

**毕 业 论 文(设 计)**

**论文（设计）题目:**

**加强过程管理的网上项目实践平台的设计与实现**

姓 名 **周钊屹**

学 号 **201600130051**

学 院 **计算机科学与技术学院**

专 业 **计算机科学与技术**

年 级 **2016级**

指导教师 **梁文革**

2020年 5月 10日

**­目 录**

[摘 要 I](#_Toc38890709)

[ABSTRACT II](#_Toc38890710)

[第1章 绪 论 1](#_Toc38890711)

[1.1 选题背景和研究意义 1](#_Toc38890712)

[1.2 Git的国内外研究现状 1](#_Toc38890713)

[1.2.1 Git的国内研究现状 1](#_Toc38890714)

[1.2.2 Git的国外研究现状 2](#_Toc38890715)

[1.3 本文主要工作 3](#_Toc38890716)

[1.4 论文组织结构 3](#_Toc38890717)

[第2章 需求分析 5](#_Toc38890718)

[2.1 系统概述 5](#_Toc38890719)

[2.2 解决问题和基本要求 5](#_Toc38890720)

[2.3 系统目标 6](#_Toc38890721)

[2.4 功能性需求 7](#_Toc38890722)

[2.5 非功能需求 8](#_Toc38890723)

[第3章 系统设计 9](#_Toc38890724)

[3.1 工作流程 9](#_Toc38890725)

[3.2 系统模块设计 9](#_Toc38890726)

[3.3 系统数据库设计 9](#_Toc38890727)

[第4章 实现与测试 10](#_Toc38890728)

[4.1 系统实现 10](#_Toc38890729)

[4.2 系统测试 10](#_Toc38890730)

[第5章 总结 11](#_Toc38890731)

[致 谢 12](#_Toc38890732)

[参考文献 13](#_Toc38890733)

[附录1 代码及相关附件 14](#_Toc38890734)

[附录2 文献英文原文 15](#_Toc38890735)

[附录3 文献中文译文 16](#_Toc38890736)

**加强过程管理的网上项目实践平台的设计与实现**

# 摘 要

不论是在大型互联网公司还是学校平时做的一些实验，一个软件的开发或者课程设计的完成，都需要很多人的分工合作才能完成。而在多人合作开发实现一个项目的时候就会出现很多问题，最主要的问题就是版本管理和如何多人合作开发，多人合作开发的流程模式是怎样的。

本课题针对多人合作的项目式开发进行过程化、系统化的管理。拟结合GitHub和项目实践的特色，解决在多人合作中出现的许多常见问题，最终便于版本管理、多人合作、项目进度管理和历史项目的学习以及持续改进。整个课题的难点在于实现版本管理和多人合作的问题，最终通过Git提供的许多功能实现了上述基本要求。整个项目通过Java Maven创建项目，利用Swing创建图形界面，版本管理和多人合作功能通过Git在Java中提供的jar包JGit实现。整体的工作流程简述如下:项目负责人先创建项目，其他合作开发同伴Fork相应项目。负责人可以对合作开发同伴授权，如果有授权，合作者就不用通过pull request功能先发起请求，才能将合作者完成的功能提交到项目负责人的项目中；如果没有授权，合作者就需要先通过pull request发起合并请求，在项目负责人看过可以合并之后才能合并，如果项目负责人不同意则不予合并。

**关键词：**项目实践平台；多人合作；Git；版本控制；

# ABSTRACT

Whether it is a large-scale Internet company or some experiments which is always done in school, the development of a software or the course design needs a lot of people to work together to complete. Many problems will arise when a project is developed by a lot of people in the same time. The main problems are version control and how to develop by a lot of people at the same time, and what is the process of multiplayer development.

This paper aims at the process-oriented and systematic management of project-based development with multi-person cooperation. It is planned to combine the characteristics of GitHub and project practice to solve many common problems that occur in multi-person cooperation, and ultimately facilitate the version control, multi-person cooperation, project progress management, learning of historical projects and continuous improvement. The difficulty of the whole subject lies in the issue of implementing version control and multi-person cooperation. Eventually, the above basic requirements are realized through many functions provided by Git. The entire project is created through Java Maven, using Swing to create a graphical interface. The jar package JGit provided by java realize the function of version control and multi-person cooperation. The overall work flow is briefly described as follows: the project leader first creates the project, and other cooperator Fork corresponding projects. The person in charge can authorize the partner to develop the project together. If authorized, the collaborator does not need to initiate a pull request before submitting the function completed by the collaborator; if there is no authorization, the collaborator needs to Initiate a pull request, and the project can only be merged after the person in charge of the project has approve the changes. If the person in charge of the project does not agree, it will not be merged anyway.

**Keywords:** Project Practice Platform; Multiplayer cooperation; Git; Version Control.

# 绪 论

## 选题背景及研究意义

很多软件的开发过程都不是单靠一个人完成的，都需要一个团队进行合作开发，而在合作开发软件过程中比较重要的就是多人合作的模式，流程，以及软件实现的版本问题，而这些问题都和Git的功能相契合。版本管理如果只需要历史的修改信息还可以直接手动创建一个日志文件用于记录修改，但是实际上自己实现起来需要考虑历史版本的存储问题，没有太好的办法。

在现在的很多软件开发中，都需要用到多人合作以及版本管理。虽然针对版本管理除了git之外还有很多的版本管理软件，比如说同样可以用于多人开发下源码维护的CVS和SVN。SVN也是采用分支管理实现多人合作以及版本管理，是一种集中式的管理[[[1]](#endnote-1)]。而Git和CVS以及SVN都不同，是一种分布式的版本管理以及代码管理工具，相比其他版本管理工具，Git的分支管理非常方便，在建立新的分支或者切换分支的时候，可以直接用一行命令解决，不需要建立多余的目录。而且和集中式的管理相比，Git绝大部分可以直接在本地完成，而不用和集中的代码管理服务器(所有代码都存放在中央服务器中)进行交互[[[2]](#endnote-2)]，本项目在设计的时候就考虑到本地仓库的管理功能。

本课题开发完成了一个软件，该软件可以针对多人合作的项目式开发进行过程化、系统化的管理，除了实现多人合作之外，还能实现版本管理等一些其他功能，最重要的两个功能式实现了多人合作以及历史的版本管理功能。

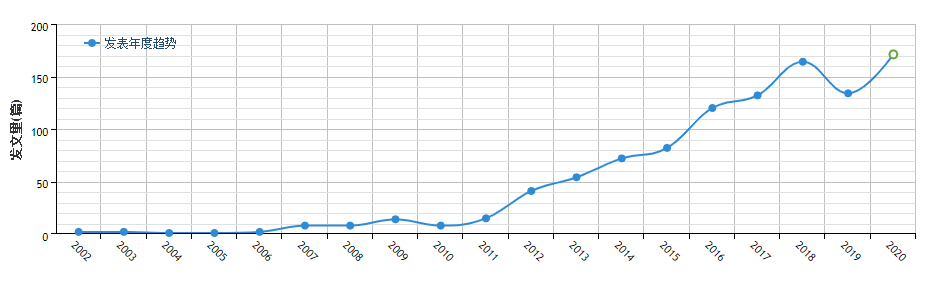
## Git的国内外研究现状

由于整个设计的功能大部分功能以及核心功能都是通过Git为Java提供的JGit包进行实现的，所以主要的研究现状就对Git的国内国外研究现状进行探索。

国内外直接研究现状没有直接文献进行叙述，所以通过一些论文网站的数据，即相关论文的发表数量作为一个研究现状的参考，国内的研究现状通过知网查询全文得到的论文作为研究现状的一个参考，而国外的部分参考ScienceDirect，而直接搜索Git会出现生物上的一些概念，应该是缩写的重复问题，需要用Git版本控制为关键字进行搜索，其中有些结果是单纯讨论Git或者GitHub的应用的文章[[[3]](#endnote-3)-5][[4]](#endnote-4)[[5]](#endnote-5)[[6]](#endnote-6)[[7]](#endnote-7)，利用Git的版本管理的特性实现各种各样不同的需求，还有些结果是一些利用Git嵌入自己的项目中实现一个具有需求功能的软件，和本文所阐述的设计有类似之处[[[8]](#endnote-8),[[9]](#endnote-9)]。

### Git的国内研究现状

国内的研究现状通过在知网用“Git版本管理”进行全文搜索，然后利用知网的计量可视化功能显示出总体趋势分析（见图 1‑1），主题分布太多太杂乱没有太多参考价值，在此不做引用，发文量的多少有一定参考价值，至少可以说明在这方面的研究或者使用量的多少。



**图 1‑1** **文献发表年度趋势**

从图上还是基本可以看出关于Git版本管理的文献数量从某个时间节点之后就开始一直上涨，到最近几年开始平缓，其中2020年的是预测值。Git这一技术是从2005年才开始有所开发，之前还有一些版本管理系统比如SVN等，但基本都是集中式版本管理，而Git是分布式版本管理。而后来很多相关文献和应用的增长或许与GitHub的出现有关，中国也有Gitee这样的基于Git的代码托管和研发协作平台[[[10]](#endnote-10)]。

### Git的国外研究现状

国外的研究采用在ScienceDirect上进行“Git version control”关键字查询，统计从2002到2020年以来的论文发表数量，最终得到的折线图见图 1‑2。

**图 1‑2 文献发表年度趋势**

在搜索结果中和国内的搜索结果一样可以大致分为两类，一类是直接利用Git或者GitHub进行辅助研究者工作的完成，简述其中的工作流程使其利于更多人使用。另一类是将Git的功能嵌入自己的软件中，利用其中的功能特性，比如基本的push，merge和branch功能，可以实现很多功能，不局限于版本管理，而且利用不同的Gitflow工作流程使得自己的工作更加容易被完成。

## 本文主要工作

文章主要用于阐述软件系统设计思路，从软件产生的背景到软件构思功能，即需求分析，然后说明软件整体需要怎样协调的设计才能完成需要功能，最后进行实现部分的详细说明和测试。本文一方面是对软件的系统化介绍，一方面用于在软件开发过程中的记录。

