    border-top-right-radius: 0;
}
.swal-button-container {
    margin: 5px;
    display: inline-block;
    position: relative;
}

.swal-button {
    background-color: #7cd1f9;
    color: #fff;
    border: none;
    box-shadow: none;
    border-radius: 5px;
    font-weight: 600;
    font-size: 14px;
    padding: 10px 24px;
    margin: 0;
    cursor: pointer;
}
.swal-button--danger {
    background-color: #e64942;
}
.swal-button--cancel {
    color: #555;
    background-color: #efefef;
}

@keyframes showSweetAlert {
 0% {
 -webkit-transform:scale(1);
 transform:scale(1)
}
1% {
 -webkit-transform:scale(.5);
 transform:scale(.5)
}
45% {
 -webkit-transform:scale(1.05);
```

```
 transform:scale(1.05)
}
80% {
 -webkit-transform:scale(.95);
 transform:scale(.95)
}
to {
 -webkit-transform:scale(1);
 transform:scale(1)
}
}



.spinner-border {
    display: inline-block;
    width: 2rem;
    height: 2rem;
    vertical-align: text-bottom;
    border: 0.35rem solid currentColor;
    border-right-color: transparent;
    border-radius: 50%;
    animation: spinner-border 1.75s linear infinite;
}

.sr-only {
    border: 0;
    clip: rect(0,0,0,0);
    height: 1px;
    margin: -1px;
    overflow: hidden;
    padding: 0;
    position: absolute;
    width: 1px;
}

.text-primary {
    color: #007bff!important;
}

@-webkit-keyframes spinner-border {
 to {
           -webkit-transform:rotate(1080deg);
           transform:rotate(1080deg)
 }
```

```
}@keyframes spinner-border {
           to {
           -webkit-transform:rotate(1080deg);
           transform:rotate(1080deg)
 }
}



.swal-icon {
    width: 80px;
    height: 80px;
    border-width: 4px;
    border-style: solid;
    border-radius: 50%;
    padding: 0;
    position: relative;
    box-sizing: content-box;
    margin: 20px auto;
}



.swal-icon--warning {
    border-color: #f8bb86;
    -webkit-animation: pulseWarning .75s infinite alternate;
    animation: pulseWarning .75s infinite alternate;
}

.swal-icon--warning__body, .swal-icon--warning__dot {
    position: absolute;
    left: 50%;
    background-color: #f8bb86;
}
.swal-icon--warning__body {
    width: 5px;
    height: 47px;
    top: 10px;
    border-radius: 2px;
    margin-left: -2px;
}
.swal-icon--warning__dot {
    width: 7px;
    height: 7px;
    border-radius: 50%;
    margin-left: -4px;
```

```
        bottom: -11px;
    }
```

2. 集成到 JS 文件中

在 static 文件夹中创建 js 文件夹，然后在配置类文件 WebMVConfig.java 中加入映射配置，代码如下：

```
    /**
     * 自定义资源映射
     */
    @Override
    protected void addResourceHandlers(ResourceHandlerRegistry registry) {

    registry.addResourceHandler("/css/**").addResourceLocations("file:src/main/resourc
es/static/css/");
    registry.addResourceHandler("/images/**").addResourceLocations("file:src/main/reso
urces/static/images/");
    registry.addResourceHandler("/js/**").addResourceLocations("file:src/main/resource
s/static/js/");

    }
```

我们封装的代码使用到 JQuery，访问 http://jquery.com/download 下载 JQuery 库，如图 5-2-2 所示。

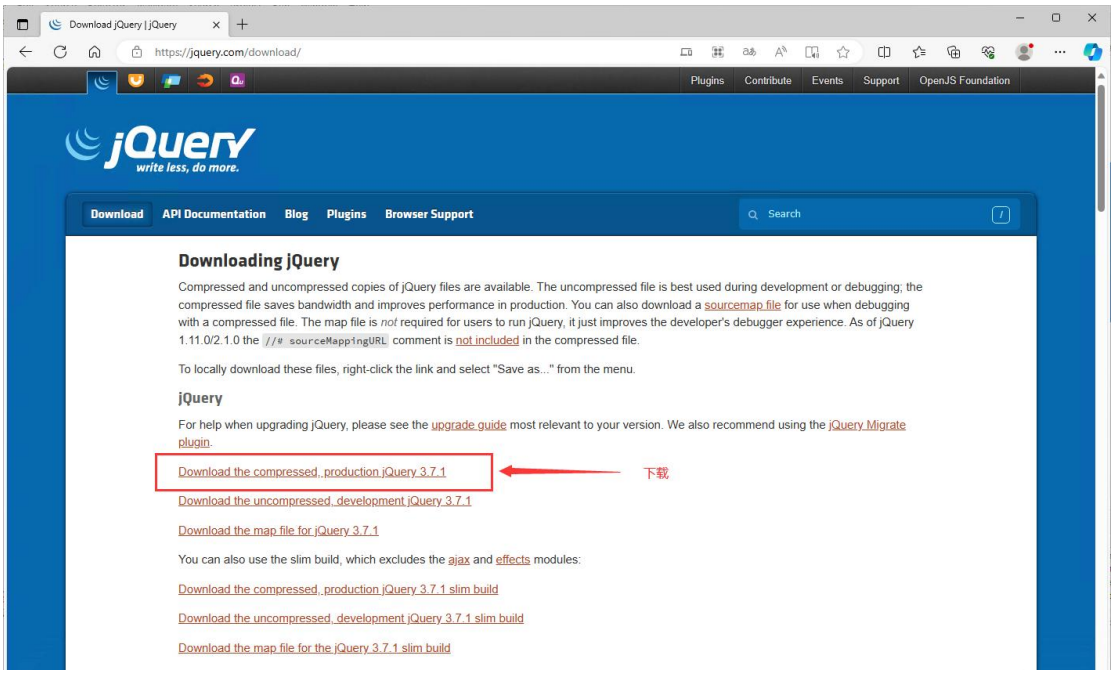


图 5-2-2 JQuery 库下载

把下载到的 jquery-3.7.1.min.js 文件放入到项目中的 js 文件夹中里。再在 js 文件夹中创建 dialog.js 文件，具体代码如下：

```
var dialog;

$(function(){

var dialogHtml = $('<div class="dialog-mask" tabindex="-1" >'+
'         <div class="dialog-modal" role="dialog" aria-modal="true">'+
'                 <div class="swal-title" style="">提示</div>'+
'                 <div class="swal-text" style="">您没有操作权限！</div>'+
'                 <div class="swal-footer">'+
'                         <div class="swal-button-container">'+
'                                 <button class="swal-button
swal-button--cancel" tabindex="0">取消</button>'+
'                         </div>'+
'                         <div class="swal-button-container">'+
'                                 <button class="swal-button
swal-button--confirm swal-button--danger">确定</button>'+
'                         </div>'+
'                 </div>'+
'         </div>'+
'</div>');

var loadingHtml = $('         <div class="loading-mask" tabindex="-1">'+
'         <div class="loading-modal">'+
```

```
    '                   <div class="spinner-border text-primary" role="status">'+
    '                   <span class="sr-only">Loading...</span>'+
    '               </div>'+
    '           </div>'+
    '</div>');



    dialog = {

              showDialog: function(_param){

                        var param = {
                                  title: _param.title || "提示",
                                  content: _param.content || "",
                                  cancel: _param.cancel || function(){},
                                  confirm: _param.confirm || function(){}
                        }

                        dialogHtml.find(".swal-title").html(param.title);
                        dialogHtml.find(".swal-text").html(param.content);

                        $("body").append(dialogHtml);

                        dialogHtml.find(".swal-button--cancel").click(function(){
                                  dialogHtml.remove();
                                  param.cancel();
                        });

                        dialogHtml.find(".swal-button--danger").click(function(){
                                  dialogHtml.remove();
                                  param.confirm();
                        });

              },
              showLoading(_param){

                        var param = {
                                  autoClose: typeof(_param.autoClose)=="boolean"?
_param.autoClose : true,  //是否自动关闭，默认为 ture 自动关闭
                                  closeTime: _param.closeTime || 1000
//自动关闭时间，默认 1000（毫米）
                        }
```

```
                $("body").append(loadingHtml);

                if(param.autoClose == true){
                        setTimeout(function(){
                                loadingHtml.remove();
                        }, param.closeTime);
                }

                return {
                        close: function(){
                                loadingHtml.remove();
                        }
                }
        }
 };
});
```

3. 页面中调用方法

在页面中把 jquery-3.7.1.min.js 和 dialog.js 文件提交到需要调用的页面，代码如下：

```
<script type="text/javascript" src="js/jquery-3.7.1.min.js"></script>
<script type="text/javascript" src="js/dialog.js"></script>
```

在页面中使用以下方式可以调用：

```
    <script>
        $(function(){
                //显示确认提示对话窗口
                dialog.showDialog({
                    title: "提示",
                    content: "你可以正常运行吗？"       ,
                    cancel: function(){
                    },
                    confirm: function(){
                    }
                });
        });
</script>
```

四、集成加载提示

1. 加载提示效果及代码

查看我们制作好的效果，如图 5-2-2 所示。

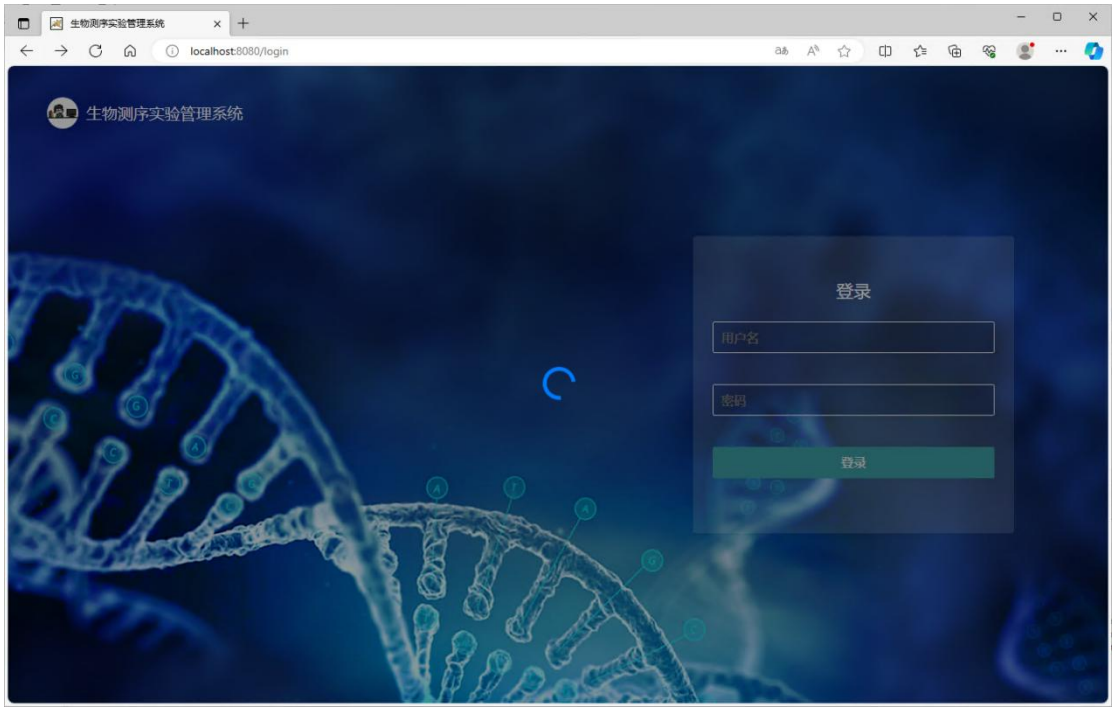


图 5-2-2 加载提示效果

在前面的代码中使用“display:none”做一下元素的隐藏调整，如图 5-2-3 所示。

```
<div class="dialog-mask" tabindex="-1" style="display: none;" >
    <div class="dialog-modal" role="dialog" aria-modal="true">
        <div class="swal-title" style="">提示</div>
        <div class="swal-text" style="">您没有操作权限！</div>
        <div class="swal-footer">
            <div class="swal-button-container">
                <button class="swal-button swal-button--cancel" tabindex="0">取消</button>
            </div>
            <div class="swal-button-container">
                <button class="swal-button swal-button--confirm swal-button--danger">确定</button>
            </div>
        </div>
    </div>
</div>

<div class="loading-mask" tabindex="-1">
    <div class="loading-modal">
        <div class="spinner-border text-primary" role="status">
            <span class="sr-only">Loading...</span>
        </div>
    </div>
</div>
```

原来的 style="display:none;"去掉

图 5-2-3 显示加载提示效果

2. 页面中调用方法

在页面中把 jquery-3.7.1.min.js 和 dialog.js 文件提交到需要调用的页面，代码如下：

```
<script type="text/javascript" src="js/jquery-3.7.1.min.js"></script>
<script type="text/javascript" src="js/dialog.js"></script>
```

在页面中使用以下方式可以调用：

```
 <script>
         $(function(){
                  //显示加载
                  var load = dialog.showLoading({
                      autoClose: false,//是否自动关闭，默认为 ture 自动关闭
                      closeTime: 1000  //自动关闭时间，默认 1000（毫米）
                  });

                  //当 autoClose 设置为 false，需要调用关闭，否则不会关闭
                  setTimeout(function(){
                      load.close();
                  }, 3000);
         });
</script>
```

任务实施

Step1 JQuery 认知

Step2 JSON 认知

Step3 集成确认提示对话窗口

Step4 集成加载提示

活动三 系统用户数据库表知识与设计

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。要求我们对系统中需要的数据库表进行建设！

任务目标

1. 能对 MySql 数据库进行 user 和 user_role 表结构及表之间关联关系的设计。
2. 能完成系统需要的数据表设计。

任务分析

1. 建表有哪些规约？
2. 索引有哪些规约？
3. 用户数据表和用户角色数据表的关系？
4. 生物测序实验管理系统都有哪些表？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、建表规约

1. 表达是与否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint（1 表示是，0 表示否）。

说明: 如果字段为非负数，必须用 unsigned。

注意：pojo 中任何布尔类型的变量，都不要加 is 前缀，所以需要在 resultMap 设置 is_xxx 到 Xxx 的映射关系。数据库表示是与否的值，使用 tinyint 类型，坚持 is_xxx 的命名方式是为了明确其取值含义与取值范围。

正例：表达逻辑删除的字段名为 is_deleted，1 表示删除，0 表示未删除。

2. 表名、字段名必须使用小写字母或数字，禁止出现数字开头，禁止两个下划线之间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名需要慎重考

虑。

说明：MySQL 在 windows 下不区分大小写，但是在 Linux 下是默认区分大小写，因此数据库名、表名和字段名都不允许出现任何大写字母，避免节外生枝。

正例：system_admin，rdc_config，level3_name。

反例：SystemAdmin，rdcConfig，level_3_name。

3. 表名不使用复数名词。

说明：表名应该仅仅表示里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

4. 禁用保留字，请参考 MySQL 官方保留字。

5. 主键索引名为 pk_字段名；唯一索引名为 uk_字段名；普通索引名为 idx_字段名

说明：pk_即 primary key；uk_即 unique key；idx_即 index 的简称。

6. 小数类型为 decimal，禁止使用 float 和 double。

说明：在存储的时候 float 和 double 都存在精度缺失的问题，很可能在比较值的时候得不到正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

7. 如果存储字符串长度几乎相等，使用 char 定长字符串类型。

8. varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text。独立出来一张表，用主键来对应，避免影响其他字段索引效率。

9. 表必备的三个字段，自增主键 id，创建时间 create_time，更新时间 update_time。

说明：其中 id 必为主键，类型为 bigint unsigned、单表时自增、步长为 1。Create_time, update_time 的类型均为 datetime 类型，前者是记录创建时间，后者是记录被更新时间。

10. 修改字段含义或一些表字段值含有追加时，应更新字段注释。

11. 字段允许适当冗余，以提高性能。

12. 单表容量超过 2GB 或者单表行数超过 500 万行，推荐进行分库分表。

二、索引规约

1. 业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。

说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

2. 超过三个表禁止 join。需要 join 的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。

说明：即使双表 join 也要注意表索引、SQL 性能。

3. 在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度。

说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度高达 90%以上，可以使用 count(distinct left(列名, 索引长度))/count(*)的区分度来确定。

4. 页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。

说明：索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

5. 如果有 order by 的场景，请注意利用索引的有序性。order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。

正例：where a=? and b=? order by c; 索引：a_b_c

反例：索引如果存在范围查询，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b; 索引 a_b 无法排序。

6. 利用覆盖索引来进行查询操作，避免回表。

说明：如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。

正例：能够建立索引的种类分为主键索引、唯一索引、普通索引三种，而覆盖索引只是一种查询的一种效果，用 explain 的结果，extra 列会出现：using index。

7. 利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

正例：先快速定位需要获取的 id 段，然后再关联：

SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20) b where a.id=b.id

8. SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts 最好。

说明：

1）consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。

2）ref 指的是使用普通的索引（normal index）。

3）range 对索引进行范围检索。

反例：explain 表的结果，type=index，索引物理文件全扫描，速度非常慢，这个 index 级别比较 range 还低，与全表扫描是小巫见大巫。

9. 建组合索引的时候，区分度最高的在最左边。

正例：如果 where a=? and b=?，a 列的几乎接近于唯一值，那么只需要单建 idx_a 索引即可。

说明：存在非等号和等号混合判断条件时，在建索引时，请把等号条件的列前置。如：where c>? and d=?那么即使 c 的区分度更高，也必须把 d 放在索引的最前列，即建立组合索引 idx_d_c。

10.防止因字段类型不同造成的隐式转换，导致索引失效。

11.创建索引时避免有如下极端误解：

1）索引宁滥勿缺。认为一个查询就需要建一个索引。

2）吝啬索引的创建。认为索引会消耗空间、严重拖慢记录的更新以及行的新增速度。

3）抵制唯一索引。认为唯一索引一律需要在应用层通过“先查后插”方式解决。

三、用户和用户角色数据表

1. 用户数据表

创建 users 用户数据表，如图 5-3-1 所示。

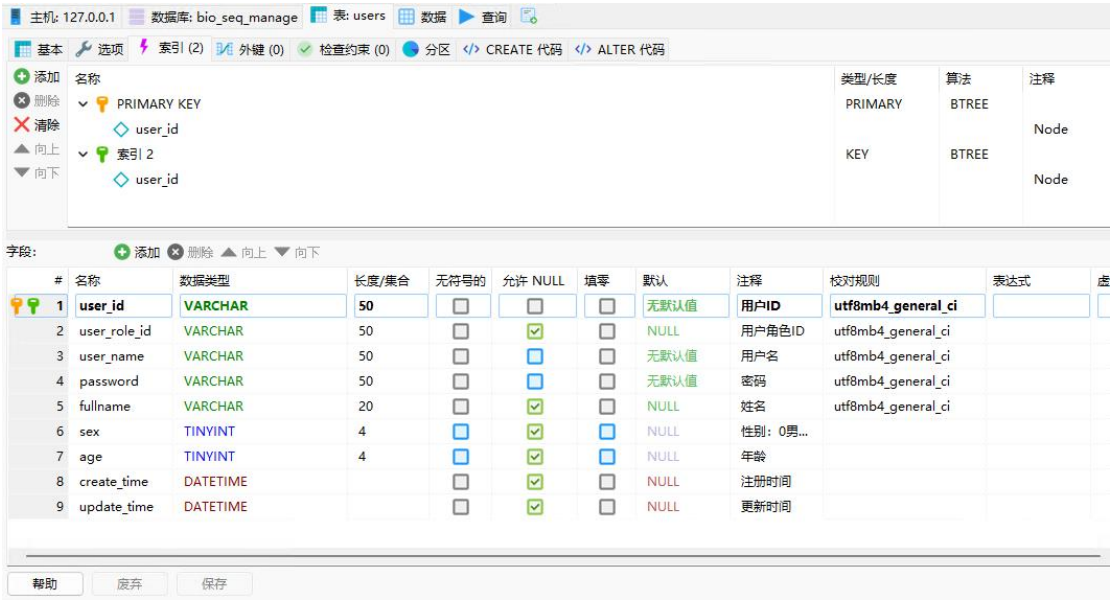


图 5-3-1 users 用户数据表

创建用户数据表的 SQL 语句，具体如下：

```
    CREATE TABLE `users` (
     `user_id` VARCHAR(50) NOT NULL COMMENT '用户 ID' COLLATE 'utf8mb4_general_ci',
     `user_role_id` VARCHAR(50) NULL DEFAULT NULL COMMENT '用户角色 ID' COLLATE
'utf8mb4_general_ci',
```

```
  `user_name` VARCHAR(50) NOT NULL COMMENT '用户名' COLLATE 'utf8mb4_general_ci',
  `password` VARCHAR(50) NOT NULL COMMENT '密码' COLLATE 'utf8mb4_general_ci',
  `fullname` VARCHAR(20) NULL DEFAULT NULL COMMENT '姓名' COLLATE 'utf8mb4_general_ci',
  `sex` TINYINT(4) NULL DEFAULT NULL COMMENT '性别：0 男，1 女',
  `age` TINYINT(4) NULL DEFAULT NULL COMMENT '年龄',
  `create_time` DATETIME NULL DEFAULT NULL COMMENT '注册时间',
  `update_time` DATETIME NULL DEFAULT NULL COMMENT '更新时间',
  PRIMARY KEY (`user_id`) USING BTREE,
  INDEX `索引 2` (`user_id`) USING BTREE
)COMMENT='用户表' COLLATE='utf8mb4_general_ci' ENGINE=InnoDB;
```

2. 用户角色数据表

创建 user_role 用户角色数据表，如图 5-3-2 所示。

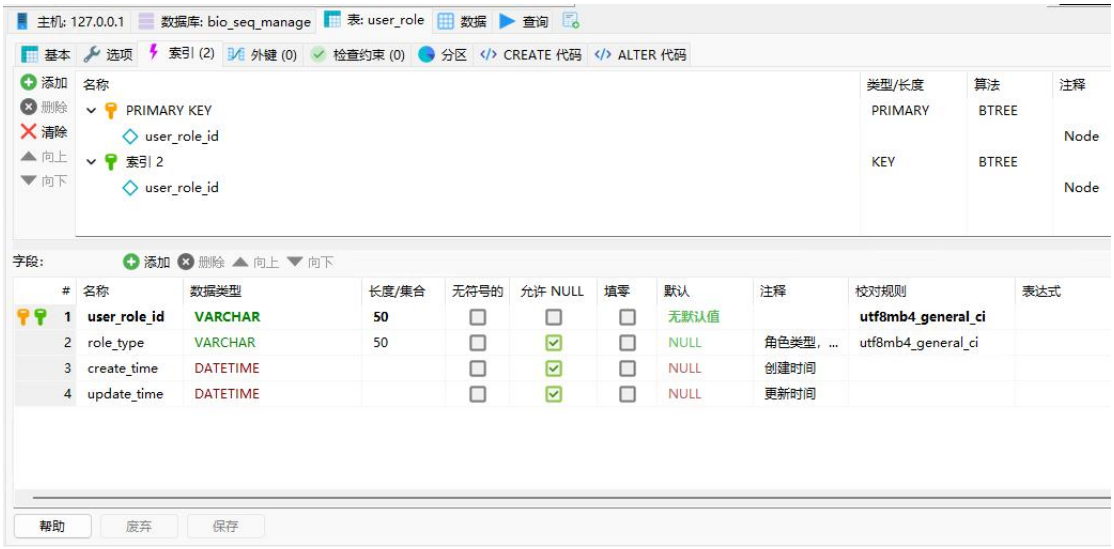


图 5-3-2 user_role 用户角色数据表

创建用户数据表的 SQL 语句，具体如下：

