封面

内封

摘要

Abstract

目录

前言

自从世界上第一台全自动电子计算机“埃尼阿克”诞生以来，计算机技术的发展极其迅速。在移动互联网及大数据的背景下，计算机已经融入到人类生活的方方面面。从衣食住行到工业生产，计算机已经成为人类社会不可或缺的重要组成部分。随着计算机科学与技术的快速发展而来的各种规模的软件工程，常常需要整合人力以及各种软件开发工具，这时版本控制的概念应运而生。

版本控制作为软件工程的一种技巧，需要让项目中的多个开发人员的开发进度保持同步。在软件开发过程中，参与项目的人数多不代表一定可以加快项目的开发速度，因为软件产品的抽象性，人员之间的沟通与管理极有可能占用很多资源，导致整体开发效率下降，而版本控制系统的作用就是协调开发人员的工作，同步不同开发者的进度，尽可能减少版本管理占用的资源。

现代软件开发已经不是“软件=数据结构+算法”的模式，而是要适应各种变化的需求，使软件架构有足够的灵活性，不至于因为需求的变化而推倒重来。现代软件源于互联网的发展，互联网使得软件进入新的时代。互联网深入生产生活的方方面面，因此需要处理一些难以用算法表达的业务逻辑，如银行的金融业务不仅很复杂，很难用算法表示，而且经常调整，导致需求多变。这就需要版本控制适应变化的需求，可以进行同一系统下的分支管理。

一个算法可以从源代码中识别出来，而业务逻辑则很难从代码中看出来。一些企业级软件即使留下源代码，后来者也很难明白其中的业务逻辑。随着老一代程序员的退休，他们也将业务逻辑带走了。以至于后来者不敢轻易重构遗留代码，怕一个误解造成巨大损失。为了解决这样的问题，版本控制系统需要记录下迭代过程中的版本描述，以简化维护人员分析项目的过程。

源代码是开发人员的产品，因此对版本控制系统的安全性及稳定性有较高的要求。软件及硬件上的错误是无法预料的，同时也很难避免，因此对于版本控制系统的文件系统这方面，分布式架构是一个非常好的选择。分布式系统具有较高容错性，将文件操作等较慢的操作分发给多个服务器也有助于提高文件读写效率，采用RAID技术的底层文件系统对文件安全也有很大保障。

目前主流的版本控制系统有github、svn、rcs和cvs等，每一种版本控制系统都有其优点，我选择具有代表性的功能进行实现。在开发这个系统的过程中，应灵活运用本科四年学到的理论知识，以理论结合实际，磨练编程技巧，拓展开发技术。

1 需求分析

1.1 需求分析引言

1.1.1 项目背景

本系统的名称为基于ASP.NET的源代码版本控制系统。版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。随着软件系统规模的日益扩大和复杂程度的日益增长，软件工程师或网页设计师可能会需要保存某一系统的源码或页面布局文件的所有修订版本，采用版本控制系统是个明智的选择。有了版本控制系统就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态。用户可以通过系统比较文件的变化细节，查出最后是团队中的哪个成员修改了哪个地方，从而找出导致问题出现的原因，是谁在何时报告了某个功能缺陷等等。使用版本控制系统通常还意味着，就算对整个项目中的文件进行大幅度改动，也照样可以轻松恢复到原先稳定版本的样子，但额外增加的工作量却微乎其微。

版本管理是软件配置管理的基础，它管理并保护开发者的软件资源。本系统作为软件开发过程中的辅助工具，目的在于减少版本控制及管理过程中使用的人力资源，协调进行不同工作的开发人员的工作，同步不同开发者的进度。系统需要尽可能保存每一阶段的工作成果，尤其是源文件，以保证每个阶段性工作成果的安全，这样任何时候都可以方便的找回原来的工作成果；另一方面系统应能够快速检索工作成果，比如很容易找到某个版本的文件，或者最主要的几个阶段性成果，并且能够很容易预览需要的文件。

1.1.2 项目风险

由于开发能力有限以及时间安排上的任务冲突，可能无法实现开题报告所述的全部功能，但可以保证实现基本的版本控制功能。对于使用者，风险主要来自于程序编写过程产生的漏洞，以及系统的初期版本可能不稳定。此外，因为系统的重点在于版本控制，分布式文件系统可能仅有部分功能可以启用。初期版本系统的部分功能在质量及性能上可能无法达到本需求文档的要求，但后续版本可能会有提升。

1.1.3 本系统的目标用户

本系统的目标用户分为普通用户及系统管理员。普通用户是本系统的最终用户，一般为软件开发人员或软件项目管理人员，这一类用户应具有版本控制的基本知识，可以理解计算机的基本操作及运行原理。使用者应尽可能保存软件开发过程中的各个阶段，以及各版本对应的描述性说明及详细开发文档。对于每一个使用本系统的软件项目，本系统的使用频度应对应于项目的开发速度，可以每次修改源代码就使用本系统，也可以在某开发进程达到某阶段后使用本系统。

系统管理员对本系统进行管理，其具有直接操作后台数据库以及配置本系统的权限（如备份、回滚数据库等），并在系统出现问题时对普通用户进行回应。

1.2 功能需求

1.2.1 系统范围

代码仓库：具有受版本控制的所有文件的完整修订历史的共享数据库。

分支：分支是指目录和文件的现有原始树的副本。分支的生命周期是从某事物的副本开始的，并从此副本处移动，生成自己的历史。通常创建分支以尝试新功能，同时不影响具有编译器错误和小问题的开发的主分支。

