主要技术难点和实现方案/算法设计：

（以下分点仅按照所对应的package名排序，与难度无关）

* Jlink打包：

Java应用程序启动时，如果需要启动参数，将不能直接鼠标双击启动，而需要在命令行中输入，或者另写一个bat文件。同时，JavaFX应用对于Java版本有一定依赖性，以本应用为例，本应用只能在Oracle JRE 9或Oracle JRE 10上运行，如果用户版本不一致或者没有安装JRE将导致无法运行，限制了产品的适应性。

为了解决这一问题，我们选择了Java 9引入的jlink工具，将项目模块化，在module-info.java中声明所需依赖，进行打包。打包后得到的是一个由部分JRE和本项目代码及其依赖构成的最小运行时，排除了Java基础库中没有使用的部分，大小仅为完整JRE的1/4，打包分发时，大小仅34.8M。解压后直接双击根目录下的“网络学堂.exe”（windows平台）即可运行，无额外依赖，适应所有能够运行JRE的平台，有效地解决了分发的适应性问题。

* Material design：

Material design是谷歌提出的一套UI设计方案，最先在谷歌自身产品中应用，后来被各个平台的开发者广泛使用。我们希望实现一个界面更具有现代风格、更符合用户使用习惯的客户端应用，因此选择了Material design作为我们的设计规范，以此进行UI设计。从零开始构建一套Swing组件对我们有限的时间来说难度过大，通过寻找，开源项目JFoenix提供了一套基于JavaFX的Material design实现。通过对该项目的学习，使用该项目的组件进行UI布局，我们构建了基本上让我们自己满意的应用外观，达到了预定目标。

* UI组件化：

JFoenix提供了基础组件，但如何将这些组件组合起来还是需要开发者自己实现。对于界面中重复出现的或相对独立的组件，如果不予以抽象，势必造成代码膨胀，难于维护。为了有效组织代码，增强复用性，我们学习了JavaFX的自定义组件，参考oracle文档中的写法，在app.controls类中定义了一些自定义组件，增强了适应性，降低了维护成本。

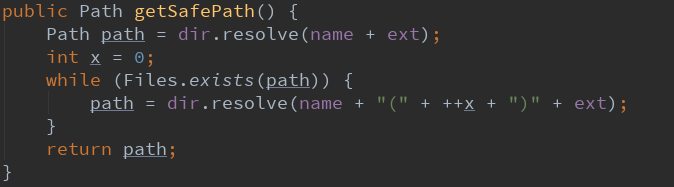
* 向JFoenix库提交代码：

在对JFoenix项目使用逐渐深入的过程中，发现选择时间面板中JFXDatePicker组件工作不正常。通过追溯异常出现的原因，发现了一个JFoenix库的bug。经过思考，我们找出了解决方案，通过GitHub的Issue和Pull request功能，向JFoenix项目指出该bug，并提交了对应的修复代码。目前JFoenix的维护者已经接收该修复，在下一个版本发布时，将包含该修复。

相关链接：[Issue](https://github.com/jfoenixadmin/JFoenix/issues/893)

[Fix](https://github.com/jfoenixadmin/JFoenix/pull/894)

* 下载文件的正确命名：

通过HTTP协议下载的文件，名字是从响应头的Content-Disposition字段取得的，由于网络学堂服务端对该字段采用了不符合规范的GB-2312编码，不能被正常解析。同时该文件名包含若干位的随机码，属于用户所不需要的部分。经过思考，我们采用了从Content-Disposition字段取得后缀名，与该文件在下载条目中所呈现的文件标题相组合的方式，为用户生成更符合使用习惯的文件名。另外，如果发生重名，直接写入可能破坏用户数据（若已更改），取消任务可能影响用户体验，我们采用如下解决方案：

如果foo.pdf存在则查找foo(1).pdf, foo(2).pdf以此类推，生成一个不会重名的安全文件名。

* Java基础库中不可序列化类型的处理：

本应用程序需要存储用户设置、登录信息等内容，需要进行对象的序列化。在简单情况下，只需要实现Serializable接口即可，但有一些需要序列化的Java基础库中的类没有实现Serializable接口，例如java.nio.Path, javafx.beans.property.StringProperty等，这给序列化造成了挑战。通过查找资料以及尝试，我们通过自定义序列化逻辑的方式解决了这一问题。通过在序列化时使用writeObject方法，将StringProperty转化为能够序列化的String类型（其他类的实现方法类似），在反序列化时用readObject将String转化为StringProperty，实现了序列化与存储。

* 用Preferences存储用户信息、设置：

用户设置、信息等采用临时文件存储需要额外的代码实现，需要考虑跨平台性，也一定程度上构成安全隐患（如果将配置文件存在客户端目录下，如果将本软件分发给他人，将导致个人信息泄露），增加了维护成本。通过查找Java基础库的API，我们发现java.util.prefs.Preferences类可以很好地满足我们的需求。它提供了跨平台的存储接口，在windows平台上使用注册表实现。通过对Preferences的学习、使用，我们很好地解决了信息持久化的问题。

* 系统原生方法的调用：

Java基础库的功能十分强大，但也有一些系统接口未能提供。例如，JavaFX的PasswordField组件无法禁用输入法（已被列入[JDK bug tracker](https://bugs.openjdk.java.net/browse/JDK-8087721)，但尚未修复），在用户输入密码时若开启了拼音等输入法，将导致密码显示在屏幕上，增加了风险，影响用户体验。通过查阅windows相关API，我发现Imm32.dll中的ImmAssociateContext方法可以实现相关控制，因此通过JNA（Java Native Access，用于简化Java调用原生系统级API）项目调用该函数，解决了该问题。

* 网络请求与解析：

网络请求具有时长的不确定性，如果在UI线程中执行相关逻辑，阻塞渲染，将有可能使应用失去响应，严重影响用户体验。因此，采用了多线程的方式进行网络请求。同时，老版网络学堂采用了服务端渲染的方式，请求到的只能是渲染好的HTML文本，因此，引入了jsoup项目进行HTML解析，通过对html结构的分析，进行相应的信息提取，转化为Java对象呈现于UI中。

* 多态处理网络学堂的三个版本：

网络学堂的三个版本差异很大，但在内容结构上又有共通点。通过研究不同版本的共性，将各种字段提取入超类供外部使用，在超类中将具体执行网络请求的方法标为抽象方法，由各个子类实现，使weblearning包对于外部将实现细节隐藏，只提供统一的接口，简化了使用。

* 在浏览器中打开：

本难点的解决很有挑战性，但与Java本身联系不紧密，故放在最后提出。通过Java使一个URI在浏览器中打开，简单来说只需要调用Desktop.getDesktop().browse即可。但在实际应用中，要在浏览器中成功打开网络学堂中的相关页面，需要浏览器中存在相关cookie，简单来说就是需要用户登录。为了在浏览器的CSP与CORS安全策略下，实现向网络学堂域名“添加”cookie，我们采用了通过Java生成中转页、在该页面中通过JavaScript代码跨域登录、再跳转至对应页面的方法，解决了相关问题（更具体的方案仅与JavaScript有关，涉及fetch、no-cors等API，在此略过）。