**DocSys文件管理系统设计**

# 1、系统简介

DocSys是一个网页版本的文件管理系统，方便用户进行文件存储和管理，用户操作需要尽可能的接近本地目录的操作习惯（例如，拖放、右键新建、删除、修改等）。

为了方便用户对文件进行分类管理和不同的存储系统，系统支持多仓库，不同仓库有自己独立的目录结构和权限管理，不同仓库可以采用不同的存储方式。

# 2、概要设计

## 2.1系统架构

DocSys采用BS架构，使用数据库记录文件节点信息，使用户本地目录存储文件，使用本地或远程SVN/GIT仓库存储文件历史颁布，系统拓扑结构如下图所示：

图1：系统拓扑图

### 2.1.1传统文件系统实现

传统文件系统设计需要以下几个关键信息：

1. 块位图信息：用于表示数据块的使用情况
2. 文件节点位图信息：用于表示文件节点的使用情况
3. 文件节点信息表：与文件节点一一对应，用于描述文件/目录的信息
4. 数据块：数据存储区

传统文件系统实现有两个难点：

1. 在于文件操作（增、删、改）过程中如何维护块位图信息和文件节点信息的正确性，这个是保证文件系统稳定性的关键，因此保证文件节点的写操作是原子操作是核心
2. 文件的查找速度，文件系统总是通过文件名找到文件节点索引，才能实现对文件内容的访问，因此一个快速高效的搜索算法是非常必要。

### 2.1.2 Web文件系统实现

Web文件系统的文件存储部分是建立在操作系统的文件系统之上的，因此不需要考虑维护块数据位图的维护，可以假设每个文件节点只使用一个数据块，而且这个数据块可以自由伸缩（因为这个数据块实际就是一个真实的文件），因此我们设计需要考虑的就是如何维护文件节点位图和文件节点信息表。

DocSys采用数据库来保存文件节点信息，相当于将节点为图和节点信息表合在了一起，每条存在一条记录就表示一个文件节点，不需要考虑如何去维护文件节点位图，但需要保证文件节点的写入操作是原子操作（即同一时间只能有一个用户在进行写入操作）。

以上提到的原子操作，只是为了保证文件系统底层数据的正常，但对于网页版文件系统而言，做到这一点还是不够的，比如用户在编辑文件的过程中（没对文件进行写操作），发现文件内容突然被其他给修改了，因此作为网页版文件系统，不仅要保证底层文件的原子操作，还需要保证在用户层的原子操作（即用户一旦启动了编辑模式，该文件的写入操作就杯占用，直到用户主动释放或者系统超时自动释放）。

### 2.1.3关键技术

1. zTree控件：

目录结构展示和文件操作

1. MD文件编辑控件

虚文件编辑功能

1. 右键菜单控件

文件右键操作

1. 文件流上传

文件大小可以脱离Tomcat的限制

1. 文件流下载

实现文件下载的权限控制

1. 文件拖拽控件

实现文件拖拽上传

1. 页面Div区域大小拖放控制

目录展示区、当前目录展示区、虚文件预览区、实文件预览区、在线编辑区、在线编辑预览区

1. SVN版本管理
2. GIT版本管理
3. Word在线预览与编辑
4. Pdf在线预览
5. Txt文件在线预览

### 2.2文件并发访问控制

由于Web文件系统允许不同用户同时访问，因此必须充分考虑各种并发访问的情况，以保证文件节点信息、文件实体和文件版本库内容的一致性。

为了保证在对文件编辑操作时（Add、Delete、Modify、Rename、Move、Copy、UpdateDocContent）出现不可控或混乱的情况，必须保证在操作过程中处于独占状态（例如对目录更名操作完成前时，其他用户不能操作该目录的文件或目录，否则会破坏文件系统信息的一致性），因此采取以下方案来实现文件的互斥访问：

（1）每个Doc都有一个Lock字段，0：表示Doc未锁定、1表示锁定自身（即使是目录也只锁定自身，主要用于Doc的虚文件编辑）、2表示锁定（如果是目录会锁定目录下的所有Doc）。

（2）为了避免查询Doc的Lock状态以及设置Lock状态时，Lock状态被其他线程修改，因此在查询和设置Lock状态时必须上线程锁，从而保证获取到的Lock状态是准确的。

（3）以下是各种编辑操作的Lock的获取和设置方法

|  |  |  |
| --- | --- | --- |
| 编辑操作 |  |  |
| Add  File  Dir | 1、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Add  2、set LockState = 2（Dir不允许增加Doc）  3、Add 失败需要删除新增的Doc |  |
| Delete  File  Dir | 1、Doc是目录，且目录非空不能Delete  2、Doc's LockStat != 0（节点本身处于编辑状态）不能Delete  3、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Delete  4、Delete 失败需要unlock Doc |  |
| Modify  File | 1、Doc's LockStat != 0（节点本身处于编辑状态）不能Modify  2、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Modify  3、set LockState =1  4、Modify 失败需要unlock Doc |  |
| UpdateDocContent  File  Dir | 1、Doc's LockStat != 0（节点本身处于编辑状态）不能UpdateDocContent  2、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能UpdateDocContent  3、set LockState =1  4、UpdateDocContent 失败需要unlock Doc |  |
| Rename | 1、Doc's LockStat != 0（节点本身处于编辑状态）不能Rename  2、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Rename  3、set LockState =2 （锁定整个目录）  4、Rename失败需要unlock Doc |  |
| Move | 1、Doc's LockStat != 0（节点本身处于编辑状态）不能Move  2、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Move  3、set LockState =2 （锁定整个目录）  4、Move失败需要unlock Doc |  |
| Copy | 1、Doc's LockStat != 0（节点本身处于编辑状态）不能Copy  2、Parent Doc's LockState == 2（父节点处于锁定状态，例如正在进行Rename、Move、Copy操作）时，不能Copy  3、set LockState =2 （锁定整个目录），新增的文件也需要Lock  4、Copy失败需要unlock Doc | Copy操作时锁定，是为了避免在copy过程中，源文件被修改，从而导致copy操作出现不可控情况 |

以下方案证明不合适==\_\_\_\_\_\_\_==!!