最终设计的系统将多人合作的项目式开发进行过程化、系统化的管理，结合项目实践的特色，首先实现了四个基本功能:版本管理、多人合作、项目进度管理、历史项目的学习，持续改进。并且为了更好地实现这些功能，还实现了一些比如对本地Git项目的管理操作，比如Push，Merge，Fetch等，以及一些为了安全性考虑并结合GitHub特色所应用的SSH Key管理和Pull request，Fork功能。

## 论文组织结构

论文第一章绪论主要介绍文章研究对象所研究的背景，由于设计是一个工程性项目，没有专门对某个学术部分进行研究，应用比较新的技术是Git的版本管理以及Git所提供的一些基本功能所实现的多人合作功能，所以绪论部分的很多背景研究都是探索的Git这个技术的背景研究。

论文第二章主要描述设计的系统基本概述，详细阐述系统设计想解决什么样的问题，以及需要具备什么样的基本功能去解决这样的问题，利用设计出来的基本功能用什么样的步骤去实现，即相应产品的工作流程。

论文第三章开始阐述系统如果需要实现之前需求分析所说的功能，需要如何去进行设计。软件一开始进入写代码阶段就比较需要有明确的目标，能够知道当前需要实现什么功能，并且实现功能的顺序也有所不同，这样有利于一边写一遍进行一些小的测试用来确保当前完成的小功能能够正常工作，如果最后一起再进行测试就会容易找不到具体错误的地方。软件如果进入代码阶段，写到后面发现该功能不能这样去实现，进行修改就会很麻烦，因为功能之间大都是有所联系的，而且修改起来需要重新测试，过程繁杂，因此第三部分的系统功能的具体到每个板块的设计还是很有必要的。

论文第四章正式阐述论文具体每个功能是如何实现的，和需求分析的功能进行对比就可以评估软件的完成程度。该章主要分成系统具体实现部分和项目测试部分。具体实现结合关键代码对细小的功能进行阐述，这些功能不一定直接就是最后的要求的功能，更有点类似数学的公理化方法，从一些定理进行推导得到最后的结论。项目测试功能利用一些设想的使用软件的操作流程，将其作为输入，然后用软件正式进行运行，最后测试这样的功能是否完成，完成情况是否符合预期。

论文第六章是总结部分，这部分主要用于会看论文设计过程遇到的问题和一些闪光点，并且对实现的功能做一些评估，是否可以用别的更好的方式将这些功能实现，如果还有时间之后可以进行的一些改进的方向。

# 需求分析

## 系统概述

项目设计主要分成两部分。第一部分是非本地部分，非本地部分主要用来实现一些远程服务器实际进行的一些操作，以及模拟一些网站才会进行的操作，比如创建仓库，pull request和fork，clone之类的一些和不同用户之间交互有关的功能。另外一部分就是本地部分，进行本地git仓库的一些操作。比如本地有一个git仓库，现在需要对这个仓库内容进行一些修改，然后才将其上传到远程服务器用于多人操作，所以有了打开本地git仓库，以及一些多人合作所需要的基本功能，比如add，commit，push到远程分支，创建删除切换本地分支，以及fetch和回到历史的某些版本功能。

## 解决问题和基本要求

项目需要对多人合作的项目式开发进行过程化以及系统化的管理。解决在多人合作中出现的许多常见问题，最终便于版本管理、多人合作、项目进度管理和历史项目的学习以及持续改进。最基本的要求就是实现上面的四个功能，每个功能初期的设想实现方式简述如下，最后详细实现会在本文后面章节详细阐述。

* + - 1. 版本管理

版本管理手动实现可以利用一个日志文件，手动记录每次Commit的内容以及修改的文件，文件改变的内容需要想办法保存下来。这一点并不好实现，设想可以通过一些第三方工具来实现，版本管理工具中有很多出色的三方工具，比如集中式管理的SVN和分布式版本控制系统Git，所以最后可以实现不用自己手动去实现版本管理这样的操作，而是直接借助于三方工具来实现。而GitHub和Gitee这样的网站也都是用了Git作为版本管理的工具实现的在线代码托管平台，并且用于多人合作，所以可以借鉴一下这些平台的实现方式，以及结合项目实践的特色完成版本管理。

* + - 1. 多人合作

多人合作需要解决如果多人完成一个项目，开始可以考虑的有腾讯文档那种直接多人进行合作，但是鉴于真正代码开发项目时的多人合作不太可能存在多人同时编辑一个文档的情况，都是分开完成相应功能之后再将各自完成的功能整合起来。另外多人合作还需要解决仓库存储问题，即需要模拟多用户创建属于自己的仓库，这样就必然需要一个空间去存储这些仓库，利用Gitee网站自己创建一个账号，命名特殊来模拟多人的一个情况是一种可能的选择，但是由于项目每次需要创建一个新项目，即需要在网页点击相应内容，所以需要网站爬虫的一些内容，结果是不太好爬。最后可行的办法就是自己租一个服务器，在服务器上用与用户名称同名的文件夹模拟用户，里面就可以用于自己存储仓库。

然后还需要如何实现多人合作的一个工作流程，这样的工作流程需要按照实际需要进行设定，不需要完全按照基本的gitflow流程，但是需要能够安全并且完全地实现多人合作的流程。

* + - 1. 项目进度管理

一个软件最好还可以实现一些进度管理之类的工作。需要知道整个项目的进程，不仅有利于项目进度的明示，而且有利于让开发不同功能的合作同伴知道当前自己的工作开发到什么步骤了，而且参考接下来可能的进度还可以让当前完成功能的人有意识地留下一些接口为了后来的功能实现更加方便。

* + - 1. 历史项目的学习以及持续改进

除了当前正在完成的项目之外，还需要将软件打造成类似于一种在线社区的 氛围，这样有利于软件开发的交流与借鉴，并且历史项目的展示也是多人软件管理的一种需要，只有将历史项目进行展示，才能让更多人可以知道这个项目，从而进行共同开发。

## 系统目标

系统的整体目标是在完成版本管理、多人合作、项目进度管理、历史项目学习，持续改进这四个主要的基本功能的前提下，为了更好地实现多人合作的项目式开发进行过程化、系统化的管理并结合一些已经运用Git技术的平台比如GitHub，更多地去实现一些方便的功能。

## 功能性需求

在正式实现软件上诉四个基本功能之前需要将功能具体细化，细分成更多可以直接明确实现的功能，然后下面具体阐述需要实现的功能。

* + - 1. 登录注册功能

需要管理历史用户和新建用户信息，数据库管理用户账号和密码对应信息并且对注册账号形式进行限制，即不规范字符不允许作为用户名。该功能重点在于实现之后代码的安全性以及鲁棒性。

* + - 1. 项目展示功能

这个功能其实是为了历史项目学习功能做铺垫。想要学习历史项目首先得能够知道历史项目有哪些，并且这个功能还可以用来让不同的用户相互交流借鉴项目。基本要求应该是需要可以展示项目目录的，在基本功能上还可以进行提升的部分就是，可以做成一种富文本，直接就可以点击查看想要查看的文本内容，至少二进制文件都可以直接查看。

* + - 1. 版本管理功能

版本管理至少应该可以按照一些历史版本的信息使得用户可以对历史版本有选择性地返回到该版本。如果在基本要求上对功能进行扩展就是在可以在返回版本前对该版本提供更多的信息，最好就是直接可以模拟回到该版本之后文件状态时怎么样的，文本中的内容状态是否是想要回到的版本，如果是再确定回到该版本，如果不是就不选择回到该版本。

* + - 1. Git基本操作功能

这是最难实现的一个功能，因为多人合作首先需要自己构思设计一个多人合作的流程，不一定需要标准的gitflow流程，但是需要一个可以方便并且安全实现多人合作的流程。但是有一点可以肯定的是，不论用什么样的流程去实现多人合作功能，必须要实现Git的一些基本操作:Clone，Push，Merge，Commit，Add等。这些功能的实现有些需要再远程服务器上实现，有些需要在本地实现。远程服务器可以考虑直接用java对远程服务器执行命令行操作实现，而本地就需要jgit提供的接口实现相应功能。

Git的基本功能还需要对本地仓库可以进行操作。一个合作者在工作时毕竟只能在本地进行代码的一些编辑，但是最后还是需要add并且commit和push修改过的内容，所以需要对本地仓库需要专门有一个模块用来实现上述功能，并且图形界面显示出来，完全不用借助于用户自己在命令行中运行git功能就实现本地仓库的管理功能。

* + - 1. 项目进度管理功能

项目进度管理功能可以简单实现，但是必须自己定义什么是项目进度，然后在完成相应任务或者功能的时候，可以对项目进度进行同步更新。

* + - 1. 界面美化功能

由于整个系统的图形界面是利用Swing来实现的，做网页版本前端需要花费很多时间，为了先把功能实现出来，所以最后还是采取了Swing来实现图形界面。而Swing本身的界面并不好看，所以在完成基本功能之后需要对Java Swing的界面进行一定程度的美化。

* + - 1. 安全性功能

Git本身在进行传输时，例如Push或者Clone，都是使用的SSH或者HTTP协议进行传输，所以如果在用SSH协议进行传输时需要对用户的SSH Key进行相应的管理，并且在每次连接的时候都采用RSA产生的Key进行确认的形式进行连接远程服务器，从远程服务器获取或者传输相应内容。

## 非功能需求

* + - 1. 正确性

基本实现所要求的功能，并且自己模拟用户实现起来时不会出现明显功能上或者明显卡住的问题。

* + - 1. 稳定性

如果软件出现一些多人同时的访问或者数据库在运行时网络中断能够保证数据不被影响，或者可以保存进度之后继续，不过前者实现起来更简单。

* + - 1. 可靠性

尽量做到软件的结果要么是正确的，要么就直接给出错误信息，不能程序还在正确运行，但是给出或者展示出的结果却是错误的。这样会给用户错误的误导，自己还认为程序运行正确，得到的结果也是正确的。

* + - 1. 性能、效率

软件在运行时不能点击之后要等很久才能执行完一个功能，需要对代码进行不断的优化，尽量使得用户等待程序响应时间很短，程序运行起来需要很流畅。

* + - 1. 安全性

数据库的安全性，不能被恶意连接，以及数据库内容需要备份等。多人项目之间需要有权限的区分，别人的项目只能进行读取，如果没有收到项目负责人的授权不能直接进行修改操作。远程服务器安全以及SSH连接安全需要有一定保障。

