logo

**Visual-Vers——可视化版本控制工具**

模块设计报告

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布  [ ] 正在修改 | 文件标识： | ZJGSU |
| 当前版本： | 1.0 |
| 作 者： | 汤英杰 顾启隽 叶扬 |
| 完成日期： | 2022-5-31 |

浙江工商大学计算机与信息工程学院

目 录

[1．文档介绍 2](#_Toc20181)

[1.1文档目的 2](#_Toc31716)

[1.2文档范围 3](#_Toc19843)

[1.3读者对象 3](#_Toc2869)

[1.4参考文献 3](#_Toc17309)

[1.5术语与缩写解释 3](#_Toc27227)

[2．模块命名规则 4](#_Toc22771)

[3．模块汇总 4](#_Toc11430)

[3.1模块汇总表 4](#_Toc15175)

[3.1.1Visual-Vers子系统模块汇总 4](#_Toc31141)

[3.1.2Visual-Vers子系统模块汇总表 4](#_Toc10623)

[3.2模块关系图 5](#_Toc7172)

[4．用户子系统的模块设计 5](#_Toc13674)

[4.1表示层模块设计 5](#_Toc28876)

[4.1.1开始界面 5](#_Toc22680)

[4.1.2主见面 6](#_Toc330)

[4.1.3差异对比界面 7](#_Toc7454)

[4.1.4弹出菜单 7](#_Toc9562)

[4.1.5绘图界面 7](#_Toc24882)

[4.2数据处理层 8](#_Toc13582)

[4.2.1命令模块 8](#_Toc1820)

[4.2.2操作模块 9](#_Toc3189)

[4.2.3节点模块 1](#_Toc24012)1

[4.2.4分支模块 1](#_Toc2510)3

[4.2.5修改记录模块 1](#_Toc2510)4

[4.2.6提交记录按钮 1](#_Toc2510)4

# 1．文档介绍

## 1.1文档目的

本文档的目的是明确系统的各模块的命名规则，明确各模块的功能，同时对每个模块进行详细设计，包括接口与属性设计、数据结构与算法设计，明确各模块的具体实现策略，使系统开发人员和产品管理人员明确产品功能，可以有针对性的进行系统开发、测试、验收等各方面的工作。

## 1.2文档范围

本文档包括各子系统的各模块命名规则以及各模块的详细设计，包括接口与属性设计、数据结构与算法设计。

## 1.3读者对象

本文档主要面向客户、数据库设计人员、架构设计人员、界面设计人员、编码人员及测试人员。

## 1.4参考文献

陈明：《软件工程》，中央广播电视大学出版社，2002年6月版

《软件工程》第五版——清华大学出版社 张国藩编

《QtCreator快速入门》，霍亚飞，北京航空航天大学出版社

《Git从入门到精通》，高见龙，北京大学出版社

## 1.5术语与缩写解释

|  |  |
| --- | --- |
| **缩写、术语** | **解 释** |
| Git | 一款分布式版本控制工具，目前为大众所使用 |
| VVS | 可视化版本库，Visual-Vers |
| DAG | 有向无环图，Directed Acyclic Graph |
| Diff | 差异，difference |
| Dagre | 一种有向无环图绘制算法 |
| DP | 动态规划，Dynamic Programming |