系统利用了文件系统本身的并发访问控制来实现底层接口的并发访问控制（换句话说，利用文件系统的文件增、删、写接口的本身机制来实现文件实体的并发访问），另外利用文件节点信息实现更高层的并发访问控制。

注意：

（1）这里有一个假设，我们认为版本仓库的增删改总是会成功的，即使不成功我们也可以通过重新同步的方式来将本地WorkingCopy和仓库进行同步。

（2）系统只考虑文件的增、删、改操作，其中的改是指发送到后台的文件覆盖操作（文件编辑属于本地操作过程），改名、移动、复制属于增删改的组合操作。

（3）为了避免Commit的过程中的文件被修改或删除，需要Lock正在Commit的文件

|  |  |  |  |
| --- | --- | --- | --- |
|  | Add | Delete | Update |
| Add | (1) 同时执行到本地文件新建，文件新建接口报错，退出  (2) A用户新建文件成功，B用户会检测到文件已存在，退出  (3) A用户更新数据库失败，删除版本仓库文件，删除新增的文件成功，B用户新增可以成功  (5) A用户更新数据库失败，删除版本仓库文件，删除新增的文件失败，B用户无法新增文件，该文件将无法再新建 | (1) Delete操作的前提是文件节点信息已存在，因此对于同一个文件的删除操作，总是发生在Add操作成功之后。  (2) A用户在删除目录，B用户在该目录下Add文件  A用户删除目录成功，B用户新增文件失败，因为上层目录不存在  B用户正在新增文件，A用户删除目录失败，因为存在无法删除的目录  (3) A用户在上传目录，B用户在删除目录，系统通过前台控制，B用户只会递归删除已经存在的文件和目录，会导致情况(2) | Update操作的前提也是文件节点已经存在，所以一定是发生在Add操作成功之后，因此不会发生冲突 |
| Delete |  | 1. A用户和B用户同时删除文件，其中一个会失败，退出 2. A用户删除成功，B用户检测到文件不存在，退出 3. A用户和B用户同时删除目录，由于前台递归次序不一样，过程中会出现情况(1)和(2) 4. A用户删除顶层目录，B用户删除底层目录，当A用户递归删除到底层目录时，会出现情况(3) | (1) A用户正在Update，B用户Delete会失败  (2) A用户Update文件完成，正在上传历史版本，B用户Delete如果会成功，则数据库记录会被删除，A用户更新数据库失败，A用户可以重新执行更新（后台需要判断文件节点是否存在执行新增）  如果上传历史版本过程中，其他用户无法删除上传中的文件实体，则不会产生冲突  (3) B用户正在删除，B用户Update失败  (4) B用户删除成功，B用户Update失败，提示文件已被删除，需要重新上传或保存 |
| Update |  |  | 1. A用户正在Update,B用户Update会失败，需要重新保存或上传 2. A用户Update文件完成，正在Commit，B用户Update文件成功，则会导致Commit的是B用户的结果，如果此时不能Update文件则不会问题 |

## 2.2文件下载

支持多文件下载和单目录下载；文件下载使用数据流方式下载，避免用户直接获取到下载链接，方便进行权限管理。

### 2.2.1 普通文件下载

普通文件下载实现分目录和文件下载：

1. 文件下载

文件下载比较简单，实际上就是将文件以数据流的方式传回前台。

1. 目录下载

目录下载需要先将目录打包成zip文件放到/tmp/UserID/目录下，再以数据流的方式传回前台。

后期也可以考虑以下载文件历史版本的方式实现，也就是说下载文件的最新版本即可。

### 2.3.2 虚拟文件下载

单个虚拟文件下载，跟普通文件系统的目录下载功能相同。但如果需要考虑以目录关系下载的方式的话，则需要将多个虚拟文件目录打包后下载。

另外考虑到以目录结构的方式展示给用户的话，还需要将md文件的内容以html文件的方式提供给用户，这个功能对于利用虚拟文件系统来记录一些接口定义是会非常方便的。

## 2.3文件编辑

文件系统包括虚拟文件和普通文件，虚拟文件是基于Markdown文件实现的，需要保证文件内容指定的外联资源的正常显示，因此实际上并不是一个单一的文件，它在后台存储上是一个以DocID标示的目录，其中包括markdown文件以及关联的资源文件/DocID/DocName.md /DocID/res/xxx.png…，其中md文件内容还有一份是放在DataBase中的。

普通文件就是一个单一的文件：例如txt、word、markdown等文件，普通文件是不保证像markdown文件这种需要链接外部资源的正常显示的。

### 2.3.1 虚拟文件编辑

当用户编辑虚文件时，需要将虚拟文件目录复制到tmp/UserID目录下，并将markdown文件内容回传给前台，前台根据当前是编辑状态和用户ID，来获取后台相应的图片文件用于展示。