```
  CREATE TABLE `user_role` (
   `user_role_id` VARCHAR(50) NOT NULL COLLATE 'utf8mb4_general_ci',
   `role_type` VARCHAR(50) NULL DEFAULT NULL COMMENT '角色类型，admin：管理员，normal：
普通用户' COLLATE 'utf8mb4_general_ci',
   `create_time` DATETIME NULL DEFAULT NULL COMMENT '创建时间',
   `update_time` DATETIME NULL DEFAULT NULL COMMENT '更新时间',
   PRIMARY KEY (`user_role_id`) USING BTREE,
   INDEX `索引 2` (`user_role_id`) USING BTREE
  )COMMENT='用户角色表' COLLATE='utf8mb4_general_ci' ENGINE=InnoDB;
```

3. 用户表与用户角色表的关系

用户表与用户角色表的关系是多对一，既一个用户对应一个用户角色，一个用户角色对应多个用户。用户表的 user_role_id 字段是用户角色表的外键。如图 5-3-3 所示。

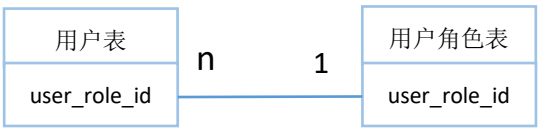


图 5-3-3 用户表与用户角色表的关系

四、系统项目的其他数据库表

1. 样本数据表

创建 sample 样本数据表，如图 5-3-4 所示。

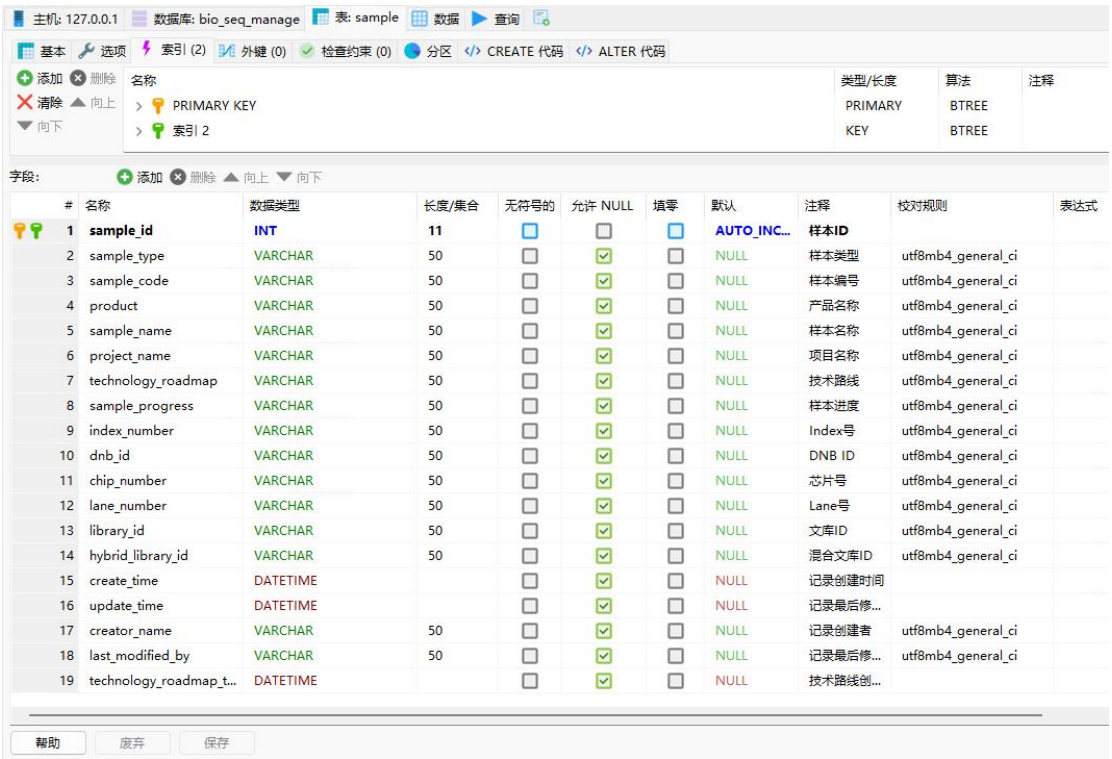


图 5-3-4 sample 样本数据表

创建样本数据表的 SQL 语句，具体如下：

```
    CREATE TABLE `sample` (
     `sample_id` INT(11) NOT NULL AUTO_INCREMENT COMMENT '样本 ID',
     `sample_type` VARCHAR(50) NULL DEFAULT NULL COMMENT '样本类型' COLLATE
'utf8mb4_general_ci',
     `sample_code` VARCHAR(50) NULL DEFAULT NULL COMMENT '样本编号' COLLATE
'utf8mb4_general_ci',
     `product` VARCHAR(50) NULL DEFAULT NULL COMMENT '产品名称' COLLATE
'utf8mb4_general_ci',
     `sample_name` VARCHAR(50) NULL DEFAULT NULL COMMENT '样本名称' COLLATE
'utf8mb4_general_ci',
     `project_name` VARCHAR(50) NULL DEFAULT NULL COMMENT '项目名称' COLLATE
'utf8mb4_general_ci',
     `technology_roadmap` VARCHAR(50) NULL DEFAULT NULL COMMENT '技术路线' COLLATE
'utf8mb4_general_ci',
     `sample_progress` VARCHAR(50) NULL DEFAULT NULL COMMENT '样本进度' COLLATE
'utf8mb4_general_ci',
     `index_number` VARCHAR(50) NULL DEFAULT NULL COMMENT 'Index 号' COLLATE
'utf8mb4_general_ci',
     `dnb_id` VARCHAR(50) NULL DEFAULT NULL COMMENT 'DNB ID' COLLATE 'utf8mb4_general_ci',
     `chip_number` VARCHAR(50) NULL DEFAULT NULL COMMENT '芯片号' COLLATE
'utf8mb4_general_ci',
     `lane_number` VARCHAR(50) NULL DEFAULT NULL COMMENT 'Lane 号' COLLATE
'utf8mb4_general_ci',
     `library_id` VARCHAR(50) NULL DEFAULT NULL COMMENT '文库 ID' COLLATE
'utf8mb4_general_ci',
     `hybrid_library_id` VARCHAR(50) NULL DEFAULT NULL COMMENT '混合文库 ID' COLLATE
'utf8mb4_general_ci',
     `create_time` DATETIME NULL DEFAULT NULL COMMENT '记录创建时间',
     `update_time` DATETIME NULL DEFAULT NULL COMMENT '记录最后修改时间',
     `creator_name` VARCHAR(50) NULL DEFAULT NULL COMMENT '记录创建者' COLLATE
'utf8mb4_general_ci',
     `last_modified_by` VARCHAR(50) NULL DEFAULT NULL COMMENT '记录最后修改者' COLLATE
'utf8mb4_general_ci',
     `technology_roadmap_time` DATETIME NULL DEFAULT NULL COMMENT '技术路线创建时间',
     PRIMARY KEY (`sample_id`) USING BTREE,
     INDEX `索引 2` (`sample_id`) USING BTREE
    )
    COMMENT='样本'
    COLLATE='utf8mb4_general_ci'
    ENGINE=InnoDB
    ;
```

2. 任务数据表

创建 task 任务数据表，如图 5-3-5 所示。

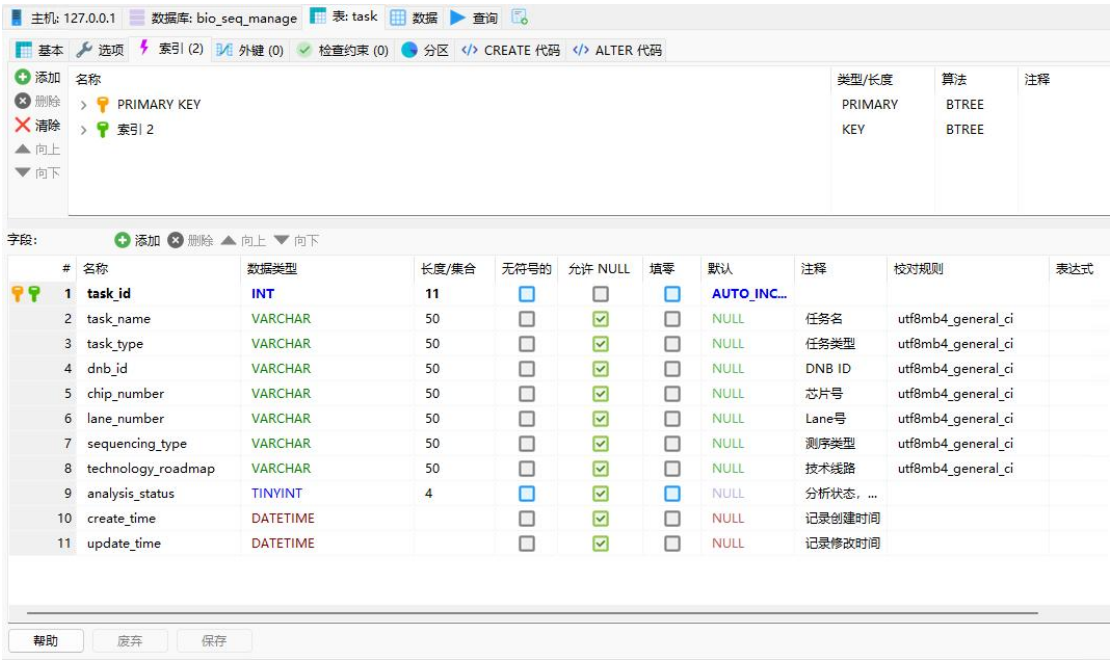


图 5-3-5 task 任务数据表

创建任务数据表的 SQL 语句，具体如下：

```
    CREATE TABLE `task` (
     `task_id` INT(11) NOT NULL AUTO_INCREMENT,
     `task_name` VARCHAR(50) NULL DEFAULT NULL COMMENT '任务名' COLLATE
'utf8mb4_general_ci',
     `task_type` VARCHAR(50) NULL DEFAULT NULL COMMENT '任务类型' COLLATE
'utf8mb4_general_ci',
     `dnb_id` VARCHAR(50) NULL DEFAULT NULL COMMENT 'DNB ID' COLLATE 'utf8mb4_general_ci',
     `chip_number` VARCHAR(50) NULL DEFAULT NULL COMMENT '芯片号' COLLATE
'utf8mb4_general_ci',
     `lane_number` VARCHAR(50) NULL DEFAULT NULL COMMENT 'Lane 号' COLLATE
'utf8mb4_general_ci',
     `sequencing_type` VARCHAR(50) NULL DEFAULT NULL COMMENT '测序类型' COLLATE
'utf8mb4_general_ci',
     `technology_roadmap` VARCHAR(50) NULL DEFAULT NULL COMMENT '技术线路' COLLATE
'utf8mb4_general_ci',
     `analysis_status` TINYINT(4) NULL DEFAULT NULL COMMENT '分析状态，0 未完成，1 完成',
     `create_time` DATETIME NULL DEFAULT NULL COMMENT '记录创建时间',
     `update_time` DATETIME NULL DEFAULT NULL COMMENT '记录修改时间',
     PRIMARY KEY (`task_id`) USING BTREE,
```

```
 INDEX `索引 2` (`task_id`) USING BTREE
)
COMMENT='任务'
COLLATE='utf8mb4_general_ci'
ENGINE=InnoDB
;
```

3. 任务与样本关联数据表

创建 task_sample 任务与样本关联数据表，如图 5-3-6 所示。

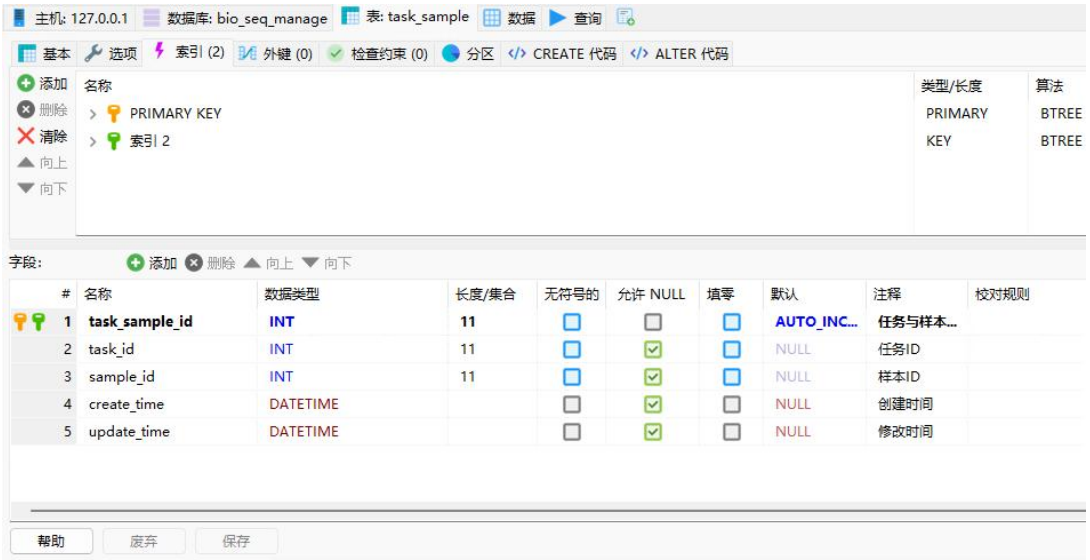


图 5-3-6 task_sample 任务与样本关联数据表

创建任务与样本关联数据表的 SQL 语句，具体如下：

```
CREATE TABLE `task_sample` (
 `task_sample_id` INT(11) NOT NULL AUTO_INCREMENT COMMENT '任务与样本的关联 ID',
 `task_id` INT(11) NULL DEFAULT NULL COMMENT '任务 ID',
 `sample_id` INT(11) NULL DEFAULT NULL COMMENT '样本 ID',
 `create_time` DATETIME NULL DEFAULT NULL COMMENT '创建时间',
 `update_time` DATETIME NULL DEFAULT NULL COMMENT '修改时间',
 PRIMARY KEY (`task_sample_id`) USING BTREE,
 INDEX `索引 2` (`task_sample_id`) USING BTREE
)
COMMENT='任务与样本的关联'
COLLATE='utf8mb4_general_ci'
ENGINE=InnoDB
;
```

任务实施

Step1 表规约认知

Step2 索引规约认知

Step3 创建用户数据表

Step4 创建用户角色数据表

Step5 创建系统项目的其他数据表

活动四 在工程项目中创建用户管理及用户登录所需要的包与类文件

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们根据分层结构建设需要的包以及用户管理和用户登录功能需要的类文件！

任务目标

1. 对实体类的认知能完成实体类的代码编写。
2. 对数据访问层以及 service 层的认知能完成用户数据的集成封装。

任务分析

1. 在 Java 开发中有哪些分层结构？为什么要分层？
2. 实现用户登录及管理功能需要哪些接口与类文件？
3. 根据系统分析与设计还需要哪些接口与类文件？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、分层结构

1. Java 分层开发概述

在项目中我们通常都采用分层的架构进行开发，这种分层结构的主要目的在于实现应用程序的模块化和解耦，简化程序的开发和维护，便于团队协作，提高代码的可重用性和可维护性。

Java 开发中我们按程序的职责和功能划分，分别是 Controller 控制层、Service 业务逻辑

层、Entity 实体层、Dao 数据访问层。每个层我们建一个包，然后在对应的包里创建类文件实现相关功能。

2. Controller 页面控制层

Controller 层是 Java Web 应用程序里面的控制层，主要负责接收客户端发送的请求、调度 Service 层的各个方法，并将处理的结果返回给客户端。同时 Controller 层也可以对请求参数进行校验，防止错误数据的处理和提交。

Controller 层主要使用 Spring MVC 相关的注解来映射请求和处理请求结果。例如，@Controller 注解用于标识该类为控制器类，@RequestMapping 注解用于配置基于请求 URL 的映射，@ResponseBody 注解用于将返回结果序列化成 JSON 等格式返回给客户端。

3. Service 业务逻辑层

Service 层是 Java Web 应用程序里面的业务逻辑层，主要负责处理业务逻辑、实现数据校验、事务控制、权限控制等方面。Service 层通过调用 Dao/Mapper 层的接口来操作数据库，以完成具体的业务目标。

为了能够实现业务逻辑的独立性和可重用性，Service 层通常采用接口+实现类的方式进行开发。Service 层主要使用@Autowired 注解来注入 Dao/Mapper 层的实例，并使用@Transactional 注解来定义事务。

4. Dao 数据访问接口层

Dao 层是 Java Web 应用程序里面的数据访问层，主要负责与数据库进行交互，进行数据读取、写入、修改、删除等操作。Dao 层通常使用 Spring Data JPA、MyBatis 等开源框架进行 ORM 操作。

在 Dao 层中通常有一个 Java 接口，用于定义 SQL 语句，以完成数据的增、删、改、查操作。Dao 层中的方法会由 Service 层来调用，通过调用 SQL 语句来完成与数据库的交互。

5. Entity 实体层

Entity 层用于定义实体、getter()和 setter()方法。实体中所包含的所有属性，与数据库中的字段是保持一致的。

二、创建用户登录及管理需要的接口与类文件

1. 用户实体类

在 Entity 实体层创建用户实体类 Users.java 文件，在实体类名上加入两个注解@Entity 和@Table(name = "config")，并且实现 Serializable 接口，具体代码如下：

```
package com.bioseqmanage.entity;

import java.io.Serializable;
import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="users")
public class Users implements Serializable {

 private static final long serialVersionUID = 1L;



 @Id
 @Column(name="user_id")
 private String userId;
 @Column(name="user_roleId")
 private String userRoleId;
 @Column(name="user_name")
 private String userName;
 @Column(name="password")
 private String password;
 @Column(name="fullname")
 private String fullname;
 @Column(name="sex")
 private int sex;
 @Column(name="age")
 private int age;
 @Column(name="create_time")
 private Date createTime;
 @Column(name="update_time")
```

```
    private Date updateTime;

    /**
     * 用户 ID
     * @return
     */
    public String getUserId() {
              return userId;
    }
    /**
     * 用户 ID
     * @param userId
     */
    public void setUserId(String userId) {
              this.userId = userId;
    }
    /**
     * 用户角色 ID
     * @return
     */
    public String getUserRoleId() {
              return userRoleId;
    }
    /**
     * 用户角色 ID
     * @param userRoleId
     */
    public void setUserRoleId(String userRoleId) {
              this.userRoleId = userRoleId;
    }
    /**
     * 用户名
     * @return
     */
    public String getUserName() {
              return userName;
    }
    /**
     * 用户名
     * @param userName
     */
    public void setUserName(String userName) {
              this.userName = userName;
    }
```

```
    /**
     * 密码
     * @return
     */
    public String getPassword() {
              return password;
    }
    /**
     * 密码
     * @param password
     */
    public void setPassword(String password) {
              this.password = password;
    }
    /**
     * 姓名
     * @return
     */
    public String getFullname() {
              return fullname;
    }
    /**
     * 姓名
     * @param fullname
     */
    public void setFullname(String fullname) {
              this.fullname = fullname;
    }
    /**
     * 性别：0 男，1 女
     * @return
     */
    public int getSex() {
              return sex;
    }
    /**
     * 性别：0 男，1 女
     * @param sex
     */
    public void setSex(int sex) {
              this.sex = sex;
    }
    /**
     * 年龄
```

```
     * @return
     */
    public int getAge() {
              return age;
    }
    /**
     * 年龄
     * @param age
     */
    public void setAge(int age) {
              this.age = age;
    }
    /**
     * 注册时间
     * @return
     */
    public Date getCreateTime() {
              return createTime;
    }
    /**
     * 注册时间
     * @param createTime
     */
    public void setCreateTime(Date createTime) {
              this.createTime = createTime;
    }
    /**
     * 更新时间
     * @return
     */
    public Date getUpdateTime() {
              return updateTime;
    }
    /**
     * 更新时间
     * @param updateTime
     */
    public void setUpdateTime(Date updateTime) {
              this.updateTime = updateTime;
    }


}
```

2. 用户数据访问接口

在 Dao 数据访问层创建数据方法接口，接口中要加入一个注解@Repository，并且继承 JpaRepository 接口，JpaRepository 需要输入实体类类型以及 ID 类型。具体代码如下：

```
package com.bioseqmanage.dao;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.bioseqmanage.entity.Users;

@Repository
public interface UsersDao extends JpaRepository<Users, String> {

 Optional<Users> findById(String userId);

}
```

3. 用户业务逻辑接口及实现类

（1）创建用户业务逻辑接口

在业务逻辑层里我们可以首先实现一些基本的功能，具体有不同功能需求的可以再加入实现。基本的工具有：判断记录是否存在、添加数据、更新数据、删除数据、获取一个实体数据和过去列表数据等功能。用户业务逻辑接口具体代码如下：

