

东软秘密

**设计报告  
*（HwcPlayer-音乐播放器）***

东软集团股份有限公司IT人才实训中心

版权所有，翻版必究

**更改履历**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 版本 | 更改时间 | 更改人 | 更改章节 | 状态 | 更改描述 |
| 1 | V1.0 | 2021-7-22 | 胡未名 | 全部 | 新建 |  |
| 2 | V2.0 | 2021-7-28 | 胡未名 | 全部 | 修改 | 重构工程结构 |
| 3 | V2.1 | 2021-7-29 | 胡未名 | 全部 | 修改 | 更新项目模块 |

状态：新建、增加、修改、删除。

目 录

[1 引言 3](#_Toc224533617)

[1.1 编制目的 3](#_Toc224533618)

[1.2 词汇表 3](#_Toc224533619)

[1.3 参考资料 3](#_Toc224533620)

[2 系统开发环境 4](#_Toc224533621)

[3 系统设计思路 4](#_Toc224533622)

[4 系统公用代码设计 4](#_Toc224533623)

[4.1 分页组件类： 4](#_Toc224533624)

[4.2 过滤组件类： 4](#_Toc224533625)

[4.3 错误处理页 4](#_Toc224533626)

[4.4 异常处理 5](#_Toc224533627)

[5 功能模块设计 5](#_Toc224533628)

[*5.1* *XXXX* 5](#_Toc224533629)

[6 数据库设计 5](#_Toc224533630)

[6.1 功能说明 5](#_Toc224533631)

[6.2 数据表设计 6](#_Toc224533632)

[6.3 视图设计 6](#_Toc224533633)

[6.4 序列设计 6](#_Toc224533634)

# 引言

## 编制目的

本报告详细完成对音乐播放器项目的整体设计，达到指导开发的目的，同时实现和测试人员及用户的沟通。

本报告面向详细设计人员、开发人员、测试人员及最终用户而编写，是了解系统的导航。

## 词汇表

|  |  |  |
| --- | --- | --- |
| 词汇名称 | 词汇含义 | 备注 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 参考资料