普通用户可以创建软件项目对应的代码仓库；代码仓库有且只有一个主分支，该分支无法被删除且至少有一个版本；用户可以在代码仓库下创建分支，分支的根可以不是主分支；用户可以查看选中的代码仓库，也可以查看选中的分支及版本；用户可以向代码库中选中的版本上传文件、下载文件或创建文件夹；系统应记录每一次版本迭代的信息，如描述和时间等;用户可以在代码仓库下提交评论和文件，文件经过仓库拥有者审核后可以合并到对应版本中。

代码仓库、分支及版本应该具有项目描述和创建时间等描述性信息。仓库拥有者可以创建、删除分支，可以将当前分支回滚到之前的某一个版本。仓库拥有者也可以删除指定仓库。

普通用户可以创建账号；系统应对已登录用户和未登录用户的权限进行验证。具有权限的用户可以预览某一版本的文件结构和文件内容，并可以签入或签出选中的文件。如果时间允许，系统应实现不同版本文件之间的差异对比。

分布式文件系统应具有文件操作的基本功能，并且分布式文件服务器应是可拓展的。

1.2.2 系统体系结构



版本控制系统下分为九个模块：

用户管理：可以进行注册、登录、记录用户信息等操作。

代码仓库管理：用户登录成功后便可以在自己的工作空间中查看已创建代码仓库的列表，同时也可以在这里创建和删除代码仓库。

分支管理：创建和删除分支，也可以添加描述信息。

版本控制：签入或签出文件，回滚分支到指定版本，推出新版本。

文件系统：由分布式服务器构成的文件系统，独立于web服务器。Web服务器与分布式文件系统之间采用自定义通许协议。

数据库封装：将页面与数据分离，把对数据库的操作封装成库。

权限检查：对于某些操作进行用户身份检查，来确定是否有操作权限。

差异对比：对比不同版本文件的差异。

1.2.3 系统总体流程



用户使用系统时需要先注册，注册成功后后台会记录用户的账号、密码、用户名、用户描述和注册时间等信息。用户可以使用注册时使用的账号登录系统，如果登录失败则返回登录界面重新登录，如果登录成功系统会重定向至用户工作空间，在该界面用户可以查看和管理已拥有的代码仓库。

如果用户想查看或管理某个代码仓库，可通过工作空间中对应的链接进入代码仓库详细信息界面，用户可以在这里查看代码仓库的描述信息，如代码仓库创建时间、说明等信息。用户可以在仓库详细信息界面进行版本控制，如管理分支和文件。分布式文件系统独立于web服务，web服务可通过特定的通讯协议对文件系统进行控制，如进行文件读写等操作。

1.2.4 用户界面概述

系统至少要具有如下界面：

主页：访问系统时打开的第一个界面。主页中有系统的介绍，用户可以通过主页跳转到注册和登录界面。

注册页面：用户在注册页面输入用户名、账户和密码等信息，通过后台审核后即可获得账户和工作空间。

登录页面：用户可在登录页面使用注册时提交的账号登录。登录成功后自动跳转至该用户工作空间界面。

用户工作空间页面：在这个页面可以查看用户名、用户描述信息，也可以看到该用户已创建的代码仓库。用户可以通过该页面跳转到代码仓库浏览页面。工作空间的拥有者可以在该页面创建新的代码仓库。

创建代码仓库页面：在该页面，工作空间拥有者可创建新的代码仓库。用户输入仓库名称和描述信息后可创建代码仓库，系统会自动初始化代码仓库的主分支和第一个版本。每个代码仓库有且仅有一个名为master的主分支，且主分支无法被删除。

代码仓库浏览界面：这个页面包含了本系统的核心功能。代码仓库的拥有者可以在这个页面进行版本控制，包括分支管理、版本管理和基本的文件管理。在该页面代码仓库的拥有者可以跳转到创建分支界面和推出新版本界面。该页面也可以进行基本的文件访问操作。

创建分支界面：仓库拥有者可以在该页面向代码仓库提交新的分支，新分支的第一个版本的文件结构由选中的起始版本决定。用户应输入分支名和描述信息。用户不能创建名为master分支，且每个分支至少会包含一个版本。