* + - 1. 可移植性

软件在不同操作系统以及不同电脑上可以同样进行运行，并且数据库中的内容需要保持一致性，这样需要在

* + - 1. 兼容性

对于软件中用到的接口之类，需要针对不同版本尽量做到更多的兼容，如果可以还可以单独写一些函数类用于提供接口。

* + - 1. 鲁棒性

软件功能虽然是这么设计的，但是用户可以进行很多匪夷所思的操作，不管用户进行了操作，系统程序尽量做到不能崩溃，如果用户进行了非常错误的操作，需要程序能够继续正常运行并且给用户提供提示信息表明刚才的操作不是正确操作，在之后用户进行正确操作之后还能正确返回结果。

# 系统设计

## 工作流程

开发的流程有很多，有很久之前有人在博客中提出的一种标准gitflow，图片引用参见图 3‑1A successful Git branching model.上述的工作流程是比较标准且复杂的git流程，如果只是一些比较简单的项目合作，可以采用GitHub式工作流程。



**图 3‑1A successful Git branching model**

本课题在结合GitHub特色的基础上自己设计了一个用于使用本软件的多人项目合作以及版本管理的工作流程。最开始设定的小型合作方式为，项目负责人创建一个项目，项目小组成员只需要使用该软件的本地仓库管理功能，将完成之后的代码直接push到项目负责人的仓库即可，即远程仓库中保存的只有项目负责人的仓库。但是由于软件本身还需要实现历史项目学习功能，所以远程服务器上的项目能被非该小组成员看见，如果有人恶意进行push毁坏项目，虽然有历史版本回滚的功能可以从侧面解决这样的问题，但是出于安全性考虑，还是应该结合GitHub工作流程的pull request特色，在想要对项目进行贡献之前进行一个请求。

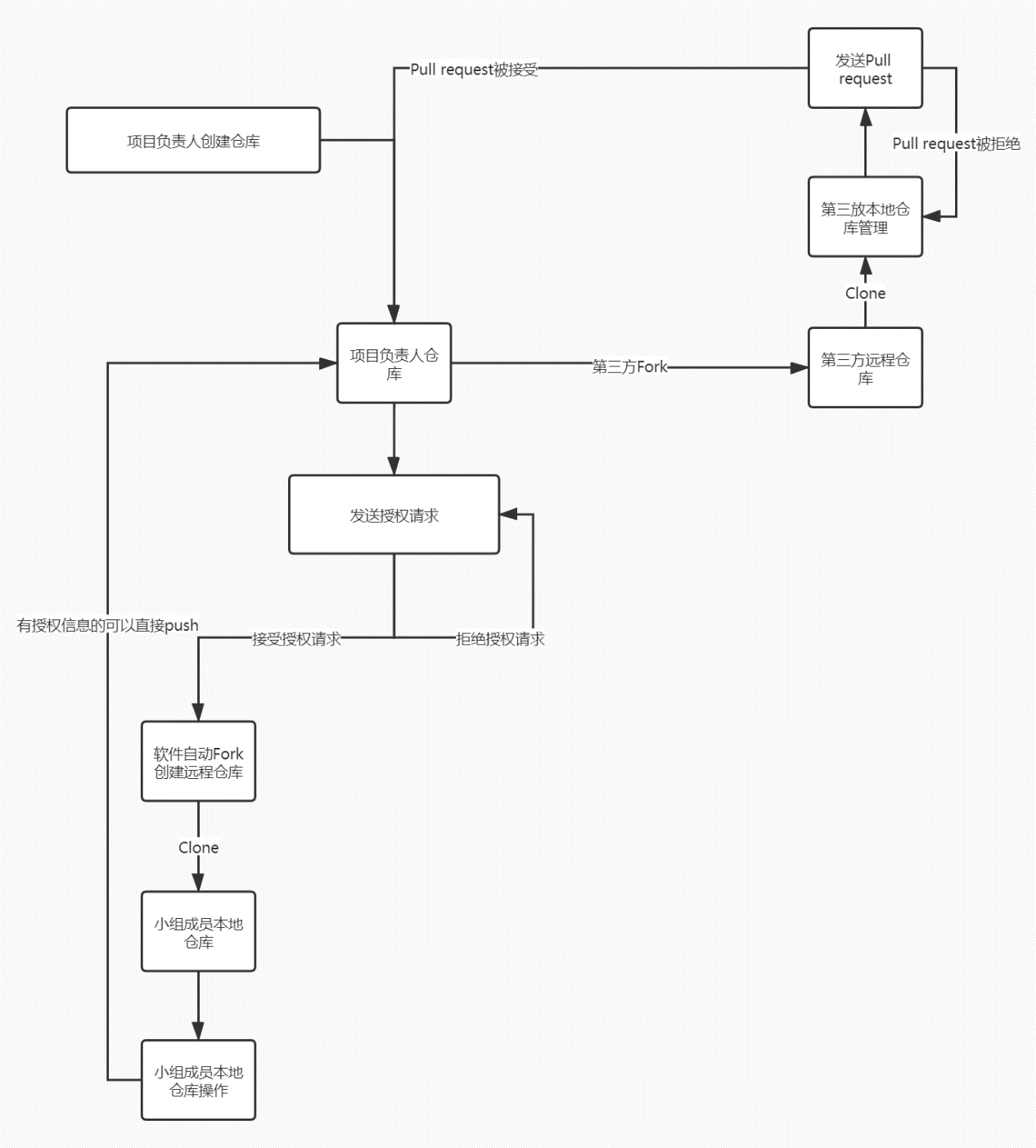
这里又出现两个问题，第一个问题是如果本身就是一起合作的，每次都需要创建pull request的请求会显得工作流程比较繁琐，因此需要添加一个授权功能，如果授权通过，团队成员就可以直接将修改好的代码推送到远程服务器上去而不需要先发送推送请求，等待请求批准之后才能推送。

第二个问题在于，甲在十点钟时向乙发送了一个pull request请求，乙在十一点看见这个请求，在十点到十一点期间，甲是不能对发送请求的分支进行修改的，否则乙看见的合并请求是会变成甲修改后的最新的请求。而且在乙同意之后，需要有一个甲的代码的备份，而此前所描述的过程中，一个团队只有一个项目负责人的代码放在远程服务器上，这显然是行不通的，团队所有成员在最开始都需要有一个同样的项目，因此需要结合GitHub的特色，加入Fork这样的功能，通过Fork项目负责人的项目，从而使得自己有一个和项目负责人同样的项目，之后再同步或者pull request请求被同意之后能够在远程服务器上也找到一个和小组成员本地仓库一样的仓库，即一个远程服务器上的备份。

综上所述，在此将工作流程清晰描述一遍。项目负责人在软件中创建一个仓库，软件实际上实现了在远程服务器上创建了一个仓库。然后项目负责人给小组成员发送授权邀请，小组成员在收到授权邀请之后可以决定是否接受这个授权邀请。如果接受了授权邀请，就会自动将负责人的项目在软件中执行Fork；如果没有接受邀请就不会进行任何操作。在此之后小组成员需要将自己远程服务器上的

项目Clone到本地，利用软件提供的本地仓库管理功能进行软件功能开发，完成开发之后可以直接push到自己的远程仓库，并且由于是已经授权的，所以也可以直接push到项目负责人的项目而不用发送合并请求。此外如果有第三方看到该项目并对此感兴趣想参与开发，就需要先自己Fork一个项目到自己的远程仓库，然后利用clone功能将自己的远程仓库放到本地，利用软件提供的本地仓库管理功能进行相应代码功能实现，之后还需要发送pull request请求，在项目负责人看到请求并同意合并之后直接将发送pull request的分支拉去到自己项目上来，从而实现多人合作。

整体的工作流程见图 3‑2使用软件工作流程图。



**图 3‑2使用软件工作流程图**

## 系统模块设计

* + - 1. 登录注册模块设计

登录注册模块用于登录软件，注册时创建相应数据更新数据库。

1. 登录

判断访问是否具有合法性，需要已经注册的用户。

1. 注册

用于注册用户账号。

* + - 1. 本地仓库管理模块设计

本地仓库模块用于管理从远程服务器上clone到本地的git项目进行管理，其中包含很多小的功能用于具体实现多人合作和版本管理。

1. Add功能

一套git仓库管理流程需要分别用到Add，Commit，Push等，因此需要Add作为该流程的一部分。Add是将文件放到暂存区(.git文件夹中的Index)，Add有助于分批次提交，降低commit每次文件上传的多少，并且可以保存一下Add的文件，如果不行方便回退。

1. Commit功能

Commit也是git仓库管理流程其中一步，用于将本地暂存区的文件，即已经add跟踪的文件提交到版本库，简单来说就是文件有三种状态，如果想要push必须要add和commit。

1. Push功能

完成Add和Commit功能之后，已经将需要修改的文件放到本地版本库中，但是多人合作需要文件在远程服务器上进行分享，所以还需要Push将本地版本库的分支推送到远程服务器上对应分支。

1. 分支功能

为了便于在本地进行管理，几乎所有的版本管理系统都支持分支。分支的主要作用在于可以把当前的工作从主线上分离开来，这样就可以在实现一些新功能的时候不会影响到主线。因此在辅助管理本地项目时，分支的作用是必要的，至少包括创建分支、切换分支、合并分支、删除分支这几项和分支有关的操作。

1. 版本管理功能

在本地需要对版本进行管理，并且需要和远程的版本进行同步，即本地版本回退之后需要远程也同时回到历史特定版本。该功能是最初对软件要求的基本功能之一，也是辅助对本地仓库进行管理的重要功能之一。

* + - 1. 项目展示模块设计

1. Pull request模块

由于安全性考虑，不能直接Push，需要添加pull request请求，具体添加Pull request原因参见第四章系统实现中有关部分。

1. Clone模块

Clone部分使得在远程服务器上进行的操作在本地同样可以进行。

1. Fork模块

Fork模块让所有的远程项目和本地项目对应了起来，即不论是否是项目负责人，都可以拥有开源的项目，并且所有用户在基本权限下只能修改自己的项目。

1. 项目进度管理模块

项目进度管理是本课题需要实现的一个基本功能，项目进度重点在与如何定义一个项目的进度，所以软件需要让项目负责人自己定义任务从而定义当前项目进度。

1. 项目展示，历史项目内容学习模块

软件开发基本要求中还有一项历史项目的学习，所以还需要有一个功能可以展示历史项目，至少可以展示历史项目中有些什么文件，进一步可以利用富文本将项目中的二进制文件直接在软件中打开。