无

# 系统开发环境

操作系统： *Windows 10*

集成开发工具：*Visual Studio 2019*

编译环境：*MSVC*

# 系统设计思路

*将不同的功能封装在不同的模块中，功能之间的切换通过模块之间的相互调用来实现。*

# 系统公用代码设计

## 胶水模块：

### 功能描述

*存储所有使用到的标准头文件和部分宏定义，被其他所有模块直接或间接调用，用于简化工程结构*

### 类、方法描述

#### *Glue.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | CLEAR\_BUF | 清空缓存区，在获取用户输入时经常被使用 | 无 |
|  |  |  |  |

## 链表模块：

### 功能描述

*定义音乐文件结构体，定义与链表操作有关的函数*

### 类，方法描述

#### *List.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| Media \* | Media \*createMedia(const char \*path); | 根据文件路径创建一个Media结构体，返回它的地址 | path:指定的文件路径 |
| MediaNodePtr | MediaNodePtr createNode(const char \*path); | 根据文件路径创建一个节点，返回它的地址 | path:指定的文件路径 |
| MediaNodePtr | MediaNodePtr createList(); | 创建一个空链表，返回这个链表的地址 |  |
| int  1 if yes, 0 if no | int isListEmpty(MediaNodePtr startPtr); | 判断链表是否为空 | startPtr:指定的链表表头 |
| const char \*：对应节点的短路径 | const char \*getNodeShortPathByNumber(MediaNodePtr startPtr, unsigned int number); | 接收一个节点编号，返回对应节点的短路径 | startPtr：指定链表的表头  number：节点编号 |
| unsigned int：对应节点的路径 | const char\* getNodePathByNumber(MediaNodePtr startPtr, unsigned int number); | 接收一个节点编号，返回对应节点的路径 | startPtr：指定链表的表头  number：节点编号 |
| unsigned int：对应节点的编号 | unsigned int getNodeNumberByShortPath(MediaNodePtr startPtr, const char\* short\_path); | 接收一个节点短路径，返回对应节点的编号 | startPtr：指定链表的表头  short\_path:指定的短路径 |
| unsigned int：对应节点的编号 | unsigned int getNodeNumberByPath(MediaNodePtr startPtr, const char \*path); | 接收一个节点路径，返回对应节点的编号 | startPtr：指定链表的表头  path:指定的路径 |
| const char \*：对应节点的路径 | const char \* getNodePathByShortPath(MediaNodePtr startPtr, const char\* short\_path); | 接收一个节点短路径，返回对应节点的路径 | startPtr：指定链表的表头  short\_path:指定的短路径 |
| int | int appendNode(MediaNodePtr &startPtr, const char \*path); | 根据文件位置生成一个Media结构体和与之对应的节点，把它添加到链表末尾 | startPtr：指定链表的表头  path：指定路径 |
| int | int deleteNode(MediaNodePtr &startPtr, unsigned int num); | 根据编号删除一个链表的节点 | startPtr：指定链表的表头  num：节点编号 |
|  | void clearList(MediaNodePtr &startPtr); | 清空以传入地址为首的整个链表 | startPtr：指定链表的表头 |
|  | void printList(MediaNodePtr &startPtr); | 打印以传入地址为首的链表的每个节点 | startPtr：指定链表的表头 |
| unsigned int | unsigned int getLength(MediaNodePtr startPtr); | 返回链表长度（在数值上等于链表最后一个节点的编号） | startPtr：指定链表的表头 |
| const char \* | const char \*getMediaNameByNumber(MediaNodePtr startPtr, unsigned int num); | 根据编号返回节点的音乐名称 | startPtr：指定链表的表头  num：节点编号 |
|  |  |  |  |

## 文件处理模块

### 功能说明

*存储与文件存储和读取有关的操作函数*

### 类，方法描述

#### *File.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void storeList(const char \*filename, MediaNodePtr currentPtr); | 将指定链表的信息以二进制形式存储在指定的文件中，如果文件不存在会被创建，已存在则会先删除后创建 | filename: 文件名  currentPtr：指定链表的表头 |
|  | void readList(const char \*filename, MediaNodePtr sPtr); | 从指定的文件中读取链表信息，并存储到指定链表中 | filename：文件名  sPtr：指定链表的表头 |

## 时间处理模块

### 功能说明

*存储所有处理时间有关的函数*

### 类，方法描述

#### *MyTime.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| int | int getCurrentYear(); | 获取当前年份 |  |
| int | int getCurrentMonth(); | 获取当前月份 |  |
| int | int getCurrentDay(); | 获取当前天数 |  |
| int | int getCurrentHour(); | 获取当前小时数（北京时间） |  |
| int | int getCurrentMinute(); | 获取当前分钟数 |  |
| int | int getCurrentSecond(); | 获取当前秒数 |  |

## 媒体处理模块

### 功能说明

*存储媒体交互基本函数*

### 类，方法描述

#### *Media.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| int，成功返回0，不成功返回非0 | int MymciSendString(const char \*szCommand, char \*szbuffer); | 调用MCI接口对音乐文件进行操作 | szCommand：指令字符串  szbuffer：缓存字符串，如果不需要可以取NULL |
| int，成功返回0，不成功返回非0 | int openMusicByShortPath(const char \*short\_path); | 调用MCI接口根据短路径打开文件，同时将相应的.lrc文件加载到歌词链表中 | short\_path：文件短路径 |
| int，成功返回0，不成功返回非0 | int closeMusic(); | 调用MCI接口关闭文件 |  |
| int，成功返回0，不成功返回非0 | int playMusic(); | 调用MCI接口播放音乐 |  |
| int，成功返回0，不成功返回非0 | int playMusicRepeat(); | 调用MCI接口循环播放音乐 |  |
| int，成功返回0，不成功返回非0 | int stopMusic(); | 调用MCI接口停止音乐 |  |
| int，成功返回0，不成功返回非0 | int pauseMusic(); | 调用MCI接口暂停音乐 |  |
| int，成功返回0，不成功返回非0 | int resumeMusic(); | 调用MCI接口继续音乐 |  |
| int，成功返回0，不成功返回非0 | int seekToPosition(int position); | 调用MCI接口将播放位置放到指定位置 | position：指定的播放位置，是一个整数 |
| int，当前播放的位置 | int getMusicCurrentPosition(); | 调用MCI接口获取当前播放位置 |  |
| int，当前音乐的总播放时长 | int getMusicLength(); | 调用MCI接口获取当前音乐的总播放时长 |  |
| int，当前音乐的播放速度，标准为1000 | int getMusicSpeed(); | 调用MCI接口获取当前音乐的播放速度 |  |
|  | void setMusicSpeed(int speed); | 调用MCI接口设置当前音乐的播放速度 | speed：指定的音乐播放速度，标准为1000 |