```
package com.bioseqmanage.service;

import java.util.List;

import com.bioseqmanage.entity.Users;

public interface UsersService {

 /**
  * 是否存在记录
```

```
     * @param userId 用户 ID
     * @return
     */
    boolean Exists(String userId);
    /**
     * 添加一条记录
     * @param model
     * @return
     */
    boolean Add(Users model);
    /**
     * 更新一条数据
     * @param model
     * @return
     */
    boolean Update(Users model);
    /**
     * 删除一条数据
     * @param userId 用户 ID
     * @return
     */
    boolean Delete(String userId);
    /**
     * 获取一条数据
     * @param userId
     * @return
     */
    Users GetModel(String userId);
    /**
     * 获取数据列表
     * @param top 指定获取的条数，为 0 时获取全部数据
     * @return
     */
    List<Users> GetTopList(int top);

}
```

(2) 创建用户业务逻辑实现类

业务逻辑实现类主要使用@Autowired 注解来注入 Dao 层的实例，然后实现业务逻辑接口的功能。用户业务逻辑实现类具体代码如下：

```
package com.bioseqmanage.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import com.bioseqmanage.dao.UsersDao;
import com.bioseqmanage.entity.Users;
import com.bioseqmanage.service.UsersService;

@Service
public class UserServiceImpl implements UsersService {

 @Autowired
 private UsersDao userDao;

 @Override
 public boolean Exists(String userId) {
          return userDao.existsById(userId);
 }

 @Override
 public boolean Add(Users model) {
          Users user = userDao.save(model);
          if(user != null) {
                    return true;
          }
          else {
                    return false;
          }
 }

 @Override
 public boolean Update(Users model) {
          if(!userDao.existsById(model.getUserId())) {
                    return false;
          }
          userDao.save(model);
          return true;
```

```
    }

    @Override
    public boolean Delete(String userId) {
            userDao.deleteById(userId);
            return true;
    }

    @Override
    public Users GetModel(String userId) {
            return userDao.findById(userId).get();
    }

    @Override
    public List<Users> GetTopList(int top) {
            Sort sort = Sort.by("createTime").descending();
            Pageable pageable = PageRequest.of(0, top, sort);
            return userDao.findAll(pageable).getContent();
    }

}
```

三、创建其他功能的接口与类文件

1. 实体类

（1）样本实体类

```
package com.bioseqmanage.entity;

import java.io.Serializable;
import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="sample")
public class Sample implements Serializable {

    private static final long serialVersionUID = 1L;
```

```
@Id
@Column(name="sample_id")
private int sampleId;
@Column(name="sample_type")
private String sampleType;
@Column(name="sample_code")
private String sampleCode;
@Column(name="product")
private String product;
@Column(name="sample_name")
private String sampleName;
@Column(name="project_name")
private String projectName;
@Column(name="technology_roadmap")
private String technologyRoadmap;
@Column(name="sample_progress")
private String sampleProgress;
@Column(name="index_number")
private String indexNumber;
@Column(name="dnb_id")
private String dnbId;
@Column(name="chip_number")
private String chipNumber;
@Column(name="lane_number")
private String laneNumber;
@Column(name="library_id")
private String libraryId;
@Column(name="hybrid_library_id")
private String hybridLibraryId;
@Column(name="create_time")
private Date createTime;
@Column(name="update_time")
private Date updateTime;
@Column(name="creator_name")
private String creatorName;
@Column(name="last_modified_by")
private String lastModifiedBy;
@Column(name="technology_roadmap_time")
private Date technologyRoadmapTime;


/**
 * 样本 ID
 * @return
 */
public int getSampleId() {
          return sampleId;
}
/**
 * 样本 ID
 * @param sampleId
 */
public void setSampleId(int sampleId) {
          this.sampleId = sampleId;
}
/**
 * 样本类型
```

```
     * @return
     */
    public String getSampleType() {
              return sampleType;
    }
    /**
     * 样本类型
     * @param sampleType
     */
    public void setSampleType(String sampleType) {
              this.sampleType = sampleType;
    }
    /**
     * 样本编号
     * @return
     */
    public String getSampleCode() {
              return sampleCode;
    }
    /**
     * 样本编号
     * @param sampleCode
     */
    public void setSampleCode(String sampleCode) {
              this.sampleCode = sampleCode;
    }
    /**
     * 产品名称
     */
    public String getProduct() {
              return product;
    }
    /**
     * 产品名称
     * @param product
     */
    public void setProduct(String product) {
              this.product = product;
    }
    /**
     * 样本名称
     * @return
     */
    public String getSampleName() {
              return sampleName;
    }
    /**
     * 样本名称
     * @param sampleName
     */
    public void setSampleName(String sampleName) {
              this.sampleName = sampleName;
    }
    /**
     * 项目名称
     * @return
     */
    public String getProjectName() {
```

```
            return projectName;
    }
    /**
     * 项目名称
     * @param projectName
     */
    public void setProjectName(String projectName) {
            this.projectName = projectName;
    }
    /**
     * 技术路线
     * @return
     */
    public String getTechnologyRoadmap() {
            return technologyRoadmap;
    }
    /**
     * 技术路线
     * @param technologyRoadmap
     */
    public void setTechnologyRoadmap(String technologyRoadmap) {
            this.technologyRoadmap = technologyRoadmap;
    }
    /**
     * 样本进度
     * @return
     */
    public String getSampleProgress() {
            return sampleProgress;
    }
    /**
     * 样本进度
     * @param sampleProgress
     */
    public void setSampleProgress(String sampleProgress) {
            this.sampleProgress = sampleProgress;
    }
    /**
     * Index 号
     * @return
     */
    public String getIndexNumber() {
            return indexNumber;
    }
    /**
     * Index 号
     * @param indexNumber
     */
    public void setIndexNumber(String indexNumber) {
            this.indexNumber = indexNumber;
    }
    /**
     * DNB ID
     * @return
     */
    public String getDnbId() {
            return dnbId;
    }
```

```
    /**
     * DNB ID
     * @param dnbId
     */
    public void setDnbId(String dnbId) {
              this.dnbId = dnbId;
    }
    /**
     * 芯片号
     * @return
     */
    public String getChipNumber() {
              return chipNumber;
    }
    /**
     * 芯片号
     * @param chipNumber
     */
    public void setChipNumber(String chipNumber) {
              this.chipNumber = chipNumber;
    }
    /**
     * Lane 号
     * @return
     */
    public String getLaneNumber() {
              return laneNumber;
    }
    /**
     * Lane 号
     * @param laneNumber
     */
    public void setLaneNumber(String laneNumber) {
              this.laneNumber = laneNumber;
    }
    /**
     * 文库 ID
     * @return
     */
    public String getLibraryId() {
              return libraryId;
    }
    /**
     * 文库 ID
     * @param libraryId
     */
    public void setLibraryId(String libraryId) {
              this.libraryId = libraryId;
    }
    /**
     * 混合文库 ID
     * @return
     */
    public String getHybridLibraryId() {
              return hybridLibraryId;
    }
    /**
     * 混合文库 ID
```

```
     * @param hybridLibraryId
     */
    public void setHybridLibraryId(String hybridLibraryId) {
              this.hybridLibraryId = hybridLibraryId;
    }
    /**
     * 记录创建时间
     * @return
     */
    public Date getCreateTime() {
              return createTime;
    }
    /**
     * 记录创建时间
     * @param createTime
     */
    public void setCreateTime(Date createTime) {
              this.createTime = createTime;
    }
    /**
     * 记录最后修改时间
     * @return
     */
    public Date getUpdateTime() {
              return updateTime;
    }
    /**
     * 记录最后修改时间
     * @param updateTime
     */
    public void setUpdateTime(Date updateTime) {
              this.updateTime = updateTime;
    }
    /**
     * 记录创建者
     * @return
     */
    public String getCreatorName() {
              return creatorName;
    }
    /**
     * 记录创建者
     * @param creatorName
     */
    public void setCreatorName(String creatorName) {
              this.creatorName = creatorName;
    }
    /**
     * 记录最后修改者
     * @return
     */
    public String getLastModifiedBy() {
              return lastModifiedBy;
    }
    /**
     * 记录最后修改者
     * @param lastModifiedBy
     */
```

```
    public void setLastModifiedBy(String lastModifiedBy) {
            this.lastModifiedBy = lastModifiedBy;
    }
    /**
     * 技术路线创建时间
     * @return
     */
    public Date getTechnologyRoadmapTime() {
            return technologyRoadmapTime;
    }
    /**
     * 技术路线创建时间
     * @param technologyRoadmapTime
     */
    public void setTechnologyRoadmapTime(Date technologyRoadmapTime) {
            this.technologyRoadmapTime = technologyRoadmapTime;
    }

}
```

(2) 任务实体类

```
package com.bioseqmanage.entity;

import java.io.Serializable;
import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="task")
public class Task implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="task_id")
    private int taskId;
    @Column(name="task_name")
    private String taskName;
    @Column(name="task_type")
    private String taskType;
    @Column(name="dnb_id")
    private String dnbId;
    @Column(name="chip_number")
    private String chipNumber;
    @Column(name="lane_number")
    private String laneNumber;
    @Column(name="sequencing_type")
    private String sequencingType;
    @Column(name="technology_roadmap")
```

```
    private String technologyRoadmap;
    @Column(name="analysis_status")
    private int analysisStatus;
    @Column(name="create_time")
    private Date createTime;
    @Column(name="update_time")
    private Date updateTime;

    /**
     * 任务 ID
     * @return
     */
    public int getTaskId() {
              return taskId;
    }
    /**
     * 任务 ID
     * @param taskId
     */
    public void setTaskId(int taskId) {
              this.taskId = taskId;
    }
    /**
     * 任务名
     * @return
     */
    public String getTaskName() {
              return taskName;
    }
    /**
     * 任务名
     * @param taskName
     */
    public void setTaskName(String taskName) {
              this.taskName = taskName;
    }
    /**
     * 任务类型
     * @return
     */
    public String getTaskType() {
              return taskType;
    }
    /**
     * 任务类型
     * @param taskType
     */
    public void setTaskType(String taskType) {
              this.taskType = taskType;
    }
    /**
     * DNB ID
     * @return
     */
    public String getDnbId() {
              return dnbId;
    }
    /**
```

```
     * DNB ID
     * @param dnbId
     */
    public void setDnbId(String dnbId) {
              this.dnbId = dnbId;
    }
    /**
     * 芯片号
     * @return
     */
    public String getChipNumber() {
              return chipNumber;
    }
    /**
     * 芯片号
     * @param chipNumber
     */
    public void setChipNumber(String chipNumber) {
              this.chipNumber = chipNumber;
    }
    /**
     * Lane 号
     * @return
     */
    public String getLaneNumber() {
              return laneNumber;
    }
    /**
     * Lane 号
     * @param laneNumber
     */
    public void setLaneNumber(String laneNumber) {
              this.laneNumber = laneNumber;
    }
    /**
     * 测序类型
     * @return
     */
    public String getSequencingType() {
              return sequencingType;
    }
    /**
     * 测序类型
     * @param sequencingType
     */
    public void setSequencingType(String sequencingType) {
              this.sequencingType = sequencingType;
    }
    /**
     * 技术路线
     * @return
     */
    public String getTechnologyRoadmap() {
              return technologyRoadmap;
    }
    /**
     * 技术路线
     * @param technologyRoadmap
```

```
     */
    public void setTechnologyRoadmap(String technologyRoadmap) {
            this.technologyRoadmap = technologyRoadmap;
    }
    /**
     * 分析状态
     * @return
     */
    public int getAnalysisStatus() {
            return analysisStatus;
    }
    /**
     * 分析状态
     * @param analysisStatus
     */
    public void setAnalysisStatus(int analysisStatus) {
            this.analysisStatus = analysisStatus;
    }
    /**
     * 记录创建时间
     * @return
     */
    public Date getCreateTime() {
            return createTime;
    }
    /**
     * 记录创建时间
     * @param createTime
     */
    public void setCreateTime(Date createTime) {
            this.createTime = createTime;
    }
    /**
     * 记录修改时间
     * @return
     */
    public Date getUpdateTime() {
            return updateTime;
    }
    /**
     * 记录修改时间
     * @param updateTime
     */
    public void setUpdateTime(Date updateTime) {
            this.updateTime = updateTime;
    }
}
```

(3) 任务与样本关联实体类

```
package com.bioseqmanage.entity;

import java.io.Serializable;
import java.util.Date;
```

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="task_sample")
public class TaskSample implements Serializable{

 private static final long serialVersionUID = 1L;

 @Id
 @Column(name="task_sample_id")
 private int taskSampleId;
 @Column(name="task_id")
 private int taskId;
 @Column(name="sample_id")
 private int sampleId;
 @Column(name="create_time")
 private Date createTime;
 @Column(name="update_time")
 private Date updateTime;


 /**
  * 任务与样本关联 ID
  * @return
  */
 public int getTaskSampleId() {
          return taskSampleId;
 }
 /**
  * 任务与样本关联 ID
  * @param taskSampleId
  */
 public void setTaskSampleId(int taskSampleId) {
          this.taskSampleId = taskSampleId;
 }
 /**
  * 任务 ID
  * @return
  */
 public int getTaskId() {
          return taskId;
 }
 /**
  * 任务 ID
  * @param taskId
  */
 public void setTaskId(int taskId) {
          this.taskId = taskId;
 }
 /**
  * 样本 ID
  * @return
  */
 public int getSampleId() {
          return sampleId;
```

```
    }
    /**
     * 样本 ID
     * @param sampleId
     */
    public void setSampleId(int sampleId) {
              this.sampleId = sampleId;
    }
    /**
     * 创建时间
     * @return
     */
    public Date getCreateTime() {
              return createTime;
    }
    /**
     * 创建时间
     * @param createTime
     */
    public void setCreateTime(Date createTime) {
              this.createTime = createTime;
    }
    /**
     * 更新时间
     * @return
     */
    public Date getUpdateTime() {
              return updateTime;
    }
    /**
     * 更新时间
     * @param updateTime
     */
    public void setUpdateTime(Date updateTime) {
              this.updateTime = updateTime;
    }

}
```

2. 数据接口

(1) 样本数据接口

接口代码如下：

```
package com.bioseqmanage.dao;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
import com.bioseqmanage.entity.Sample;

@Repository
public interface SampleDao extends JpaRepository<Sample, Integer>{

 Optional<Sample> findById(int sampleId);

}
```

(2) 任务数据接口

接口代码如下：

```
package com.bioseqmanage.dao;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.bioseqmanage.entity.Task;

@Repository
public interface TaskDao extends JpaRepository<Task, Integer> {

 Optional<Task> findById(int taskId);

}
```

(3) 任务与样本关联数据接口

接口代码如下：

```
package com.bioseqmanage.dao;

import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.bioseqmanage.entity.TaskSample;

@Repository
public interface TaskSampleDao extends JpaRepository<TaskSample, Integer> {

 Optional<TaskSample> findById(int taskSampleId);

}
```

3. 业务逻辑接口及实现类

（1）样本业务逻辑接口及实现类

接口代码如下：

```
package com.bioseqmanage.service;

import java.util.List;

import com.bioseqmanage.entity.Sample;

public interface SampleService {

 /**
  * 是否存在记录
  * @param sampleId 样本 ID
  * @return
  */
 boolean Exists(int sampleId);
 /**
  * 添加一条记录
  * @param model
  * @return
  */
 boolean Add(Sample model);
 /**
  * 更新一条数据
  * @param model
  * @return
  */
 boolean Update(Sample model);
 /**
  * 删除一条数据
  * @param sampleId 样本 ID
  * @return
  */
 boolean Delete(int sampleId);
 /**
  * 获取一条数据
  * @param sampleId 样本 ID
  * @return
  */
 Sample GetModel(int sampleId);
 /**
  * 获取数据列表
  * @param top 指定获取的条数，为 0 时获取全部数据
  * @return
  */
 List<Sample> GetTopList(int top);

}
```

实现类如下：

```
package com.bioseqmanage.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import com.bioseqmanage.dao.SampleDao;
import com.bioseqmanage.entity.Sample;
import com.bioseqmanage.service.SampleService;

@Service
public class SampleServiceImpl implements SampleService {

 @Autowired
 private SampleDao sampleDao;

 @Override
 public boolean Exists(int sampleId) {
          return sampleDao.existsById(sampleId);
 }

 @Override
 public boolean Add(Sample model) {
          Sample sample = sampleDao.save(model);
          if(sample != null) {
                    return true;
          }else {
                    return false;
          }
 }

 @Override
 public boolean Update(Sample model) {
          if(!sampleDao.existsById(model.getSampleId())) {
                    return false;
          }
          sampleDao.save(model);
          return true;
 }

 @Override
 public boolean Delete(int sampleId) {
          if(!sampleDao.existsById(sampleId)) {
                    return false;
          }
          sampleDao.deleteById(sampleId);
          return true;
 }

 @Override
```

```
    public Sample GetModel(int sampleId) {
            return sampleDao.findById(sampleId).get();
    }

    @Override
    public List<Sample> GetTopList(int top) {
            Sort sort = Sort.by("createTime").descending();
            Pageable pageable = PageRequest.of(0, top, sort);
            return sampleDao.findAll(pageable).getContent();
    }
}
```

(2) 任务业务逻辑接口及实现类

接口代码如下：

```
package com.bioseqmanage.service;

import java.util.List;

import com.bioseqmanage.entity.Task;

public interface TaskService {

    /**
     * 是否存在记录
     * @param taskId 任务 ID
     * @return
     */
    boolean Exists(int taskId);
    /**
     * 添加一条记录
     * @param model
     * @return
     */
    boolean Add(Task model);
    /**
     * 更新一条数据
     * @param model
     * @return
     */
    boolean Update(Task model);
    /**
     * 删除一条数据
     * @param taskId 任务 ID
     * @return
     */
    boolean Delete(int taskId);
    /**
     * 获取一条数据
     * @param taskId 任务 ID
     * @return
     */
    Task GetModel(int taskId);
```

```
    /**
     * 获取数据列表
     * @param top 指定获取的条数，为0时获取全部数据
     * @return
     */
    List<Task> GetTopList(int top);

}
```

实现类如下：

```
package com.bioseqmanage.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import com.bioseqmanage.dao.TaskDao;
import com.bioseqmanage.entity.Task;
import com.bioseqmanage.service.TaskService;

@Service
public class TaskServiceImpl implements TaskService {

    @Autowired
    private TaskDao taskDao;

    @Override
    public boolean Exists(int taskId) {
              return taskDao.existsById(taskId);
    }

    @Override
    public boolean Add(Task model) {
              Task task = taskDao.save(model);
              if(task != null) {
                        return true;
              }else {
                        return false;
              }
    }

    @Override
    public boolean Update(Task model) {
              if(!taskDao.existsById(model.getTaskId())) {
                        return false;
              }
              return true;
    }

    @Override
```

```
    public boolean Delete(int taskId) {
            taskDao.deleteById(taskId);
            return true;
    }

    @Override
    public Task GetModel(int taskId) {
            return taskDao.findById(taskId).get();
    }

    @Override
    public List<Task> GetTopList(int top) {
            Sort sort = Sort.by("createTime").descending();
            Pageable pageable = PageRequest.of(0, top, sort);
            return taskDao.findAll(pageable).getContent();
    }

}
```

(3) 任务与样本关联业务逻辑接口及实现类

接口代码如下：

```
package com.bioseqmanage.service;

import java.util.List;

import com.bioseqmanage.entity.TaskSample;

public interface TaskSampleService {

    /**
     * 是否存在记录
     * @param taskSampleId 任务与样本关联 ID
     * @return
     */
    boolean Exists(int taskSampleId);
    /**
     * 添加一条记录
     * @param model
     * @return
     */
    boolean Add(TaskSample model);
    /**
     * 更新一条数据
     * @param model
     * @return
     */
    boolean Update(TaskSample model);
    /**
     * 删除一条数据
     * @param taskSampleId 任务与样本关联 ID
     * @return
     */
```

```
 boolean Delete(int taskSampleId);
 /**
  * 获取一条数据
  * @param taskSampleId 任务与样本关联 ID
  * @return
  */
 TaskSample GetModel(int taskSampleId);
 /**
  * 获取数据列表
  * @param top 指定获取的条数，为 0 时获取全部数据
  * @return
  */
 List<TaskSample> GetTopList(int top);
}
```

实现类如下：

```
package com.bioseqmanage.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import com.bioseqmanage.dao.TaskSampleDao;
import com.bioseqmanage.entity.TaskSample;
import com.bioseqmanage.service.TaskSampleService;

@Service
public class TaskSampleServiceImpl implements TaskSampleService {

 @Autowired
 private TaskSampleDao taskSampleDao;

 @Override
 public boolean Exists(int taskSampleId) {
          return taskSampleDao.existsById(taskSampleId);
 }

 @Override
 public boolean Add(TaskSample model) {
          TaskSample taskSample = taskSampleDao.save(model);
          if(taskSample != null) {
                   return true;
          }else {
                   return false;
          }
 }

 @Override
 public boolean Update(TaskSample model) {
          if(!taskSampleDao.existsById(model.getSampleId())) {
```

```
                    return false;
            }
            taskSampleDao.save(model);
            return true;
    }

    @Override
    public boolean Delete(int taskSampleId) {
            taskSampleDao.deleteById(taskSampleId);
            return true;
    }

    @Override
    public TaskSample GetModel(int taskSampleId) {
            return taskSampleDao.findById(taskSampleId).get();
    }

    @Override
    public List<TaskSample> GetTopList(int top) {
            Sort sort = Sort.by("createTime").descending();
            Pageable pageable = PageRequest.of(0, top, sort);
            return taskSampleDao.findAll(pageable).getContent();
    }

}
```

任务实施

Step1 认知 Java 开发中的分层结构

Step2 创建用户登录及管理需要的接口与类文件

Step3 创建其他功能的接口与类文件

活动五 实现用户登录管理系统

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中的用户登录功能，没有登录的跳转到登录页面！

任务目标

1. 能通过读数据库的用户表进行查询并且判断用户的账号密码是否正确。
2. 能对查询出来的数据进行判断是否登录成功并且做出页面跳转处理。
3. 能通过过滤器 Filter 和 session 会话判断当前访问是否已经登录。

任务分析

1. Java 中如何定义变量？都有哪些类型？
2. Java 中都有哪些流程控制语句？
3. 如何使用过滤器 Filter 以及监听器 Listener？
4. Session 有什么作用？
5. Spring Boot JPA 中如何通过账号密码查找用户信息？
6. Java 中如何进行页面跳转？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、Java 基本语法