推出新版本界面：用户在该界面输入新版本名称和描述，服务器审核成功后就可推动选中分支发展。

评论查看界面：代码仓库的拥有者可以在这里查看其他用户对该仓库提交的评论。

用户界面的总体色调以蓝色和白色为主，文件系统在初期版本可以使用控制台界面。界面为级简风格，尽量采用风格相近的颜色搭配，尽量减少颜色种类以突出网页重点内容。

1.2.5 需求分析

软件需求分析是启动一个软件工程的重要工作阶段，本小节将概述性的说明本系统中用户、需求、系统功能单元之间的联系。

系统整体用例图



|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 游客注册账户 |
| 用例标识号 | 1 |
| 简要说明 | 为游客创建新账户。数据库应记录用户名、登录账户、密码、用户描述和账户创建时间。 |
| 前置条件 | 用户已打开注册界面。 |
| 基本事件流 | 1.用户输入用户名。  2.用户输入用于登录的账户。  3.用户输入登录密码。  4.用户输入用户描述。  5.用户单击创建按钮。  6.服务器校验用户输入。如果输入符合规则，系统自动生成新用户的ID并记录账户创建时间；否则要求用户重新输入注册信息。 |
| 其他事件流 | 如果用户关闭注册页面，注册程序终止。 |
| 异常事件流 | 1.用户输入不符合规则或输入为空，要求用户重新输入注册信息。  2.用户输入的账号已存在，要求重新输入数据。 |
| 后置条件 | 1.数据库记录新用户信息。  2.页面跳转到登录界面。 |
| 注释 | 可以有同名用户，但用户ID是唯一的。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 用户登录 |
| 用例标识号 | 2 |
| 简要说明 | 让拥有帐号的用户登录系统。 |
| 前置条件 | 用户已打开登录界面。 |
| 基本事件流 | 1.用户输入账户及密码。  2.用户单击登录按钮。  3.服务器校验登录信息。如果账户密码不匹配，要求重新输入登录信息。 |
| 其他事件流 | 1.如果关闭页面，登录程序终止。  2.如果用户无账户，可跳转到登录界面。 |
| 异常事件流 | 1.如果输入不符合规则，要求重新输入。  2.如果密码错误，要求重新输入密码。 |
| 后置条件 | 1.用户登录系统。  2.页面跳转到用户工作空间。 |
| 注释 | - |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 系统管理员帐户管理 |
| 用例标识号 | 3 |
| 简要说明 | 系统管理员帐户管理 |
| 前置条件 | 1.系统管理员成功登录数据库。  2.系统管理员具备访问用户表的权限。 |
| 基本事件流 | 1.系统管理员选择操作：对用户进行增加、删除、修改或查询操作。  2.提交修改。 |
| 其他事件流 | - |
| 异常事件流 | - |
| 后置条件 | 数据库对用户表进行修改。 |
| 注释 | - |



|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 浏览代码仓库 |
| 用例标识号 | 4 |
| 简要说明 | 所有用户都可以浏览公开的代码仓库。 |
| 前置条件 | 用户处于代码仓库界面。 |
| 基本事件流 | 1.显示版本关系图。  2.用户选择查看关注的版本。  3.预览显示用户选中的文件。 |
| 其他事件流 | 用户可评论代码仓库。 |
| 异常事件流 | 1.非仓库拥有者尝试修改代码仓库，系统提示无权限。  2.版本跳转参数无效提示参数错误。 |
| 后置条件 | - |
| 注释 | - |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 查看工作空间 |
| 用例标识号 | 5 |
| 简要说明 | 用户可在工作空间中查看已创建的代码仓库。 |
| 前置条件 | 用户处在工作空间页面。 |
| 基本事件流 | 1.显示该工作空间拥有者的用户名，描述信息。  2.显示工作空间拥有者创建的仓库列表。 |
| 其他事件流 | 1.用户可以通过仓库列表跳转到被选中的浏览代码仓库页面。  2.用户可以通过单击创建按钮跳转到创建仓库页面。 |
| 异常事件流 | 非工作空间拥有者尝试创建仓库是显示无创建权限。 |
| 后置条件 | - |
| 注释 | - |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 创建代码仓库 |
| 用例标识号 | 6 |
| 简要说明 | 为已登录用户创建新的代码仓库。 |
| 前置条件 | 1.用户已登录。  2.用户从工作空间页面跳转至该页面。  3.URL参数正确。 |
| 基本事件流 | 1.用户输入代码仓库名称和描述。  2.用户单击创建按钮。  3.服务器校验用户输入。如果输入为空或内容不符合要求，提示重新输入。 |
| 其他事件流 | 如果用户关闭页面，创建程序终止。 |
| 异常事件流 | 1.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。  2.URL参数错误，系统提示参数错误并关闭当前页面。 |
| 后置条件 | 1.系统为新代码仓库分配唯一的代码仓库ID，并记录代码仓库创建时间。  2.系统为新仓库提交第一个分支和第一个版本。新分支与新版本的描述信息与代码仓库的描述信息一致，名称统一为master。  3.设定主分支的版本范围。  4.将代码仓库更新为指向最新版本。  5.在文件系统中创建对应文件。  6.反馈是否创建成功。如果失败进行异常处理。 |
| 注释 | 1.每个代码仓库有且仅有一个主分支，其名称为master，且无法被移除。  2.每个分支（包括主分支）至少包含一个版本。 |