1. 搜索模块

历史项目展示出来会很多，需要对项目有一个搜索操作，主要用于对项目名称进行关键词搜索，即如果搜索AB，则所有项目名称中将AB作为子序列的都可以被显示出来。

1. 授权模块

如果本身是项目小组成员中的一员，每次完成功能的一小部分的时候都要进行Pull request会使得开发过程显得很繁琐，所以可以添加授权功能使得项目负责人信任的人可以直接push到负责人仓库中去。

* + - 1. 其他模块设计

1. SSH Key管理

Git主要利用SSH和HTTP进行传输，这里主要使用SSH进行传输，考虑到传输过程中的安全性，以及在代码中不含有服务器的账号和密码情况，需要将SSH产生的公钥进行保存，管理SSH Key这样一个功能。

1. 消息设计

主要涉及授权消息传递和Pull request消息传递，需要传递这两种消息，如果还有别的消息需要传递也可以添加进来。

## 系统数据库设计

由于Git中已经存在一些数据库用于查询分支等，所以数据库不需要建立很多，比如分支查询可以不用建立数据库去存储分支的相关信息，只需要用Git提供的本地的分支查询以及远程服务器上运行Git命令查询远程服务器的分支即可。为了本文的思路逻辑，还是将一些没有实际在软件中创建的数据库，但是用到了Git内涵的数据库展示出来。

**表格 3‑1用户账号密码管理**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | 备注 |
| username | Varchar(20) | 非空 | 存储用户名，主键 |
| password | Varchar(20) | 非空 | 存储用户密码 |

表格 3‑1用户账号密码管理展示的是对用户账号密码进行管理的数据库，是比较基本的数据库。

表格 3‑2用户仓库管理用于记录用户和仓库的对应关系以及一些权限信息。在服务器上创建了与用户名同名的文件夹用于存放该用户的远程仓库，因此需要用数据库将这些文件夹和已经注册的用户建立一个连接关系。本身只有在表格 3‑1用户账号密码管理中出现的用户名称才能在表格 3‑2中出现，由于软件的图形界面关系这一点已经做到，不用再在数据库中添加条件。表格 3‑2中的fork\_from是用于支持Fork功能，用于存放fork项目拥有者的用户名，便于后续功能的操作。权限位用于判断该用户是否对该仓库拥有权限，即是否是该用户直接创建的，而非fork的。权限位的数据格式为tinyint类型，利用这一特点拓展了权限位的功能，使得其可以传输授权消息，利用权限位扩展的数字，比如2可以表示邀请授权消息，-2表示拒绝等，从而可以省略表格 3‑3授权请求。

**表格 3‑2用户仓库管理**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | 备注 |
| username | Varchar(20) | 非空 | 存储用户名，主键 |
| Repo\_name | Varchar(100) | 非空 | 存储项目名称，主键 |
| Fork\_from | Varchar(20) | 可以为空 | 记录fork项目的创建者 |
| Auth | Tinyint(1) | 可以为空 | 对项目是否具有权限 |

**表格 3‑3授权请求**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | 备注 |
| From\_user | Varchar(20) | 非空 | 发送授权请求用户，主键 |
| To\_user | Varchar(20) | 非空 | 接收授权请求用户，主键 |
| flag | Int | 可以为空 | 授权消息状态 |

Pull request消息由于存在分支到分支的细节，所以不能只改变表格 3‑2中的权限位从而扩展pull request消息功能，因此需要单独创建数据库表格用于该功能的传输，具体内容见表格 3‑4Pull request消息传输。Pull request消息传输表的功能和授权请求表的结构类似，只是在其上加入了分支之间的对应关系。

**表格 3‑4Pull request消息传输**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | 备注 |
| from\_user | Varchar(20) | 非空 | 发送拉取请求用户，主键 |
| From\_branch | Varchar(20) | 非空 | 发送拉取请求分支，主键 |
| Repo\_name | Varchar(100) | 非空 | 项目名称，主键 |
| To\_user | Varchar(20) | 非空 | 接收拉取请求用户 |
| To\_branch | Varchar(20) | 非空 | 接收拉取请求分支 |
| Flag | Int | 非空 | 消息状态，标志位 |

关于分支管理原本应该有表格 3‑5分支管理这样的表格，但是由于Git本身就有功能用于查询仓库分支，所以在本地进行仓库管理时可以直接通过JGit接口进行管理，而对于远程服务器上的仓库，可以利用Java对远程服务器进行操控运行相关命令，获取命令输出得到。

**表格 3‑5分支管理**

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Null | 备注 |
| From\_user | Varchar(20) | 非空 | 发送授权请求用户，主键 |
| To\_user | Varchar(20) | 非空 | 接收授权请求用户，主键 |
| flag | Int | 可以为空 | 授权消息状态 |

# 实现与测试

## 系统实现

文章该部分详细对基本功能进行阐述，每个功能主要分成对功能实现的描述和界面效果展示以及关键代码示例。

* + - 1. 登录界面以及新项目的创建

1. 功能实现

登录界面是软件的基本功能，主要涉及对数据库中用户的账号密码的一些管理，用户注册和登录需要在数据库中进行相应的查找和插入。这个部分还需要进行整体构思，之后用户的合作项目需要存储在什么地方。最开始的思路是不租用服务器，直接利用Gitee提供的空间来模拟有服务器的情况，即自己注册一个Gitee的账号，然后在自己软件创建用户时，只在数据库中进行记录，真正需要创建项目时，直接通过网络爬虫的方式发送请求，在Gitee中创建一个用户名加项目名称作为项目名称的项目，只是由于Gitee或者GitHub每次创建项目时都需要手动点击创建项目，网络爬虫写起来有点繁琐，所以还是租借了服务器。

整个流程为，在软件中创建用户时，除了在数据库中插入用户和密码信息，还需要通过Java对远程服务器进行操纵，在远程服务器根目录下创建一个和创建的用户名同名的目录，这个目录之后就用于存放该用户创建的所有新项目。并且创建的项目都直接用裸仓库实现，即不让用户可以在远程服务器中进行任意的更改，在目录中只能看见git运行所需要的文件，而没有用户自己创建的文件，无法当作工作仓库。

1. 界面展示

登录界面的效果见图 4‑1登录界面，注册时进行两项操作，一是插入数据库内容，二是在服务器根目录创建同名文件夹，在此不将服务器根目录创建文件夹的图片进行展示。



**图 4‑1登录界面**

1. 关键代码

**代码 4‑1登录界面关键代码**

|  |
| --- |
| 登录界面关键代码 |
| 1. //查询数据库 2. ResultSet rs = stmt.executeQuery("select \* from user where username = \'"+username+"\'"); 3. //限制注册用户名规范 4. if(username.matches("[a-zA-Z0-9\_]+")) 5. //对数据库进行操作 6. stmt.executeUpdate("insert into user values(\'"+username+"\',\'"+passwd+"\')"); 7. //利用Java对服务器进行操作，即在根目录创建文件夹 8. SSH ssh = new SSH(); 9. ssh.exec("mkdir "+username); |

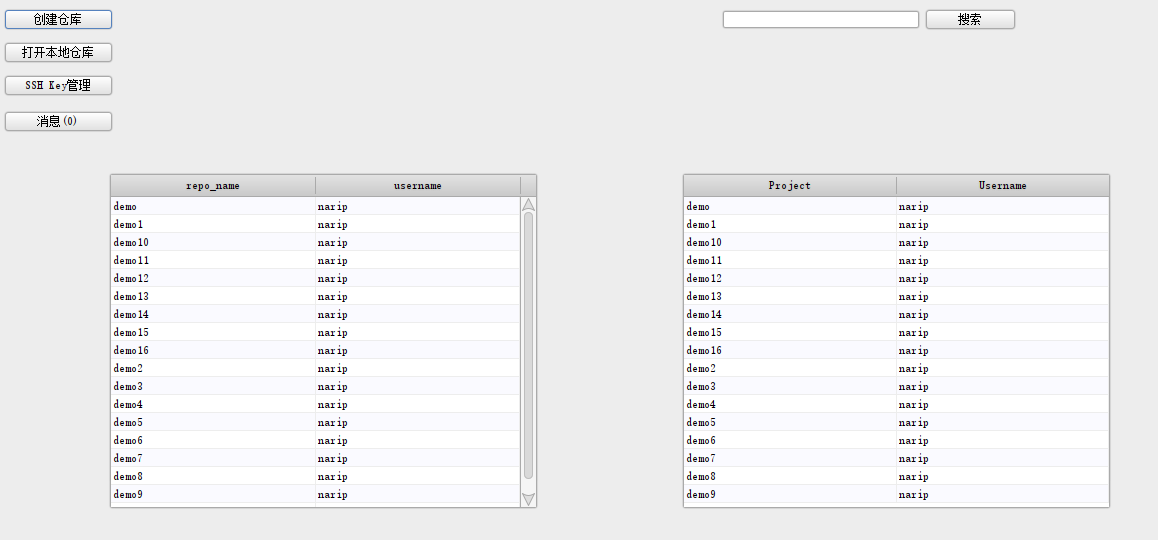
* + - 1. 历史项目展示和查询历史项目

1. 功能实现

该部分功能是为了后来的历史项目学习以及多人合作的功能需要。实现主要依靠数据库进行查询，然后利用Swing的表格将内容进行相应的展示，并且为了便于很多项目的观察，还需要加入边框滚轮。而查询功能最开始打算利用一些算法来快速实现，因为暴力查询速度会很慢，字符串匹配也不应该一个字一个字慢慢进行比较。但是软件开发过程中发现项目名称和对应的项目创建者都是存储在数据库中的，所以最后还是直接使用数据库的like关键字进行模糊查询，最后将查询到的结果放在另外一个界面展示。