1. 基本数据类型

1）整型类型

Java 的整型数据类型有 byte、short、int、long，它们都是用来存储整数数值，但是它们的取值范围可不一样。

- byte 的取值范围：-128～127（-2 的 7 次方到 2 的 7 次方-1）
- short 的取值范围：-32768～32767（-2 的 15 次方到 2 的 15 次方-1）
- int 的取值范围：-2147483648～2147483647（-2 的 31 次方到 2 的 31 次方-1）
- long 的取值范围：-9223372036854774808～9223372036854774807（-2 的 63 次方到 2 的 63 次方-1）

由上可以看出 byte、short 的取值范围比较小，而 long 的取值范围最大的，所以占用的空间也是最多的。int 取值范围基本上可以满足我们的日常计算需求了，所以也是我们使用的最多的一个整型类型。

2）浮点类型

float 和 double 都是表示浮点型的数据类型，它们之间的区别在于精确度的不同。

- float（单精度浮点型）取值范围：3.402823e+38～1.401298e-45（e+38 表示乘以 10 的 38 次方，而 e-45 表示乘以 10 的负 45 次方）
- double（双精度浮点型）取值范围：1.797693e+308～4.9000000e-324（同上）
- double 类型比 float 类型存储范围更大，精度更高。

带小数点的字面量默认属于 double 类型，所以声明一个 float 类型的变量时，都要在数字后面加上"F"或"f"。

在 Java 中，对浮点型数据使用基本的加减乘除运算符，计算的数据可能不是完全精确的，有时候可能出现小数点后几位浮动。对于金融行业或者和钱有关的业务来说，这是不可接受的，当出现与金钱数值相关的场景，建议使用 BigDecimal 进行运算。

3）字符类型

char 有以下的初始化方式：

```
// 可以是汉字，因为是 Unicode 编码
char ch = 'a';
// 可以是十进制数、八进制数、十六进制数等等。
char ch = 1010;
// 可以用字符编码来初始化，如：'\0' 表示结束符，它的 ascll 码是 0，这句话的意思和 ch = 0 是一个意思。
char ch = '\0';
```

Java 是用 unicode 来表示字符，“中”这个中文字符在 unicode 就是两个字节。

unicode / gbk / gb2312 是两个字节，utf-8 是 3 个字节。

对于字符串（String），可以通过 String.getBytes(encoding) 方法，获取指定编码类型的 byte 数组。

4）布尔类型

boolean 型只有两个取值 true 和 false，它的默认值是 false。

对于布尔型占用的空间，得看 JVM 对于它的具体实现，有些 JVM 底层其实是使用 0 和 1 来表示 true 和 false，那么就是 4 字节。

2. 变量与常量

1）声明变量

声明变量就是告诉编译器这个变量的数据类型，这样编译器才知道需要配置多少空间给它，以及它能存放什么样的数据。在程序运行过程中，空间内的值是变化的，这个内存空间就称为变量。变量的命名必须是合法的标识符。内存空间内的值就是变量值。在声明变量时可以没有赋值，也可以是直接赋给初始值。

例如，声明一个整数类型变量和声明一个字符类型变量，代码如下：

```
int age;                    //声明 int 类型变量
char charl = 'r';           //声明 char 类型变量并赋值
```

对于变量的命名并不是随意的，应遵循以下几条规则：

- 变量名必须是一个有效的标识符。
- 变量名不可以使用 Java 中的关键字。
- 变量名不能重复。
- 应选择有意义的单词作为变量名。

2）声明常量

在程序运行过程中一直不会改变的量称为常量（Constant），通常也被称为“final 变量”。常量在整个程序中只能被赋值一次。在为所有的对象共享值时，常量是非常有用的。

在 Java 语言中声明一个常量，除了要指定数据类型，还需要通过 final 关键字进行限定。声明常量的标准语法如下:

```
final 数据类型 常量名称 [= 值]
```

常量名通常使用大写字母，但这并不是必须的。很多 Java 程序员使用大写字母表示常量，是为了清楚地表明正在使用常量。

例如，声明常量π（程序中用 PI 表示），代码如下：

```
final double PI = 3.1415926D;
```

当变量被 final 关键字修饰时，该变量就变成了常量，必须在定义时就设定它的初值，否则将会产生编译错误。

3）变量的有效范围

由于变量被定义出来后只是暂存在内存中，等到程序执行到某-一个点，该变量会被释放掉，也就是说变量有它的生命周期。因此，变量的有效范围是指程序代码能够访问该变量的区域，若超出该区域，则在编译时会出现错误。在程序中，一般会根据变量的 “有效范围”将变量分为“成员变量”和“局部变量”。

①成员变量

在类体中所声明的变量被称为成员变量，成员变量在整个类中都有效。类的成员变量又可分为两种，即静态变量和实例变量。例如下面这段代码：

```
class Demo{
    int x = 45;
    static int y = 90;
}
```

其中，x 为实例变量，y 为静态变量(也称类变量)。如果在成员变量的类型前面加上关键字 static，这样的成员变量称为静态变量。静态变量的有效范围可以跨类，甚至可达到整个应用程序之内。对于静态变量，除了能在声明它的类内存取，还能直接以“类名:静态变量”的方式在其他类内使用。

②局部变量

在类的方法体中声明的变量（方法内部定义，“{”与“}”之间的代码中声明的变量）称为局部变量。局部变量只在当前代码块中有效，也就是只能在“{”与“}”之内使用。

在类的方法中声明的变量，包括方法的参数，都属于局部变量。局部变量只在当前定义的方法内有效，不能用于类的其他方法中。局部变量的生命周期取决于方法，当方法被调用时，Java 虚拟机会为方法中的局部变量分配内存空间，当该方法的调用结束后，则会释放方法中局部变量占用的内存空间，局部变量也将会被销毁。

局部变量可与成员变量的名字相同，此时成员变量将被隐藏，即这个成员变量在此方法中暂时失效。

变量的有效范围如图 3-5-1 所示。

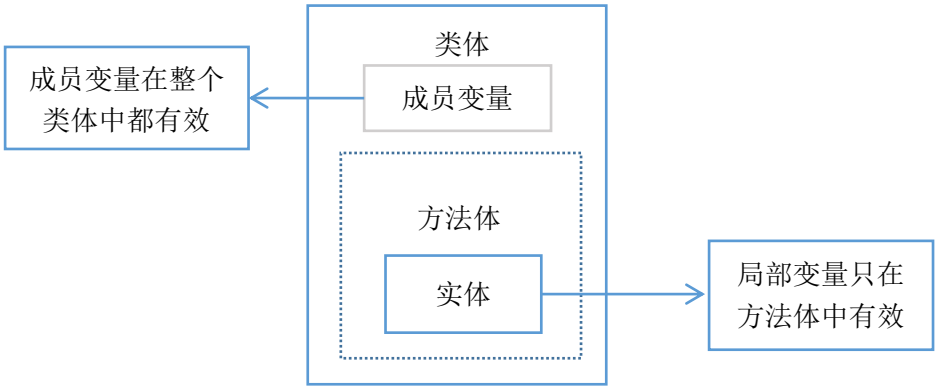


图 3-5-1 变量的有效范围

3. 运算符

运算符是一些特殊的符号，主要用于数学函数、一些类型的赋值语句和逻辑比较方面。Java 中提供了丰富的运算符，如赋值运算符、算术运算符、比较运算符等。

1）赋值运算符

赋值运算符以符号“=”表示，它是一个二元运算符（对两个操作数作处理），其功能是将右方操作数所含的值赋给左方的操作数。例如：

```
int a = 100;
```

该表达式是将 100 赋值给变量 ao 左方的操作数必须是一个变量，而右边的操作数则可以是任何表达式，包括变量(如 a、number)、常量(如 123、'book')、有效的表达式(如 45* 12)。

由于赋值运算符“=”处理时会先取得右方表达式处理后的结果，因此一个表达式中若含有两个以上的“=”运算符，会从最右方的“=”开始处理。

```
int a,b,c;
a = 15;
c = b = a + 4;
system.out.println("c 值为：" + c);
system.out.println("b 值为：" + b);
```

运行结果为：

```
c 值为：19;
b 值为：19;
```

2）算术运算符

Java 中的算术运算符主要有+（加）、-（减）、*（乘）、/（除）、%（求余），它们都是二元运算符，Java 中算术运算符的功能及使用方式如下所示。

运算符	说明	实例	结果
+	加	12.45f + 15	27.45
-	减	456 - 0.16	4.4
*	乘	5L * 12.45f	62.26
/	除	7 / 2	3
%	求余	12 % 10	2

其中，“+”和“-”运算符还可以作为数值的正负符号，如+5、-7。

3）自增和自减运算符

自增、自减运算符是单目运算符，可以放在操作元之前，也可以放在操作元之后。操作元必须是一个整型或浮点型变量。自增、自减运算符的作用是使变量的值增 1 或减 1。放在操作元前面的自增、自减运算符，会先将变量的值加 1 (减 1)，然后再使该变量参与表达式的运算。放在操作元后面的自增、自减运算符，会先使变量参与表达式的运算，然后再将该变量加 1 (减 1)。例如：

```
++a(--a)            //表示在使用变量 a 之前，先使 a 的值加(减) 1
a++(a--)            //表示在使用变量 a 之后，使 a 的值加(减) 1

b = ++a;            //先将 a 的值加 1,然后赋给 b,此时 a 值为 5，b 值为 5
```

```
int a = 4;
b = a++;                //先将 a 的值赋给 b，再将 a 的值变为 5，此时 a 值为 5，b 值为 4
```

4）比较运算符

比较运算符属于二元运算符，用于程序中的变量之间、变量和自变量之间以及其他类型的信息之间的比较。比较运算符的运算结果是 boolean 型。当运算符对应的关系成立时，运算结果为 true，否则为 false。所有比较运算符通常作为判断的依据用在条件语句中。比较运算符共有 6 个，如下所示。

运算符	作用	举例	操作数据	结果
>	比较左方是否大于右方	'a'> 'b'	整型、浮点型、字符型	false
<	比较左方是否小于右方	156< 456	整型、浮点型、字符型	true
==	比较左方是否等于右方	'c' = 'c'	基本数据类型、引用型	true
>=	比较左方是否大于等于右方	479 >= 426	整型、浮点型、字符型	true
<=	比较左方是否小于等于右方	12.45 <= 45.5	整型、浮点型、字符型	true
!=	比较左方是否不等于右方	'y' !=t	基本数据类型、引用型	true

5）逻辑运算符

返回类型为布尔型的表达式，如比较运算符，可以被组合在一起构成一个更复杂的表达式。这是通过逻辑运算符来实现的。逻辑运算符包括& (&&) (逻辑与)、|| (逻辑或)、! (逻辑非)。逻辑运算符的操作元必须是 boolean 型数据。在逻辑运算符中，除了“!”是一元运算符，其他都是二元运算符。如以下给出了逻辑运算符的用法和含义。

运算符	作用	举例	结合方向
&&、&	逻辑与	op1 && op2	从左到右
||	逻辑或	op1 || op2	从左到右
!	逻辑非	!op	从右到左

6）运算符的优先级

Java 中的表达式就是使用运算符连接起来的符合 Java 规则的式子。运算符的优先级决定了表达式中运算执行的先后顺序。通常，优先级由高到低的顺序依次是：

●增量和减量运算。

●算术运算。

●比较运算。
●逻辑运算。
●赋值运算。

如果两个运算有相同的优先级，那么左边的表达式要比右边的表达式先被处理。在 Java 中众多运算符特定的优先级，如下表所示。

优先级	描述	运算符
1	圆括号	()
2	正负号	+、-
3	一元运算符	++、--、!
4	乘除	*、/、%
5	加减	+、-
6	移位运算	>>、>>>、<<
7	比较大小	<、>、>=、<=
8	比较是否相等	==、!=
9	按位与运算	&
10	按位异或运算	^
11	按位或运算	|
12	逻辑与运算	&&
13	逻辑或运算	||
14	三元运算符	?:
15	赋值运算符	=

4. 数据类型转换

类型转换是将一个值从一种类型更改为另一种类型的过程。例如，可以将 String 类型的数据“457”转换为数值型，也可以将任意类型的数据转换为 String 类型。

如果从低精度数据类型向高精度数据类型转换，则永远不会溢出，并且总是成功的；而把高精度数据类型向低精度数据类型转换时，则会有信息丢失，有可能失败。

数据类型转换有两种方式，即隐式转换与显式转换。

1）隐式类型转换

从低级类型向高级类型的转换，系统将自动执行，程序员无须进行任何操作。这种类型的转换称为隐式转换。下列基本数据类型会涉及数据转换，不包括逻辑类型和字符类型。这些类型按精度从低到高排列的顺序为 byte < short < int < long < float < double。

例如，可以将 int 型变量直接赋值给 float 型变量，此时 int 型变量将隐式转换成 float 型变量。代码如下：

```
int x = 50;                //声明 int 类型变量 x
float y = x;               //将 x 赋值给 y，y 的值为 50.0
```

隐式转换也要遵循一定的规则，来解决在什么情况下将哪种类型的数据转换成另一种类型的数据。下表列出了各种数据类型隐式转换的一般规则。

操作数 1 的数据类型	操作数 2 的数据类型	转换后的数据类型
byte、short、char	int	int
byte、short、char、int	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

2）显示类型转换

当把高精度的变量的值赋给低精度的变量时，必须使用显式类型转换运算（又称强制类型转换）。语法如下：

```
(类型名)要转换的值
```

将高精度数值转换为低精度数字。代码如下：

```
int a = (int)45.23;                    //此时输出 a 的值为 45
long y = (long)456.6F;                 //此时输出 y 的值为 456
int b = (int)'d';                      //此时输出 b 的值为 100
```

执行显式类型转换时，可能会导致精度损失。除 boolean 类型外，其他基本类型都能以显示类型转换的方式实现转换。

5. 字符串

字符串是 Java 程序中经常处理的对象，如果字符串运用的不好，将会影响到程序的运行效率。在 Java 中字符串作为 String 类的实例来处理。以对象的方式处理字符串，将使字符串更加灵活、方便。了解字符串上可用的操作可以节省程序编写与维护的时间。

单个字符可以用 char 类型保存，多个字符组成的文本就需要保存在 String 对象中，String 通常被称为字符串，一个 String 对象最多可以保存（2 的 32 次方-1）个字节（占用 4GB 空间大小）的文本内容。

1）声明字符串

在 Java 语言中，字符串必须保存在一对双引号（“”）之内。例如：

```
String str = "字符串";
```

2）连接字符串

使用“+”运算符可以实现连接多个字符串的功能。“+”运算符可以连接多个 String 对象并产生一个新的 String 对象，如下代码：

```
String s1 = new String("春色绿千里");   //声明 String 对象 s1
String s2 = new String("马蹄香万家");   //声明 String 对象 s2
String s = s1 + "\n" + s2;              //将对象 s1、"\n"和对象 s2 连接并且将结果赋值给 s
System.out.println(s);                  //将 s 输出
```

字符串也可同其他基本数据类型进行连接。如果将字符串同其他数据类型数据进行连接，会将其他数据类型的数据直接转换成字符串。

```
int bookting = 4;
float practice = 2.5f;
System.out.println("我每天花费" + booktime + "小时看书；“ + practice + "小时上机练习");
```

运行结果如下：

```
我每天花费 4 小时看书；2.5 小时上机练习
```

3）获取字符串信息

使用 String 类的 length()方法可以获取声明的字符串对象的长度。语法如下：

```
str.length();
```

String 类提供了两种查找字符串的方法，即 indexOf()与 lastIndexOf()方法。这两个方法都允许字符串搜索指定条件的字符或字符串。indexOf()方法返回的是搜索的字符或字符串搜词出现的位置。lastIndexOf()方法返回的是搜索的字符或字符串最后一次出现的位置。

语法如下：

```
str1.indexOf(str2);
str1.lastIndexOf(str2);
```

获取指定索引位置的字符，使用 charAt()方法，它会将指定索引处的字符返回，语法如下：

```
str.charAt(int index)
```

4）字符串操作

String 类中包含了很多方法，可以满足在实际编程中对字符串进行操作的需要。
substring 方法，截取字符串，语法如下：

```
//返回的是从指定的索引位置开始截取直到该字符串结尾的字串
str.substring(int beginIndex)

//返回的是从字符串某一索引位置开始截取至某一索引位置结束的子串
Str.substring(int beginIndex, int endIndex)
```

trim()方法，返回去除前后空格字符串的副本，原字符串不改变，如下：

```
String str2 = str1.trim();
```

replace()方法，将指定的字符或字符串替换成新的字符或字符串，如下：

```
String str2 = str1.replace("a", "A");
```

startsWith()和 endsWith()方法分别是用于判断字符串是否以指定的内容开始或结束，它们的返回值都是 boolean 类型。语法如下：

```
str.startsWith(String prefix);
str.endsWith(String suffix);
```

equals()和 equalsIgnoreCase()方法都是用来判断字符串是否相等，前者区分大小写，后者不区分大小写，它们返回的值都是 boolean 类型。语法如下：

```
str.equals(String otherstr);
str.equalsIgnoreCase(String otherstr);
```

compareTo()方法，按字典顺序比较两个字符串，该比较基于字符串中的各个字符的 Unicode 值，按字典顺序将 String 对象表示的字符序列与参数字符串所表示的字符序列进行比较。如果按字典顺序此 String 对象位于参数字符串之前，则比较结果为一个负整数；如果按字典顺序此 String 对象位于参数字符串之后，则比较结果为一个正整数；如果这两个字符串相等，则结果为 0。语法如下：

```
Str.compareTo(String otherstr)
```

toLowerCase()和 toUpperCase()方法，前者是将字符串改成小写字母，后者是将字符串改成大写字母。

```
String str = new String("Oh My God");
String newStr1 = str.toLowerCase();
String newStr2 = str.toUpperCase();
```

split()方法，可以是字符串按指定的分割字符或字符串进行分割，分割的结果返回在字符串数组中。语法如下：

```
str.split(String sign)
str.split(String sign, int limit)     //限定拆分的次数
```

6. 代码注释

在 Java 源程序文件的任意位置都可以添加注释语句。注释中的文字 Java 编译器不进行编译，所以代码中的注释文字对程序不产生任何影响。在 Java 语言中提供 3 种添加注释的方法，分别是：单行注释、多行注释和文档注释。

“//”为单行注释标记，从符号“//”开始直到换行为止的所有内容均作为注释而被编译器忽略。

“/* */”多行注释标记，符号“/*”与“*/”之间的所有内容均为注释内容。注释中的内容可以换行。

“/** */”为文档注释标记。符号“/**” 与“*/”之间的内容均为文档注释内容。当文档注释出现在声明(如类的声明、类的成员变量的声明、类的成员方法的声明等)之前时，会被 JavaDoc 文档工具读取作为 JavadDoc 文档内容。除注释标记不同外，文档注释的格式与多行注释的格式相同。

二、Java 的流程控制条件语句

条件语句可根据不同的条件执行不同的语句。条件语句包括 if 条件语句与 switch 多分支语句。

1. if 条件语句

简单 if 条件语句，语法如下：

```
if(条件表达式){
    执行语句
}
```

If...else 语句，如果满足某种条件就进行某种处理，否则进行另一种处理，语法如下：

```
if(条件表达式){
    执行语句 1
}else{
    执行语句 2
}
```

If...else if 多分支语句，如果满足某种条件，就进行某种处理，否则如果满足另一种条件则执行另一种处理，语法如下：

```
if(条件表达式 1){
    执行语句 1
}else if(条件表达式 2){
    执行语句 2
}
...
else if(条件表达式 n){
    执行语句 n
}
```

2. switch 多分支语句

在程序中，一个常见的问题就是检测一个变量是否符合某个条件，如果不符合， 再用另一个值来检测，以此类推。当然，这种问题使用 if 条件语句也可以完成。这里用 switch

语句，语法如下：

```
switch(表达式){
case 常量值 1：
    语句块 1
    [break];
...
case 常量值 n：
    语句块 n
    [break];
default：
    语句块 n+1
    [break];
}
```

switch 语句中表达式的值必须是整型、字符型、字符串类型或枚举类型，常量值 1~n 的数据类型必须与表达式的值的类型相同。

switch 语句首先计算表达式的值，如果表达式的计算结果和某个 case 后面的常量值相同，则执行该 case 语句后的若干个语句直到遇到 break 语句为止。此时，如果该 case 语句中没有 break 语句，将继续执行后面 case 中的若干个语句，直到遇到 break 语句为止。若没有一.个常量的值与表达式的值相同，则执行 default 后面的语句。default 语句为可选的，如果它不存在，且 switch 语句中表达式的值不与任何 case 的常量值相同，switch 语句则不做任何处理。

三、过滤器 Filter

过滤器英文名称为 Filter，是处于客户端与服务器资源文件之间的一道过滤网，它是 Servlet 技术中最激动人心的技术之一。Web 开发人员通过 Filter 技术管理 Web 服务器的所有资源，例如 JSP、Servlet、静态图片文件或静态 HTML 文件等进行拦截，从而实现些特殊的功能。例如实现 URL 级别的权限访问控制、过滤敏感词汇、压缩响应信息等些高级功能。

在 Spring Boot 中使用 Filter 过滤器很简单，在入口类 App.java 文件中添加注解 @ServletComponentScan，使用该注解后只需要在过滤器响应的类中添加@WebFilter 注解即可自动注册，无需其他代码。

```
@SpringBootApplication
@ServletComponentScan
public class App
{
    public static void main( String[] args )
    {
      SpringApplication.run(App.class, args);
```

```
    }
}
```

四、监听器 Listener

监听器也叫 Listener，是 Servlet 的监听器，它可以用于监听 Web 应用中某些对象、信息的创建、销毁、增加、修改、删除等动作的发生，然后做出相应的响应处理。当范围对象的状态发生变化的时候，服务器自动调用监听器对象中的方法。常用于统计在线人数和在线用户，系统加载时进行信息初始化，统计网站的访问量，等等。

根据监听对象，可把监听器分为 3 类: ServletContext (对应 application)、HttpSession (对应 session)、ServletRequest (对应 request)。Application 在整个 Web 服务中只有一个，在 Web 服务关闭时销毁。Session 对应每个会话，在会话起始时创建，一端关闭会话时销毁。Request 对象是客户发送请求时创建的(一同创建的还有 Response)，用于封装请求数据，再次请求处理完毕时销毁。