|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 仓库管理 |
| 用例标识号 | 7 |
| 简要说明 | 代码仓库拥有者对代码仓库进行版本控制。 |
| 前置条件 | 1.用户已登录。  2.用户对当前代码仓库具有修改权限。 |
| 基本事件流 | 1.用户可通过单击创建分支按钮跳转到创建分支界面。  2.用户可签入签出文件。  3.用户可以在该页面查看版本关系图，并可跳转到选中版本，也可以回滚到选中版本。  4.用户可以删除分支。  5.用户可以通过页面中的文件浏览器管理文件。 |
| 其他事件流 | 1.如果页面关闭，终止管理程序。 |
| 异常事件流 | 1.如果无操作权限，系统提示无权限。  2.如果未登录，跳转到登录界面。  3.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。 |
| 后置条件 | 反馈操作状态。 |
| 注释 | 主分支无法删除。每个分支至少包含一个版本。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 创建分支 |
| 用例标识号 | 8 |
| 简要说明 | 为代码仓库创建分支。 |
| 前置条件 | 1.用户已登录。  2.用户拥有指定代码仓库。  3.URL参数正确。  4.用户已选择作为分支起点的目标版本。 |
| 基本事件流 | 1.用户输入分支名和分支描述。  2.用户单击创建按钮。 |
| 其他事件流 | 如果用户关闭页面，创建程序终止。 |
| 异常事件流 | 1.输入为空或内容不符合规则，要求重新输入。  2.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。  3.URL参数错误，系统提示参数错误并关闭当前页面。 |
| 后置条件 | 1.以用户输入创建分支，自动生成ID和创建时间戳。  2.创建新分支的第一个版本，其名称与描述对应于分支的名称和描述。  3.设定新分支的版本范围。  4.反馈操作结果。如果失败则进行异常处理。 |
| 注释 | 用户创建的分支不能与主分支同名（master）。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 文件系统 |
| 用例标识号 | 9 |
| 简要说明 | 封装对文件的基本操作。 |
| 前置条件 | 1. 分布式文件系统服务启动成功。  2.web服务已启动。 |
| 基本事件流 | 根据接收到的指令进行对应的文件操作。 |
| 其他事件流 | 文件读写过程中与web服务器连接中断，文件操作终止。 |
| 异常事件流 | 程序崩溃，文件系统自动重启。 |
| 后置条件 | 反馈操作结果 |
| 注释 | 文件系统可以独立于web服务运行。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 删除分支 |
| 用例标识号 | 10 |
| 简要说明 | 删除指定代码仓库下选中的分支。 |
| 前置条件 | 1.用户已登录。  2.用户拥有指定代码仓库。  3.URL参数正确。  4.用户已选择分支。 |
| 基本事件流 | 1.从选中分支的起点，迭代删除分支中的版本和对应的文件结构。  2.从数据库清除选中的分支。 |
| 其他事件流 | 如果用户关闭页面，删除程序终止。 |
| 异常事件流 | 1.如果没有选中要删除的分支，系统提示参数错误。  2.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。  3.URL参数错误，系统提示参数错误并关闭当前页面。 |
| 后置条件 | 1.反馈操作结果，如果操作失败则进行异常处理。  2.操作成功后跳转到选定的代码仓库。 |
| 注释 | 主分支无法删除。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 回滚到选中版本 |
| 用例标识号 | 11 |
| 简要说明 | 把指定代码仓库下某一分支回滚到选中版本。 |
| 前置条件 | 1.用户已登录。  2.用户拥有指定代码仓库。  3.URL参数正确。  4.用户已选择分支和目标版本。 |
| 基本事件流 | 1.从选中版本的后面第一个版本开始，迭代删除分支中的版本和对应的文件结构。  2.在数据库中更新选中的分支。 |
| 其他事件流 | 如果用户关闭页面，则该进程终止。 |
| 异常事件流 | 1.如果没有选中目标分支和版本，系统提示参数错误。  2.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。  3.URL参数错误，系统提示参数错误并关闭当前页面。 |
| 后置条件 | 1.反馈操作结果，如果操作失败则进行异常处理。  2.操作成功后跳转到选定的代码仓库。 |
| 注释 | 每个分支至少有一个版本存在。 |

|  |  |
| --- | --- |
| 用例说明 | 详细信息 |
| 用例名称 | 推进新版本 |
| 用例标识号 | 12 |
| 简要说明 | 用户可以通过此页面手动推出新版本。 |
| 前置条件 | 1.用户已登录。  2.用户拥有指定代码仓库。  3.URL参数正确。  4.用户已选择目标版本。 |
| 基本事件流 | 1.用户输入新版本名称和描述。  2.用户单击创建按钮。 |
| 其他事件流 | 如果用户关闭当前页面，创建进程终止。 |
| 异常事件流 | 1.输入为空或内容不符合规则，要求重新输入。  2.如果数据库或文件系统的操作中出现异常，则删除新增数据，回滚到未添加新代码仓库的状态。  3.URL参数错误，系统提示参数错误并关闭当前页面。 |
| 后置条件 | 1.使用用户的输入信息创建新版本。  2.更新分支起止范围，如果更新的是主版本，则还要更新代码仓库记录信息中的最新主版本。  3.反馈操作结果。如果操作失败则启用异常处理流程。 |
| 注释 | - |

1.3 非功能需求

初期版本Web服务器应至少可承受50人同时在线，数据库服务器应至少可承受每秒500次查询，单个文件服务器在极限情况下应至少能承受每个会话125kb/s的文件传输流量。Web页面的加载时间应小于3秒。

1.4 非功能需求

1.4.1 运行环境规定

版本控制系统运行环境：

处理器：Intel(R) Pentium(R) 4 2.4 GHz 或 AMD(R) Athlon(TM) 64 2800+ 处理器 或任何 1.8Ghz Dual Core处理器。