1. 界面展示

历史项目的展示见图 4‑2历史项目展示，左边的窗口用于显示所有人的项目，右边的项目用于显示登录人创建或拥有的项目。



**图 4‑2历史项目展示**

1. 关键代码

**代码 4‑2历史项目展示关键代码**

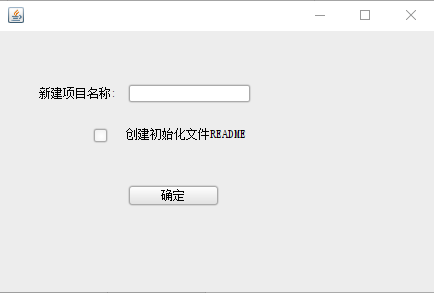
|  |
| --- |
| 历史项目展示关键代码 |
| 1. //利用vector存储表格中内容 之后将vector传入table容器，用于项目展示 2. ResultSet rs = stmt.executeQuery("select \* from repo"); 3. column\_name.add("repo\_name"); 4. column\_name.add("username"); 5. while(rs.next()) { 6. Vector rowdata = new Vector(); 7. rowdata.add(rs.getString("repo\_name")); 8. rowdata.add(rs.getString("username")); 9. row.add(rowdata); 10. } 11. //数据库like关键字进行查询 如果查询 demo 先处理成 %d%e%m%o% 然后再查询 12. StringBuffer text = new StringBuffer(search\_text); 13. for(int i = 0; i < search\_text.length(); i++) { 14. text.insert(2\*i + 1, "%"); 15. } 16. text.insert(0, "%"); 17. //然后再查询 18. ResultSet rs = stmt.executeQuery("select \* from repo where repo\_name like \'"+text+"\'"); |

* + - 1. 仓库创建

1. 功能实现

在图 4‑2历史项目展示中的左上角有显示创建仓库按钮，用于创建一个属于登录者自己的仓库，创建仓库一方面需要在数据库中记录创建者和仓库的信息，一方面需要记录是否是从别人的项目fork过来的，fork功能在后续会详细说明，还需要知道当前用户对该仓库的权限是怎样的，即是否直接拥有关系，也是用于区别从别人仓库中直接fork过来的原因。在图 4‑3仓库创建中还可以看到有是否需要用一个README的文件初始化仓库，这个功能GitHub上也有，这里不是完全照搬的。因为远程仓库需要安全的缘故，禁止有人在上面进行创建仓库文件等直接当成工程仓库的操作，所以远程仓库的创建都是创建的裸仓库，从代码 4‑3创建仓库关键代码中的init –bare选项知道是创建的裸仓库。但是创建裸仓库有一个问题在于，裸仓库才开始的时候没有任何分支，连master分支都没有，需要在最开始的时候上传文件到远程服务器上才能进行使用，即刚创建裸仓库必须使用push操作向上进行推送，不能进行fetch等操作，因为没有分支可以提供fetch。而初始化创建README文件是在远程服务器上，在根目录下创建.init目录，先在里面克隆相应项目，创建README文件之后再push到远程仓库上面，上面一系列的操作都是通过Java操作远程服务器运行代码实现的。

1. 界面展示



**图 4‑3仓库创建**

1. 关键代码

**代码 4‑3创建仓库关键代码**

|  |
| --- |
| 创建仓库关键代码 |
| 1. //更新数据库 2. stmt.executeUpdate("insert into repo (username,repo\_name,auth) values(\'"+username+"\',\'"+project\_name+"\',1)"); 3. //服务器上执行相应操作，下面显示的是有README版本的 4. SSH ssh = new SSH(); 5. ssh.exec("mkdir "+username+"/"+project\_name); 6. ssh.exec("git init --bare "+username+"/"+project\_name); 7. //克隆对应项目到默认目录 .init 然后上传README 最后删除.init中文件 8. ssh.exec("mkdir .init"); 9. ssh.exec("git clone root@39.97.255.250:/root/"+username+"/"+project\_name+" /root/.init/"); 10. ssh.exec("touch /root/.init/README"); 11. ssh.exec("cd /root/.init && git add . && git commit README -m \"README\" && git push origin master"); 12. ssh.exec("rm -rf .init"); |

代码 4‑3创建仓库关键代码中的SSH()是自己写的一个Java利用SSH操作远程服务器的类，详细代码不在此罗列。

* + - 1. 本地仓库管理

1. 功能实现

这个功能基本是多人合作功能的核心，这里需要实现很多细节的功能。首先需要从图 4‑2历史项目展示处点击打开本地仓库，然后在图 4‑4本地仓库文件目录中选择Git仓库就能打开本地仓库。打开之后如图 4‑5本地仓库管理所示，有很多细节的功能，后面一一展示。

文件状态显示。本地仓库打开之后会有四个部分用于显示，左上角的框图用于显示还没有add的部分，左下角用于显示已经add到但是还没有commit的文件，commit之后才算保存，否则并没有实际保存。右上角部分用于显示文件的改变，改变之后有什么地方不同。右下角用于用户在commit的时候输入commit的描述，每次commit时添加相应的描述有助于之后在想要返回历史版本的时候，按照commit的信息可以选择回到特定的版本。而且这个commit的信息是必须添加的，除了之前说的历史版本的问题之外，在本地使用Java操作Git的时候是利用的JGit这个Jar包，而这里面的方法是必须添加相应的commit信息的，所以这个地方如果没有填，软件会提示必须填。

同步功能，也就是add功能。Add功能利用JGit直接实现，主要是用于完成git的基本流程中的一步。

刷新功能。刷新前文提到的四个窗口界面，比如在本地用文件系统删除或者创建修改了文本之后，需要通过刷新功能使得四个展示窗口的内容更新。

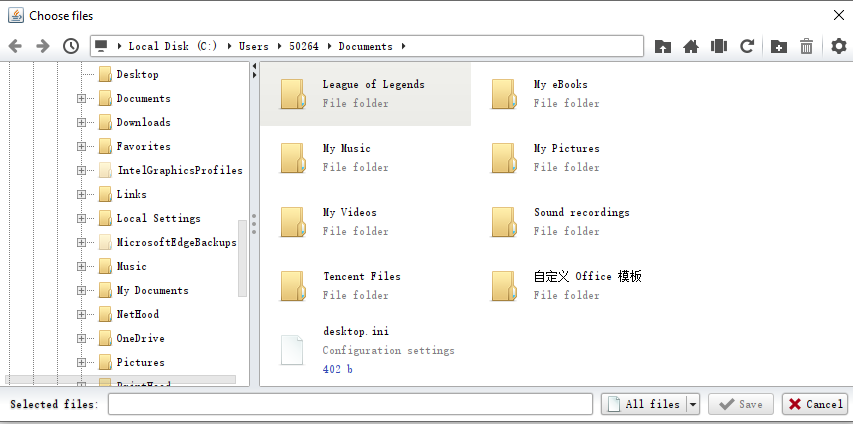
Commit功能。该功能的实现直接利用JGit实现，也是用于完成git中的基本流程的一步，类似于如果要实现数据库也必须有commit进行真正保存，将之前对文件所做的一些改变进行保存。Commit必须有commit的信息，一方面是代码实现的需要，另一方面是版本回退的需要。

Push功能。Push功能用于将在本地已经完成的操作传送到远程仓库，将之前commit了的新的改变传送到远程仓库。Push默认是推送到远程同名的分支上面，因为远程仓库是裸仓库的原因，所以Push推送到同名分支也是有理由的，如果在本地用了一个分支，再Push到远程仓库没有利用不用同样的名字，如果对分支名称不满意，完全可以在本地使用另外一个满意的名称作为分支名称，然后再进行推送。

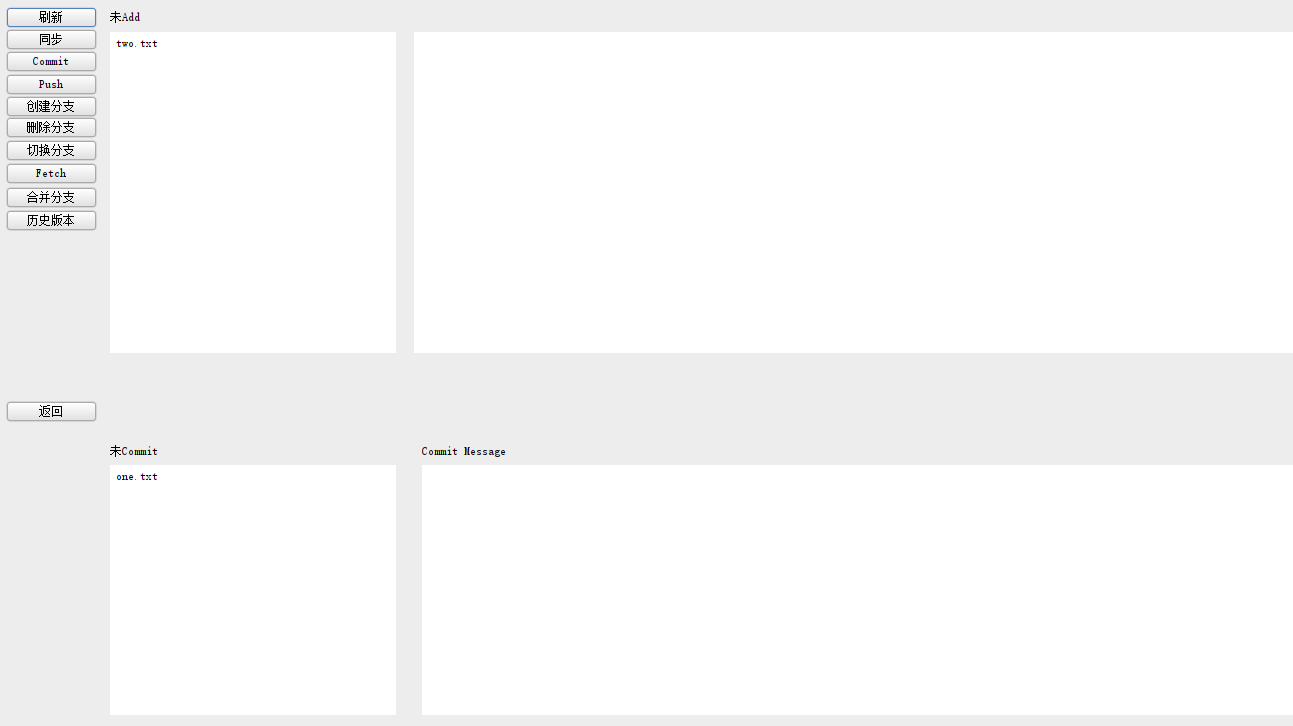
创建分支，删除分支，合并分支，切换分支功能。上述四个功能都是和git分支有关的功能。虽然只需要前文所述的功能就能基本完成多人合作功能，但是为了更好地进行多人合作，还可以利用git中分支的特色，在多人开发不同新功能的时候，可以自己在本地新建一个分支，将功能完成之后先存放到该分支，最后如果没有问题再将功能整合到一起。创建分支就是在原基础上创建一个分支，仓库在用README初始化之后有一个默认的master分支，如果还需要其他分支就需要用到创建分支功能。删除分支就是对分支进行删除。合并分支除了可以合并本地的分支之外，还可以合并远程的分支，这一点是因为前面的push功能只能push到远程同名分支上面去，而且不能对远程仓库进行操作，所以删除或者合并分支是可以进行远程仓库的分支合并到本地或者删除远程仓库的分支，这个功能也完成了git的pull功能，所就没有单独再列举pull功能。切换分支功能就是切换当前分支，push的时候也用到当前分支的概念，只能push当前分支，所以在push之前需要切换到想要的分支。