编辑过程中，如果加入了图片，则图片会被上传到tmp/UserID/DocID/res目录，用于编辑过程中的预览。

编辑过程中，前台会定时将修改内容保存到md文件。

用户点击保存后，后台需要将tmp/UserID/DocID目录替换/vdata/DocID目录。

用户点击退出编辑时，如果文件有修改，则提示用户是否先保存修改，如果否的话，则直接进入预览状态，否则后台保存后，再进入预览状态。

预览实际上就是展示/vdata/DocID的md文件。

#### 2.3.2 普通文件编辑

当用户编辑普通文件时，需要将文件复制到/tmp/UserID目录，并根据不同的文件类型将文件内容解析成前台可以展示的数据，前台需要根据不同文件类型选择不同的编辑容器。

用户插入的图片会被保存在/tmp/UserID/res目录下，如果Word之类的文档，则保存时图片会以数据的格式进入文件，但如果类似与md文件本身不保存res文件的话，那么这些图片只是用于临时的展示用途。

编辑过程中，前台会定时将修改内容保存到文件。

用户点击保存后，后台需要将tmp/UserID/目录下的文件替换/rdata/目录下的文件。同时将/tmp/UserID/res目录下的文件复制到/web/images/fordoc

用户点击退出编辑时，如果文件有修改，则提示用户是否先保存修改，如果否的话，则直接进入预览状态，否则后台保存后，再进入预览状态。

预览实际上就是展示/rdata/文件，如果文件使用了图片，则尝试在/web/images/fordoc目录下去找。

### 2.3.3编辑撤销功能

对于本地文件系统而言，由于操作系统和编辑软件的支持，通常不会发生编辑过程中，文件内容彻底丢失的问题，但对于网页来说，彻底丢失是很正常的事情，因此如何保证在网页编辑过程可以跟在本地操作一样安全是一个必须考虑的设计。

用户启动编辑时，后台将原始文件复制到临时目录下，用户点击保存时才会用临时文件替换原始文件，用户可以执行撤销（CTRL + Z）时，系统会将指令传递给后台，由后台来对临时文件的内容编辑进行撤销操作（这个能否实现未验证）；另外一个方案就是保留最近50次操作的文件备份，但这样对于后台的存储文件有着非常高的要求。

仓库目录下共三个目录: data、vdata和tmp目录，其中data用于存放实际的文件，vdata用于存放虚文件（全部是markdown文件： ID+文件名），tmp目录是用于实行文件编辑过程中的撤销功能存在，用于保存用户编辑过程中的内容（用户编辑过程中没隔一段时间保存一个备份，从而实现在线编辑的撤销功能，撤销功能只是一个提高用户体验的一个东西，系统不能保证撤销总是会成功）。

## 2.4文件版本管理

文件版本管理是一个非常重要的功能，用户主动提交（或者叫保存，可以参考VSS）才会产生历史版本，历史版本的保存使用SVN仓库或GIT仓库进行管理。

文件版本管理相关有以下几个操作节点：

### 2.4.1新建仓库时的版本仓库初始化

仓库存储目录创建成功时，如果该仓库指定了版本控制，则根据设置的SVN或GIT路径进行版本仓库初始化（SVN或GIT路径未设置，需要先创建一个本地的SVN仓库或GIT 仓库，作为版本仓库使用），版本仓库初始化具体操作如下：

（1）远程目录结构修改

如果版本仓库目录结构与本地目录结构不一致的时候，需要先删除版本仓库上存在而本地仓库不存在的文件或目录。

（2）CheckOut

将版本仓库Checkout到本地目录，形成working copy，注意CheckOut时不得对本地文件进行修改。

（3）提交Working Copy的Changes

如果本地目录的文件与版本仓库不一致，则需要将这些Changes全部提交到版本仓库中去，完成同步。

过程中发生错误，新建仓库都会失败，避免出现初始状态不正确的仓库。

### 2.4.2仓库参数修改时的版本仓库变更

仓库参数如果修改了仓库存储地址和版本管理仓库地址，需要进行处理。

1. 仓库存储地址改变时，需要将当前仓库目录整个移动到新的位置（包括了仓库信息），如果版本管理仓库地址未修改则可以考虑不重新进行版本仓库初始化，（理论上是可以的，当前安全起见还是执行一次）。

如果发生错误，需要删除刚复制过来的目录，以还原现场。

1. 版本管理仓库地址改变时，需要重新进行仓库初始化。

如果发生错误的话，有可能出现不可逆的问题，安全起见就是将WorkingCopy先保护起来，因此在真正完成仓库初始化之前，需要备份本地仓库目录，如果失败的话需要还原回来（我们需要假设总是可以还原的，否则这将是一个永远无法结束的循环）。

### 2.4.3 新增文件

本地文件已存在 🡪 失败

版本仓库文件已存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（删除新增的本地文件）

### 2.4.4 删除文件

本地文件不存在 🡪 失败

版本仓库文件不存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（恢复被删除的本地文件）

### 2.4.5 编辑文件

本地文件不存在 🡪 失败

本地文件替换失败 🡪 失败

版本仓库文件不存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（恢复被修改的本地文件）

### 2.4.6 上传文件

上传文件如果是新增操作，流程同新增文件。

上传文件如果是覆盖操作，流程同修改文件。

### 2.4.7 重命名文件

源本地文件不存在 🡪 失败

目标本地文件已存在🡪 失败

版本仓库源文件不存在🡪失败

版本仓库目标文件已存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（将目标本地文件重命名回源本地文件）