显卡：NVIDIA(R) Geforce(TM) 6600 以上或 ATI(R) Radeon(R) 9800Pro以上 。

内存：8GB。

硬盘空：500GB。

操作系统：Windows8.1及Windows Server 2012以上的Windows操作系统。

客户端运行环境：

主流浏览器如chrome、firefox，IE等。

1.4.2 支持软件

开发环境使用Visual Studio 2015，开发语言采用C#、Javascript、C++、Html。数据库采用Microsoft SQL Server 2012，数据库控制中心使用Visual Studio 2010。数据库连接使用微软企业库。

2 系统总体设计

2.1 系统设计原则

一个优良的系统设计，强调模块间保持低耦合、高内聚的关系。本系统在设计和实现的过程中应尽可能遵守以下原则：

开闭原则(OCP):一个软件实体应当对扩展开放，对修改关闭。“抽象化”是OCP的关键。

里氏代换原则(LSP)：在一个软件系统中，子类应该可以替换任何基类能够出现的地方，并且经过替换以后，代码还能正常工作。“继承”是LSP的关键。

依赖倒转原则(DIP)：要依赖于抽象,不要依赖于具体。或者说是：要针对接口编程，不要对实现编程。“规范抽象”是DIP的关键。

接口隔离原则(ISP)：使用多个专门的接口比使用单一的总接口要好。也就是说，一个类对另外一个类的依赖性应当是建立在最小的接口上的。“多重继承”是ISP的关键。

组合/聚合复用原则(CARP)：在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分：新的对象通过向这些对象的委派达到复用已有功能的目的。“组合/聚合”是CARP的关键。聚合指的是整体与部分的关系，在定义一个整体类后，再去分析这个整体类的组成结构。从而找出一些组成类，该整体类和组成类之间就形成了聚合关系。组合表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期，一旦整体对象不存在，部分对象也将不存在。

迪米特法则(LoD)：一个对象应当对其他对象有尽可能少的了解。“传递间接的调用”是LoD的关键。

2.2 系统结构化分析

系统主要采用“用户界面-业务逻辑层-数据访问层”的结构。实体类用于映射数据库的表结构。使用这种设计模式的理由是，用一种业务逻辑、数据访问和界面显示分离的方法来组织代码，将业务逻辑集中到一个部件里面，在改进和个性化定制用户界面及用户交互过程的同时，不需要重新编写业务逻辑。



系统功能逻辑关系图



上下文图

普通用户和游客可以浏览系统中文件，可以进行注册和登录等操作，也可以对代码仓库进行版本控制；系统管理员有对整个系统的完全控制权限，能调整系统的运行状态，修改数据库，更新文件。在正常情况下系统会对普通用户、游客和系统管理员的一切操作进行反馈，提示用户的操作是否成功，显示用户所处的状态。

用户在版本控制系统中选择对文件系统和代码仓库的操作。用户可以选择对代码仓库、分支或版本进行版本控制操作,这些操作由数据库记录，系统会反馈对数据库的操作结果。版本控制系统通过内部协议，对分布式文件系统进行远程过程调用，完成文件操作。分布式文件系统会维持版本控制系统中，用户创建的代码仓库、分支和各版本的对应文件结构。

0层图

用户管理模块通过数据库操作模块完成注册、登录等操作,用户操作结果由控制系统模块直接返回。控制系统通过操作数据库和分布式文件系统来达到目的，数据库操作结果先返回到版本控制模块，将异常信息和错误代码转换为可读性较强的文本，作为普通用户的操作结果返回给用户。系统管理员对整个系统具有完全的控制权限，可以直接操作各个模块，也可以直接修改文件系统中的文件结构和数据库的内容。



用户管理1层图

没有帐号的游客可以通过用户管理模块注册新账户。已拥有账户的用户可以通过用户管理模块登录到系统，登录后对工作空间进行管理，也有修改已拥有代码仓库和账户信息的权限，同时可以向任意代码仓库提交评论。所有操作都经过控制系统，对数据库中的用户表进行操作；操作结果有控制系统反馈回用户。

控制系统1层图

控制系统作为业务逻辑层，整合对数据库实体的操作。Web服务通过该模块修改数据库，该模块将异常、错误代码和运行状态封装为可读性较强的文本返回给用户。

用户表数据字典

|  |  |
| --- | --- |
| 项目 | 内容 |
| 名称 | 用户表 |
| 别名 | user\_table |
| 使用地点 | 用户管理、控制系统。 |
| 使用方法 | 作为储存用户信息的容器。 |
| 描述 | 用户表=用户ID+用户名+用户帐户+用户密码+注册时间+用户描述+用户类型。 |

代码仓库数据字典

|  |  |
| --- | --- |
| 项目 | 内容 |
| 名称 | 代码仓库表 |
| 别名 | warehouse\_table |
| 使用地点 | 版本控制、控制系统 |
| 使用方法 | 记录用户创建的代码仓库的描述性信息。 |
| 描述 | 代码仓库表=代码仓库ID+用户ID+所属组织ID+代码仓库类型+创建时间+代码仓库描述+主分支最新版本ID+代码仓库名称。 |

分支表数据字典

|  |  |
| --- | --- |
| 项目 | 内容 |
| 名称 | 分支表 |
| 别名 | branch\_table |
| 使用地点 | 版本控制、控制系统。 |
| 使用方法 | 记录代码仓库下的所有分支。 |
| 描述 | 分支表=分支ID+代码仓库ID+用户ID+起始版本ID+结束版本ID+时间戳+分支名称+分支描述。 |