## 音量控制模块

### 功能说明

*存储控制媒体与播放器音量的基本函数*

### 类，方法描述

#### *Volume.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void setAudioVolume(int audio\_volume); | 设置系统音量 | audio\_volume：指定的系统音量，范围0-100 |
| 当前的系统音量，范围0-100 | int getAudioVolume(); | 获取当前系统音量 |  |
| true：当前播放器静音  false：当前播放器没有静音 | bool isSongMute(); | 检测当前播放器是否静音 |  |
|  | void setSongMute(bool mute); | 设置播放器是否静音 | mute：指定播放器是否静音，是为true，否为false |
| 当前的播放器音量，范围0-1000 | int getSongVolume(); | 获取当前播放器音量 |  |
|  | void setSongVolume(int song\_volume); | 设置播放器音量 | song\_volume：指定的播放器音量，范围0-1000 |

# 功能模块设计

*从本章开始，逐个地给出各个功能模块的设计考虑。以下给出的提纲是针对一般情况的。对于一个具体的模块，尤其是层次比较低的模块或子程序，其很多条目的内容往往与它所隶属的上一层模块的对应条目的内容相同，在这种情况下，只要简单地说明这一点即可。*

## *播放列表页面*

### 功能说明

*显示播放列表页面，同时承担交互等功能。*

### 类、方法设计

5.1.2.1 ListPage.h  
 *该类的功能：*显示播放列表页面，同时承担交互等功能，可以被其他模块调用

功能：

从指定文件夹中导入一个音乐

从指定文件夹中导入所有音乐

删除音乐列表中的一个音乐

清空播放列表

返回主菜单界面

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void listPage(); | 显示播放列表页面，同时承担交互等功能，可以被其他模块调用  功能：  从指定文件夹中导入一个音乐  从指定文件夹中导入所有音乐  删除音乐列表中的一个音乐  清空播放列表  返回主菜单界面 |  |
|  | void showListPage(); | 只显示播放列表界面的样子，不承担交互等功能，不可以被其他模块调用 |  |
| true 表示点击确定，false 表示不选择和取消。 | bool getFolder(std::string& folderpath, const char\* szCaption, HWND hOwner); | 打开并获取文件夹对话框。 | folderpath:对文件夹路径的引用  szCaption: 对话框窗口标题  hOwner: 父窗口句柄 |
| 成功返回0，不成功返回非0 | int loadPathInFolder(MediaNodePtr sPtr); | 将指定文件夹中的音乐文件信息存到指定链表中 | sPtr：指定链表的表头 |