### 2.4.8 移动文件

源本地文件不存在 🡪 失败

目标本地文件已存在🡪 失败

版本仓库源文件不存在🡪失败

版本仓库目标文件已存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（将目标本地文件复制回源本地文件）

### 2.4.9 复制文件

源本地文件不存在 🡪 失败

目标本地文件已存在🡪 失败

版本仓库源文件不存在🡪失败

版本仓库目标文件已存在🡪失败

提交版本成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次成功🡪成功

提交版本失败🡪更新WorkingCopy后再提交一次失败🡪失败

失败处理：还原现场（将目标本地文件删除）

## 2.5权限管理

### 2.5.1 权限规则

系统需要能够对仓库以及文件设置每个用户的访问权限，访问权限包括管理员权限、读、写、增加、删除权限。

系统用户数据库的ReposAuth表和DocAuth表来记录权限设置，为了减少权限设置的数量，系统采用权限继承的方式来实现用户文件权限控制（也就是说如果数据库中没有某个用户对某个用户的权限设置，那么这个用户的对这个目录的访问权限将由其父目录的权限决定）。

注意：

系统使用UserId=0来表示任意用户，任意用户是一个虚拟的用户，在数据库中不存在这样的用户实体。

#### 2.5.1.1仓库权限

仓库权限包括：超级权限、管理员权限和访问权限（读、写、增加、删除）

1. 超级权限
2. 超级权限：删除仓库、修改仓库基本设置；
3. 超级管理员和仓库拥有人拥有超级权限；
4. 其他用户没有超级权限。
5. 管理员权限
6. 管理员权限：可以设置添加仓库访问用户、设置用户的仓库和目录访问权限、仓库密码设置；
7. 超级管理员和仓库拥有人拥有仓库的管理员权限，不受权限表控制
8. 其他用户根据权限表确定是否拥有管理员权限；
9. 访问权限
10. 访问权限：读、写、增加、删除权限。
11. 所有用户的访问权限根据权限表的设置决定。
12. 权限申请

用户可以申请自己看得到得仓库的访问权限，仓库的管理员会受到申请通知；

因此能够申请权限的前提是仓库设置了任意用户的访问权限。

#### 2.5.1.2文件权限

文件权限包括：管理员权限和访问权限（读、写、增加、删除）

1. 管理员权限
2. 管理员权限：可以设置目录访问密码；
3. 超级管理员和仓库拥有人拥有目录的管理员权限，不受权限表控制
4. 其他用户根据权限表确定是否拥有管理员权限；
5. 访问权限
6. 访问权限：读、写、增加、删除权限。
7. 所有用户的访问权限根据权限表的设置决定。
8. 权限申请

用户可以申请自己看得到得仓库的访问权限，仓库的管理员会受到申请通知；

因此能够申请权限的前提是仓库设置了任意用户的访问权限。

### 2.5.2 权限设置

#### 2.5.2.1 添加仓库访问用户

1. **工作原理**

添加仓库访问用户的原理实际上就是给指定用户增加一个仓库权限记录（默认权限为空），因此添加访问用户需要两个信息，用户ID和仓库ID。

1. **实现方法**

查询出所有用户列表或可见用户列表（需要带权限信息，用于区分哪些用户已经是仓库访问用户）。

超级管理员可以设置所有用户的仓库权限，因此只要查询出所有用户列表即可，权限信息通过LEFT JION repos\_auth on repos\_auth.user\_id = user.id来获取。

其他用户可以设置可见用户的仓库权限，目前可见用户是指该用户的好友，因此查询用户时需要结合好友表进行筛选。

#### 2.5.2.2仓库与文件权限设置

1. **工作原理**

（1）仓库权限设置的原理是删除或修改仓库权限记录，因此需要仓库权限记录ID信息。

（2）文件权限设置的原理是增加、修改或删除文件权限记录，增加权限记录时需要用户ID和文件ID信息，删除时需要文件权限记录ID信息。

（3）仓库权限列表

仓库权限列表实际包括了仓库的权限和文件的权限列表（带用户信息和文件信息，用户展示用户名和路径）。

选中仓库目录或没有选中目录时，展示仓库权限列表

（4）文件权限列表

文件权限列表是指仓库可访问用户的权限设置列表，包括继承过来的权限。

选中文件时，展示文件权限列表

1. **实现方法**

**（1）仓库权限列表展示**

查询出所有repos\_auth.repos\_id = reposId仓库权限设置，用户信息通过LEFT JION user on user.id=repos\_auth.user\_id来获取。

查询出所有doc\_auth.repos\_id=reposId的文件权限设置，用户信息通过LEFT JION user on user.id=doc\_auth.user\_id来获取，文件信息通过LEFT JION doc on doc.id=doc\_auth.doc\_id来获取。

合并两个查询结果返回给前台进行展示，通过区分docAuthId是否大于0来区分仓库权限还是文件权限，从来在前台来进行不同的操作（前台根据docAuthId是否大于0来确定删除仓库权限还是文件权限，权限的设置则是根据docId是否大于0来区分时仓库还是文件）。

**（2）文件权限设置**

查询出所有repos\_auth.repos\_id = reposId仓库权限设置，用户信息通过LEFT JION user on user.id=repos\_auth.user\_id来获取。

