SmartWord设计文档

1. 设计需求

设计实现一个背单词的工具

- 以动态链接库形式提供算法核心API
- 算法与实现分离，程序模块可复用
- 可以根据用户实际英语能力的变化动态设定单词级别（如罕见词、生词、熟词、易忘常见词等等）
- 可以调整记忆策略
- 可以对指定的文本文件进行生词统计并给出生词解释
- 控制台窗口界面
- 可以进行在线单词测试
- 允许用户为单词添加例句，以后显示单词含义时一并显示用户添加的例句
- 以交互方式来查询用户输入单词的解释，记录查询历史。
- 除C++标准库外不得使用其他程序库
- 单词库为纯文本文件，一行一个单词

2. 开发人员

学号	姓名	负责模块
2015011372	张钰晖	1. 整体框架提出 2. 算法核心类（Word, Dictionary, History, Recite, Translation, Test, ChineseTest, EnglishTest, SpellTest, Random, Testlist）的设计与实现 3. 交互类(Login, 部分Handler派生类)的实现 4. Qt中GUI绘制及实现GUI版本 5. MacOS平台程序测试与Bug修复 6. Qt平台测试与Bug修复 7. PPT及开发文档制作
2015011355	范臻	1. 算法核心类（Word, Dictionary, Test, ChineseTest, EnglishTest, SpellTest, History, Translation）的设计与实现 2. 交互类(Processor, Handler及其所有派生类)的设计与实现 3. Windows平台程序测试与Bug修复 4. API文档撰写 5. PPT制作 6. GUI界面的绘制 7. Qt平台测试与Bug修复
2015011304	罗华一	1. 算法核心类(Word, Dictionary, History, Translation)的设计与实现 2. GUI界面的绘制 3. API文档撰写 4. 获取并处理词库 5. Linux平台程序测试与Bug修复 6. 使用说明撰写 7. Qt平台测试与Bug修复

3.版本控制

3.1 终端版本

版本号	完成时间	实现功能
Version 0.0	2016-04-18	核心API框架的构想
Version 1.0	2016-04-20	设计Word类，存储单词以释义，提供部分接口
Version 1.1	2016-04-21	完善Word类功能及接口
Version 2.0	2016-04-23	Dictionary类的提出与实现
Version 2.1	2016-04-25	终端用户界面制作，交互式查词的实现
Version 3.0	2016-05-03	History类的设计与实现
Version 3.1	2016-05-05	Translation类的设计与实现
Version 3.2	2016-05-07	Random类、Recite类设计与实现
Version 3.3	2016-05-09	Test类及其派生类设计与实现
Version 3.4	2016-05-10	TestList类设计及实现
Version 3.5	2016-05-11	单词本功能的实现
Version 4.0	2016-05-13	MVC设计思路的提出，重构交互类代码, 所有功能基本实现
Version 4.1	2016-05-14	完善部分功能
Version 4.2	2016-05-14	完善终端界面
Version 5.0	2016-05-18	多次测试并完成Bug修复，增加登录和个人信息储存功能
Version 5.1	2016-05-20	完善细节，跨平台的实现，Bug修复
Version 6.0	2016-05-21	最终版