根据监听事件分为监听对象创建与销毁，如 ServletContextListener、监听对象域中属性的增加和删除，如: HttpSessionListener、ServletRequestListener 和监听绑定到 Session 上的某个对象的状态，如 ServletContextAttributeListener、HttpSessionAttributeListener 和 ServletRequestAttributeListener 等。

在 Spring Boot 中使用 Listener 监听器和 Filter 基本一样，在入口类 App.java 文件中添加注解@ServletComponentScan，使用该注解后只需要在过滤器响应的类中添加@WebListener 注解即可自动注册，无需其他代码。

五、用户登录功能实现

1. 在用户的数据接口中加入方法

用户登录需要实则是从用户数据表中通过对用户名和密码的查找，同时都满足的时候就能登录成功，否则无法登录。在数据访问接口中加入对用户名和密码查询的方法，具体代码如下：

```
    Optional<Users> findByUserNameAndPassword(String userName, String
password);
```

参数 userName 和 password 对应的是数据库中的 user_name 和 password 两个字段，然后 findByUserNameAndPassword 则是同时查找用户名和密码这两个条件。通过此方法不需要写 SQL 语句即可实现查找。

2. 在用户的业务接口及实现类中加入方法

在业务逻辑接口层中的 UsersServic.java 中加入以下代码：

```
/**
 * 通过用户和密码获取用户
 * @param userName 用户名
 * @param password 密码
 * @return
 */
Users getUserByUserNameAndPassword(String userName, String password);
```

在业务逻辑实现层中的 UsersServiceImpl.java 中加入以下代码：

```
/**
 * 通过用户和密码获取用户
 * @param userName 用户名
 * @param password 密码
 * @return
 */
@Override
public Users getUserByUserNameAndPassword(String userName, String password) {

        Optional<Users> user = userDao.findByUserNameAndPassword(userName, password);
        if(user.isEmpty()) {
                return null;
        }else {
                return user.get();
        }
}
```

注意这里的变量 user 需要用 isEmpty 方法进行判断，否则直接使用可能会报错。

3. 在用户的控制类实现登录

在控制层中的 LoginController.java 中加入以下代码：

```
@PostMapping("/login")
@ResponseBody
public Map<String, Object> login(HttpServletRequest request){

        String userName = request.getParameter("userName");
        String password = request.getParameter("password");

        Map<String, Object> map = new HashMap<>();

        if(userName == null || "".equals(userName)) {
```

```
                    map.put("status", "fail");
                    map.put("msg", "用户名不能为空");
                    return map;
                }

                if(password == null || "".equals(password)) {
                    map.put("status", "fail");
                    map.put("msg", "密码不能为空");
                    return map;
                }

                Users user = usersService.getUserByUserNameAndPassword(userName,
password);

                if(user != null) {

                    HttpSession session = request.getSession();
                    session.setAttribute("loginUser", user);
                    map.put("status", "success");
                    map.put("msg", "登录成功");
                }
                else {
                    map.put("status", "fail");
                    map.put("msg", "登陆失败");
                }

                return map;

        }
```

代码中的两个注解@PostMapping("/login")和@ResponseBody，它们分别表示的是：

@PostMapping("/login")是“/login”路径下接收 POST 方式的访问，其他访问方式不进入这个方法。

@ResponseBody 是将方法的返回值，以特定的格式写入到 response 的 body 区域，进而将数据返回给客户端。当方法上面没有写 ResponseBody，底层会将方法的返回值封装为 ModelAndView 对象。如果返回值是字符串，那么直接将字符串写到客户端；如果是一个对象，会将对象转化为 JSON 串，然后写到客户端。

以上代码返回的类似是 Map<String, Object>，所以前端页面的内容是 JSON 字符串。

4. 在登录页面模板实现登录

登录页面中引入 JQuery 库，因为我们使用 JQuery 并且提交登录数据使用的是 AJAX。同时我们还需要引入已经集成好的 dialog.js。具体的登录脚本代码如下：

```
    <script>
    $(function(){
        $("#btnLogin").click(function(){
            var load = dialog.showLoading({
              autoClose: true,
              closeTime: 1000
          });
```

```
        $.ajax({
            url: "/login",
            type: "post",
            data:{
                username: $("#userName").val(),
                password: $("#password").val()
            },
            dataType:'json',
            success:function (data) {
                var msg = $("#msg");
                load.close();
                if(data.status == "success"){
                  //登录成功
                  window.location.href = "index";
                  msg.html("<span style='color:#ffffff;'>提示：登录成功</span>")
                 }else{
                  //登录失败，则给出提示信息
                  msg.html("<span style='color:#ffffff;'>提示："+data.msg+"</span>")
                 }
            }
        })
        });
});
</script>
```

dialog.showLoading 是我们集成的加载提示，在点击登录按钮是显示。$.ajax 是无刷新提交数据到后台。“window.location.href = ‘index’;”登录成功后跳转到系统首页。

5. 用户是否登录判断

用户访问每个系统页面我们都需要检查用户是否登录，没有登录则是去到登录页面。我们使用过滤器 Filter。创建 java 包“com.bioseqmanage.controller.filter”，然后创建 SystemFilter.java 类文件。这个类需要实现 Filter 接口，然后完成一些方法的重写。如图代码如下：

```
package com.bioseqmanage.filter;

import java.io.IOException;

import com.bioseqmanage.entity.Users;

import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.FilterConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
```

```
    @WebFilter
    public class SystemFilter implements Filter {

          @Override
          public void init(FilterConfig filterConfig) throws ServletException {

          }

          @Override
          public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain)
                                throws IOException, ServletException {

            HttpServletRequest servletRequest = (HttpServletRequest) request;
            HttpServletResponse servletResponse = (HttpServletResponse) response;
            HttpSession session = servletRequest.getSession();
            String vpath = request.getServletContext().getContextPath();

            // 获得用户请求的 URI
            String uri = servletRequest.getRequestURI();

            // 无需过滤页面或路径
            if(uri.indexOf("/login") > -1
                     || path.indexOf("/css") > -1
                     || path.indexOf("/image") > -1
                     || path.indexOf("/js") > -1){

                chain.doFilter(servletRequest, servletResponse);
                return;
            }

            /**
             * 获取 session 判断是否有登录
             */
            Users user = null;
          try{
                    user = (Users)session.getAttribute("loginUser");
          }catch(Exception e){
                    // TODO: handle exception
          }

          if(user == null) {
                    servletResponse.sendRedirect(vpath + "/login");
          }
          else {
                    chain.doFilter(servletRequest, servletResponse);
          }

          }

          @Override
          public void destroy() {

          }

    }
```

6. 退出登录

退出登录的实现主要是清除 Session 对话信息，具体代码如下：

```
        @RequestMapping(value={"/logout"})
          public void logout(HttpServletRequest request, HttpServletResponse response) throws
IOException {
          HttpSession session = request.getSession();
          session.removeAttribute("loginUser");
          response.sendRedirect(request.getServletContext().getContextPath()+"/login");
        }
```

任务实施

Step1 认知 Java 基本语法

Step2 认知 Java 的条件语句

Step3 认知过滤器和监听器

Step4 实现用户登录功能

活动六 实现样本管理模块的导入功能开发

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中样本数据的导入功能！

任务目标

1. 对文件的下载功能的实现认知然后完成样本模板文件的下载。
2. 能对导入的 Excel 文件里的数据进行读取并且能插入数据库保存。

任务分析

1. 如何实现文件的下载？
2. 如何上传 Excel 文件？
3. 如何读取 Excel 里的数据？
4. 如何把从 Excel 读取到的数据保存到数据库？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、Java 数组的使用

在 Java 中将数组看作一个对象，虽然基本数据类型不是对象，但由基本数据类型组成的数组却是对象。在程序设计中引入数组可以更有效的管理和处理数据，可根据数组的维数分为一维数组、二维数组等。

1. 一维数组

一维数组的声明如下：

```
数组元素类型 数组名字[];
数组元素类型[] 数组名字;
```

数组元素类型决定了数组的数据类型，它可以是 Java 中的人员数据类型，包括简单类型和组合类型。数组名字为一个合法标识符，符号“[]”指明该变量是一个数组类型。具体声明语句如下：

```
int arr[];   //或者 int[] arr;
```

在没有给数组分配空间，既还没有给数组指定长度，数组是不能访问的。具体如下：

```
int arr[];   //或者 int[] arr;
arr = new int[5];

//或者在定义的时候指定数组长度
int arr[] = new int[5];
```

数组在声明的同时还可以初始化，具体如下：

```
int arr[] = new int[]{1,2,3,4,5,8};   //创建 6 个元素的数组，既是长度为 6 的数组
//还可以这样初始化
int arr2[] = {1,2,3,4,5,8};
```

2. 二维数组

二维数组和一维数组相似，符号“[]”有几个就表示是几维数组。二维数组的声明化具体如下：

```
int arr[][];   //或者 int[][] arr;
```

为二维数组分配空间，具体如下：

```
int arr[][];   //或者 int[][] arr;
arr = new int[2][5];

//或者在定义的时候指定数组长度
```

```
int arr[] = new int[2][5];
```

二维数组在声明的同时也可以初始化，具体如下：

```
int arr[][] = new int[][]{{1,2,3,4,5,8},{3,5,6,5,6,7}};   //创建 2 行 6 列的二维的数组
//还可以这样初始化
int arr2[][] = {{1,2,3,4,5,8},{3,5,6,5,6,7}};
```

3. List 集合

Java.util 包中提供了一些集合类，这些集合类又被称为容器。提到容器不难想到数组。集合类与数组的不同之处是：数组的长度是固定的，集合的长度是可变的；数组用来存放基本类型的数据，集合用来存放对象的引用。常用的集合有 List 集合、Set 集合和 Map 集合，其中 List 集合与 Set 集合继承了 Collection 接口，各接口还提供了不同的实现类。这里我们讲解 List 集合的使用。

使用 List 集合时通常声明为 List 类型，可以通过不同的实现类来实例化集合。通过 ArrayList 类、LinkedList 类分别实例化 List 集合的代码如下：

```
List<E> list = new ArrayList<>();
List<E> list2 = new LinkedList<>();
```

上面代码中 E 可以是合法的 Java 数据类型。如果集合中的元素为字符串类型，那么 E 就修改为 String。

List 集合的使用举例代码如下：

```
List<String> list = new ArrayList<>();

list.add("字符串 1");            //添加元素，该元素为字符串
list.add("字符串 2");
list.add("字符串 3");

list.size();                    //集合中的长度
list.get(1);                    //获取集合中的元素
list.remove(2);                 //把指定索引的元素从集合中移除
```

二、Java 流程操作语句 for 循环

for 循环是 Java 程序设计中最有用的循环语句之一。一个 for 循环可以用来重复执行某条语句，直到某个条件得到满足。for 循环有两种，一种是传统的 for 语句，一种是 foreach 语句。

for 语句语法如下：

```
for(表达式 1;表达式 2;表达式 3){
     执行语句
}
```

foreach 语句语法如下：

```
for(元素类型 x: 遍历对象 obj){
     执行语句
}
```

举例说明它们的用法。先声明一个一维数组和一个 List 集合，然后分别使用 for 和 foreach 语句遍历它们，具体代码如下：

```
int arr[] = new int[]{1,2,3,4,5};
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.add(4);
list.add(5);

for(int i=0; i<arr.length; i++){
    System.out.println(arr[i]);
}

for(Integer x: arr){
     System.out.println(x);
}

for(int i=0; i<list.size(); i++){
    System.out.println(list.get(i));
}

for(Integer x: list){
```

```
        System.out.println(x);
    }
```

三、实现模板文件下载

在资源目录下的 static 创建文件夹 template，这个是目录和资源目录下的 templates 存放的东西不一样，static 目录下的 template 存放的是我们 Excel 模板，这些 Excel 模板是定义好了导入数据的格式，让我们可以正确的导入数据。

创建的 template 需要进行资源映射配置，代码如下：

```
    /**
     * 自定义资源映射
     */
    @Override
    protected void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/css/**").addResourceLocations("file:src/main/resources/static/css/");
        registry.addResourceHandler("/images/**").addResourceLocations("file:src/main/resources/static/images/");
        registry.addResourceHandler("/js/**").addResourceLocations("file:src/main/resources/static/js/");
        registry.addResourceHandler("/template/**").addResourceLocations("file:src/main/resources/static/template/");
    }
```

我们在该目录下放入 Excel 文件 sample_import_template.xlsx，此时我们就可以做一个下载链接，如下：

```
    <a href="/template/sample_import_template.xlsx" target="_blank">样本导入模板下载</a>
```

四、实现 Excel 文件的上传

在控制层的 SampleController.java 控制类文件中，添加以下上传方法：

```
        /**
         * 文件上传
         * @param file
         * @return
```

```
     * @throws IOException
     * @throws InvalidFormatException
     */
    @PostMapping("/sample/upload")
    @ResponseBody
    public Map<String, Object> upload(MultipartFile file) throws IOException,
InvalidFormatException {

        // 1.获取文件名
          String fileName = file.getOriginalFilename();
          Integer index = fileName.lastIndexOf(".");
          Integer pointIndex = fileName.length()-(fileName.length()-index);
          String name = fileName.substring(0,pointIndex);
          String etc = fileName.substring(pointIndex);

          // 2.给文件重命名，避免重复覆盖的情况
          String uuid = UUID.randomUUID().toString().replace("-","");
          fileName = uuid + etc;

          // 3.获取要存储的路径
          String targetPath = "";
          targetPath = this.getProjectPath();
          targetPath = targetPath + "/src/main/resources/static/upload/" + fileName;

          Map<String, Object> map = new HashMap<>();

          // 4.写入文件
          try{
              file.transferTo(new File(targetPath));
              List<Sample> list = readExcel(targetPath);

              saveData(list);

              map.put("status", "success");
              map.put("msg","上传成功");

          } catch (IOException e){
              e.printStackTrace();
              map.put("status","fail");
              map.put("msg","登录失败");
          }

          return map;
```

```
    }
```

MultipartFile 是 SpringMVC 提供简化文件流操作的接口，文件是以二进制流传递到后端的，然后需要我们自己转换为 File 类。使用 MultipartFile 接口中提供的实现方法，我们对文件处理的操作就会变得很便捷。MultipartFile 接口方法 transferTo(new File(targetPath)把文件写到指定的 targetPath 路径下，这样实现了文件上传。

客户端可以通过提交表单的形式上传文件，这里我们使用 AJAX 上传，前端页面脚本代码如下：

```
$(function(){
        var $fileUpload = $("#fileUpload");
        var $mark = $("#fileUploadMark");
        $("#importBtn").click(function(){
                $mark.show();
                $fileUpload.show();
        });
        $fileUpload.find(".close,#uploadFileClose").click(function(){
                $mark.hide();
                $fileUpload.hide();
        });

        function importSample(){
          var fileName=$("#inputSampleExcel").val();
            if(fileName==null||fileName==""){
                dialog.showDialog({
                  title: "提示",
                  content: "请选择上传文件？"    ,
                  cancel: function(){
                  },
                  confirm: function(){
                  }
            });
             return;
          }

          var formData = new FormData();
          formData.append("file", $("#inputSampleExcel")[0].files[0]);
          if(fileName){

                $("#uploadFileSave").attr("disabled",true);

                $.ajax({
                  url: "/sample/upload",
```

```
                    type: "post",
                data: formData,
                processData: false,//告诉 JQuery 不对数据进行处理，设置为 true
时 JQuery 将会序列号数据
                contentType: false,////告诉 JQuery 不对请求进行设置
                dataType:'json',
                success: function (data) {
                    $("#uploadFileSave").attr("disabled",false);
                    $mark.hide();
                    $fileUpload.hide();
                    location.reload();
                 },
                error: function (e) {
                    $("#uploadFileSave").attr("disabled",false);
                 }
               });
          }

        }

        $("#uploadFileSave").click(function(){
                importSample();
        });

        });
```

FormData 是 JavaScript 中的对象，用以将数据编译成键值对，以便用 XMLHttpRequest 来发送数据。提交表单数据时可以把 form 表单元素的 name 与 value 进行组合，实现表单数据的序列化，从而减少表单元素的拼接，提高工作效率。我们可以通过 append 方法追加表单数据，或使用 set 修改数据。

当 processData: true 的时候，jquery 会序列化数据，用于 GET 请求，为 false 时，当 processData: false 的时候，jquery 不会对数据进行处理。

contentType 是指发送信息到服务器时内容的编码类型，contenttype 发送数据流的形式，服务器根据编码类型使用特定的解析方式。

这里如果不把 contentType 设置为 false，后台则不能通过 get 的方式获取 image_url。

因为当有文件传输时，会设置以 contentType=multipart/form-data 的形式，这时是不能以 get 获取表单的，因为 multipart/form-data 不是以传统的键值对形式传递数据，后端将获取不到数据，form 表单中可以定义 enctype 属性，该属性的含义是在发送到服务器之前应该如何对表单数据进行编码。在默认的情况下，表单数据编码为 application/x-www-form-unlencoded。

五、读取 Excel 文件数据

在 Spring Boot 中我们可以使用 Apache POI 来读取 Excel 文件数据，Apache POI 是一个开源的 Java API，用于处理 Microsoft Office 文档，包括 Excel 电子表格。在 Spring Boot 中，可以使用 Apache POI 创建 Excel 文档，并将其写入 HTTP 响应中，以实现 Excel 表格的导出。

在使用前需要在 pom.xml 中加上依赖配置，如下：

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>4.1.2</version>
</dependency>
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>4.1.2</version>
</dependency>
```

在 SampleController.java 控制类文件中添加读取 Excel 文件数据的方法，具体代码如下：

```
/**
 * 读取 excel 中的数据
 * @param targetPath
 * @return
 */
private List<Sample> readExcel(String targetPath){

      //1.创建文件对象
      File file = new File(targetPath);

      //2.创建工作簿对象
      XSSFWorkbook workbook = new XSSFWorkbook(file);

      //3.获取工作表
      XSSFSheet sheet = workbook.getSheetAt(0);

      List<Sample> list = new ArrayList<Sample>();

      //4.遍历工作表中的每一行
      for(Row row : sheet) {
```

```
                if(row.getRowNum() == 0) {
                    continue;
                }

                //5.创建样本实体类对象
                Sample sample = new Sample();
                sample.setProduct(row.getCell(0).getStringCellValue());
                sample.setProjectName(row.getCell(1).getStringCellValue());
                sample.setSampleCode(row.getCell(2).getStringCellValue());
                sample.setSampleName(row.getCell(3).getStringCellValue());
                sample.setSampleType(row.getCell(4).getStringCellValue());
                sample.setDnbId(row.getCell(5).getStringCellValue());
                sample.setIndexNumber(row.getCell(6).getStringCellValue());

                list.add(sample);

        }

        return list;
    }
```

六、数据保存到数据库

遍历数据集 List 中的数据并且保存到数据库中，保存代码如下：

```
        /**
         * 保存数据到数据库中
         * @param list
         */
        private void saveData(List<Sample> list) {

                for(int i=0; i<list.size(); i++) {
                        Sample sample = list.get(i);

                        sample.setCreateTime(new Date());
                        sample.setUpdateTime(new Date());
                        sample.setTechnologyRoadmapTime(new Date());

                        sampleService.Add(sample);
                }

        }
```

任务实施

Step1 认知 Java 数组使用

Step2 认知 Java 的 for 语句

Step3 模板文件下载

Step4 Excel 文件上传

Step5 读取 Excel 文件数据

Step6 数据保存到数据库

活动七 实现样本管理模块的导出功能开发

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中样本数据的导出功能！

任务目标

能读出数据库数据通过 Java 语言编程实现对 Excel 文件的数据写入。

任务分析

1. 如何把数据从数据库批量中读取出来？
2. 如何把数据写入到 Excel 文件中？
3. 如何把从 Excel 文件输出到用户电脑？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、批量读取样本数据

调用业务逻辑层的方法读出样本数据，具体代码如下：

```
List<Sample> list = sampleService.GetTopList(0);
```

业务逻辑层的实现方法如下：

```
@Override
public List<Sample> GetTopList(int top) {
      if(top != 0) {
                Sort sort = Sort.by("createTime").descending();
                Pageable pageable = PageRequest.of(0, top, sort);
                return sampleDao.findAll(pageable).getContent();
```

```
        }
        else {
                return sampleDao.findAll();
        }
    }
```

二、把数据写入 excel

```
                //创建工作簿
                XSSFWorkbook workbook = new XSSFWorkbook();
                //创建工作表
                XSSFSheet sheet = workbook.createSheet("样本数据");

                //创建表头
                XSSFRow header = sheet.createRow(0);
                header.createCell(0).setCellValue("产品");
                header.createCell(1).setCellValue("项目名称");
                header.createCell(2).setCellValue("样本编号");
                header.createCell(3).setCellValue("样本名称");
                header.createCell(4).setCellValue("样本类型");
                header.createCell(5).setCellValue("DNB ID");
                header.createCell(6).setCellValue("Index 号");

                //填充数据
                int rowIndex = 1;
                for(Sample sample : list) {

                        XSSFRow row = sheet.createRow(rowIndex);
                        row.createCell(0).setCellValue(sample.getProduct());
                        row.createCell(1).setCellValue(sample.getProjectName());
                        row.createCell(2).setCellValue(sample.getSampleCode());
                        row.createCell(3).setCellValue(sample.getSampleName());
                        row.createCell(4).setCellValue(sample.getSampleType());
                        row.createCell(5).setCellValue(sample.getDnbId());
                        row.createCell(6).setCellValue(sample.getIndexNumber());
                        rowIndex++;
                }
```

三、Excel 文件的输出

通过 HttpServletResponse 的实现方法 getOutputStream 获取响应输出流，向页面输出 Excel 文件，同时需要设置一下响应头，告诉浏览器输出的内容是 excel 文件以及文件名称。具体代码如下：

```
                //定义输出文件名称
                String fileName = "sample_export_";
                SimpleDateFormat format = new SimpleDateFormat("yyyyMMdd_HHmmss");
                String datetime = format.format(new Date());
```

```
                    //文件名称最后以年月人与时间结尾
                    fileName = fileName + datetime + ".xlsx";

                    //设置响应头信息
                    response.setContentType("application/vnd.ms-excel");
                    response.setHeader("Content-Disposition", "attachment;
filename="+fileName);