遍历仓库权限列表，根据userId获取该用户的DocAuth，如果DocAuth不存在则获取其继承过来的DocAuth，如果是继承过来的权限docAuthId不要设置，前台需要通过docAuthId是否大于0来判断是否是继承过来的权限（继承过来的权限无法删除，直接权限可以删除）。

## 2.6管理后台

1. 系统管理（超级管理员）

用户自主注册功能关闭与打开

用户自主新建仓库功能的打开与关闭

1. 角色管理（超级管理员）

🡪以列表方式展示所有用户；

🡪角色增加、删除、修改

1. 用户管理（超级管理员）

🡪以列表方式展示所有用户；

🡪用户增加、删除、修改

1. 仓库管理（管理员、超级管理员）

🡪以列表方式展示所有仓库；

🡪仓库增加、删除、修改功能；

🡪仓库管理功能（点击仓库管理进入仓库管理页面）；

## 2.7用户

系统的用户类型有三种0、1、2，分别是普通用户、管理员、超级管理员：

1. 超级管理员

拥有最大权限，系统的所有功能访问都不受限制；

为了保证系统的安全，超级管理员目前只能设置一个，用户管理里不能添加超级管理员。

1. 管理员

管理员可以访问管理后台，但可访问的管理后台的模块受角色的权限表控制，因此避免管理员通过修改角色定义和用户角色来修改自己的权限，角色管理和用户管理只有超级管理员才拥有权限。

1. 普通用户

普通用户不能访问管理后台，其他功能跟管理员相同。

### 2.7.1 用户登录

用户登录包括系统登录和管理后台登录，后台需要使用两个不同接口，但用户基本信息检查使用相同的接口，管理后台登录除了检查基本信息外，还需要检查用户角色是否是管理员，只有是管理员才能登录管理后台。

登录成功后，用户信息需要放入session中，用于后续用户信息展示和系统访问控制。

用户登录时，勾选了记住用户，则登录信息会保存进cookie里，方便用户下次自动登录，有效期为1周。

### 2.7.2 用户注册

用户可以通过邮箱和手机进行自主注册，完成注册需要输入验证码。

# 3、详细设计

## 3.1 数据库设计

为了数据库能够兼容Oracle数据库，因此所有字段全部设计成大写字母。

### 3.1.1仓库表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 仓库ID，每个仓库的唯一标示，不可重复 |  |
| NAME | 仓库名字 |  |
| TYPE | 仓库存储类型：  0: 虚拟文件系统  1: 实文件系统 |  |
| PATH | 远程仓库根目录 |  |
| VER\_CTRL | 版本控制类型  0：无  1：SVN  2：GIT |  |
| SVN\_PATH | SVN/GIT路径 |  |
| SVN\_USER | SVN/GIT访问帐户 |  |
| SVN\_PWD | SVN/GIT访问账户密码 |  |
| INFO | 仓库简介 |  |
| MENU | 仓库目录结构 |  |
| PWD | 仓库访问密码 |  |
| OWNER | 仓库所有人（默认是创建人） |  |
| CREATE\_TIME | 仓库创建时间 |  |

### 3.1.2 文件节点表

这里的文件包括文件和目录

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 文件ID,每个文件的唯一ID |  |
| NAME | 文件名称 |  |
| TYPE | 文件类型   1. 目录 2. 文件 |  |
| CONTENT | 虚文件内容 |  |
| PATH | 目录路径：相对于仓库根目录的路径 |  |
| PID | 父目录ID |  |
| VID | 所属仓库ID |  |
| PWD | 文件节点密码 |  |

常见错误:

1、数据库新增了三个字段，导致addDoc失败，提示没有Default值，将空打勾就可以了

### 3.1.3用户表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 用户ID |  |
| NAME | 用户名（字母+数字） |  |
| PWD | 用户密码 |  |
| TYPE | 用户类型：0:普通用户; 1:管理员；2超级管理员 |  |
| ROLE | 角色ID：不同角色可访问的管理后台的模块不同，可自定义 |  |
| REAL\_NAME | 真实姓名 |  |
| NICK\_NAME | 昵称 |  |
| INTRO | 用户简介 |  |
| IMG | 用户头像信息 |  |
| TEL | 手机 |  |
| TEL\_VALID | 手机是否已验证 |  |
| EMAIL | 邮箱 |  |
| EMAIL\_VALID | 邮箱是否已验证 |  |
| LAST\_LOGIN\_TIME | 最近登陆时间 |  |
| LAST\_LGOIN\_IP | 最近登陆IP |  |
| LAST\_LOGIN\_CITY | 最近登陆城市 |  |
| CREATE\_TYPE | 用户创建类型：  0：管理员创建  1：用户注册 |  |
| CREATE\_TIME | 用户创建时间 |  |

### 3.1.4角色表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | ID |  |
| ROLE\_ID | 角色ID |  |
| NAME | 角色名 |  |
| AUTH | 角色的访问权限，用json格式数据来保存，方便扩展 |  |

### 3.1.5 仓库权限表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 每个权限的唯一ID |  |
| USER\_ID | 用户ID |  |
| REPOS\_ID | 仓库ID |  |
| IS\_ADMIN | 是否是仓库管理员：仓库管理员可以管理其他人的权限 |  |
| ACCESS | 访问权限：  0：不可访问  1：可访问 |  |
| EDIT\_EN | 编辑权限：  0：不可编辑  1：可编辑 |  |
| ADD\_EN | 文件新增权限：  0：无  1：有 |  |
| DELETE\_EN | 文件删除权限  0：无  1：有 |  |