3.2 Qt版本

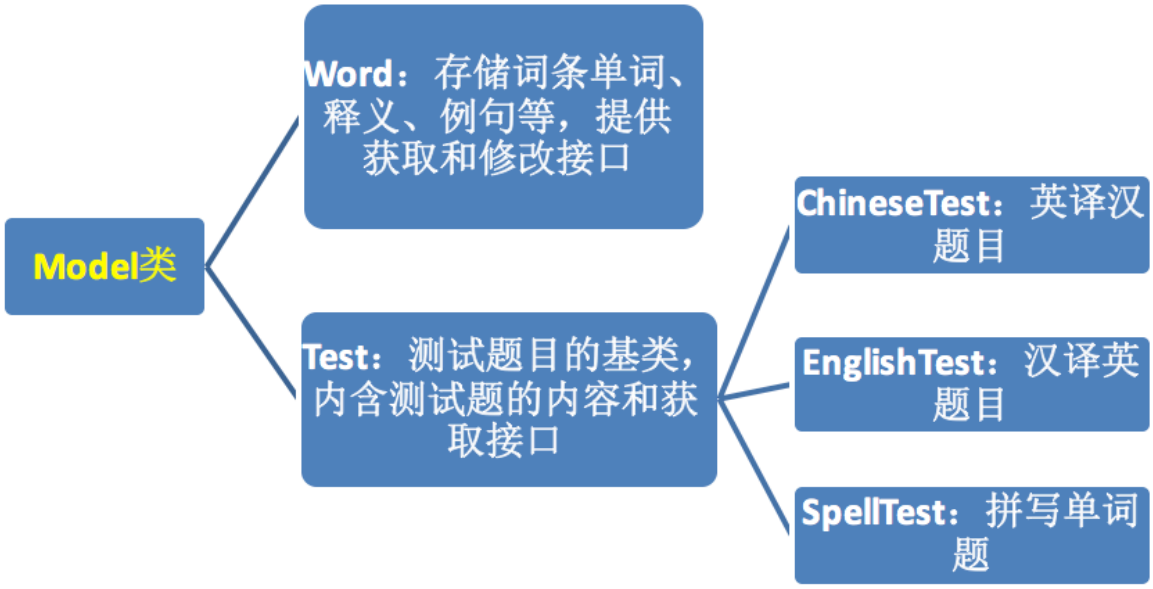
版本号	完成时间	实现功能
Version 1.0	2016-05-04	GUI绘制
Version 2.0	2016-05-06	交互式查词和导入词典功能
Version 2.1	2016-05-07	细节优化，用户操作不当输出提示信息
Version 3.0	2016-05-12	美化GUI，增加历史记录、我的单词本功能
Version 4.0	2016-05-15	增加文本翻译、背诵、测试功能，基本功能基本实现
Version 5.0	2016-05-17	美化GUI，实现登录、个人信息管理功能
Version 5.1	2016-05-18	一些细节Bug修复
Version 5.2	2016-05-19	跨平台测试，Bug修复
Version 6.0	2016-05-20	最终版

4. 终端版设计思路

在大作业设计方面我们采用了Model-View-Controller的MVC设计模式。

4.1 Model——数据储存

- Word类：整个字典中最重要的信息类，储存词条信息（如拼写，释义等），并提供获取和修改词条的接口。
- Test类及其派生类：储存考试题目（由于测试题目结构特殊我们专门做了一个类。Test为虚基类，由Test派生出了之后需要用到的ChineseTest，EnglishTest，SpellTest类。分别实现汉译英、英译汉、拼写测试功能）
- History类：由于需要按顺序加入列表，选用核心数据结构std::vector，管理用户查询或记录历史。