                    //把 Excel 文件写入响应流，输出文件
                    ServletOutputStream outputStream = response.getOutputStream();
                    workbook.write(outputStream);
                    outputStream.flush();
                    outputStream.close();
                    workbook.close();
```

任务实施

Step1 批量读取样本数据

Step2 把数据写入 excel

Step3 Excel 文件下载

活动八 实现样本管理模块的数据管理开发

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中样本数据的管理功能！

任务目标

1. 能完成数据列表的展示。
2. 能完成数据的分页功能。
3. 能完成数据的删除功能。

任务分析

1. 如何使用 Thymeleaf 模板在页面中实现列表展示？
2. 实现数据分页的原理是什么？
3. Spring Data JPA 的分页如何实现？
4. 如何实现数据删除功能？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、分页读取样本数据

要能进行分页，首先能分页读取数据库中的样本数据，所以我们需要在业务逻辑层中加入读取分页数据的方法。

样本数据业务逻辑接口，代码如下：

```
/**
 * 获取数据列表（分页）
 * @param pageable
 * @return
 */
```

```
        Page<Sample> GetPageList(Pageable pageable);
```

样本数据业务逻辑实现类，代码如下：

```
        /**
         * 获取数据列表（分页）
         * @param pageable
         * @return
         */
        public Page<Sample> GetPageList(Pageable pageable){
                return sampleDao.findAll(pageable);
        }
```

在控制层中接收当前显示的页码和对其该页面的数据，代码如下：

```
        @RequestMapping(value={"/sample"})
        public String sample(@RequestParam(defaultValue = "1") int page,
@RequestParam(defaultValue = "20") int size, Model model) {


                Sort sort = Sort.by("createTime").descending();
                Pageable pageable = PageRequest.of(page - 1, size, sort);

                Page<Sample> pageData = sampleService.GetPageList(pageable);

                List<Sample> list = pageData.getContent();

                com.bioseqmanage.common.Page pageObj = new
com.bioseqmanage.common.Page();
                pageObj.setCurrent(page);
                pageObj.setPageSize(size);
                pageObj.setTotalRecord(pageData.getTotalElements());

                model.addAttribute("list", list);
                model.addAttribute("page", pageObj);

                return "sample";
        }
```

方法中的参数@RequestParam(defaultValue = "1") int page 和@RequestParam(defaultValue = "20") int size，page 是当前页页码，size 是每页记录数。注解@RequestParam 我们用来设置

参数的默认值。

代码段 Pageable pageable = PageRequest.of(page - 1, size, sort);和 Page<Sample> pageData = sampleService.GetPageList(pageable); 其中使用的 Pageable 和 Page 是 spring 提供来帮助实现分页的。在 Spring Boot JPA 中已经直接提供 Pageable 作为参数的方法来获取参数。

我们执行创建的 Page 类，用来帮助我们在页面中实现分页页面，代码如下：

```
package com.bioseqmanage.common;

public class Page {

	//当前页面
	private int current = 1;
	//每页大小
	private int pageSize = 10;
	//总记录数
	private long totalRecord;

	/**
	 * 当前页面
	 * @return
	 */
	public int getCurrent() {
		return current;
	}
	/**
	 * 当前页面
	 * @param current
	 */
	public void setCurrent(int current) {
		this.current = current;
	}
	/**
	 * 每页大小
	 * @return
	 */
	public int getPageSize() {
		return pageSize;
	}
	/**
	 * 每页大小
	 * @param pageSize
	 */
	public void setPageSize(int pageSize) {
		this.pageSize = pageSize;
```

```
	}
	/**
	 * 总记录数
	 * @return
	 */
	public long getTotalRecord() {
		return totalRecord;
	}
	/**
	 * 总记录数
	 * @param totalRecord
	 */
	public void setTotalRecord(long totalRecord) {
		this.totalRecord = totalRecord;
	}

	/**
	 * 页面总数
	 * @return
	 */
	public long getTotalPage() {
		if(this.totalRecord % this.pageSize == 0) {
			return this.totalRecord / this.pageSize;
		}else {
			return this.totalRecord / this.pageSize + 1;
		}
	}

	/**
	 * 上一页页码
	 * @return
	 */
	public long getPreviousPage() {
		return this.current <= 1 ? 1 : this.current - 1;
	}
	/**
	 * 下一页页码
	 * @return
	 */
	public long getNextPage() {
		long total = this.getTotalPage();
		return this.current >= total ? total : this.current + 1;
	}
```

```
        /**
         * 显示的起始页码
         * @return
         */
        public long getShowFrom() {
                int from = this.current - 2;
                return from <= 1 ? 1 : from;
        }
        /**
         * 显示的终止页码
         * @return
         */
        public long getShowTo() {
                int to = this.current + 2;
                long total = getTotalPage();
                return to >= total ? total : to;
        }

}
```

二、分页数据在列表展示

在模板中使用 Thymeleaf 的循环语法显示数据，代码如下：

```
<tr th:each="item: ${list}">
    <td width="60" class="tc">1</td>
    <td width="60" class="tc">操作</td>
    <td width="60" class="tc" th:text="${item.sampleType}">样本类型</td>
    <td width="60" th:text="${item.sampleCode}">样本编号</td>
    <td width="60" th:text="${item.product}">产品名称</td>
    <td width="60" th:text="${item.sampleName}">样本名称</td>
    <td width="60" th:text="${item.projectName}">项目名称</td>
    <td width="60" th:text="${item.technologyRoadmap}">技术路线</td>
    <td width="60" th:text="${item.sampleProgress}">样本进度</td>
    <td width="60">--</td>
    <td width="60" th:text="${item.indexNumber}">Index 号</td>
    <td width="60" th:text="${item.dnbId}">DNB ID</td>
    <td width="60" th:text="${item.chipNumber}">芯片号</td>
    <td width="60" th:text="${item.laneNumber}">Lane 号</td>
    <td width="60" th:text="${item.libraryId}">文库 ID</td>
    <td width="60" th:text="${item.hybridLibraryId}">混合文库 ID</td>
    <td width="60" th:text="${item.createTime}">记录创建时间</td>
    <td width="60" th:text="${item.creatorName}">记录创建者</td>
    <td width="60" th:text="${item.updateTime}">记录最后修改时间</td>
    <td width="60" th:text="${item.lastModifiedBy}">记录最后修改者</td>
    <td width="60" th:text="${item.technologyRoadmapTime}">技术路线创建时间</td>
</tr>
```

三、前端分页页码制作

模板中的分页页码实现，代码如下：

```
    <div class="list-footer">
          <div class="record">
                    一共 <span th:text="${page.totalRecord}"></span> 条记录  共 <span
th:text="${page.totalPage}"></span> 页
          </div>
          <div class="page-box">
                    <div class="previous"><a
th:href="@{/sample(page=${page.previousPage})}"><</a></div>
                    <div th:class="|page ${page.current==i?'current':''}"
th:each="i:${#numbers.sequence(page.showFrom, page.showTo)}">
                              <a th:href="@{/sample(page=${i})}" th:text="${i}"></a>
                    </div>
                    <div class="next"><a
th:href="@{/sample(page=${page.nextPage})}">></a></div>
          </div>
    </div>
```

四、数据删除功能

样本控制类中，添加一下方法：

```
          /**
           * 删除样本数据
           * @return
           */
          @RequestMapping("/sample/delete")
          @ResponseBody
          public Map<String, Object> delete(HttpServletRequest request) throws Exception {

                    String deleteId = request.getParameter("deleteId");

                     Map<String, Object> map = new HashMap<>();

                    if(deleteId == null || "".equals(deleteId)) {
                              map.put("status", "fail");
                  map.put("msg","deleteId 不能为空");

                  return map;
                    }
```

```
            System.out.println(deleteId);

            String[] idArray = deleteId.split(",");
            try {
                    for(int i=0; i<idArray.length; i++) {
                            if(!"".equals(idArray[i])) {
                                    int id = Integer.parseInt(idArray[i]);
                                    sampleService.Delete(id);
                            }
                    }

            } catch (Exception e) {
                    // TODO: handle exception
            }

            map.put("status", "success");
      map.put("msg","删除成功");

      return map;
    }
```

前面模板中，给删除按钮绑定点击事件，在该事件中实现一下代码：

```
$("#deleteBtn").click(function(){

     function deleteData(){
            var count = 0, deleteId = "";
              $("input:checkbox:checked").each(function (index) {
                       count++;
                       deleteId += $(this).val() + ","
             });

             if(count <= 0){
                       dialog.showDialog({
                                title: "提示",
                                content: "请选择删除的项目"   ,
                                cancel: function(){
                                },
                                confirm: function(){
                                }
                       });
                        return;
             }

             var load = dialog.showLoading({
                       autoClose: true,
                       closeTime: 1000
```

```
                });

                    $.ajax({
                type: "post",
                url: "/sample/delete",
                data: {
                                        deleteId: deleteId
                            },
                dataType:'json',
                success: function (data) {
                    load.close();
                    location.href = "/sample";
                },
                error: function (e) {
                    load.close();
                }
            });

        }

        dialog.showDialog({
                  title: "提示",
                  content: "确定要删除吗？删除后无法恢复" ,
                  cancel: function(){
                  },
                  confirm: function(){
                            deleteData();
                  }
       });

});
```

任务实施

Step1 从数据库分页读取样本数据

Step2 分页数据在列表展示

Step3 前端分页页码制作

Step4 数据删除功能

活动九 实现任务管理的创建分析任务开发

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中分析任务的创建功能！

任务目标

1. 能通过 URL 传递参数和服务器端接收传递的数据来完成不同的分析任务的创建。
2. 能完成表单数据提交前的校验和进行服务器校验，并能输出错误提示。

任务分析

1. 不同页面直接如何传递参数？服务器段如何接收参数？
2. 如何把数据绑定到表单中？
3. 如何把表单数据提交到服务器端？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、参数传递

1. @RequestParam

针对 QueryString 参数传递，在请求 URL 中直接拼接请求参数，如 URL?param1=value1&param2=value2。

传递基本类型参数，在接收的方法中用@RequestParam 注解修饰，指定前端传递的参数名称。

```
@RequestMapping("QueryStringTest1")
```

```
    public String QueryStringTest1(@RequestParam("id") String id,@RequestParam("name") String
name){
        return "id is "+ id+",name is "+ name;
    }
```

如果传递的参数和接收的参数名一致，甚至可以省略相关配置，如下：

```
    @RequestMapping("QueryStringTest2")
    public String QueryStringTest2(@RequestParam String id,@RequestParam String name){
        return "id is "+ id+",name is "+ name;
    }
```

注意：因为有 RequestParam 注解的存在，所以传参必须包含所有使用 RequestParam 注解的接收参数，否则会报错（允许传空，但是必须包含参数）；可以使用@RequestParam(value =“id”,required = false) 的方式，让 id 未传也可以。

@RequestParam 注解直接被省略。

```
    @RequestMapping("QueryStringTest3")
    public String QueryStringTest3(String id, String name){
        return "id is "+ id+",name is "+ name;
    }
```

允许其中的参数不传，不会报错，但取值为 null，比如 name 不传。

传参和入参名不一样的情况。

```
    @RequestMapping("QueryStringTest4")
    public String QueryStringTest4(@RequestParam("id") String myId,@RequestParam("name")
String myName){
       return "myId is "+ myId+",myName is "+ myName;
    }
```

传递对象类型参数，定义一个对象，属性名称和前端传递的参数名称一致即可。

```
    @RequestMapping("objectTest1")
     public String objectTest1(User user){
         return "user.id is "+ user.getId()+",user.name is "+ user.getName();
     }
```

未传的属性值为 null。

加上注解@RequestParam，会怎么样呢？会报错，不要这样用。

```
@RequestMapping("objectTest3")
public String objectTest3(@RequestParam User user){
    return "user.id is "+ user.getId()+",user.name is "+ user.getName();
}
```

传递数组、集合类型
数组
拼接多个参数名称一样的参数即可，如 URL?param=value1¶m=value2¶m=value3

```
@RequestMapping("arrayTest1")
public String[] arrayTest1(@RequestParam String[] name){
    for (String temp:name){
        System.out.println(temp);
    }
    return name;
}
```

对同一个参数赋多个值，多个值之间用“,”隔开，如 URL?param=value1,value2,value3

集合
只举例 List，其他类似。

```
@RequestMapping("listTest1")
public List listTest1(@RequestParam List name){
    for(int i=0;i<name.size();i++){
        System.out.println("list name==="+name.get(i));
    }
    return name;
}
```

2. @PathVariable

路径传参方式是将参数直接包含在 URL 路径中，比如 URL/paramValue1/paramValue2。
在接口对应的请求路径中用{参数名}形式标出路径参数。
在接口方法的参数上标注@PathVariable 指名对应路径参数的参数名。
传递基本类型参数，如下：

```
@RequestMapping("pathTest1/{id}/{name}")
 public String pathTest1(@PathVariable("id") String id, @PathVariable("name") String name){
     return "id is "+ id+",name is "+ name;
 }
```

能省略这个注解吗？我的结论是不可以，请看下面。

```
@RequestMapping("pathTest2/{id}/{name}")
public String pathTest2(String id,String name){
    return "id is "+ id+",name is "+ name;
}
```

传递数组、集合类型
数组

```
@RequestMapping("pathArrayTest1/{name}")
public String[] pathArrayTest1(@PathVariable("name") String[] names){
    for (String name:names){
        System.out.println("array name:"+name);
    }
    return names;
}
```

集合

```
@RequestMapping("pathListTest1/{name}")
public List pathListTest1(@PathVariable("name") List names) {
    for (int i=0;i<names.size();i++) {
        System.out.println("list name:" + names.get(i));
    }
    return names;
}
```

文件

```
// MultipartFile 接收
@PostMapping("upload")
public String uploadFile(@RequestParam("file") MultipartFile myFile){
    return "type: "+myFile.getContentType()
```

```
                +" fileName: "+myFile.getOriginalFilename()
                +" size: "+myFile.getSize();
    }
```

测试的注意点：

●用 post。

●用 Body 的 form-data。

●参数类型选择 File 而不是默认的 Text。

3. @RequestBody

针对 body 表单传参，RequestBody 只能有一个。

1）传递基本类型参数

```
@RequestMapping("bodyTest1")
public String bodyTest1(@RequestBody String name){
   return "name is: "+name;
}
```

2）传递对象类型参数

比如 User 对象：

```
@RequestMapping("bodyTest2")
public String bodyTest2(@RequestBody User user){
    return "name is: "+user.getName() + " age is: "+user.getAge();
}
```

3）传递数组、集合类型

①数组

```
@RequestMapping("bodyTest3")
public String bodyTest3(@RequestBody String[] users){
    return Arrays.asList(users).toString();
}
```

②集合

User 对象

```
@RequestMapping("bodyTest4")
public String bodyTest4(@RequestBody List<User> users){
    return users.toString();
}
```

String 类型

```
@RequestMapping("bodyTest5")
public String bodyTest5(@RequestBody List<String> users){
    return users.toString();
}
```

*Json 对象

添加依赖

```
<dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>fastjson</artifactId>
      <version>1.2.56</version>
</dependency>
```

后端代码

```
@RequestMapping("bodyTest6")
public String bodyTest6(@RequestBody JSONObject user){
    return "name is : "+user.get("name")+
            " age is : "+user.get("age");
}
```

4. HttpServletRequest

```
@RequestMapping("httpRequest")
public String httpRequest(HttpServletRequest request){
```

```
        return "name is : "+request.getParameter("name")+
                " age is : "+request.getParameter("age");
    }
```

5. @PathVariable+@RequestParam 混合

```
    @RequestMapping("pathAndParam/{id}")
    public String pathAndParam(@PathVariable("id") String id, @RequestParam("name") String
name){
        return "id is "+ id+",name is "+ name;
    }
```

6. @PathVariable+@RequestBody 混合

```
    @RequestMapping("pathAndBody/{id}")
    public String pathAndBody(@PathVariable("id") String id, @RequestBody User user){
        return "id is "+ id+",name is "+ user.getName();
    }
```

7. 无注解传参

```
    @RequestMapping("other")
    public String other(User user){
        return "name is: "+user.getName() + " age is: "+user.getAge();
    }
```

二、绑定数据到表单

通过 url 获取参数值，这里 productName 是产品名称。

```
    @RequestMapping(value={"/task/dnbSampleEnter"})
    public String dnbSampleEnter(String productName, Model model) {
```

```
        model.addAttribute("productName", productName);

        return "dnbSampleEnter";

    }
```

在前端模板中把 productName 绑定到隐藏域中，代码如下：

```
<input type="hidden" id="product" name="product" th:value="${productName}" />
```

三、表单数据校验

获取表单中域的值，在提交的后台前需进行数据有效性验证，具体如以下代码：

```
<script>
$(function(){
      $("#btnSave").click(function(){

         var product = $("#product").val(),
             dnbId = $("#dnbId").val(),
             indexNumber = $("#indexNumber").val(),
             sampleCode = $("#sampleCode").val(),
             sampleName = $("#sampleName").val(),
             sampleType = $("#sampleType").val();

         if(product == null || product == ""){
             dialog.showDialog({
                 content: "产品名不能为空！"
             });
             return;
         }

         if(dnbId == null || dnbId == ""){
             dialog.showDialog({
                 content: "DNB ID 不能为空！"
             });
             return;
         }

         if(indexNumber == null || indexNumber == ""){
             dialog.showDialog({
```

```
            content: "Index 号不能为空！"
        });
        return;
    }
    if(sampleCode == null || sampleCode == ""){
        dialog.showDialog({
            content: "样本编号不能为空？"
        });
        return;
    }
    if(sampleName == null || sampleName == ""){
        dialog.showDialog({
            content: "样本名称不能为空？"
        });
        return;
    }
    if(sampleType == null || sampleType == ""){
        dialog.showDialog({
            content: "样本类型不能为空？"
        });
        return;
    }

    $.ajax({
        url: "/task/dnbSampleEnterSave",
        type: "post",
        data: {
            product: product,
            dnbId: dnbId,
            indexNumber: indexNumber,
            sampleCode: sampleCode,
            sampleName: sampleName,
            sampleType: sampleType
        },
        dataType:'json',
        success: function (data) {
           if(data.status == "success"){
               dialog.showDialog({
                    content: "保存成功？",
                    confirm: function(){
                        window.location.href = "/task";
                    }
                });
           }else{
```

```
                    dialog.showDialog({
                         content: data.msg
                     });
                }
            },
            error: function (e) {
            }
        });
    });
});
</script>
```

四、保存表单数据

在控制层 TaskController.java 中加入以下代码：

```
@PostMapping(value={"/task/dnbSampleEnterSave"})
@ResponseBody
public Map<String, Object> saveData(String product,
                String dnbId, String indexNumber,
                String sampleCode, String sampleName,
                String sampleType){

     Map<String, Object> map = new HashMap<>();

     if(product == null || "".equals(product)) {
       map.put("status", "fail");
       map.put("msg", "产品名称不能为空");
       return map;
     }
     if(dnbId == null || "".equals(dnbId)) {
       map.put("status", "fail");
       map.put("msg", "DNB ID 不能为空");
       return map;
     }
     if(indexNumber == null || "".equals(indexNumber)) {
       map.put("status", "fail");
       map.put("msg", "Index 号不能为空");
       return map;
     }
     if(sampleCode == null || "".equals(sampleCode)) {
       map.put("status", "fail");
```

```
        map.put("msg", "样本编号不能为空");
        return map;
    }
    if(sampleName == null || "".equals(sampleName)) {
        map.put("status", "fail");
        map.put("msg", "样本名称不能为空");
        return map;
    }
    if(sampleType == null || "".equals(sampleType)) {
        map.put("status", "fail");
        map.put("msg", "样本类型不能为空");
        return map;
    }


    Sample sample = new Sample();
    sample.setProduct(product);
    sample.setSampleCode(sampleCode);
    sample.setSampleName(sampleName);
    sample.setSampleType(sampleType);
    sample.setDnbId(dnbId);
    sample.setIndexNumber(indexNumber);
    sample.setCreateTime(new Date());
    sample.setUpdateTime(new Date());
    sample.setTechnologyRoadmapTime(new Date());
    sample = sampleService.AddSample(sample);

    Task task = new Task();
    task.setTaskName("MGI");
    task.setTaskType("分析");
    task.setDnbId(dnbId);
    task.setTechnologyRoadmap("分析");
    task.setAnalysisStatus(0);
    task.setCreateTime(new Date());
    task.setUpdateTime(new Date());

    task = taskService.AddTask(task);

    TaskSample taskSample = new TaskSample();
    taskSample.setTaskId(task.getTaskId());
    taskSample.setSampleId(sample.getSampleId());
    taskSample.setCreateTime(new Date());
    taskSample.setUpdateTime(new Date());
```

```
        taskSampleService.Add(taskSample);

        map.put("status", "success");
        map.put("msg", "新建成功");
        return map;

    }
```

在业务逻辑层分布中心定义了方法 AddSample 和 AddTask，用于接收添加新记录后返回实体数据，主要目的是获取到它们的 ID。获取自增长必须要在实体类的关键字段中加入注解，代码如下：

```
@Id
@GeneratedValue(strategy= GenerationType.IDENTITY)          //自增长字段，加了才会返回
@Column(name="sample_id")
private int sampleId;
```

任务实施

活动十 实现任务管理的数据搜索

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求我们完成项目中任务管理的数据搜索功能！

任务目标

1. 能通过前端表单向服务器提交搜索数据
2. 服务器端接收数据完成数据库数据搜索

任务分析

1. 如何多条件数据查询？
2. 如何进行数据库的模糊查找？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、Spring Data JPA 的条件查询

1. findBy 自定义查询

在 JPA 中 JPA 使用 findBy 方法自定义查询。也可以使用 findAllBy。这两个没有区别实际上还是使用的 findBy…进行查询的。

在 Dao 层中我们可以进行定义：

```
Optional<Users> findByUserName(String userName);
Optional<Users> findByUserNameAndPassword(String userName, String password);
```

除了 And 关键字，还支持多种关键字查询，如下所示：