版本表数据字典

|  |  |
| --- | --- |
| 项目 | 内容 |
| 名称 | 版本表 |
| 别名 | version\_table |
| 使用地点 | 版本控制、控制系统。 |
| 使用方法 | 记录版本控制系统中所有的版本信息。 |
| 描述 | 版本表=当前版本ID+所属代码仓库ID+创建者ID+前一版本ID+下一版本ID+时间戳+版本名称+版本描述+所属分支ID。 |

2.3 数据库整体设计



版本控制系统实体关系图

2.3.1 数据表设计

版本控制系统中数据库表应遵守的原则：关系中的每个属性都不可再分；数据库表中的每个实例或行必须可以被唯一地区分；一个数据库表中不包含已在其它表中已包含的非主关键字信息。所有字段不可以是null，且必须具有初始值。默认情况下，值类型的字段默认值为0，字符串类型的默认值为长度为0的字符串。

用户表是用于记录用户信息的表。用户表具有一个作为主键的用户ID，用来唯一识别用户，用户名可以不唯一。用户账户必须唯一，因为用户将使用账户与密码登录系统。用户表还应记录账户的创建时间和用户的个人描述，以方便用户之间的理解和交流。

代码仓库表用于记录用户创建的代码仓库的信息。代码仓库表具有一个具有一个作为主键的代码仓库ID，用于唯一识别每个代码仓库。该表应记录是哪个组织的哪个用户在什么时间创建了代码仓库，还要记录代码仓库的名称、类型、描述信息和主分支的最新版本ID。在创建代码仓库时，代码控制系统会自动创建主分支，并添加主分支下的第一个版本。每个代码仓库只能有一个名为master的分支作为主分支。版本控制操作应遵循每个分支至少具有一个版本这一原则。

分支表用于记录代码仓库中出现的分支信息。分支表具有一个作为主键的分支ID，用于唯一标识每个分支；具有记录该分支属于哪个代码仓库的字段，同时具有记录该代码仓库拥有者的字段。分支表记录一个分支的名称、创建时间、描述，还应记录分支下各个版本的起点和终点。代码仓库的主分支无法被删除，用户无法创建名为master的分支，每个分支至少包含一个版本。

版本表用于记录版本控制系统中的所有版本。版本表内的数据可以通过前向和后向指针形成有向图的结构。该表应记录版本是属于哪个代码仓库的哪个分支，记录是由哪个用户在什么时间创建，还要记录版本的名称和描述。版本由主键版本ID唯一标识。

组织表用于记录用户所属的组织机构，该表记录组织机构的名称描述，具有一个组织ID作为主键。评论表记录用户向代码仓库提交的评论的内容。

2.3.2 数据访问层设计

数据访问层即DAL层，也称为是久层，其功能主要是负责数据库的访问。实现对数据表的Select（查询），Insert（插入），Update（更新），Delete（删除）等操作。如果加入ORM的元素，那么也包括对象和数据表之间的映射以及对象实体的持久化。数据库访问层的主要职责是：读取数据和传递数据。

2.4 用户界面总体设计

Index.aspx为进入系统的主页。主页中概述的介绍系统的功能，用户也可以通过此页面跳转到注册新账户或登录页面。

Register.aspx为注册账户页面。该界面上有记录新账户用户名、登录帐号、密码和用户描述信息的文本输入框，此外还有一个创建按钮。用户单击创建按钮后，如果输入的内容满足数据库的存储要求，则新账户创建成功，自动跳转至登录窗口，引导用户登录新账户；如果创建失败，则要求用户重新输入注册信息。

Login.aspx为用户登录界面。该界面上有记录用户帐户和密码的输入框，还有一个登录按钮。用户单击登录按钮后，服务器校验用户输入的账户与密码是否匹配。如果账户和密码正确，服务器记录登录状态，用户浏览器自动跳转到用户工作空间界面；如果出现错误则要求用户重新输入登录凭据。

User\_page.aspx为用户工作空间。该界面的上方显示用户名和用户描述，这里的数据来自于注册时用户填写的数据；通过这里的查看评论按钮，用户空间拥有者可以跳转到查看评论的页面。页面的中部有创建代码仓库的按钮，用户单击后可跳转到创建代码仓库页面。界面的下方显示该用户创建的代码仓库的预览信息，包括跳转链接、创建时间和项目名称，点击跳转链接，用户浏览器可跳转至选中代码仓库的详细信息页面。

Create\_page.aspx为创建代码仓库页面。该页面具有记录新代码仓库名称、项目描述的文本输入框，此外还有一个创建按钮。用户单击创建按钮，服务器检查用户权限和用户提交的数据，如果符合创建规则，服务器在数据库中创建对应数据，在文件系统中创建对应的文件结构。如果用户输入为空或不符合数据库存储条件，提示用户重新输入相关内容。

Warehouse\_page.aspx为代码仓库详细信息页面，也是实现版本控制功能的核心页面之一。该页面的顶端显示当前代码仓库的名称和项目描述，这部分数据来自于代码仓库被创建时用户提交的数据。该页面的中上部分为该代码仓库的版本结构图，该部分显示项目的分支和版本之间的联系。