### 3.1.6 文件节点权限表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 每个权限的唯一ID |  |
| USER\_ID | 用户ID |  |
| DOC\_ID | 文件节点ID |  |
| REPOS\_ID | 仓库ID |  |
| IS\_ADMIN | 是否是文件节点管理员，该属性继承父节点的属性，如果父节点在根目录下，则继承仓库的属性 |  |
| ACCESS | 访问权限：  0：不可见  1：可见 |  |
| EDIT\_EN | 编辑权限：  0：不可编辑  1：可编辑 |  |
| ADD\_EN | 文件新增权限：  0：无  1：有 |  |
| DELETE\_EN | 文件删除权限  0：无  1：有 |  |

### 3.1.7 系统配置表

|  |  |  |
| --- | --- | --- |
| 字段 | 描述 | 备注 |
| ID | 配置表ID |  |
| REG\_ENABLE | 用户自主注册功能  0：关闭  1：打开 |  |
| PRIVATE\_REPOS\_ENABLE | 普通用户私有仓库创建功能  0：关闭  1：打开 |  |

## 3.2 系统详细设计

### 3.2.1 系统主页

系统主页需要提供以下功能：

1. 用户注册、用户登录、忘记密码功能。
2. 用户头像与下拉功能选项。

#### 3.2.1.1用户注册

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showRegisterPanel() | （1）用户在主页点击注册，弹出注册窗口  （2）用户填写邮箱或手机，点击获取验证码  （3）后台检查邮箱或手机是否已注册，发送验证码到邮箱或手机  （5）验证码输入后，点击注册，后台新增一个新用户  （6）系统自动弹出登陆页面，登陆用户为新注册的用户 | Index.html  showRegisterPanel()  register.html  register() |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /register.do | 用户注册 | UserController.java  register() |

#### 3.2.1.2 用户登录

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showLoginPanel () | （1）系统主页用户点击登陆弹出登陆窗口  （2）有登陆历史则使用上次的登陆用户，否则用户填写邮箱或手机  （3）输入密码  （4）点击登陆，后台验证成功后，进入系统仓库列表页 | Index.html  showLoginPanel ()  login.html  login() |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /login.do | 用户登录  函数：  参数：userName 登录用户名，邮箱或手机  pwd 登录密码  rememberMe 是否记住登录信息  功能：  根据用户名和密码信息，检查该用户是否存在，如果信息符合则返回登录成功信息（当前登录用户信息），当前登录用户信息保存在session中用于权限控制。 | UserController.java  login() |

#### 3.2.1.3 忘记密码

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| changePwd () | （1）用户登录对话框，点击忘记密码，跳转至密码找回页面  （2）用户填写邮箱或手机，点击获取验证码  （3）后台检查邮箱或手机是否已注册，发送验证码到邮箱或手机  （5）验证码输入后，点击继续，则弹出密码修改页面  （6）密码修改完成后，自动进入用户登陆页面 | login.html  showLoginPanel ()  getPassword.html  changePwd() |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /changePwd.do | 修改用户密码 | UserController.java  changePwd() |

#### 3.2.1.4 用户登录信息与功能菜单

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| pageInit() | （1）用户登陆成功后，系统右侧显示用户头像  （2）点击后有下拉菜单，点击可以进入用户主页的对应页面  （3）下拉菜单包括个人主页、我的好友、权限管理、密码修改、意见反馈  🡪个人主页：进入个人主页的用户信息页面  🡪我的好友：进入我的好友页面  🡪仓库管理：进入仓库管理页面  🡪密码修改：进入个人主页的密码修改页面  🡪意见反馈：弹出意见与反馈页面 |  |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /getLoginUser.do | 获取登录用户信息 | UserController.java  login() |

### 3.2.2 仓库列表页

仓库列表页面需要实现以下功能：

1. 展示用户可访问的所有仓库
2. 提供新建仓库功能
3. 提供进入管理后台链接
4. 仓库右键设置密码功能

#### 3.2.2.1 仓库列表显示

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| **$(function (){})** | 1. 页面加载完成时，首先获取登录用户，获取失败提示先登陆 2. 获取仓库列表，并展示 | pageInit()  onChange() |
| $(function () {  pageInit(); //获取登录用户，并展示用户信息  onChange(); //获取仓库列表并展示仓库列表  } |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /getLoginUser.do | 获取登录用户信息 | UserController.java  getLoginUser () |
| /Repos/getReposList.do | 获取仓库列表 | ReposController.java  getReposList() |

#### 3.2.2.2 仓库新建

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| newRepos() | 用户登录成功后，显示新建仓库按键newReposBtn，点击“新建”弹出新建仓库页面 | Projects.html  pageInit() |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /getLoginUser.do | 获取登录用户信息 | UserController.java  getLoginUser() |
| /Repos/addRepos.do | 新增仓库 | ReposController.java  addRepos() |

#### 3.2.2.3 管理后台链接

#### 3.2.2.4 仓库访问密码设置

右键弹出仓库密码设置对话框，可以对仓库设置访问密码；

设置了访问密码的仓库，点击仓库时弹出密码输入界面，严重正确后进入仓库主页。

### 3.2.3 仓库主页

仓库主页需要提供以下功能：

1. 展示当前用户可见的仓库目录；
2. 右键功能：
   1. 新建文件、文件夹
   2. 上传文件、文件夹
   3. 重命名
   4. 删除
   5. 下载
   6. 密码设置
3. 文件复制与移动