4.2 Controller——数据处理

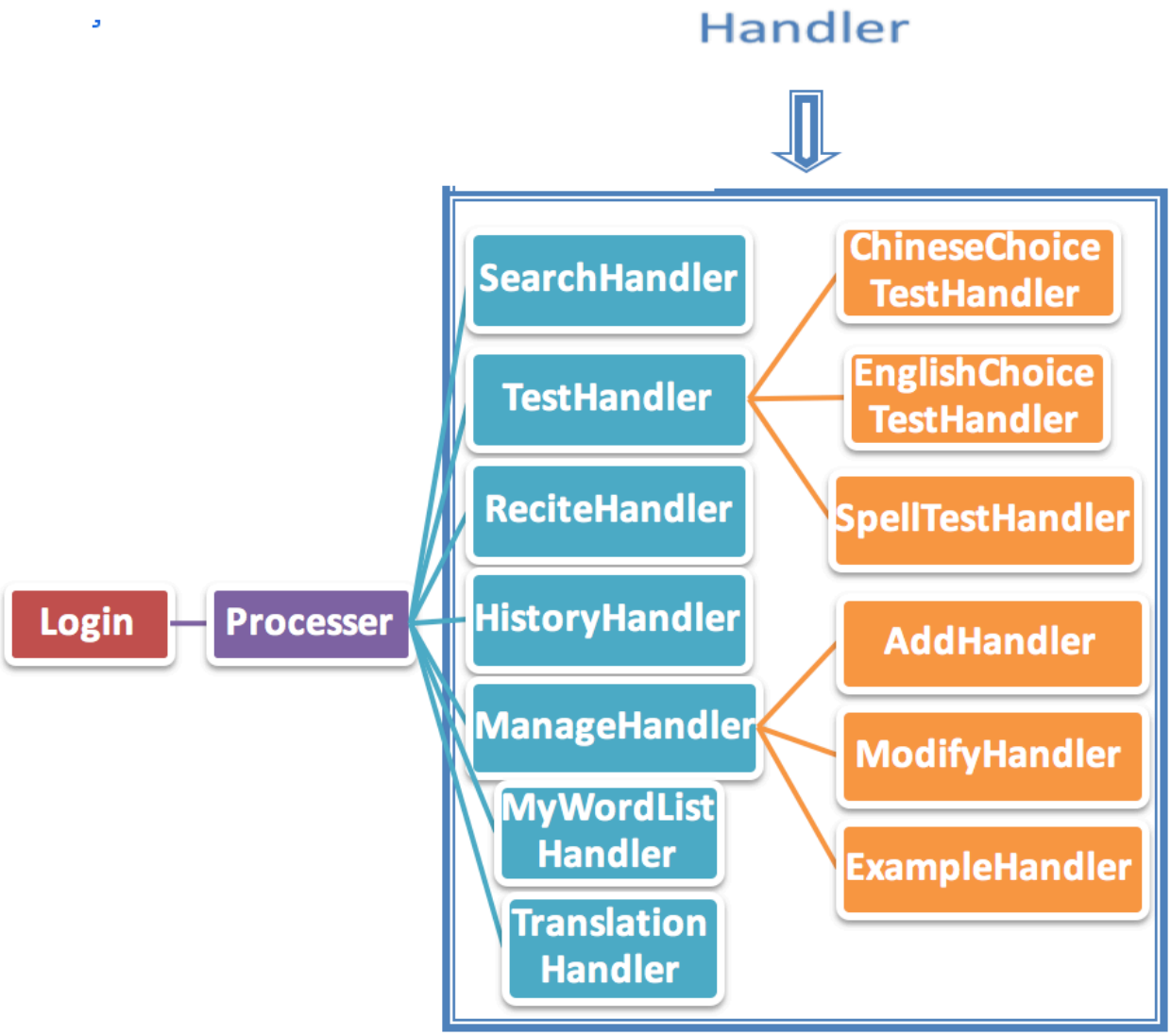
- Dictionary：储存和管理词典的核心结构，核心数据结构std::set。用于储存词典单词，并涉及多种获取信息、判断和修改词典的接口，如下所示：

```cpp
class Dictionary
{
private:
    //数据
    std::set<Word> Dict;
public:
    //构造与析构
    Dictionary();
    virtual ~Dictionary();
    //接口函数
    std::string getMeaning(const std::string &word) const;
    int getDegree(const std::string &word) const;
    std::string getExample(const std::string &word, const int &pos) const;
    std::string getTranslation(const std::string &word, const int &pos) const;
    int getNumOfExample(const std::string &word) const;
    std::string getAllWord() const;
    std::string getAllExample(const std::string &word) const;
    std::set<Word>::iterator searchWordWithIter(const std::string &word) const;
    Word getSearchWord(const std::string &word) const;
    Word getSpecificWord(const int &pos) const;
    int getDictSize() const;
    //文件函数
    bool importDict(const std::string &filename);
    bool importExample(const std::string &filename);
    bool exportDict(const std::string &filename);
    bool exportExample(const std::string &filename);
    //功能函数
    bool isExisted(const std::string &word);
    bool addWord(const std::string &word, const std::string &meaning, const int &degree = 0);
    bool addExample(const std::string &word, const std::string &example, const std::string &translation);
    bool modifyWord(const std::string &word, const std::string &new_meaning, const int &new_degree = 0);
    bool setDegree(const std::string &word, const int &new_degree = 0);
```

- Recite：生成、管理用户背诵内容
- Testlist：生成、管理测试题目列表
- Translation：处理文档或屏幕输入，从词典查询后，生成翻译

4.3 View——交互类

- 这里我们创建了一系列Handler类，由Handler作为基类，派生出一系列Handler负责词典各个功能的用户交互。用户打开词典（Main）并登录（Login）后首先进入主菜单（Processer），在其中选择各种功能，此时Processer根据用户指令跳转到相应Handler。



- 在解决Processer向Handler跳转方面，我们使用了策略模式（Strategy Pattern）以增强程序的可扩展性。在Processer中我们定义了指针数组handlers，储存指向各个特定Handler的指针。

```
class Processer
{
private:
    Handler *handlers[10];
```

```
handlers[0] = NULL;
handlers[1] = new SearchHandler(dict, hist, mydic);
handlers[2] = new ReciteHandler(dict, hist, mydic);
handlers[3] = new TestHandler(dict, hist, mydic);
handlers[4] = new ManageHandler(dict, hist, mydic);
handlers[5] = new HistoryHandler(dict, hist, mydic);
handlers[6] = new MyWordListHandler(dict, hist, mydic);
handlers[7] = new TranslationHandler(dict, hist, mydic);
```