页面的中下部分为版本控制功能选择区。在这里首先显示用户当前浏览的版本的详细信息、版本所属分支、版本创建时间、版本描述，用户在这里可以选择对当前版本的操作，这里有签出当前版本、推送新版本、创建分支、删除当前分支按钮。用户单击按钮后服务器先验证当前用户是否是当前代码仓库的拥有者，如果是则执行对应操作，并返回最新版本信息；如果不是则提示没有权限。通过单击推送新版本和创建分支按钮，用户可以跳转到对应页面。该部分也会显示用户在版本结构图选中版本的部分信息，如所在分支和版本ID。用户可以通过单击跳转到选中版本按钮重定向至选中版本的详细信息页面。此外用户单击回滚至选中版本按钮将选中分支回滚到指定版本。

该页面的下部为一个简单的文件浏览器，用户可以在此浏览当前版本的文件结构，浏览选中的文件，上传或下载选中的文件。该部分还包含文件列表，文件内容查看组件，返回至版本根目录按钮，返回上级菜单按钮。

Create\_branch.aspx为创建新分支界面。该界面包含记录新分支名称和描述信息的文本输入框，还有创建按钮。用户单击创建按钮，服务器检查提交内容和用户权限，如果符合创建规则，则记录新分支并创建对应的文件结构，如果不符合创建规则，提示用户重新输入并提交数据。

Create\_version.aspx为推送新版本页面。与创建新分支界面类似，该界面包含记录新版本名称和描述信息的文本输入框，以及创建按钮。用户单击创建按钮，服务器检查提交内容和用户权限，如果符合创建规则，则在对应分支创建新版本并创建对应的文件结构，如果不符合创建规则，提示用户重新输入并提交数据。

Comments.aspx为查看评论页面。该页面显示一个，每一项包含其他用户对代码仓库、版本和分支的评论信息。

2.5 文件系统设计



分布式文件系统结构图

分布式文件系统在版本控制系统中，是一个独立于Web服务的组件。将支撑业务的文件系统与Web服务分离，一方面模块化功能加快设计及开发进度，另一方面尽可能增加文件系统的灵活性，满足业务对存储容量、访问速度、文件安全性的需求。



服务器集群结构

服务器集群包含一个主节点，负责负载均衡和记录操作状态；集群中也可以添加多个从节点，拓展主节点的功能，或完全承担业务功能，其功能根据应用可以不同，执行的功能由集群内部通讯协议决定。而在版本控制系统中，文件系统作为一个服务运行于分布式系统中，所有节点执行相同的功能。文件系统服务采用异步架构完成文件操作请求。

一个文件系统应具备组织文件和文件夹的能力，如创建、删除文件或文件夹，遍历目录，获取文件或文件夹属性，读写文件等功能，版本控制系统中的文件操作接口对应于以上这几方面。

分布式文件系统并不是版本控制系统的核心内容，因此重要程度较低，视时间安排尽可能实现功能。版本控制系统中留出文件系统的接口，如果分布式文件系统实现对应的功能，就将其添加到版本控制系统中；而未实现的功能则使用Web服务器本身的文件操作功能。

3 系统详细设计

3.1 数据库定义及初始化

创建用户表：create table user\_table( user\_id int primary key,username nvarchar(64) not null,user\_account nvarchar(64) not null,user\_password nvarchar(64) not null,register\_time nvarchar(32) default '' not null,user\_description nvarchar(256) default '' not null,user\_type int default 0 not null );

创建代码仓库表：Create table warehouse\_table(warehouse\_id int primary key,user\_id int not null,organization\_id int default 0 not null,warehouse\_type int default 0 not null,create\_time varchar(32) default '' not null, warehouse\_description varchar(512) default '' not null, master\_version\_id int default 0 not null, warehouse\_name varchar(64) not null default '');

创建分支表：create table branch\_table(branch\_id int primary key not null, warehouse\_id int not null default 0, user\_id int not null default 0,start\_id int not null default 0, end\_id int not null default 0,timestamp varchar(32) not null default '', branch\_name varchar(64) not null default '', description varchar(1024) not null default '');

创建版本表：create table version\_table(version\_id int primary key, warehouse\_id int not null default 0, user\_id int not null default 0, prev\_id int not null default 0, next\_id int not null default 0,timestamp varchar(32) not null default '',version\_name varchar(64) not null default '', description varchar(512) not null default '', branch\_id int not null default 0);

创建评论表：create table comment\_table(comment\_id int primary key,user\_id int not null default 0,target\_user\_id int not null default 0,warehouse\_id int not null default 0,content nvarchar(1024) not null default'');

初始化各表：

insert into branch\_table values(0,0,0,0,0,'time','master','description');

insert into version\_table values(0,0,0,0,0,'0','0','0',0);

insert into warehouse\_table values(0,0,0,0,'0','0',0,'0');

insert into user\_table values(0,’nemo’,’nemo’,’password’,’0’,’null’,0);

insert into comment\_table(0,0,0,0,’null’);

3.2 数据库实体



用户类

User类用于记录用户信息。用户名（user\_name）、用户帐户（user\_account）和用户密码（user\_password）的长度限制为64字节，注册时间长度限制为32字节，用户描述（user\_description）长度限制为256字节。用户名中不可以出现下划线以外的符号，用户帐户只能由数字、字母和下划线构成，密码和用户描述在长度限制内，可以为任意内容。可以有多个用户使用相同的用户名，但用户帐户是全局唯一的。