# 2．模块命名规则

类和接口均采用名词，按照其功能含义,界面类小写开头，按驼峰规则命名；数据层模块大写开头，后面每个单词首字母大写。

# 3．模块汇总

## 3.1模块汇总表

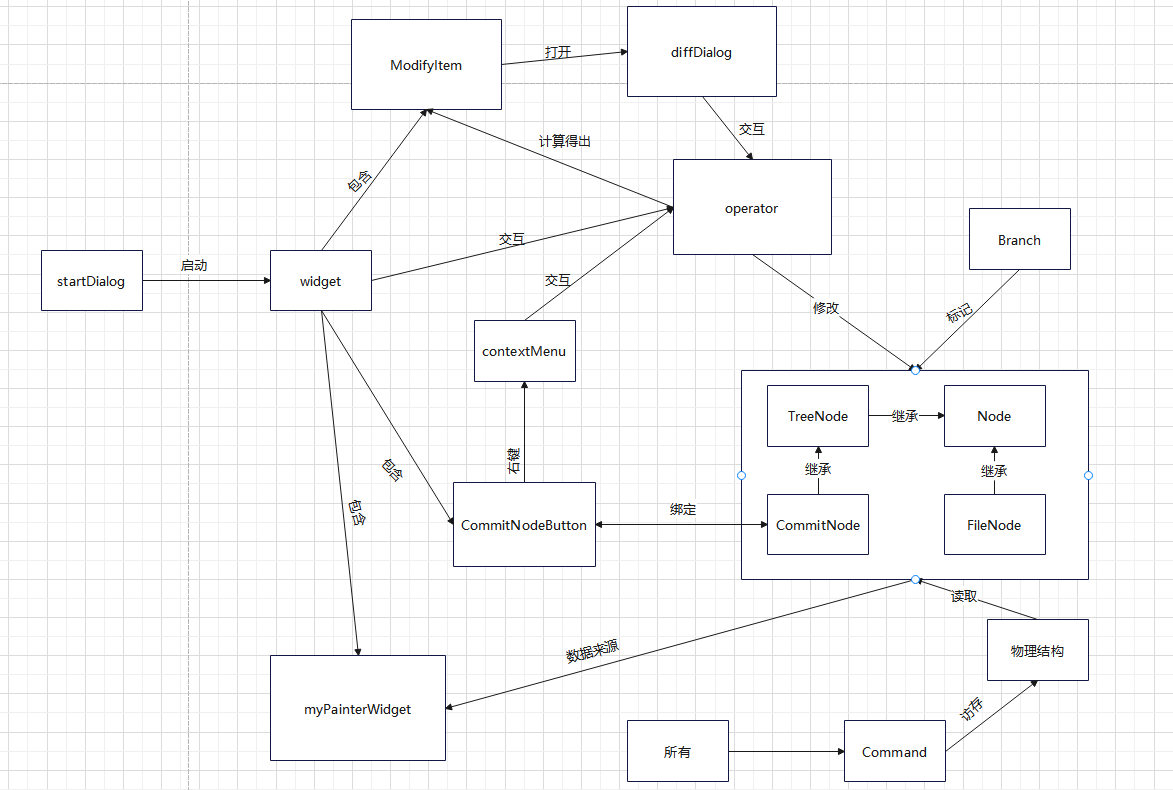
### 3.1.1Visual-Vers子系统模块汇总

1. **Command命令模块**
2. **Operator系统操作模块**
3. **Operator用户操作模块**
4. **Node节点结构模块**
5. **Branch分支模块**
6. **ModifyItem差异记录模块**
7. **CommitNodeButton数据界面衔接模块**
8. **myPainterWidget绘图界面**
9. **startDialog初始界面**
10. **widget用户操作界面**
11. **diffDialog差异对比界面**
12. **contextMenu弹出式交互界面**

### 3.1.2Visual-Vers子系统模块汇总表

|  |  |  |
| --- | --- | --- |
| **表示层** | | |
| 页面名称 | 模块名称 | 功能简述 |
| startDialog | 开始界面 | 初始界面，打开/新建仓库，最近仓库 |
| widget | 用户操作界面 | 处理大多交互内容 |
| diffdialog | 差异对比界面 | 显示差异内容 |
| contextMenu | 弹出式交互页面 | 对具体节点的交互操作 |
| myPainterWidget | 绘图界面 | 绘制版本图 |
| **数据处理层** | | |
| 模块名称 | 功能简述 | |
| Command | 常用命令，文件操作，格式转换 | |
| Operator | 封装大多数操作，实现用户可直接调用的增删查改操作 | |
| Node | 项目核心数据结构 | |
| Branch | 实现多分支，管理当前分支 | |
| ModifyItem | 项目对比，文件对比时记录差异，打开diffdialog | |
| CommitNodeButton | 与版本节点一一对应，将用户点击操作转换为对节点的具体操作 | |

## 3.2模块关系图



# 4．用户子系统的模块设计

## 4.1表示层模块设计

## 4.1.1 开始界面

|  |  |
| --- | --- |
| 模块名称 | startDialog |
| 功能描述 | 初始界面，打开/新建仓库，最近仓库 |
| 接口与属性 | private vector<string> lines; //记录最近仓库  private void openRepo(); //打开选中的仓库 |
| 数据结构与算法 |  |
| 补充说明 |  |