1. 界面展示

限于篇幅原因，这里不将所有功能界面一一展示，只用于展示主界面，具体功能阐述实现原理以及对功能进行详细介绍。



**图 4‑4本地仓库文件目录**



**图 4‑5本地仓库管理**

1. 关键代码

代码中为了节省篇幅将try{}catch{}省略掉。

**代码 4‑4本地仓库管理功能**

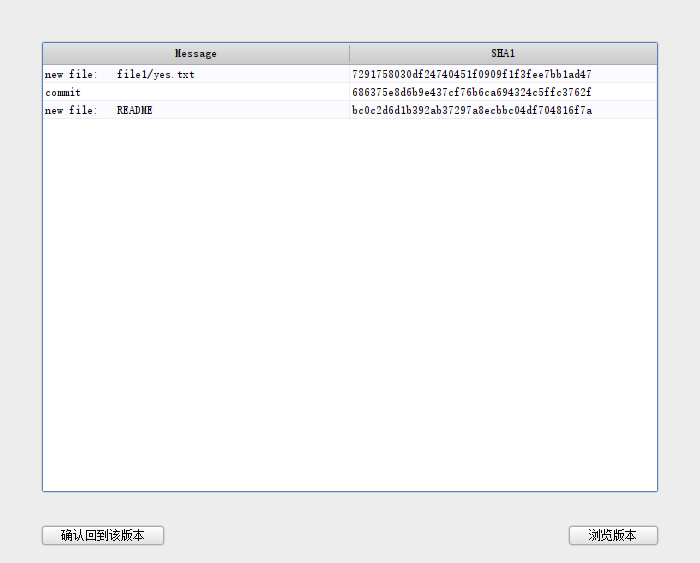
|  |
| --- |
| 本地仓库管理功能 |
| 1. Git git = new Git(repo); 2. //**add操作** 相当于命令行执行 git add 操作 3. Status status = git.status().call(); 4. git.add().addFilepattern(".").call(); 5. for(String set:status.getMissing()) { 6. git.rm().addFilepattern(set).call(); 7. } 8. **//commit操作** 9. git.commit().setMessage(textArea.getText()).call(); 10. //push操作，不过首先需要获取当前分支，然后再push 11. List<Ref> call; 12. call = git.branchList().call(); 13. for(Ref ref:call) { 14. branches.add(ref.getName().split("/")[ref.getName().split("/").length-1]); 15. } 16. //先切换分支 17. git.checkout().setCreateBranch(false).setName(from\_branch).call(); 18. git.push().setRemote("root@39.97.255.250:/root/"+login.username+"/"+repo\_name).setTransportConfigCallback(new TransportConfigCallback() {…}).call(); 19. //**分支操作** 以merge为例 20. git.pull().setRemote("origin").setRemoteBranchName(branch.get(comboBox\_1.getSelectedIndex())).call(); 21. //**fetch操作** 22. git.fetch().setRefSpecs("refs/heads/"+branch.get(index)).setTransportConfigCallback(new TransportConfigCallback() {…}).call(); |

* + - 1. 版本管理

1. 功能实现

版本管理功能从图 4‑5本地仓库管理中的历史版本中进入。版本管理依照commit的信息进行查看，用于决定回到什么版本。同样浏览历史版本也是为了方便查看选中版本里面的文件状态是怎样的，是否是想要返回的版本。版本信息在git中通过SHA-1值进行索引，按照相应的commit id回到相应版本。由于版本回退是直接在本地进行的，所以需要先push到远程服务器，将最近的版本信息先保存一下，或者当前版本是不需要的，就直接版本回退。在版本回退中加入了强行和远程仓库更新的步骤，即强行push，这样也会使得远程仓库和本地仓库的版本信息也变得一致，不过不能再回到返回版本之后的版本。

1. 界面展示



**图 4‑6版本管理**

1. 关键代码

**代码 4‑5版本管理**

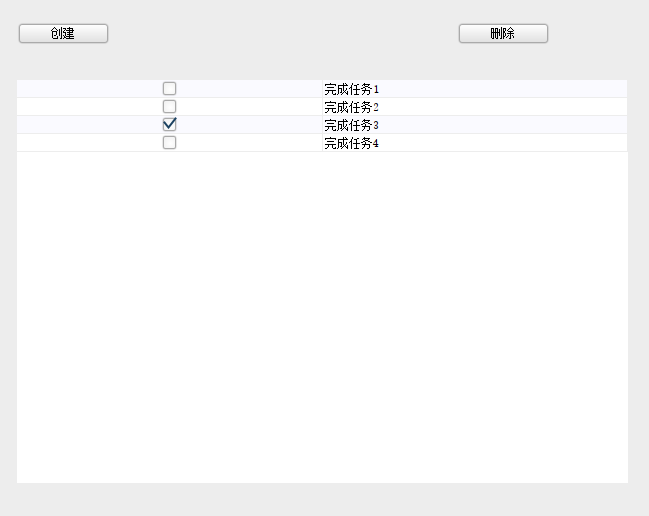
|  |
| --- |
| 版本管理功能 |
| 1. //展示版本信息 2. Git git = new Git(repo); 3. Iterable<RevCommit> log = git.log().call(); 4. for(RevCommit rev:log) { 5. model.addRow(new String[] {rev.getFullMessage(),rev.getName()});//message,sha-1 6. } 7. table = new JTable(model); 8. //版本回退 使用hard强制回退 9. git.reset().setMode(ResetType.HARD).setRef(SHA1).call(); 10. //然后强制push到远程服务器用于同步 11. git.push().setForce(true).setRemote("root@39.97.255.250:/root/"+login.username+"/"+proname).call(); |

* + - 1. 项目进度

1. 功能实现

项目进度功能的实现比较简单。主要因为项目进度到底进行到什么地方不是很好去定义，所以只设计了一个表格，然后项目负责人可以在其中自己定义项目任务，完成任务之后项目负责人可以勾选任务从而将其标志为项目已经完成。任务是否勾选对应于任务是否完成，这些信息需要保存到数据库中用于同步。

1. 界面展示



**图 4‑7项目进度管理**

1. 关键代码

**代码 4‑6项目进度管理**

|  |
| --- |
| 项目进度管理功能 |
| 1. //创建进度 2. stmt.executeUpdate("insert into process values(\'"+prouser+"\',\'"+proname+"\',\'"+desc+"\',0)"); 3. //更改状态信息，即完成改为未完成，未完成改为完成 4. stmt.executeUpdate("update process set isfinish = 0 where username = \'"+prouser+"\' and repo\_name = \'"+proname+"\' and description = \'"+description+"\'"); 5. stmt.executeUpdate("update process set isfinish = 1 where username = \'"+prouser+"\' and repo\_name = \'"+proname+"\' and description = \'"+description+"\'"); 6. //删除状态信息 7. Stmt.executeUpdate(“delete from ...”) 8. //table用于显示checkbox 9. tc.setCellEditor(table.getDefaultEditor(Boolean.class)); 10. tc.setCellRenderer(table.getDefaultRenderer(Boolean.class)); |

* + - 1. 项目展示

1. 功能实现

项目展示功能主要是用来展示项目的结构，也是为了实现历史项目学习功能的一个基本功能。项目展示主要是以目录结构的形式对项目进行描述，没有对项目中的文件内容进行展示，这是之后可以改进的地方。项目展示是以分支为基础的，不同的分支有不同的文件内容和目录内容，所以需要先对想要展示的分支进行相应选择之后才能进行相应展现，默认显示的是master分支的内容。