CTRL+鼠标拖动为复制操作；

鼠标直接拖动为移动操作；

1. 拖拽上传功能

🡪支持文件和文件夹拖拽上传

🡪文件拖拽到目录时，需要能够自动展开

🡪拖拽的文件超过显示范围时，需要能够移动

1. 目录区域与预览区域拖动缩放功能
2. 文件和文件描述预览功能

🡪点击文件时显示文件的图标和文件描述；

🡪双击文件或双击文件图标时预览文件内容（支持doc\pdf\图片），符合人们的操作习惯

🡪邮件选择文件图标时可以选择如下操作：

编辑文件、编辑文件描述、修改文件名

1. 文件和文件描述编辑功能

#### 展示用户可见目录树

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
|  | 获取用户的可见仓库文件列表 | project.html  getMenu() |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /getLoginUser.do | 获取登录用户信息 | UserController.java  getLoginUser() |
| /Repos/getReposMenu.do | 获取当前用户的可见文件列表，因为权限涉及继承问题，因此无法实现通过合并查询用户列表和权限列表的方式来实现一次性查询获取，为了避免过多次数的数据库查询操作，目前采用从顶至底的递归查询方式，通过传递顶部文件权限的方式，最大程度的减少了用户权限的查询次数（保证每个目录的权限只会被查询一次）。 | ReposController.java  getReposMenu()  getAuthedDocList()  getReposAuth()  getDocAuth() |

#### 右键功能

##### 3.2.2.1.1新建文件、文件夹

文件的操作权限（增、删、改），由该用户的文件权限而定。

//该接口根据用户是否有仓库的直接权限分开两条支线来获取用户权限

**private** DocAuth getDocUserAuth(Integer userId, Integer docId, Integer reposId) {

// **TODO** Auto-generated method stub

//获取仓库权限

ReposAuth qReposAuth = **new** ReposAuth();

qReposAuth.setReposId(reposId);

qReposAuth.setUserId(userId);

ReposAuth reposAuth = reposService.getReposAuth(qReposAuth);

**if**(reposAuth == **null**) //该用户没有该仓库的直接权限设置，获取是否设置任意用户的访问权限

{

System.*out*.println("用户不是该仓库的直接访问用户，检查该仓库是否开放了任意用户访问");

qReposAuth.setUserId(0); //获取任意用户的仓库访问权限

reposAuth = reposService.getReposAuth(qReposAuth);

**if**(reposAuth == **null**)

{

System.*out*.println("该仓库未开放了任意用户访问");

**return** **null**;

}

//递归任意用户的文件的访问权限

System.*out*.println("该仓库开放了任意用户访问，递归获取任意用户的目录访问权限");

userId = 0; //任意用户

**return** recurGetDocAuthEx(userId,docId,reposId,reposAuth);

}

**else**

{

System.*out*.println("该用户是仓库的直接访问用户，递归获取该用户的目录访问权限");

//递归获取文件的访问权限

**return** recurGetDocAuthEx(userId,docId,reposId,reposAuth);

}

}



##### 3.2.2.1.2上传文件、文件夹

##### 3.2.2.1.3重命名

##### 3.2.2.1.4删除

##### 3.2.2.1.5下载

##### 3.2.2.1.6文件复制与移动

##### 3.2.2.1.7文件拖拽上传

##### 3.2.2.1.8文件列表区与文件预览区拖拽缩放

##### 3.2.2.1.9文件与文件描述预览

##### 3.2.2.1.10文件与文件描述编辑

##### 3.2.2.1.11密码设置

### 3.2.4 仓库管理

仓库管理页面需要提供以下功能：

1. 仓库选择列表显示
2. 仓库目录显示
3. 仓库访问用户添加功能
4. 仓库与文件权限设置功能（权限设置、权限删除）
5. 右键功能

仓库设置：修改仓库的基本信息

访问密码设置：设置仓库和目录的访问密码

删除仓库：

#### 3.2.4.1仓库选择列表

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showReposSelectList() | 页面加载完成后，系统调用showReposSelectList接口获取仓库下拉选择列表 | reposManager.html  showReposSelectList() |
| $(document).ready(function(){  showReposSelectList()  }) |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/getReposList.do | 根据当前登录用户，获取其可管理的仓库列表 | ReposController.java  getReposList() |

#### 3.2.4.2仓库目录显示

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| getMenu() | 更具当前选择的仓库，列出仓库目录（仓库本身作为根目录存在） | reposManager.html  getMenu () |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/getReposManagerMenu.do | 获取仓库目录树（仓库作为根目录，其docID为0） | ReposController.java  getReposManagerMenu() |

#### 3.2.4.3仓库访问用户添加

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showAddReposUserPanel()  showReposAllUserList() | 1. 点击添加访问用户按键弹出添加访问用户页面 2. 列出可见用户列表 3. 添加成功后刷新用户列表和权限列表显示 | reposManager.html  showAddReposUserPanel()  addReposUser.html  showReposAllUserList()  addReposUser(userId) |
|  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/getReposAllUsers.do | （1）获取所有可见用户列表，带repos\_auth信息（LEFT JOIN repos\_auth on repos\_auth.user\_id = user.id）  （2）获取任意用户信息  （3）合并所有用户列表和任意用户，返回给前台 | ReposController.java  getReposAllUserList()  getAnyUserReposAuth() |
| /Repos/addUserReposAuth.do | 增加一个仓库权限记录with UserID 和 ReposID | ReposController.java  addUserReposAuth() |
|  |  |  |