- 在用户输入指令后由Processer中判断语句完成跳转。

```
else if ( choice[0] >= '1' && choice[0] <= '7' )
{
    handlers[choice[0] - '0']->MainFunction();
}
```

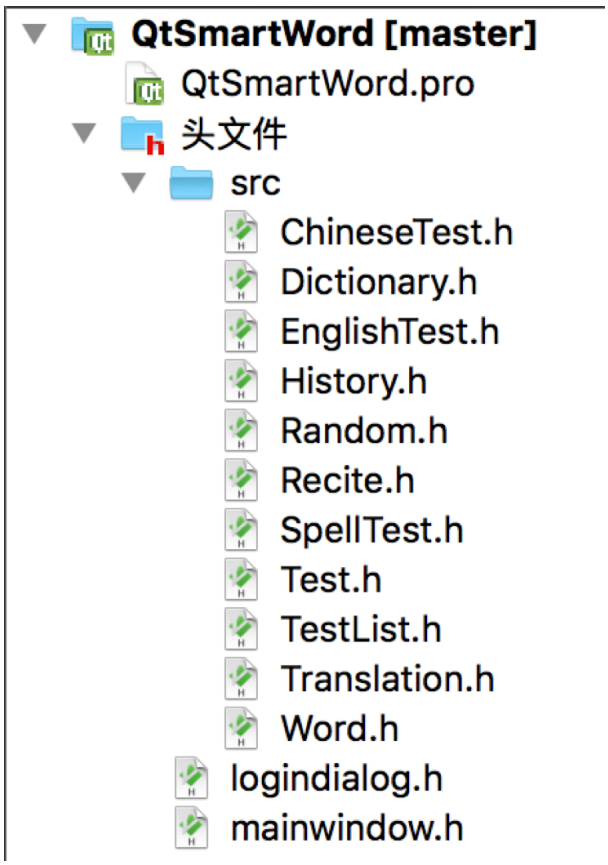
- ManageHandler和TestHandler中也采用此结构。

```
void ManageHandler::MainFunction()
{
    managehandlers[1] = new AddHandler(dict, hist, mydic);
    managehandlers[2] = new ModifyHandler(dict, hist, mydic);
    managehandlers[3] = new ExampleHandler(dict, hist, mydic);
```

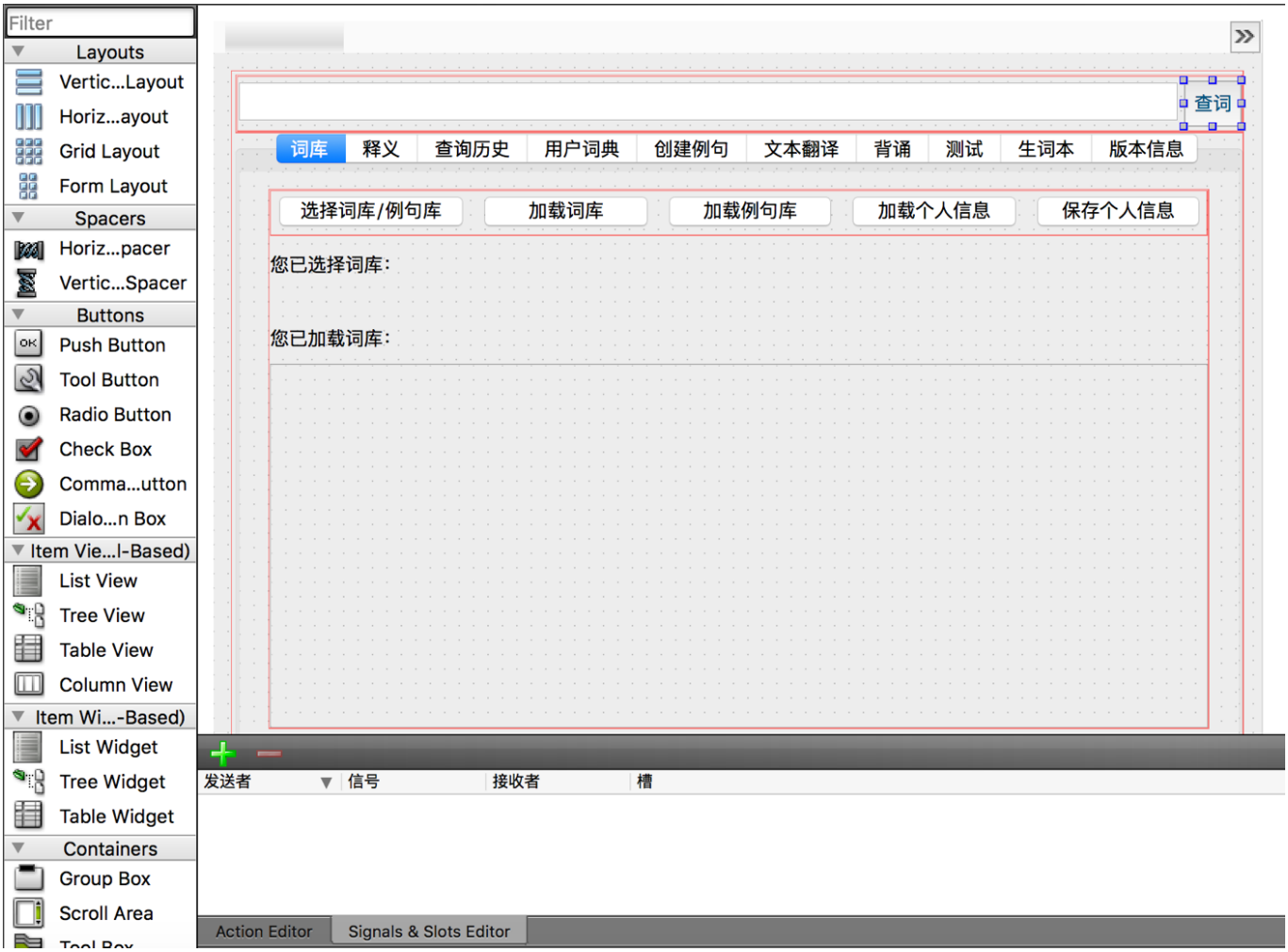
```
    testtype[1] = new EnglishChoiceTestHandler(dict, hist, mydic);
    testtype[2] = new ChineseChoiceTestHandler(dict, hist, mydic);
    testtype[3] = new SpellTestHandler(dict, hist, mydic);
```