### 相关数据表

*无*

## *播放器页面*

### 功能说明

展示播放器页面，同时承担页面交互

交互功能：

1. 播放与暂停

2. 切换播放模式：列表循环、单曲循环和随机播放

3. 跳转到播放列表的指定曲目

4. 调整歌曲播放的倍速：1.25,1.0，0.8

5. 上一曲、下一曲

6. 快退10秒、快进10秒

7. 减小、增大系统音量

8. 减小、增大播放器音量

9. 静音、取消静音

10. 返回主菜单页面

### 类、方法设计

#### PlayerPage.h *该类的功能：中止音乐文件的播放，在中断点处继续播放*。

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void loadProcessBarBuf(int nowTime, int musicTime); | 根据现在播放时间与播放总时长，生成一个代表进度条的字符串并存储在全局变量processBarBuf中 | nowTime：当前已播放的时间  musicTime：播放总时间 |
|  | void loadStatusBuf(); | 获取当前播放状态，并存储在全局变量szStatusBuf中 |  |
|  | void loadPlayStatusBuf(); | 获取当前播放模式，并存储在全局变量szPlayStatusBuf中 |  |
|  | void loadMediaNameBuf(); | 获取当前播放的音乐名，并存储在全局变量szMediaNameBuf中 |  |
|  | void loadMediaSpeedBuf(); | 获取当前播放速度，并存储在全局变量szMediaSpeedBuf中 |  |
|  | void loadLrcBuf(); | 用到PlayerPage.cpp一个全局变量lrcBuf  首先检查歌词链表是否为空，如果为空，向lrcBuf添加提示信息；  如果不为空（歌词加载成功），检查当前播放时间是否有一句对应的歌词  如果有，将歌词存储在lrcBuf中  如果没有，不改变lrcBuf的内容 |  |
|  | void playMusicUp(); | 播放列表的上一首歌，可以循环 |  |
|  | void playMusicDown(); | 根据当前播放模式，选择下一首要播放的歌曲并播放 |  |
|  | void stepBackward10Sec(); | 将当前播放进度倒退10秒左右 |  |
|  | void stepForward10Sec(); | 将当前播放进度前进10秒左右 |  |
|  | void switch\_play\_status(); | 切换当前的播放模式，在以下三种模式间循环切换：  列表循环  单曲循环  随机播放 |  |
|  | void switchMusicSpeed(); | 切换当前的播放速度，在以下三种速度间循环切换  1倍速  1.25倍速  0.8倍速 |  |
|  | void loadIsSongMute(); | 获取当前播放器是否静音，将状态存储在全局变量isSongMuteBuf中 |  |
|  | void switchSongMute(); | 将播放器在静音与非静音间切换  如果从非静音切换到静音，会先将当前播放音量存储在全局变量temp\_song\_volume中  如果从静音切换到非静音，会自动将播放音量设为temp\_song\_volume，从而恢复之前的播放音量 |  |
|  | void audioVolumeUp(); | 将系统音量增大固定值 |  |
|  | void audioVolumeDown(); | 将系统音量减小固定值 |  |
|  | void songVolumeUp(); | 将播放器音量增大固定值 |  |
|  | void songVolumeDown(); | 将播放器音量减小固定值 |  |