1. 界面展示

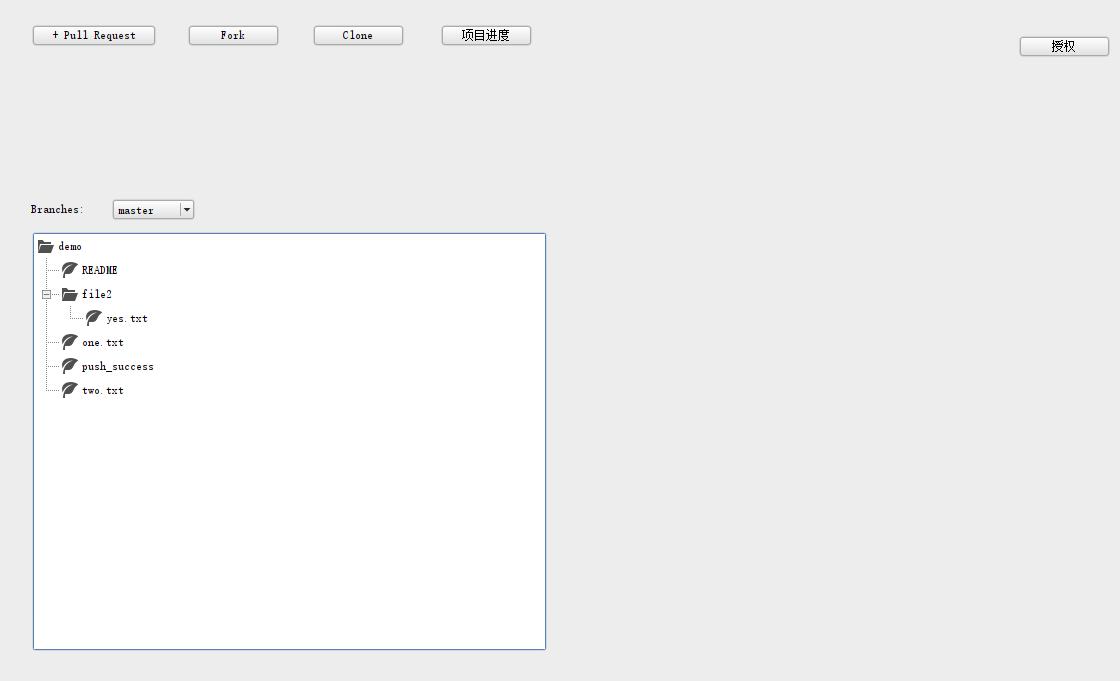


图 4‑8项目展示功能

1. 关键代码

**代码 4‑7项目展示**

|  |
| --- |
| 项目展示功能 |
| 1. //预先写好的递归构造文件目录的方法 2. temp = (Vector<String>) output.clone(); 3. if(temp.isEmpty()) { 4. DefaultMutableTreeNode son = new DefaultMutableTreeNode(nodename,false);//false表示没有子目录 5. }else { 6. for(int i = 0; i < temp.size(); i++) { 7. if(temp.get(i).split(" ")[1].equals("tree")) { 8. //是目录，生成节点，并递归调用，添加里面的节点 9. parent.add(GetAllFiles(temp.get(i).split(" ")[1],temp.get(i).split(" ")[2].split(" ")[0])); 10. }else if(temp.get(i).split(" ")[1].equals("blob")){ 11. //是文件，直接生成节点，加到父节点 12. tempnode = new DefaultMutableTreeNode(temp.get(i).split(" ")[1]);//false表示没有子目录 13. parent.add(tempnode); 14. } 15. } 16. //调用上面的方法进行构造树结构 用于展示 17. Node = login.ssh.GetAllFiles(project\_name, combobranch.get(index)); 18. tree\_model = new DefaultTreeModel(Node); |

* + - 1. 项目间交互

1. 功能实现

项目间交互包括fork功能和clone功能。Clone功能就是将远程服务器的仓库拷贝到本地仓库，然后在本地进行编辑，本地的仓库就可以利用前文描述道的打开本地仓库的功能进行操作。Fork功能是必须的，是在pull request请求的基础上加上的。最开始的工作流程是直接所有项目成员都可以直接访问项目，但是这样并不安全，因为存在第三方可以恶意提交的情况，主要为了安全性考虑，需要添加pull request的功能，即在代码直接对远程仓库造成影响之前，需要项目负责人判断是否需要这些代码，如果需要就拉取，如果不需要就废弃表示拒绝了pull request。但是pull request如果被另一个人同意了，需要直接就拉取，因此需要发起pull request的人在远程服务器上保存有项目信息，所以存在fork这样一个功能，fork实现起来就是远程服务器的自我clone，只是由于远程仓库需要全部是裸仓库，所以在clone时，也需要添加上clone的条件参数以表明此次克隆是裸仓库克隆，即克隆的结果也需要是裸仓库。

1. 界面展示

界面和图 4‑8项目展示功能一样。

1. 关键代码

**代码 4‑8项目间交互**

|  |
| --- |
| 项目间交互 |
| 1. //clone功能的实现 2. //打开窗口用以选择存储路径 3. download\_path window = new download\_path(); 4. if(temp.isEmpty()) { 5. DefaultMutableTreeNode son = new DefaultMutableTreeNode(nodename,false);//false表示没有 6. //clone操作 7. CloneCommand cloneCommand = Git.cloneRepository(); 8. cloneCommand.setURI("root@39.97.255.250:/root/"+download\_user+"/"+download\_project); 9. cloneCommand.setDirectory(new File(textField.getText()+"/"+download\_project)); 10. cloneCommand.setTransportConfigCallback(new TransportConfigCallback(){…}; 11. cloneCommand.call(); 12. //Fork操作 13. //服务器端操作 14. ssh.exec("git clone --bare root@39.97.255.250:/root/"+project\_user+"/"+project\_name+" /root/"+username+"/"+project\_name); 15. //数据库操作 16. stmt.executeUpdate("insert into repo values (\'"+username+"\',\'"+project\_name+"\',\'"+project\_user+"\',0)"); |

* + - 1. 消息管理

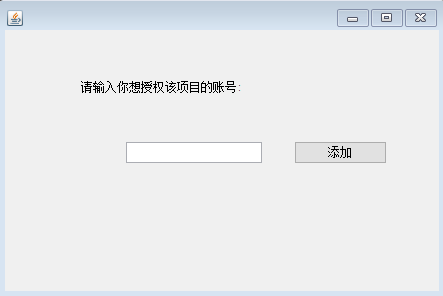
1. 功能实现

消息管理主要用于两个方面，一个是pull request的消息，另外一个是用于授权是否接受的消息。

授权功能。如果每次都使用pull request功能让同一个团队中的人将代码上传到负责人处，会显得很麻烦，所以最好是如果确实同时在一个项目合作，项目负责人可以选择是否授予其中成员一些权限，使其可以直接push到自己的项目中来，而不用每次都发送一个pull request，因为发送pull request不是每次都能立马收到信息的，如果负责人没有立马收到信息而发送消息的人在发送合并请求之后需要必须等待合并请求的结果，因为如果在此期间发送消息的人修改了自己的代码，合并的时候会将修改过后的代码一起合并，即有一个信息是否收到的时间差。所以如果授权，就可以直接push而省略时间差，让多人合作更流畅。授权的信息传递比较简单，授权的信息传递不需要再在数据库中新建一个表格用于授权信息的传递，可以直接利用已经存在的表格:repo，这个表格中记录了项目创建用户名，项目名称，项目从谁那里fork的以及是否对该项目有权限。如果是从别人那里fork过来的，项目就没有权限，如果是自己创建的项目，项目就有权限。而数据库中存储权限这一项的时候是用的tinyint类型，在数据库中用来表示布尔类型，但是也可以表示除0和1之外的数字，所以如果授权了，可以将此处的数值从0变为1，并且如果要授权的人没有这个项目会在他点击确认授权的时候自动fork过去，如果对方拒绝，此处的数值会变为-2，如果同意会变成1用于表示同意并且对该项目具有了权限。

Pull request消息传递。Pull request本来也想利用上面的技巧省略创造一个数据库的表格，但是在pull request时候还需要加入分支的关系，即从申请者的什么分支合并到对方的什么分支，需要有分支的选择，存储的信息量比较大，所以不能再像之前那样利用技巧省略数据库的一个表格，最后还是得利用数据库创建一个表格专门用于pull request消息的处理，并且需要利用标志位告知对方是否接受了pull request的请求或者拒绝了pull request的请求。

1. 界面展示



**图 4‑9授权功能**

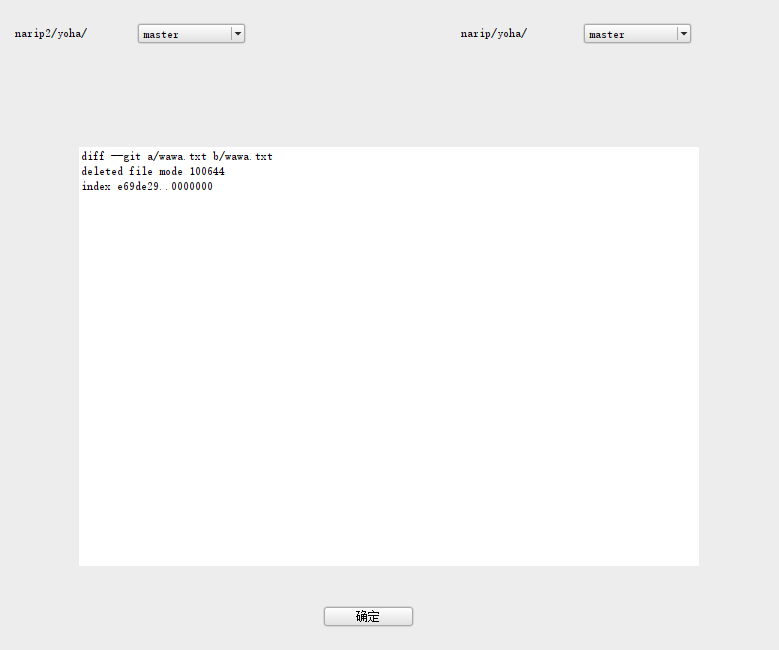
在图 4‑2历史项目展示中有一个消息的图标，点击打开就是消息的处理界面，软件主要有和授权相关的消息以及和pull requst相关的消息。



**图 4‑10消息交互的展示**

图 4‑10消息交互的展示中只用授权消息进行了示例展示，从表格中消息的区分和处理是通过鼠标点击之后获取的一行的数据，然后进行字符串分割获取的关键字进行处理的。

如图 4‑11pull request界面显示，可以选择创建从什么分支到什么分支的pull request，并且在下方会将这次将要创建的pull request信息显示出来。



**图 4‑11pull request界面**

1. 关键代码

获取界面的字符之后进行字符串处理的条件判断太多，此处重点指出一些代码。

**代码 4‑9消息处理**

|  |
| --- |
| 消息处理 |
| 1. //授权信息的一些数据库操作 利用repo表格权限位进行信息传输 2. stmt.executeUpdate("insert into repo values(\'"+textField.getText()+"\',\'"+project\_name+"\',\'"+project\_user+"\',2)"); 3. //如果接受pull request 4. //在服务器上进行执行Merge操作 5. ssh.exec("cd /root/"+from\_user+"/"+repo\_name 6. +" &&git push origin "+from\_branch+":"+to\_branch); 7. //更新数据库 8. stmt.executeUpdate("update pull\_request set flag = 1 where from\_user = \'" 9. +from\_user+"\' and from\_branch = \'"+from\_branch+"\' and repo\_name = \'"+repo\_name+"\'"); |