## 4.2.2 主界面

|  |  |
| --- | --- |
| 模块名称 | widget |
| 功能描述 | 处理大多交互内容 |
| 接口与属性 | class CommitNode; //前置声明  class Widget : public QWidget  {  Q\_OBJECT  public:  static Widget \*ins;// 单例  Ui::Widget \*ui;  std::string curAvatar; //当前用户头像  explicit Widget(QWidget \*parent = nullptr);  ~Widget();  void updateGraph(); //计算并刷新绘图区  void updateAvatar();//更新用户头像  //在某处放置按钮  float putButton(CommitNode \*p, float xPos, float yPos, std::unordered\_set<CommitNode\*> &vis);  protected:  bool eventFilter(QObject \*watched, QEvent \*event); //事件过滤器处理头像点击事件  void resizeEvent(QResizeEvent \*event); //窗口变化事件  public slots:  bool on\_freshButton\_clicked(); //刷新建交互  void on\_commitButton\_clicked(); //提交按钮交互  void on\_closeButton\_clicked(); //关闭按钮交互  // void on\_switchToBranchButton\_clicked();  // void on\_createBranchButton\_clicked();  void on\_restoreButton\_clicked(); //撤销按钮交互  void on\_workSpaceWidget\_itemDoubleClicked(QListWidgetItem \*item); //diff列表项查看  };  //便捷使用  #define MainWidget Widget::ins  #define MainUI Widget::ins->ui  #define isWorkSpaceClean MainWidget->on\_freshButton\_clicked() |
| 数据结构与算法 | 整个主界面应用单例模式，计算绘图区内容和按钮位置时用到了树形dp |
| 补充说明 |  |

## 4.2.3 差异对比界面

|  |  |
| --- | --- |
| 模块名称 | diffdialog |
| 功能描述 | 显示差异内容 |
| 接口与属性 | //构造函数中传入差异内容，传入应用差异时的源和目标  Public explicit diffDialog(vector<string> &diff, const string &name,  const string &f = "", const string &t = "", QWidget \*parent = nullptr); |
| 数据结构与算法 | vector |
| 补充说明 |  |

## 4.2.4弹出菜单

|  |  |
| --- | --- |
| 模块名称 | contextMenu |
| 功能描述 | 对具体节点的交互操作 |
| 接口与属性 | Public slots:  void on\_switchToNodeButton\_clicked(); //切换到节点按钮  void on\_pullFromCommitButton\_clicked(); //拉取合并按钮  void on\_diffWithNodeButton\_clicked(); //对比差异按钮 |
| 数据结构与算法 | 无 |
| 补充说明 |  |

## 4.2.5 绘图界面

|  |  |
| --- | --- |
| 模块名称 | myPainterWidget |
| 功能描述 | 绘制版本图 |
| 接口与属性 | Private:  std::vector<QLineF> lines; //保存需要绘制的内容，目前只画线（自带箭头  public:  explicit myPainterWidget(QWidget \*parent = nullptr);  void clearContents(); //清除内容  void addLine(QLineF l); //加入线条 |
| 数据结构与算法 | Vector，箭头绘制算法，计算位置时用到的树形dp |
| 补充说明 |  |

## 4.2数据处理层

## 4.2.1命令模块

|  |  |
| --- | --- |
| 模块名称 | Command |
| 功能描述 | 常用命令，文件操作，格式转换 |
| 接口与属性 | //#define ROOT\_PATH ("D:\\repo")  extern string ROOT\_PATH; //各种常量定义  #define REPO\_PATH (ROOT\_PATH + "\\.vvs")  #define DATA\_PATH (REPO\_PATH + "\\data")  #define DATA\_PATH\_ (REPO\_PATH + "\\data\\")  #define COMMIT\_PATH (REPO\_PATH + "\\commits")  #define COMMIT\_PATH\_ (REPO\_PATH + "\\commits\\")  #define BRANCH\_FILE\_PATH (REPO\_PATH + "\\.branch")  #define DEFAULT\_AVATAR ":/images/img/default.png"  const string EMPTY\_HASH = "00000000000000000000000000000000";  const string ROOT\_ID = "yemmm000000000000000000000000000t";  bool doCmd(const string &cmd, string &res); //命令行Api  bool doCmd(const string &cmd);  bool CopyAFile(const string &a, const string &b); //各种文件操作  void DeleteAny(const string &a);  void CreateFolder(const string &a);  const int FOLDER\_PATH = 1; //路径判断和路径宏  const int FILE\_PATH = 2;  const int EMPTY\_PATH = 0;  bool isempty(const string &path); //判断文件是否为空  int judgePath(const string &path); //判断路径类型  //更高级的文件操作  vector<string> fileList(const string &path, bool filter = true); //列出一个目录下所有文件名，filter可选是否应用过滤  void readFile(const string &path, vector<string> &lines); //读取一个文件的内容，写入lines中  void loadFile(const string &path, vector<string> &lines); //将lines内容写入一个文件  bool isTextFile(const string &path); //判断是否为文本类型文件  void Hint(const string &info, QWidget \*parent = MainWidget); //提示Api  void Warning(const string &info, QWidget \*parent = MainWidget);  void Error(const string &info, QWidget \*parent = MainWidget);  void mergeHash(string &hs, const string &name); //将名称混入哈希值  string getHash(const string &path, const string &name, const char tag); //获取文件哈希值  int str2int(const string &str); //常用类型转换  string int2str(int x);  QString Str2Q(const string &str);  string Q2Str(const QString &qstr);  string gbk2utf(const string &gbkStr); //用于windows文件操作的utf-gbk转换  string utf2gbk(const string &utf8Str); |
| 数据结构与算法 | Windows自带的MD5哈希算法，gbk，utf转换算法 |
| 补充说明 | 由于windows系统是gbk的，本项目所有涉及文件名的操作都需要经过utf-gbk转换，故不直接读取文件，全部通过command封装的函数，输入是utf-8编码的字符串。 |