### 相关数据表

*无*

## 链表模块：

### 功能描述

*定义音乐文件结构体，定义与链表操作有关的函数*

### 类，方法描述

#### *List.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| Media \* | Media \*createMedia(const char \*path); | 根据文件路径创建一个Media结构体，返回它的地址 | path:指定的文件路径 |
| MediaNodePtr | MediaNodePtr createNode(const char \*path); | 根据文件路径创建一个节点，返回它的地址 | path:指定的文件路径 |
| MediaNodePtr | MediaNodePtr createList(); | 创建一个空链表，返回这个链表的地址 |  |
| int  1 if yes, 0 if no | int isListEmpty(MediaNodePtr startPtr); | 判断链表是否为空 | startPtr:指定的链表表头 |
| const char \*：对应节点的短路径 | const char \*getNodeShortPathByNumber(MediaNodePtr startPtr, unsigned int number); | 接收一个节点编号，返回对应节点的短路径 | startPtr：指定链表的表头  number：节点编号 |
| unsigned int：对应节点的路径 | const char\* getNodePathByNumber(MediaNodePtr startPtr, unsigned int number); | 接收一个节点编号，返回对应节点的路径 | startPtr：指定链表的表头  number：节点编号 |
| unsigned int：对应节点的编号 | unsigned int getNodeNumberByShortPath(MediaNodePtr startPtr, const char\* short\_path); | 接收一个节点短路径，返回对应节点的编号 | startPtr：指定链表的表头  short\_path:指定的短路径 |
| unsigned int：对应节点的编号 | unsigned int getNodeNumberByPath(MediaNodePtr startPtr, const char \*path); | 接收一个节点路径，返回对应节点的编号 | startPtr：指定链表的表头  path:指定的路径 |
| const char \*：对应节点的路径 | const char \* getNodePathByShortPath(MediaNodePtr startPtr, const char\* short\_path); | 接收一个节点短路径，返回对应节点的路径 | startPtr：指定链表的表头  short\_path:指定的短路径 |
| int | int appendNode(MediaNodePtr &startPtr, const char \*path); | 根据文件位置生成一个Media结构体和与之对应的节点，把它添加到链表末尾 | startPtr：指定链表的表头  path：指定路径 |
| int | int deleteNode(MediaNodePtr &startPtr, unsigned int num); | 根据编号删除一个链表的节点 | startPtr：指定链表的表头  num：节点编号 |
|  | void clearList(MediaNodePtr &startPtr); | 清空以传入地址为首的整个链表 | startPtr：指定链表的表头 |
|  | void printList(MediaNodePtr &startPtr); | 打印以传入地址为首的链表的每个节点 | startPtr：指定链表的表头 |
| unsigned int | unsigned int getLength(MediaNodePtr startPtr); | 返回链表长度（在数值上等于链表最后一个节点的编号） | startPtr：指定链表的表头 |
| const char \* | const char \*getMediaNameByNumber(MediaNodePtr startPtr, unsigned int num); | 根据编号返回节点的音乐名称 | startPtr：指定链表的表头  num：节点编号 |

## 文件处理模块

### 功能说明

*存储与文件存储和读取有关的操作函数*

### 类，方法描述

#### *File.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void storeList(const char \*filename, MediaNodePtr currentPtr); | 将指定链表的信息以二进制形式存储在指定的文件中，如果文件不存在会被创建，已存在则会先删除后创建 | filename: 文件名  currentPtr：指定链表的表头 |
|  | void readList(const char \*filename, MediaNodePtr sPtr); | 从指定的文件中读取链表信息，并存储到指定链表中 | filename：文件名  sPtr：指定链表的表头 |

## 时间处理模块

### 功能说明

*存储所有处理时间有关的函数*

### 类，方法描述

#### *MyTime.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| int | int getCurrentYear(); | 获取当前年份 |  |
| int | int getCurrentMonth(); | 获取当前月份 |  |
| int | int getCurrentDay(); | 获取当前天数 |  |
| int | int getCurrentHour(); | 获取当前小时数（北京时间） |  |
| int | int getCurrentMinute(); | 获取当前分钟数 |  |
| int | int getCurrentSecond(); | 获取当前秒数 |  |

## 媒体处理模块

### 功能说明

*存储媒体交互基本函数*

### 类，方法描述

#### *Media.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| int，成功返回0，不成功返回非0 | int MymciSendString(const char \*szCommand, char \*szbuffer); | 调用MCI接口对音乐文件进行操作 | szCommand：指令字符串  szbuffer：缓存字符串，如果不需要可以取NULL |
| int，成功返回0，不成功返回非0 | int openMusicByShortPath(const char \*short\_path); | 调用MCI接口根据短路径打开文件，同时将相应的.lrc文件加载到歌词链表中 | short\_path：文件短路径 |
| int，成功返回0，不成功返回非0 | int closeMusic(); | 调用MCI接口关闭文件 |  |
| int，成功返回0，不成功返回非0 | int playMusic(); | 调用MCI接口播放音乐 |  |
| int，成功返回0，不成功返回非0 | int playMusicRepeat(); | 调用MCI接口循环播放音乐 |  |
| int，成功返回0，不成功返回非0 | int stopMusic(); | 调用MCI接口停止音乐 |  |
| int，成功返回0，不成功返回非0 | int pauseMusic(); | 调用MCI接口暂停音乐 |  |
| int，成功返回0，不成功返回非0 | int resumeMusic(); | 调用MCI接口继续音乐 |  |
| int，成功返回0，不成功返回非0 | int seekToPosition(int position); | 调用MCI接口将播放位置放到指定位置 | position：指定的播放位置，是一个整数 |
| int，当前播放的位置 | int getMusicCurrentPosition(); | 调用MCI接口获取当前播放位置 |  |
| int，当前音乐的总播放时长 | int getMusicLength(); | 调用MCI接口获取当前音乐的总播放时长 |  |
| int，当前音乐的播放速度，标准为1000 | int getMusicSpeed(); | 调用MCI接口获取当前音乐的播放速度 |  |
|  | void setMusicSpeed(int speed); | 调用MCI接口设置当前音乐的播放速度 | speed：指定的音乐播放速度，标准为1000 |