* + - 1. 界面美化

1. 功能实现

Java本身自带的Swing界面太丑陋，需要使用其中的UIManager处理，运用LookandFeel，不过需要一些主题包或者是皮肤，一些细节部分也需要更详细的处理，所以最后直接用的一个接口，直接调用现成的界面功能。不过美化后的界面的一个缺点就是一打开的时候需要载入几秒钟。

1. 界面展示

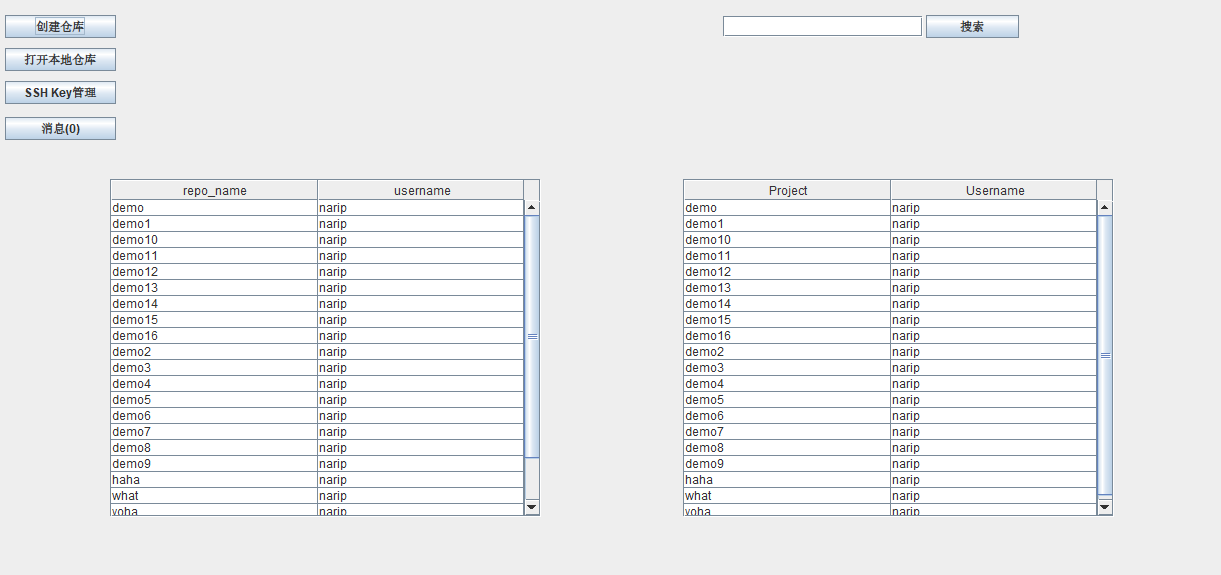
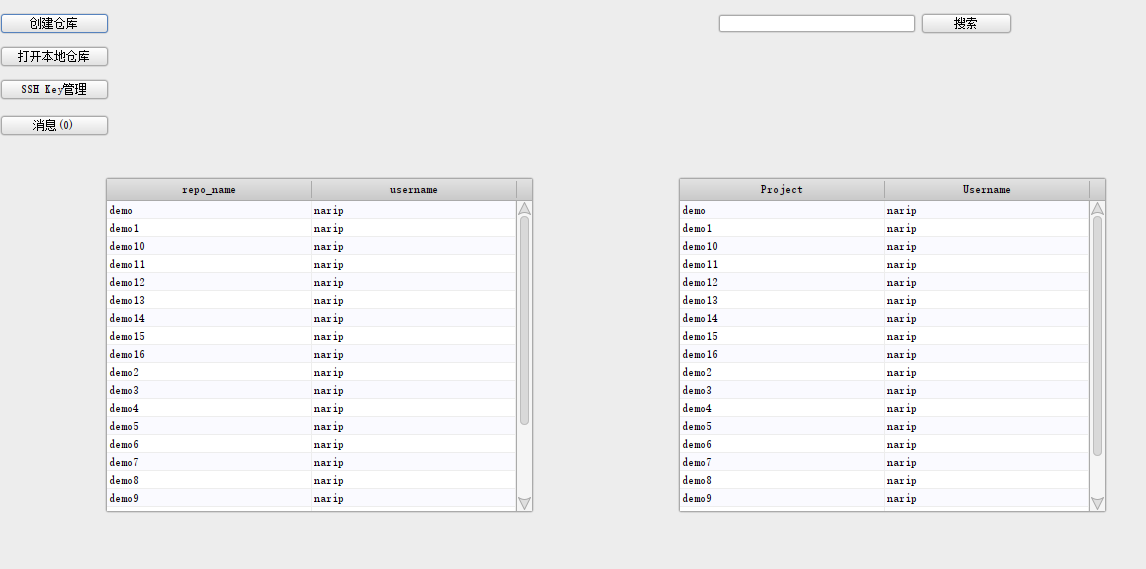


图 4‑12界面美化前的界面



**图 4‑13界面美化后的界面**

1. 关键代码

**代码 4‑10界面美化**

|  |
| --- |
| 界面美化 |
| 1. //利用lookandfeel进行界面美化 2. WebLookAndFeel.install (); |

* + - 1. SSH Key管理

1. 功能实现

由于软件中设计到数据的传输，Git一般采用的是ssh协议和http协议，而在软件中没有进行选择，一律默认的是ssh协议，可以直接将服务器账号和密码写入代码中从而使得所有用户可以直接对远程服务器的内容进行传输，但是为了安全性考虑还是需要利用每个人的ssh密匙进行连接，将公钥存储到远程服务器上的authorized\_keys中，然后将私钥在本机中的目录存到数据库中，之后在clone或者push等需要认证的时候，通过私钥的目录索引以及之前已经加在远程服务器上的公钥可以安全并正确进行连接从而实现数据传输。由于同一个用户可以在不同电脑上进行开发，所以同一个用户可以绑定多个ssh的key，因此在数据库中不应当用用户名作为主键，而应当直接用ssh的key作为主键。在利用ssh的key直接进行克隆的时候，如果一个用户拥有多个绑定的密匙，就无法知道用哪个密匙进行连接，所以在数据库中还应该添加计算机的mac地址这一项，用于在连接计算机的时候根据当前主机使用的mac地址直接进行连接。

1. 界面展示

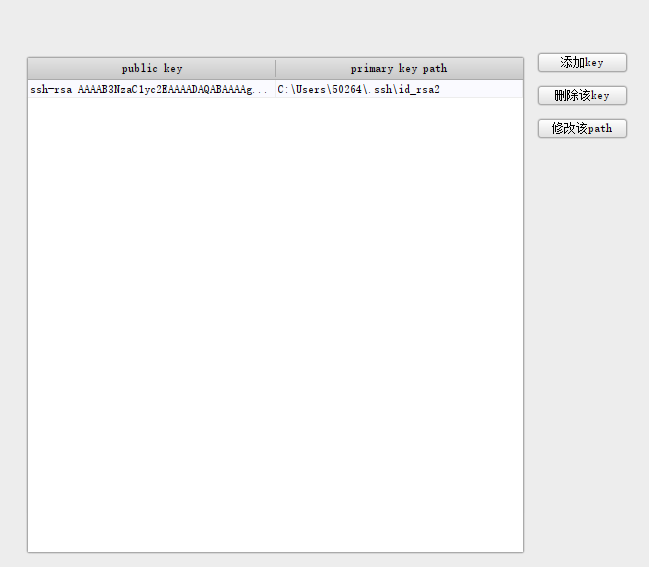


图 4‑14SSH Key管理

1. 关键代码

代码 4‑11SSH Key管理

|  |
| --- |
| SSH Key管理 |
| 1. //table中存放key 2. while(rs.next()) { 3. model.addRow(new String[] {rs.getString("pub\_key"),rs.getString("path")}); 4. } 5. //更新数据库 6. path\_Create = textField.getText().replace("\\", "\\\\"); 7. stmt.executeUpdate("insert into rsa values(\'"+key\_Create+"\',\'"+login.username+"\',\'"+path\_Create+"\',\'"+login.Mac+"\')"); 8. //远程服务器的autotized\_key添加新的public key到文件末尾 9. ssh.exec("echo \'"+key\_Create+"\' >> /root/.ssh/authorized\_keys"); |

## 系统测试

# 总结

项目展示功能可以扩展成富文。

合并有冲突之类

将四个基本要求的功能进行阐述，看最后是如何完成的

# 致 谢

不需要“花团锦簇”，不需要“无病呻吟”，需要的是：洗尽铅华、朴素是真。

# 参考文献

# 附录1 代码及相关附件

# 附录2 文献英文原文

按前述规范排版！

# 附录3 文献中文译文

按前述规范排版！

1. [] 张萱.用SVN实现软件的版本控制[J].电子技术与软件工程,2017(10):63. [↑](#endnote-ref-1)
2. [] 刘悦之.基于Git的分布式版本控制系统的设计与实现[J].科技传播,2012,4(22):197-198. [↑](#endnote-ref-2)
3. [] 包永红. Git在 JavaWeb技术实践教学课程中的应用与探索[J]. 内蒙古农业大学学报，2019，21(107):32-36. [↑](#endnote-ref-3)
4. [] 周伟，陈柳. Git 在软件版本管理实验教学中的应用[J]. 信息技术与信息化，2016，5:97-100. [↑](#endnote-ref-4)
5. [] 徐娅. Gitb版本控制工具在团队协作项目中的应用[J]. 智能计算机与应用，2019，9(5):341-343. [↑](#endnote-ref-5)
6. [] 曾少宁. 基于GitHub平台的协同式实验教学方法[J]. 计算机教育，2016，(12):144-148. [↑](#endnote-ref-6)
7. [] 任维，李伟，袁海娣. 基于GitHub平台的高校软件类课程教学方法研究[J]. 电脑知识与技术，2019，15(2):81-82. [↑](#endnote-ref-7)
8. [] 陶健. 基于git托管平台的代码存储系统的设计与实现[D]. 武汉:华中科技大学. 2018. [↑](#endnote-ref-8)
9. [] 侯效永，李良伟，孙召. 基于Git的代码托管平台JLUCODE[J]. Computer Era，2016，(12):32-38. [↑](#endnote-ref-9)
10. [] 字凤芹. 基于Git的协作小组学习资源库的建设与研究[D]. 云南:云南大学. 2016. [↑](#endnote-ref-10)