#### 3.2.4.4 仓库与文件权限设置

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showAuthList() | （1）根据选择的文件显示仓库或文件的权限列表  （2）docId>0表示选择的目录，否则表示选择的是仓库 | reposManager.html  showAuthList()  showDocAuthList()  showReposAuthList() |
| **function** showAuthList()  {  **if**(docId > 0)  {  showDocAuthList();  }  **else**  {  showReposAuthList();  }  } |
| showReposAuthList() | 显示仓库以及目录的所有权限设置，通过docAuthId是否大于0来区分是仓库权限和文件权限 |  |
| showDocAuthList() | 显示目录的所有权限设置和继承权限，通过docAuthId是否大于0来区分是设置权限还是继承权限 |  |
| deleteDocAuth() | 根据docAuthId删除DocAuth |  |
| deleteReposAuth() | 根据reposAuthId删除ReposAuth |  |
| saveAuth() | 遍历界面上处于勾选状态的权限记录，然后依次一个一个进行设置 |  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/getReposAuthLis.do | （1）获取所有仓库权限列表和仓库文件权限列表  （2）仓库权限列表，带user信息（LEFT JOIN user on user.id = epos\_auth.user\_id）  （3）文件权限列表带user信息带user信息（LEFT JOIN user on user.id = epos\_auth.user\_id）和文件信息（LEFT JOIN doc on doc.doc\_id = doc.id）  （4）合并仓库权限列表和文件权限列表，返回给前台 | ReposController.java  getReposAuthLis () |
| /Repos/getDocAuthList.do | (1)查询出所有repos\_auth.repos\_id = reposId仓库权限设置，用户信息通过LEFT JION user on user.id=repos\_auth.user\_id来获取。  (2)遍历仓库权限列表，根据userId获取该用户的DocAuth，如果DocAuth不存在则获取其继承过来的DocAuth，如果是继承过来的权限docAuthId不要设置，前台需要通过docAuthId是否大于0来判断是否是继承过来的权限（继承过来的权限无法删除，直接权限可以删除）。 | ReposController.java  getDocAuthList() |
| /Repos/deleteUserReposAuth.do | 根据reposAuthId、userId、reposId删除仓库权限记录 | ReposController.java  deleteUserReposAuth() |
| /Repos/configUserAuth.do | 根据docAuthId、userId、docId  reposId删除文件权限记录 | ReposController.java  deleteUserDocAuth |
| /Repos/configUserAuth.do | 根据docId来确定是设置用户的仓库库权限还是文件权限 | ReposController.java  configUserAuth() |

#### 3.2.4.5 右键功能

##### 3.2.4.5.1仓库设置

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| showReposBasicSettingPanel() | 弹出仓库设置页面 | reposManger.html |
| **saveReposBasicSetting()** | 修改仓库的基本设置 | reposBasicSetting.html |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/updateReposInfo.do | 根据设置的参数修改仓库基本设置，包括仓库名、仓库描述、仓库存储路径、仓库版本仓库； | ReposController.java  updateReposInfo() |

##### 3.2.4.5.2访问密码设置

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
|  | 弹出仓库密码设置页面 | reposManger.html |
|  | 设置仓库访问密码 |  |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
|  | 设置仓库访问密码 | ReposController.java |

##### 3.2.4.5.3删除仓库

1. **前端界面与接口**

|  |  |  |
| --- | --- | --- |
| **前端页面与接口** | **功能描述** | **相关页面与接口** |
| deleteRepos(vid) | 提示用户是否删除仓库  删除仓库 | reposManger.html  deleteRepos() |

1. **后台接口**

|  |  |  |
| --- | --- | --- |
| **后台接口** | **功能描述** | **相关接口** |
| /Repos/deleteRepos.do | 删除仓库记录，会保留仓库所有文件和版本仓库 | ReposController.java  deleteRepos() |

### 3.2.5 管理后台

管理后台部分的代码的网址配置在config.js中设置，需要修改的话请在config.js中进行设置。

#### 3.2.5.1 管理后台登陆页面

|  |
| --- |
| [Logo] DocSys |
| 用户名 [ ]  密 码 [ ]  [ ]记住我 忘记密码  [ 登陆按键] |

#### 3.2.5.2 管理后台主页面

|  |  |
| --- | --- |
| [Logo] DocSys [头像] 名字[角色] | |
| 系统管理  角色管理  系统参数  用户管理  仓库管理 | 数据列表区 [增加]  数据1 [删除] [修改] [设置] |

1、点击左侧的功能按键，右侧显示不同的管理页面；

2、右侧管理页面均以列表方式展示需要管理的内容

3、角色管理

角色管理包括角色的增加、删除、修改

角色信息包括角色ID（由系统自动生成无法修改）、角色名字、角色权限

角色权限用于设置角色能够进入的管理后台的哪些模块

1. 系统参数

系统参数管理包括参数的增加、删除、修改、启用

系统参数包括：是否开始开启自主注册、是否开启私人仓库、是否开启私人仓库的删除功能、私人仓库的个数等参数

可以增加多组系统参数，但只能使用其中一组，默认为ID为1的系统参数

1. 用户管理

用户管理包括用户的增加、删除、修改

用户信息包括用户名、邮箱、手机等信息

1. 仓库管理

仓库管理包括仓库的增加、删除、修改、仓库管理

仓库信息包括：仓库名、仓库路径、仓库描述、仓库所有人、仓库密码