## 音量控制模块

### 功能说明

*存储控制媒体与播放器音量的基本函数*

### 类，方法描述

#### *Volume.h*

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void setAudioVolume(int audio\_volume); | 设置系统音量 | audio\_volume：指定的系统音量，范围0-100 |
| 当前的系统音量，范围0-100 | int getAudioVolume(); | 获取当前系统音量 |  |
| true：当前播放器静音  false：当前播放器没有静音 | bool isSongMute(); | 检测当前播放器是否静音 |  |
|  | void setSongMute(bool mute); | 设置播放器是否静音 | mute：指定播放器是否静音，是为true，否为false |
| 当前的播放器音量，范围0-1000 | int getSongVolume(); | 获取当前播放器音量 |  |
|  | void setSongVolume(int song\_volume); | 设置播放器音量 | song\_volume：指定的播放器音量，范围0-1000 |

## *歌词处理模块*

### 功能说明

*处理有关歌词的函数和相关参数。*

### 类、方法设计

#### Lrc.h *该类的功能：存储处理有关歌词的函数和相关参数*。

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
| LrcNodePtr: 歌词链表表头 | LrcNodePtr createLrcList(); | 创建歌词链表，返回链表的表头 |  |
|  | void appendLrcNode(Lrc lrc); | 向歌词链表中追加一个节点 | lrc：要加入的一句歌词的信息 |
| 成功返回0，不成功返回非0 | int loadIrcList(const char\* filename); | 打开一个.lrc文件，并将其信息加载到歌词链表中 | filename：指定的文件名称（绝对路径） |
|  | void clearLrcList(); | 清空歌词链表 |  |
| 是，返回true；否，返回false | bool isLrcListEmpty(); | 检测歌词链表是否为空 |  |
| const char \*：对应的歌词字符串，默认为“\0” | const char\* getLrcByTime(int time); | 接收一个整数作为时间(单位为秒），返回指定歌词节点的字符串，如果没找到返回"\0" | time：指定的时间（单位为秒） |

### 相关数据表

*无*

## *字体颜色控制模块*

### 功能说明

存储有关控制台文本颜色操作的函数和相关参数*。*

### 类、方法设计

#### TextColor.h *该类的功能：*存储有关控制台文本颜色操作的函数和相关参数。

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  | void setConsoleTextColor(WORD fore\_color); | 设置控制台标准输出的字体颜色 | fore\_color: 指定的字体颜色 |

### 相关数据表

*无*

## *XXXX*

### 功能说明

*说明该模块应具有的功能。*

### 类、方法设计

*说明实现该功能模块所涉及的重要类和方法，模板如下表。*

#### XXXX类 *该类的功能：XXXXXXXXXXXXXXXXXXX*。

|  |  |  |  |
| --- | --- | --- | --- |
| 返回值 | 方法名 | 功能 | 参数说明 |
|  |  |  |  |

### 相关数据表

*XXXXXX表*

# 数据库设计

## 功能说明

*对数据库设计中涉及到的各种项目，如数据项、记录、表、数据库等建立数据字典，以说明它们的标识符、同义名及有关信息，并说明对此数据字典设计的基本考虑，如有单独数据库设计文档，此处标明文件名和存储位置即可。*

## 数据表设计

### *表名：XXXXX,表功能说明*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名** | **类型** | **可为空** | **默认** | **注释** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 视图设计

## 序列设计