代码仓库类

代码仓库类用于存储用户创建的代码仓库的信息，在这里不保存文件结构和内容。关于数据的长度限制，创建时间（create\_time）为32字节，代码仓库描述（warehouse\_description）为512字节，代码仓库名称（warehouse\_name）为64字节。代码仓库名称只能由数字、字母和下划线构成，代码仓库描述对内容无要求。Master\_version\_id记录的是主分支最新版本ID。



版本类

版本类用于记录版本控制过程中产生的版本，版本的文件结构和数据由文件系统保存，这里只保存版本的描述性信息。关于字符串的长度限制，时间戳（timestamp）为32字节，版本名称（verson\_name）为64字节，版本描述（description）为512字节。版本名称只能由数字、字母和下划线构成，版本描述对内容无要求。版本表的每一条记录，都是某个版本图的一部分，其结构类似于双向链表，prev\_id和next\_id分别记录着前一版本和后一版本的ID，如果不存在则为0。分支的起点和重点在分支表中记录。分支ID（branch\_id）记录当前版本属于哪个分支。



分支类

分支类用于记录版本控制过程中产生的分支。start\_id和end\_id分别记录分支起点版本和终点版本ID。每个代码仓库至少有一个名为“master”的主分支，该分支有服务器在创建代码仓库时自动创建，该分支无法被删除，用户无法创建名为“master”的分支。每个分支中至少有一个版本。字符串的长度限制：时间戳（timestamp）为32字节，分支名称（branch\_name）为64字节，分支描述（description）为1024字节。分支名称只能由数字、字母和下划线构成，分支版本描述对内容无要求。



评论类

评论类用于记录用户提交的评论。评论内容（content）的长度限制为1024字节。评论提交者（user\_id）向评论接受者（target\_user\_id）提交评论，评论的代码仓库由warehouse\_id记录。

3.3 数据访问层



UserDAL类

UserDAL用于操作用户类在数据库中的记录。包含对数据库的增删改查操作。



WarehouseDAL类图

WarehouseDAL用于操作代码类在数据库中的记录。



VersionDAL类图

VersionDAL用于操作版本类在数据库中的记录。



BranchDAL类图

BranchDAL用于操作分支类在数据库中的记录。



CommentDAL类图

CommentDAL用于操作评论类在数据库中的记录。

3.4 用户页面设计

3.5 业务逻辑层

3.6 文件系统设计

服务器集群内部通讯协议与HTTP类似，协议头部采用键值对组，每一组键值对后有\r\n两个字符。数据段在一个空行之后，如果在通讯中携带数据，则需在头部指定Content-Length字段，即系带数据的长度，服务器协议组件会解析该字段数据并和数据长度对比，如果长度不符，则丢弃数据。协议格式如下：

key1:value1\r\n

key2:value2\r\n

[ContentLength:2333\r\n]

\r\n

[Content]

同步IO：假如对一个文件（socket也同理）进行处理，那么一般的流程就是：

1 fstream file;

2 file.open();

3 file.read();

4 //do something

5 file.close();

通常情况下，当这个线程运行到read（）时会被阻塞，直到文件读取完成。

在异步情况下：还是上面的代码，我在read（）时通过操作系统或库提供的异步机制，告诉操作系统我想读一个文件，数据读完后执行某个功能；而当前线程在交代完操作系统该做什么工作之后，还可以做些别的事情（线程不必等待文件IO完成）。

线程池：为了避免IO阻塞线程导致程序无响应，完全可以为每一个文件操作创建一个线程，这样就可以同时处理多个文件了。但是创建线程，切换线程，销毁线程也是一笔资源开销，如果想重复使用已有的线程，就可以使用线程池。作为线程池，至少要提供创建线程和提交任务的功能，复杂一点可以智能控制线程池里的线程数量,还应该具有基本的负载均衡功能。这个文件系统中就会使用线程池。

这个模块只使用stl 和boost 两个库。stl主要涉及容器和fstream。boost涉及到智能指针shared\_ptr，线程同步shared\_mutex,lock\_guard, boost::filesystem中的path和一些文件操作，线程操作创建退出等。

关于智能指针。自古以来内存管理都是C/C++中的重头戏，智能指针的功能就是分配出来的内存由库管理，如果某个智能指针指向的内存，通过其他的智能指针也能访问到（即有多个引用），那么该智能指针即时被销毁，指向的内存也不会销毁；只有这块内存没有引用，才会被库释放。

boost::filesystem库提供了一些跨平台文件操作的API，如文件夹遍历，查看属性，删除文件等。path类可以记录跨平台的路径。

数据结构及定义：

AsyncStatus是异步IO中需要实现的功能。像读，写，放弃异步操作，错误处理等。

ErrorCode会出现在回调函数中，表示之前异步读写的结果，如正在处理，出错，EOF等。

FS\_Handle\_ST：这个结构体对应一个文件路径。在系统中每个handle都是唯一的，系统有一个map，通过handle可以找到它对应的路径。

FS\_AsyncHandle\_ST:标识某个handle需要执行的任务，每个AsyncHandle都需要指定status即任务。一个handle可以有多个异步任务，但多个任务在系统中按照队列顺序执行。

回调函数FileSystemIOCallback:异步操作完成之后要做什么。使用的时候把功能在派生类里实现，重载虚函数run就可以了。