5. Qt版设计思路

1. 核心API: 调用之前写好的SmartWord的核心API，仅改变Shell交互方式



2. UI绘制: 利用Qt设计者视图绘制



3. Qt中信号与槽机制:

信号

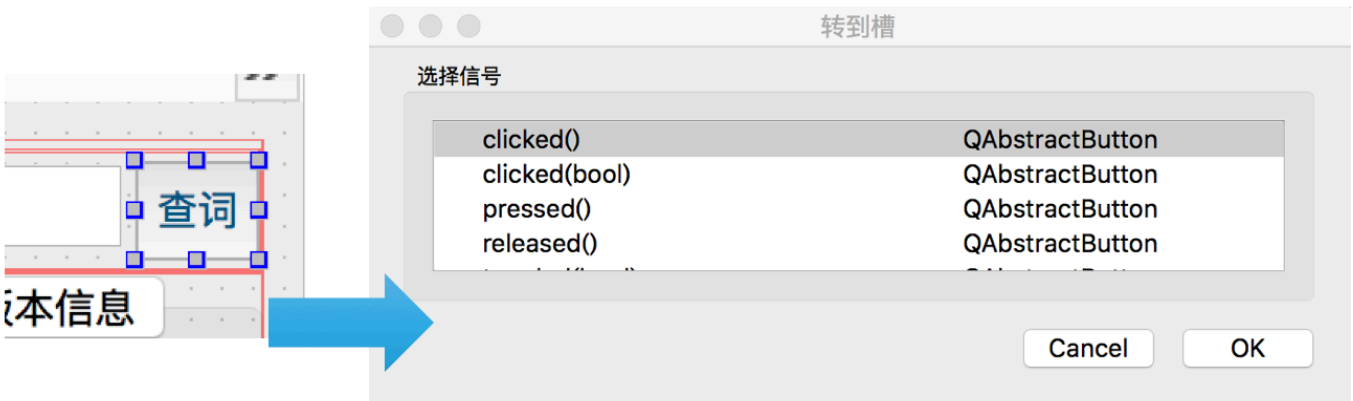
当对象改变其状态时，信号就由该对象发射 (emit) 出去，而且对象只负责发送信号，它不知道另一端是谁在接收这个信号。这样就做到了真正的信息封装，能确保对象被当作一个真正的软件组件来使用。

槽

用于接收信号，而且槽只是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。而且对象并不了解具体的通信机制。