## 4.2.2操作模块

|  |  |
| --- | --- |
| 模块名称 | Operator |
| 功能描述 | 实现大多数具体操作，分两部分，一部分是处理函数，以递归调用的函数为主；另一部分用户可直接调用，参数更通俗，经过检查和处理后进入递归函数处理。 |
| 接口与属性 | void readAllNodes(); //读取项目仓库中所有节点（硬盘到内存）  void readSubNode(TreeNode \*p); //读取一个TreeNode下所有子节点  void readCommit(CommitNode \*p); //读取一个版本的所有子节点  void readAllCommits(); //读取所有版本  void readBranch(); //读写分支  void loadBranch();  void readAvatar(string &s); //读写头像  void loadAvatar(string &s);  //---------------------------------------------------------------------------------//  //递归处理部分  Node\* commitFile(const string &path, const string &name); //提交工作区路径下的文件，当前执行到path  void loadNode(const string &path, Node \*node); //加载node及其子节点到工作区path处  void compareAndSwitch(const string &path, TreeNode \*from, TreeNode \*to); //将工作区从from切换到to，当前处于path  bool compareAndMerge(const string &path, TreeNode \*lca, TreeNode \*work, //合并work与tar，其公共祖先为lca，冲突结果写入diff，当前处于path  TreeNode \*tar, vector<ModifyItem\*> &diff, int option = 0);  void diffWithNode(TreeNode \*from, const string &path, vector<ModifyItem\*> &result); //对比，将from节点到工作区path的修改内容写入result  //---------------------------------------------------------------------------------//  //面向用户部分  bool isFileNode(const string &id); //判断一个id对应的节点类型  bool isTreeNode(const string &id);  bool isCommitNode(const string &id);  CommitNode\* getCommitLca(CommitNode \*a, CommitNode \*b); //获取两个版本的Lca  void init(); //初始化项目仓库  bool commitAllWork(const string &comment); //提交工作区所有内容  bool forceLoad(CommitNode \*target); //强制转换到节点  bool forceLoad(const string &id);  void restore(); //撤销工作区内容  bool switchToNode(CommitNode \*target); //切换工作区到节点  bool switchToNode(const string &id);  bool switchToBranch(const string &brname);  //对比两个文件，将它们的具体修改内容（具体道行）写入result  bool getDiffBetween(const string &path1, const string &path2, vector<string> &result);  //从某节点拉取合并  bool pullFromCommit(CommitNode \*target, const string &comment, vector<ModifyItem\*> &diff, int option = 0);  bool pullFromBranch(const string &brname, const string &comment, vector<ModifyItem\*> &diff, int option = 0);  //创建分支  bool createBranch(const string &brname); |
| 数据结构与算法 | commitFile，loadNode, compareAndSwitch, compareAndMerge, diffWithNode均采用文件树上递归的方式处理。  getDiffBetween运用到了求最小编辑次数的myvers算法 |
| 补充说明 |  |