```
And —— 等价于 SQL 中的 and 关键字，比如 findByUserNameAndPassword(String user, Striang pwd);
Or —— 等价于 SQL 中的 or 关键字，比如 findByUserNameOrAddress(String user, String addr);
Between —— 等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min);
LessThan —— 等价于 SQL 中的 “<”，比如 findBySalaryLessThan(int max);
GreaterThan —— 等价于 SQL 中的">"，比如 findBySalaryGreaterThan(int min);
IsNull —— 等价于 SQL 中的 “is null”，比如 findByUserNameIsNull();
IsNotNull —— 等价于 SQL 中的 “is not null”，比如 findByUserNameIsNotNull();
NotNull —— 与 IsNotNull 等价;
Like —— 等价于 SQL 中的 “like”，比如 findByUserNameLike(String user);
NotLike —— 等价于 SQL 中的 “not like”，比如 findByUserNameNotLike(String user);
OrderBy —— 等价于 SQL 中的 “order by”，比如 findByUserNameOrderBySalaryAsc(String user);
Not —— 等价于 SQL 中的 “！ =”，比如 findByUserNameNot(String user);
In —— 等价于 SQL 中的 “in”，比如 findByUserNameIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数;
NotIn —— 等价于 SQL 中的 “not in”，比如 findByUserNameNotIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数;
```

2. 自定义 SQL 查询

关于自定义 SQL 查询，方法名可以随意取（遵循驼峰命名法，方法名需能一眼看出此方法的用处），不需要遵循 JPA 制订的规则来起方法名。

自定义 SQL 是指：在方法上使用 @Query 注解，然后写 sql。

@Query 注解中有两个常用的属性，value(定义 sql)，nativeQuery (true 表示数据的 sql，false 表示 HQL，默认值是 false) 。当 @Query 注解中设置 nativeQuery = true 时即可以使用原生 SQL 进行查询，sql 里面的表名和字段都是使用的数据库里面的（带有下划线），当@Query 注解中设置 nativeQuery = false 时，sql 里面表名和字段都是使用定义实体类的变量名（驼峰命名）。

查询参数设置，使用 @Query 注解来指定查询语句，然后使用 @Param 注解来指定方法参数与查询语句中的参数对应关系，代码如下：

```
// 根据用户名查询用户
@Query("SELECT u FROM users u WHERE u.userName = :userName")
Users findByUserName(@Param("userName") String userName);
```

使用 @Query 注解直接指定查询语句。@Query 注解的 value 属性表示查询语句，可以使用占位符 ?1、?2 等表示方法参数。

```
// 根据用户名查询用户
@Query(value = "SELECT * FROM user WHERE user_name = ?1", nativeQuery = true)
Users findByUserName(String userName);
```

二、模糊查询

1. JPA 的 like 模糊查找

```
List<Users> findByFullnameLike(String fullname);
```

findByFullameLike 是 Spring Data JPA 定义的查询方法，其中 fullname 为要查询的字段名称，使用 Like 关键字可以实现模糊匹配。在使用时只需传入模糊匹配的字符串即可。

2. @Query 模糊查找

```
@Query("SELECT * FROM users WHERE fullname LIKE CONCAT('%',:fullname ,'%')")
List<User> getUserByFullname(@Param("fullname ") String fullname );

//或者

@Query(value = "SELECT * FROM users WHERE fullname LIKE CONCAT('%',?1,'%')", nativeQuery
= true)
List<User> getUserByFullname(String fullname);
```

任务实施

Step1 Dao 数据访问层中添加查询方法

在 TaskDao.java 中添加查询方法，代码如下：

```
    // 根据用户名和密码查询用户
    @Query(value = "select * from task where task_name LIKE CONCAT('%', ?1, '%') AND
analysis_status = ?2", nativeQuery = true)
    Page<Task> getPageListLike(String keywork, int analysisStatus, Pageable pageable);

    // 根据用户名和密码查询用户
    @Query(value = "select * from task where task_name LIKE CONCAT('%', ?1, '%')", nativeQuery
= true)
    Page<Task> getPageListLike(String keywork, Pageable pageable);
```

Step2 Server 业务逻辑层中添加方法

在 TaskService.java 中添加查询方法，代码如下：

```
    /**
     * 模糊查找（分页）
     * @param keywork
     * @param analysisStatus
     * @param pageable
     * @return
     */
    Page<Task> getPageListLike(String keywork, int analysisStatus, Pageable pageable);

    /**
     * 模糊查找（分页）
     * @param keywork
     * @param pageable
     * @return
     */
    Page<Task> getPageListLike(String keywork, Pageable pageable);
```

在 TaskServiceImpl.java 中添加查询方法，代码如下：

```
    /**
     * 模糊查找（分页）
     */
    @Override
    public Page<Task> getPageListLike(String keywork, int analysisStatus, Pageable pageable){
          return taskDao.getPageListLike(keywork, analysisStatus, pageable);
    }

    /**
```

```
     * 模糊查找（分页）
     */
    @Override
    public Page<Task> getPageListLike(String keywork, Pageable pageable){
          return taskDao.getPageListLike(keywork, pageable);
    }
```

Step3 Controller 中使用条件模糊查找的方法

在 TaskController.java 中添加查询方法，代码如下：

```
    @RequestMapping(value={"/task"})
    public String sample(@RequestParam(defaultValue = "1") int page,
                    @RequestParam(defaultValue = "3") int size,
                    @RequestParam(defaultValue = "")String keyword,
                    @RequestParam(defaultValue = "-1") int analysisStatus,
                    Model model) {

          Sort sort = Sort.by("create_time").descending();
          Pageable pageable = PageRequest.of(page-1, size, sort);



          Page<Task> pageData = null;
          List<Task> list = null;

          if(analysisStatus == -1) {
                    pageData = taskService.getPageListLike(keyword, pageable);
                    list = pageData.getContent();
          }else {
                    pageData = taskService.getPageListLike(keyword, analysisStatus,
pageable);
                    list = pageData.getContent();
          }

          com.bioseqmanage.common.Page pageObj = new com.bioseqmanage.common.Page();
          pageObj.setCurrent(page);
          pageObj.setPageSize(size);
          pageObj.setTotalRecord(pageData.getTotalElements());

          model.addAttribute("list", list);
          model.addAttribute("page", pageObj);
          model.addAttribute("keyword", keyword);
          model.addAttribute("analysisStatus", analysisStatus);
```

```
        return "task";
    }
```

Step4 在模板文件中做响应代码处理

在 task.html 模板中添加查询关键字和状态两个条件，然后绑定查询按钮的点击事件，代码如下：

```
    <div class="search">
         <div><input type="text" name="keyword" id="keyword" placeholder="搜索关键字"
th:value="${keyword}"></div>
         <div>
                  <select id="analysisStatus" name="analysisStatus">
                           <option value="-1">分析状态</option>
                  <option value="0">未完成</option>
                  <option value="1">完成</option>
           </select>
         </div>
         <div id="btnSearch" class="btn">搜索</div>
    </div>


    <script type="text/javascript">
       $(function(){
          $("#btnSearch").click(function(){
             location.href = "/task?keyword="+ $("#keyword").val() + "&analysisStatus=" +
$("#analysisStatus").val();
         });
         });
    </script>
```

任务 5 生物测序实验管理系统发布

项目描述

项目开发完成后，需要打开程序，然后部署到服务器上。Spring Boot 有两种不同打包方式：jar 打包和 war 打包。打包完成后部署到服务器中并且能运行。

项目目标

1. 能对项目程序进行 jar 打包。
2. 能对项目程序进行 war 打包。
3. 能运行打包好的包。
4. 能在服务器上运行。

活动一 项目打包部署到服务器

情景导入

某软件公司项目开发小组收到一张任务工作单：为某生物实验室开发一套生物测序实验管理系统，能对测序数据进行管理。现要求对完成的项目程序进行打包并且部署到服务器，让我们一起完成打包和部署吧！

任务目标

1. 能把项目打包成 jar 并且部署到服务器。
2. 能把项目打包成 war 并且部署到服务器。
3. 部署到服务器后能完成启动。

任务分析

1. Spring Boot 程序有几种打包方式？分布是什么？
2. 它们打包原理？
3. 如何上传服务器？
4. 如何在服务器上运行？

任务准备

一台开发使用的 Windows 操作系统的电脑、生物测序实验管理 Java 应用开发学习工作页。

知识链接

一、jar 打包

Spring Boot 的可执行 jar 包又称作“fat jar”，是包含所有三方依赖的 jar。Spring Boot 的 jar 包项目发布形式简单、快捷且内置 web 容器，因此 Spring Boot 将其作为默认选项。

1. 修改 pom.xml

在项目打开 pom.xml 加入标签<packaging>jar</packaging>，告诉 maven 打包的是 jar，如图 5-1-1 所示。

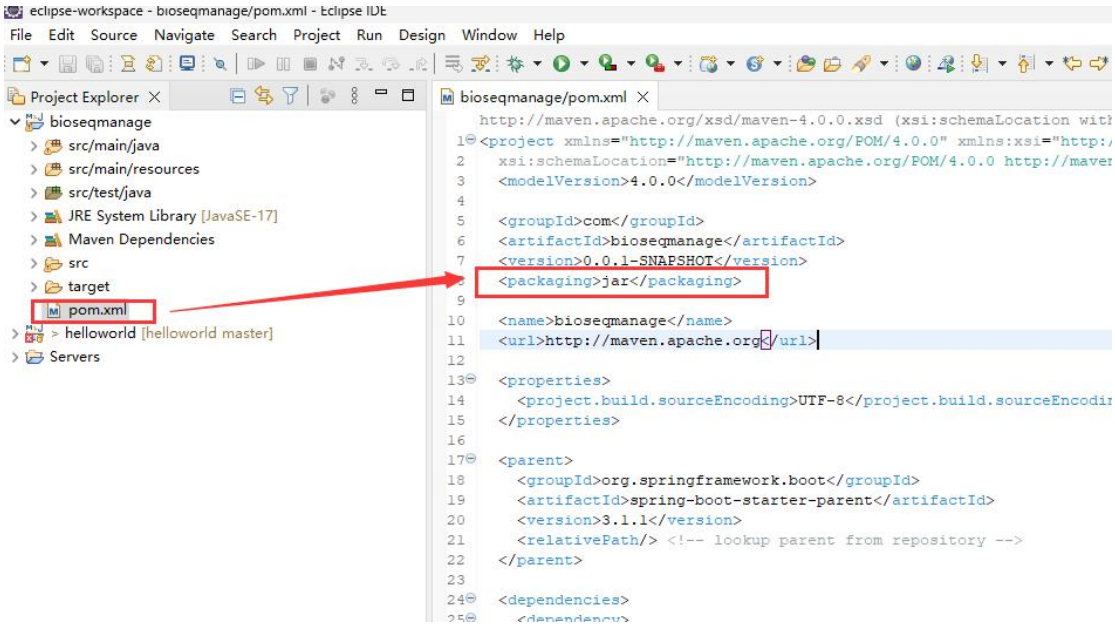


图 5-1-1 jar 打包

jar 包通常是由集成在 pom.xml 文件中的 Maven 插件来生成的。在项目打开 pom.xml 加入插件 spring-boot-maven-plugin，如图 5-1-2 所示，具体代码如下：

```
<build>
    <plugins>
        <plugin>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

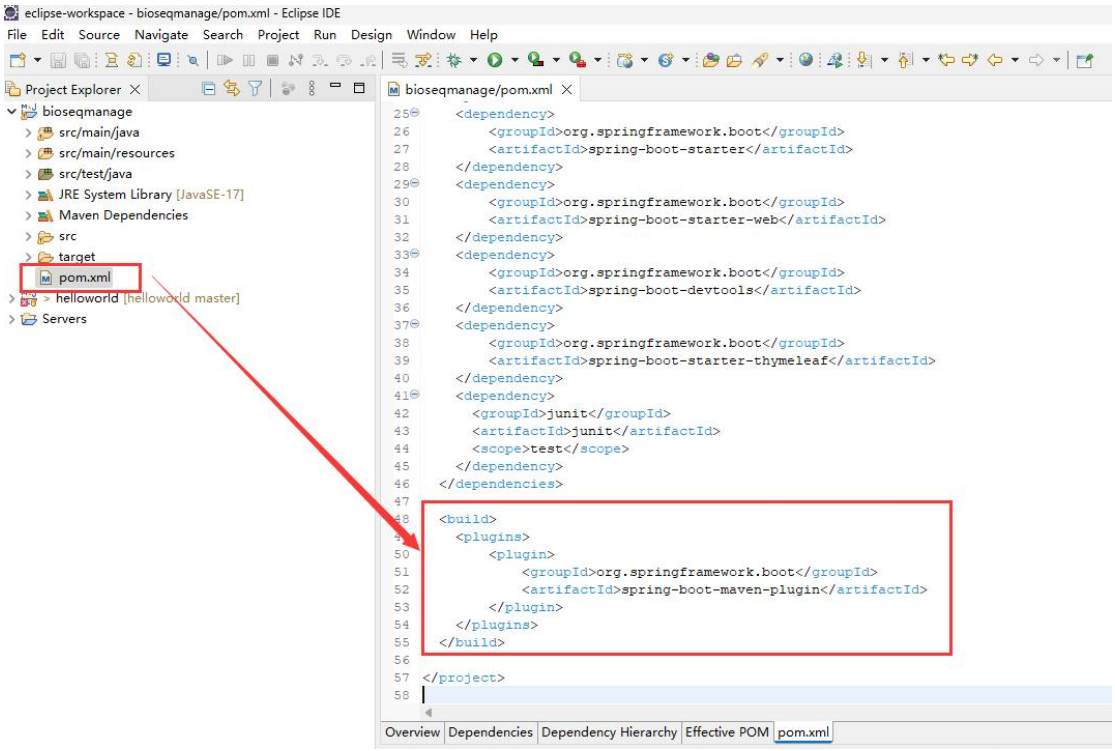


图 5-1-2 加入 Maven 插件

2. 修改端口号

通常网站的方式默认端口是 80，所以我们需要修改端口号（以实际项目需求为准），打开 application.properties 文件，修改 server.port 的值为 80，如果没有添加进入，如图 5-1-3 所示。

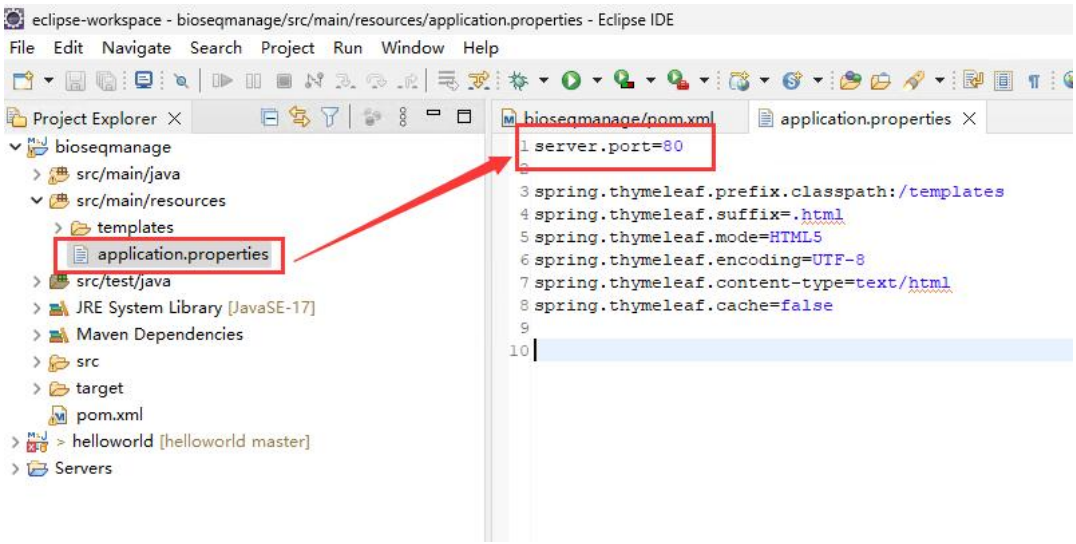


图 5-1-3 修改访问端口号

3. 生成 jar 包

先清除一下原来的打包文件，在项目中鼠标右击找到【Run As】→【Maven clean】如图 5-1-4 所示。

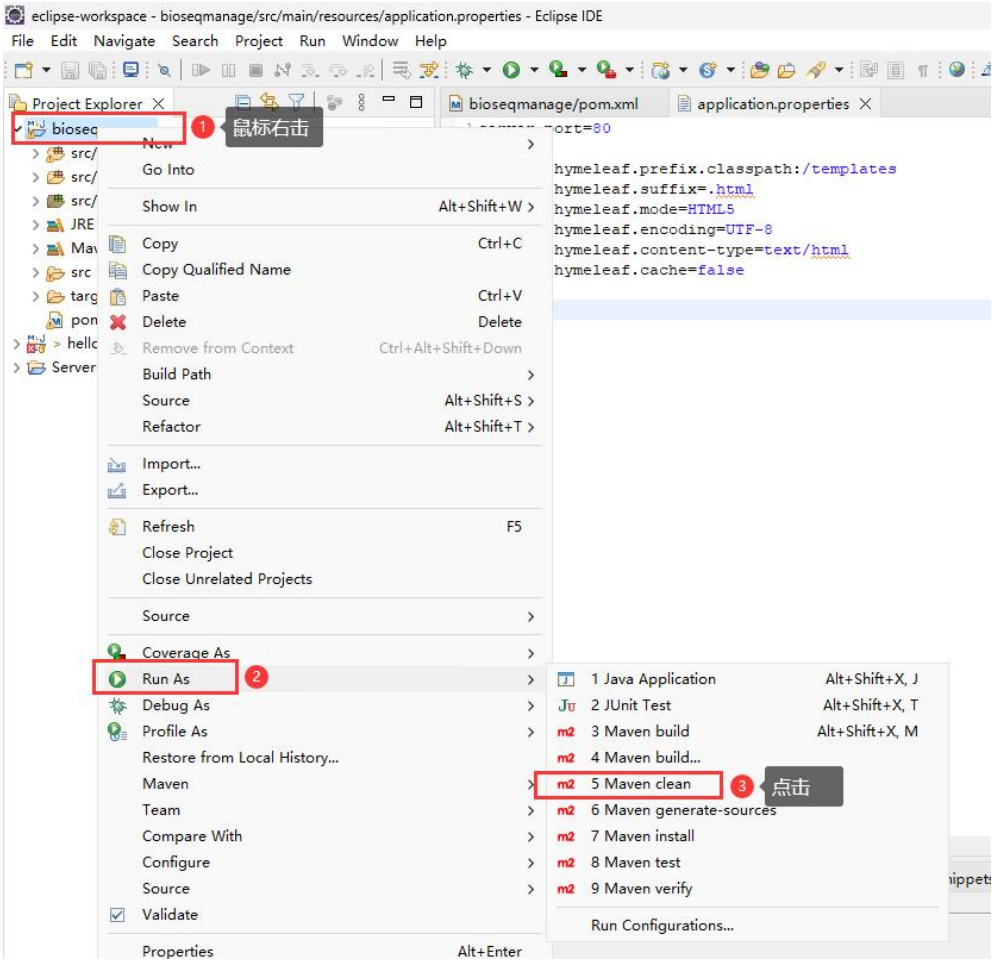


图 5-1-4 清除打包项目

生成打包文件 jar，在项目中鼠标右击找到【Run As】→【Maven install】如图 5-1-5 所示。

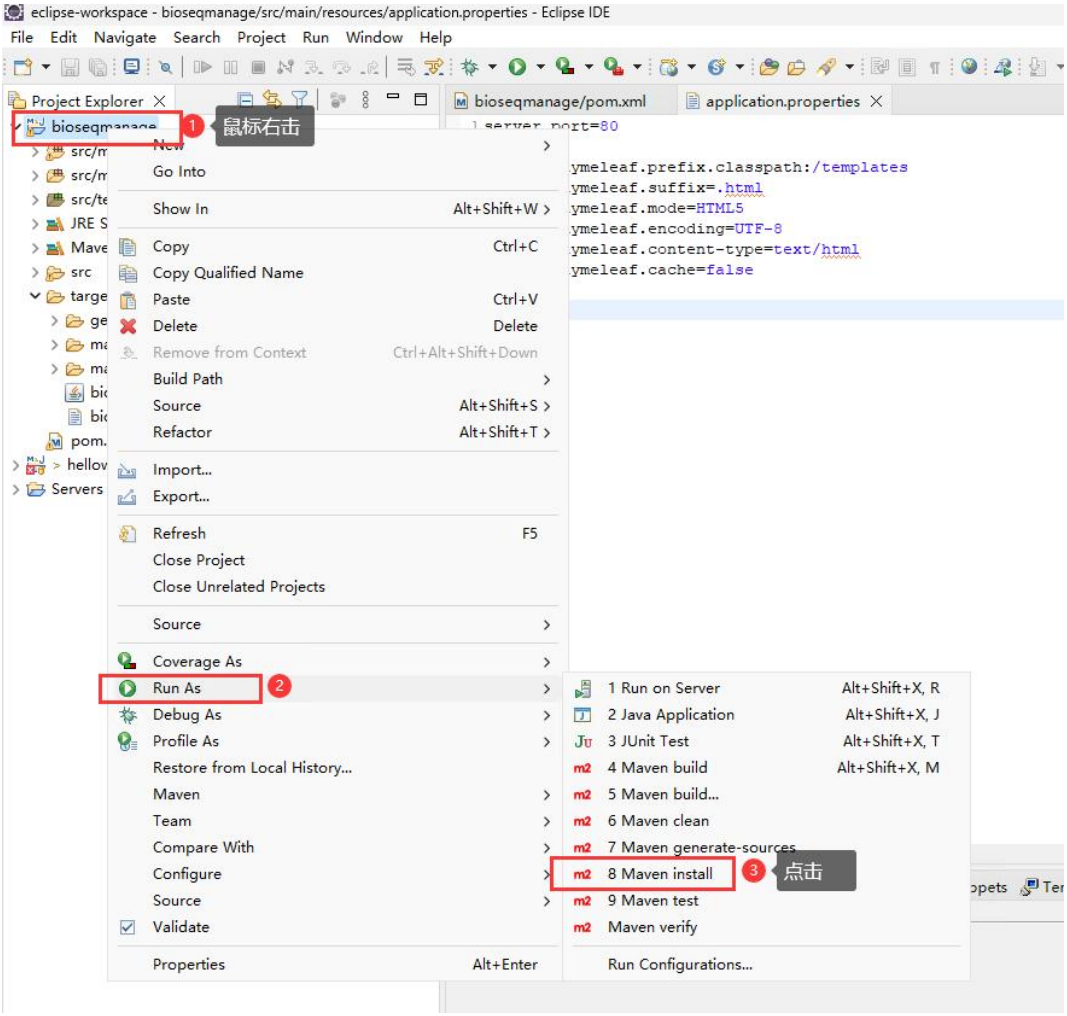


图 5-1-5 生成 jar 文件

最后在项目中 target 目录下生成了一个 jar 文件“bioseqmanage-0.0.1-SNAPSHOT.jar”，如图 5-1-6 所示。

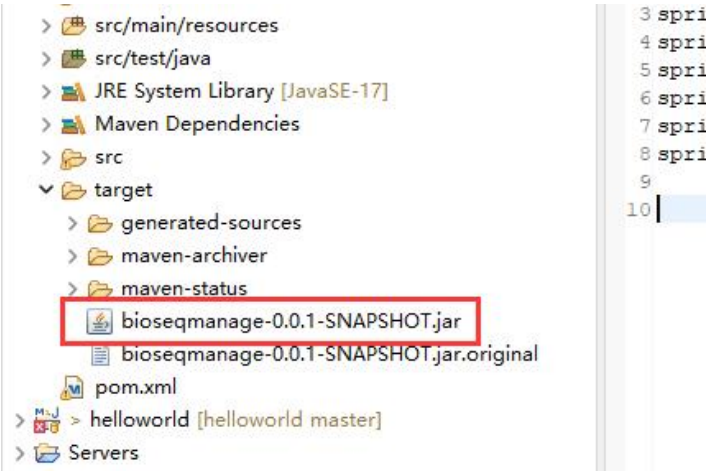


图 5-1-6 jar 文件

运行 jar 文件，在 target 目录下打开命令提示符窗口，输入以下代码：

```
java -jar bioseqmanage-0.0.1-SNAPSHOT.jar
```

运行效果，如图 5-1-7 所示，jar 成功运行，然后可以在浏览器输入 http://localhost 来进行访问。

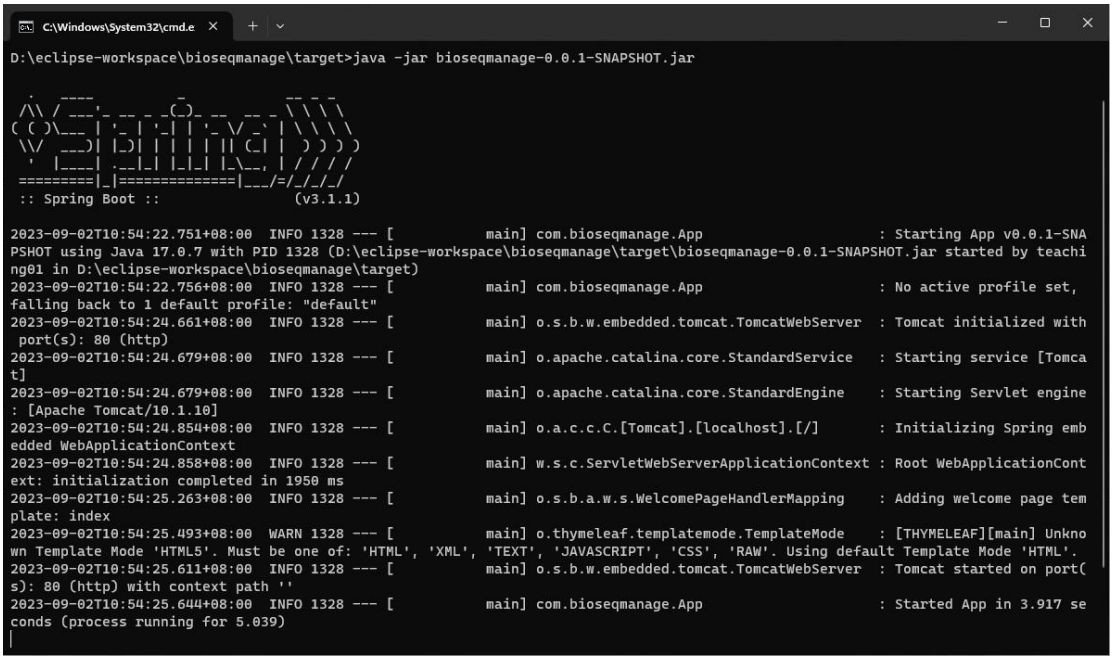


图 5-1-7 运行 jar 文件

二、war 打包

1. 修改 pom.xml

在项目打开 pom.xml 修改标签<packaging>jar</packaging>为<packaging>war</packaging>，告诉 Maven 打包的是 war，如图 5-1-8 所示。

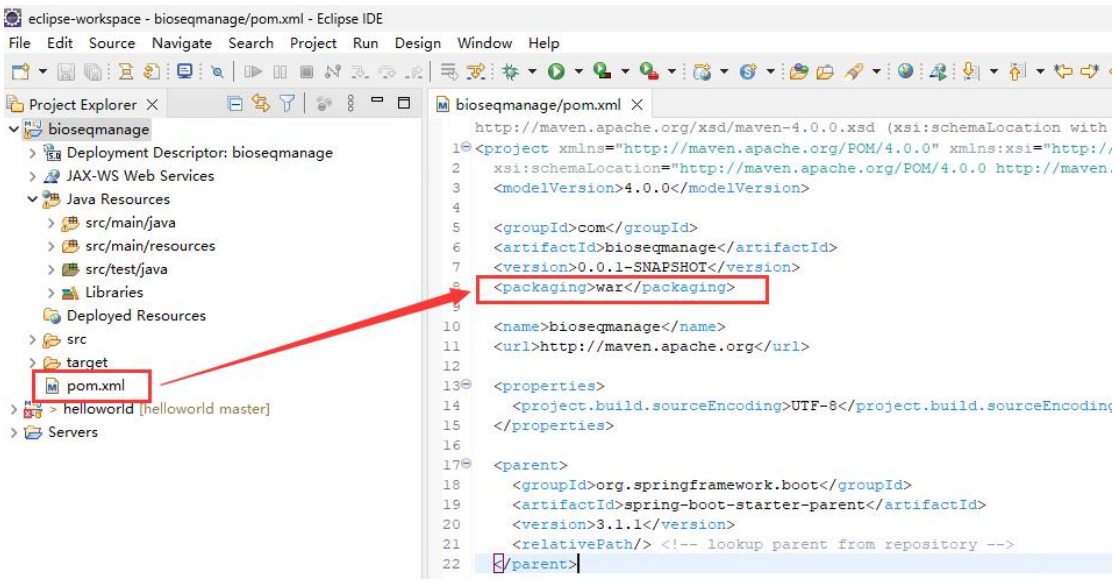


图 5-1-8 war 打包配置

war 包不需要 sprint boot 内置的 tomcat，所以需要排除内置的 tomcat。在项目打开 pom.xml 加入以下代码：

```
<!-- war 打包，排除自带的 tomcat -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

scope 属性设置为 provided，这样在打包产生的 war 包就不会包含 Tomcat 相关的 jar。

2. 改写启动类

打开启动类文件 App.java，首先继承 SpringBootServletInitializer，然后重写 Configure 方法，具体代码如下：

```
@SpringBootApplication
@ServletComponentScan
public class App extends SpringBootServletInitializer
{
    public static void main( String[] args )
    {
      SpringApplication.run(App.class, args);
    }
```

```
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
            return application.sources(App.class);
    }
}
```

3. 生成 war 包

先清除一下原来的打包文件，在项目中鼠标右击找到【Run As】→【Maven clean】如图 5-1-9 所示。

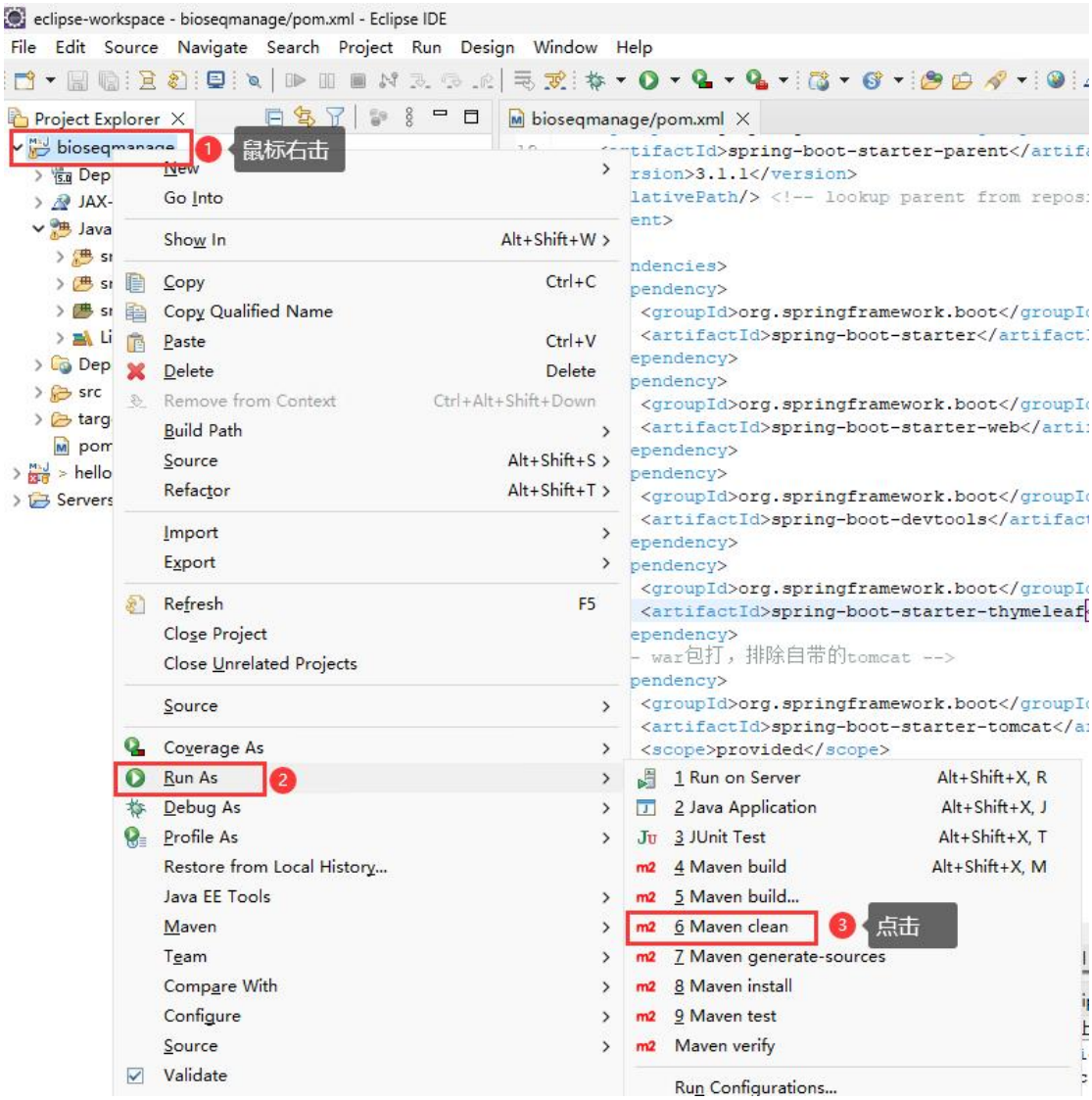


图 5-1-9 清除打包项目

生成打包文件 war，在项目中鼠标右击找到【Run As】→【Maven install】如图 5-1-10 所示。

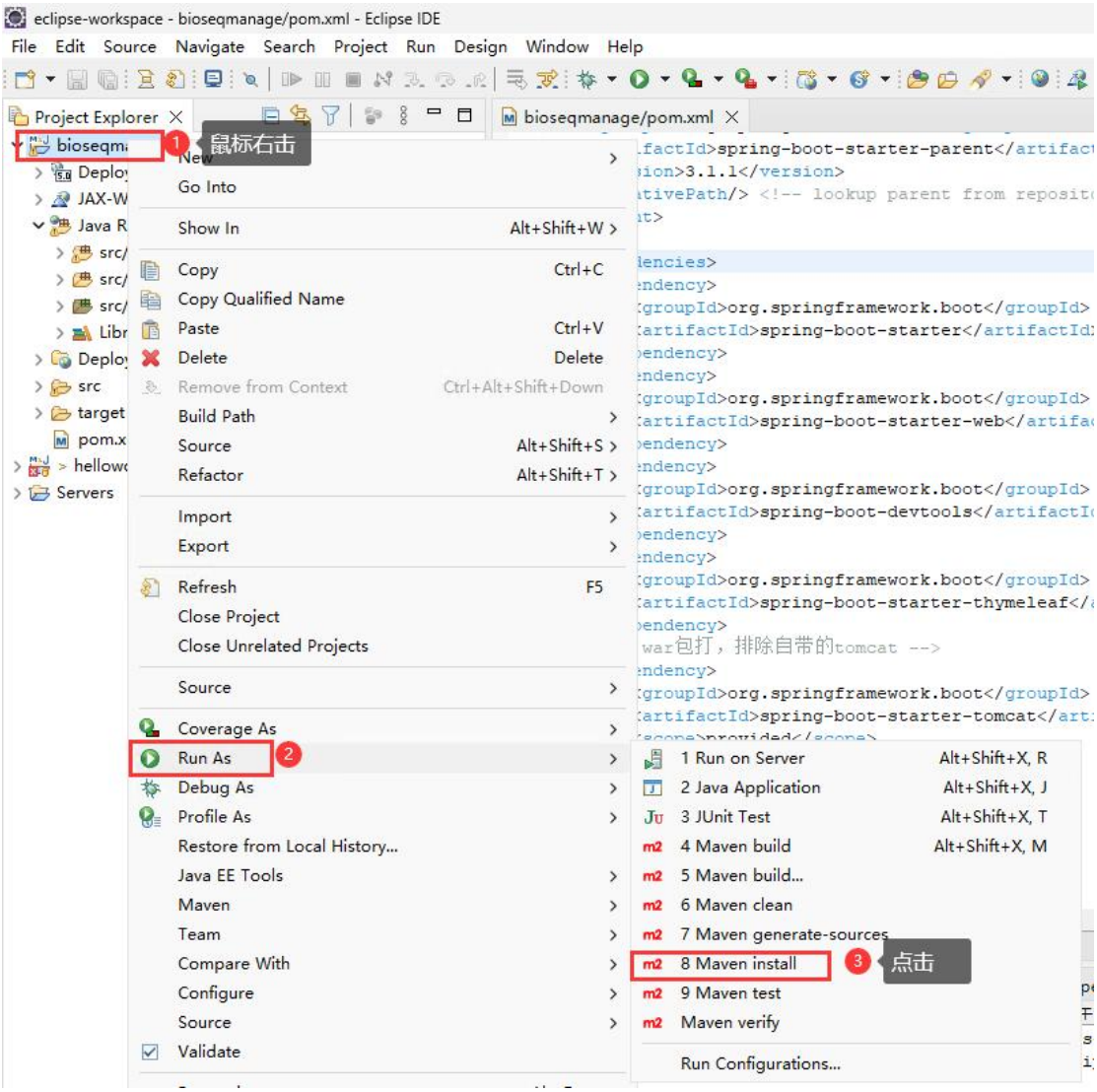


图 5-1-10 生成 war 文件

最后在项目中 target 目录下生成了一个 war 文件“bioseqmanage-0.0.1-SNAPSHOT.war”，如图 5-1-11 所示。

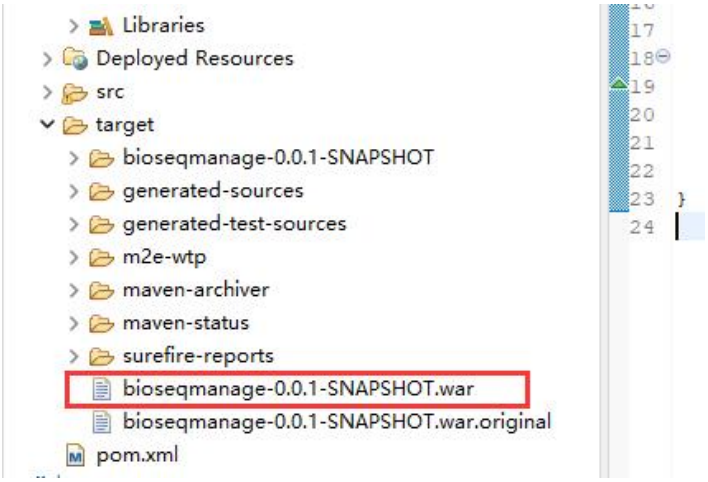


图 5-1-11 war 文件

把文件“bioseqmanage-0.0.1-SNAPSHOT.war”文件拷贝到 tomcat 的 webapps 目录下，并且把文件名改成“bioseqmanage.war”，如图 5-1-12 所示。

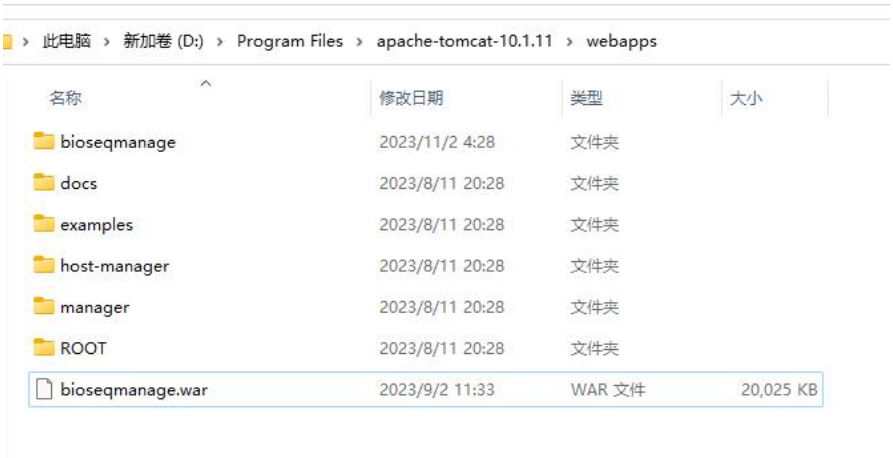


图 5-1-12 war 文件放 webapps 目录下

启动 tomcat，然后 tomcat 运行的命令提示符窗体出现 Spring Boot 的信息，表面运行成功，如图 5-1-13 所示。

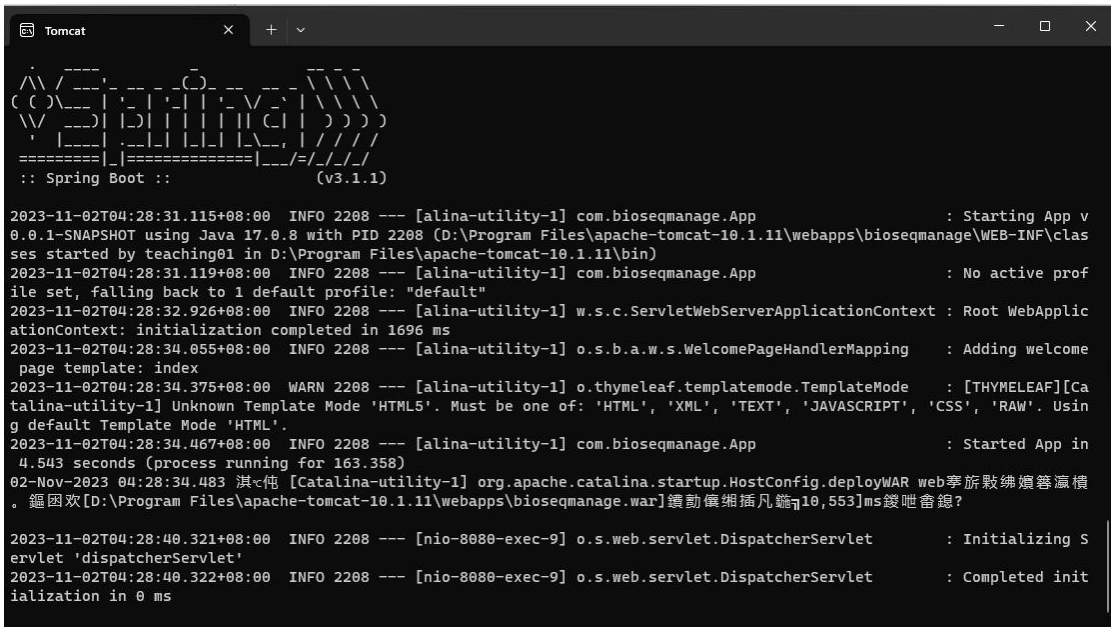


图 5-1-13 tomcat 运行出现 Spring Boot 信息

浏览器中输入 http://localhost:8080/bioseqmanage 既可以访问，其中 8080 端口是 tomcat 的默认端口，可以自行修改，“bioseqmanage”是 war 的文件名。

更改 tomcat 的 server.xml 配置，去掉“bioseqmanage”的目录名称，如下图 5-1-14 所示，增加 <Context reloadable="true" privileged="true" debug="0" docBase="D:/Program Files/apache-tomcat-10.1.11/webapps/bioseqmanage" path="" />。

```
<Host name="localhost"  appBase=""  unpackWARs="true" autoDeploy="true">
  <Context reloadable="true" privileged="true" debug="0" docBase="D:/Program Files/apache-tomcat-10.1.11/webapps/bioseqmanage" path="" />

  <!-- SingleSignOn valve, share authentication between web applications
       Documentation at: /docs/config/valve.html -->
  <!--
  <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
  -->

  <!-- Access log processes all example.
       Documentation at: /docs/config/valve.html
       Note: The pattern used is equivalent to using pattern="common" -->
  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
         prefix="localhost_access_log" suffix=".txt"
         pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>
```

图 5-1-14 sever.html 配置

三、上传到 Linux 服务器

1. 上传 jar 包

把 bioseqmanage.jar 文件上传到服务器的指定目录，然后执行命令，如下：

```
java -jar ./bioseqmanage.jar
```

如果需要在后台运行：

```
nohup java -jar ./bioseqmanage.jar &
```

2. 上传 war 包

上传到服务器上 tomcat 的 webapps 目录下，启动 tomcat 即可。

任务实施

Step1 Spring Boot 项目的 jar 打包

Step2 Spring Boot 项目的 war 打包

Step3 上传 Linux 服务器