## 4.2.3 节点模块

|  |  |
| --- | --- |
| 模块名称 | Node |
| 功能描述 | 项目核心数据结构，逻辑结构 |
| 接口与属性 | //所有节点的抽象类  class Node {  public:  string id, name;  Node(string \_id, string \_name):  id(move(\_id)), name(move(\_name)){}  virtual void save() = 0; //保存/更新节点到磁盘  virtual int getType() = 0; //获取类型  bool isTreeNode();  bool isFileNode();  bool isCommitNode();  };  //树节点，对应文件夹  class TreeNode: public Node {  public:  vector<Node\*> son; //文件树下的子节点  TreeNode(string \_id, string \_name):  Node(move(\_id), move(\_name)){}  void appendSon(Node \*node);  void save() override;  int getType() override;  };  //文件节点，对应单个文件  class FileNode: public Node {  public:  FileNode(string \_id, string \_name):  Node(move(\_id), move(\_name)){}  void save() override; //避免成为抽象类，实际FileNode的存放不在此处进行（因为我设计一个FileNode不关注其所在路径，仅关注内容）  int getType() override; //type: 1 file, 2 Tree, 4 commit  };  class CommitNodeButton; //前置声明  //提交记录节点，同时作为一次提交的文件树根节点  class CommitNode: public TreeNode {  public:  static CommitNode \*rootCommit; //记录所有提交记录的根节点（最早的空版本，在readAllCommit时指定）  CommitNodeButton \*myButton; //每个提交节点和一个button关联，创建CommitNode时自动创建button  CommitNode \*lastCommitNode[2]; //版本树上的父节点  vector<CommitNode\*> nextCommit; //版本树上的子节点  string avatar; //头像（用户）  string time; //提交时间  string comment; //评论  int dep; //深度  CommitNode(string \_comment, string \_id, CommitNode \*fa1, CommitNode \*fa2 = nullptr);//可能有两个父提交  CommitNode(string \_comment, TreeNode \*\_tnode, CommitNode \*fa1, CommitNode \*fa2 = nullptr); //直接由Treenode转换而来  ~CommitNode();  void save() override;  int getType() override;  };  extern map<string, Node\*> nodePool; //id索引 |
| 数据结构与算法 | Id索引，指向Node的map；  两棵树衔接的数据结构，文件树为可持久数据结构（类似主席树），版本树为有向无环图，具体结构见下图：  nodes |
| 补充说明 |  |

## 4.2.4 Branch分支模块

|  |  |
| --- | --- |
| 模块名称 | Branch |
| 功能描述 | 管理分支，主要管理当前分支 |
| 接口与属性 | //分支 即为CommitNode指针  class Branch {  public:  CommitNode \*position; //指向的CommitNode  string name, id; //快捷获取指向的CommitNode的name和id  Branch(string \_name, CommitNode \*\_position): name(move(\_name)), position(\_position){  if(position != nullptr) id = position->id;  }  void moveTo(CommitNode \*\_pos); //切换指向的目标  };  extern map<string, Branch\*> branch; //name索引  extern Branch\* currentBranch; //当前分支 |
| 数据结构与算法 | 名称索引，指向Branch的Map |
| 补充说明 |  |

## 4.2.5 修改记录模块

|  |  |
| --- | --- |
| 模块名称 | ModifyItem |
| 功能描述 | 记录修改内容并加以呈现，和diffDialog紧密相关 |
| 接口与属性 | class ModifyItem: public QListWidgetItem{  public:  vector<QListWidgetItem\*> linkItem; //相关联项  char type; //+, -, \*三种样式, ' '表无样式  string info; //显示内容，实际链接都是path，info为路径时，自动去除前面的RootPath  string path1, path2; //产生diff的两个文件，path1->path2  string applyTar; //强制更新目标，若不为空则允许应用path2->applyTar  ModifyItem(char \_type, const string &\_info, const string &\_p1 = "", const string &\_p2 = "", QListWidget \*parent = nullptr);  void beclicked();  }; |
| 数据结构与算法 | Vector |
| 补充说明 |  |

## 4.2.6 提交记录按钮

|  |  |
| --- | --- |
| 模块名称 | CommitNodeButton |
| 功能描述 | 衔接用户界面与提交记录节点 |
| 接口与属性 | class CommitNode; //前置声明  class CommitNodeButton : public QPushButton {  Q\_OBJECT  static set<CommitNodeButton\*> allbut; //静态存放所有按钮  public:  CommitNode \*myNode; //对应的CommitNode，和数据层衔接  float xPos, yPos; //所处位置，与myPainterWidget衔接  float occupyHeight; //这个按钮后续占用的总高度，部署前计算。  CommitNodeButton(CommitNode \*\_node, QWidget \*parent = nullptr);  ~CommitNodeButton();  void setImage(QString \_img); //设置图片  void setPosition(float \_x, float \_y);//设置位置  protected:  void contextMenuEvent(QContextMenuEvent \*ev); //右键事件，和contextMenu衔接  public slots:  void beclicked(); //点击事件  }; |
| 数据结构与算法 | Set |
| 补充说明 |  |