信号与槽的连接

所有从 QObject 或其子类派生的类都能够包含信号和槽。因为信号与槽的连接是通过 QObject 的 connect() 成员函数来实现的。




```
void MainWindow::on_queryButton_clicked()
{
    QString query = ui->lineEdit->text();
    std::string stdquery = query.toStdString();
    std::string meaning = dict->getMeaning(stdquery);
    std::string example = dict->getAllExample(stdquery);
    int difficult = dict->getDegree(stdquery);

    std::stringstream in;
    in << difficult;
    std::string diff = in.str();

    if(dict->isExisted(stdquery)) hist->addWord(Word(stdquery, meaning));

    QString qmeaning = QString::fromStdString(meaning);
    QString qexample = QString::fromStdString(example);

    ui->meaningText->setText(qmeaning);
    ui->historyLabel->setText(QString::fromStdString(hist->getAllWordString()));
    ui->difficultText->setText(QString::fromStdString(diff));
    if(qexample.length()==0) {
        ui->exampleAllText->setText("没有例句");
    } else {
        ui->exampleAllText->setText(qexample);
    }
}
```

6. SmartWord API文档

6.1 Model类

类名	功能
Word	存储单词，提供修改接口
History	存储有序单词列表，提供修改接口
Test	题目的基类
ChineseTest	英译汉题目继承Test
EnglishTest	汉译英题目继承Test
SpellTest	拼写题目继承Test

- Word类
    - Public Member Functions

返回值	名称和参数	说明
	Word(const std::string &word)	构造函数
	Word(const std::string &word, const std::string &meaning)	构造函数
	~Word()	析构函数
std::string	getWord() const	返回单词
std::string	getMeaning() const	返回单词释义
std::string	getExample(const int &pos) const	返回第pos例句
std::string	getTranslation(const int &pos) const	返回第pos例句释义
std::string	getAllExample() const	返回所有例句及释义
int	getDegree() const	返回单词难度
int	getNumOfExample() const	返回例句数量
bool	setMeaning(const string &meaning)	设置释义
bool	addExample(const string &example, const string &translation)	增加例句
bool	modifyWord(const string &new_word, const string &new_meaning)	改变单词
bool	setDegree(const int &degree)	设置难度
bool	operator <(const Word &next) const	比较两个对象大小
bool	operator ()(const Word &n1, const Word &n2) const	比较两个对象相等

    - Friend Functions

返回值	名称和参数	说明
friend std::ostream	&operator <<(std::ostream &out, Word &obj)	输出对象

- History类
    - Public Member Functions

返回值	名称和参数	说明
	History()	构造函数
	virtual ~History()	析构函数
Word	getWord(const int &index) const	获取第index个历史
std::vector(Word)	getAllWord() const	获取所有历史
std::string	getAllWordString() const	获取所有历史string
bool	addWord(const Word &word)	把word加到history
bool	isExisted(const Word &word) const	判断是否有word
bool	addWordWithoutDuplicate(const Word &word)	去重增加word到history
bool	deleteWord(const int &index)	删掉第index项
bool	deleteAllWord()	清空历史
bool	importHistory(const std::string &filename)	导入历史
bool	exportHistory(const std::string &filename)	导出历史
int	getHistorySize() const	共有多少项

- Test类
  - Public Member Functions

返回值	名称和参数	说明
	Test(Dictionary *dict, const int &rand)	构造函数
	virtual ~Test()	析构函数
std::string	virtual std::string getAnswer() const	返回答案单词
std::string	virtual std::string std::string getAnswerMeaning() const	返回答案释义
std::string	virtual std::string std::string getTest() const = 0	返回测验题目
std::string	virtual std::string std::string getTestAns() const = 0	返回测验答案
bool	virtual bool makeTest() = 0	生成测验
bool	virtual bool isCorrect(const std::string &userans) const = 0	判断用户正误

- 剩余三个test类 **均继承自test类，实现各自的makeTest(), getTest(), getAnswer()功能**

6.2 Controller类

类名	功能
Dictionary	以set(Word)结构存储的单词本，提供查询等接口
Recite	生成背诵列表，提供接口
Testlist	生成测试列表，提供接口
Translation	实现翻译文本功能，提供接口

- Dictionary类
  - Public Member Functions

返回值	名称和参数	说明
	Dictionary()	构造函数
	virtual ~Dictionary()	析构函数
std::set(Word)iterator	searchWordWithIter(const std::string &word) const	返回迭代器
std::string	getMeaning(const std::string &word) const	返回word释义
std::string	getExample(const std::string &word, const int &pos) const	返回word的第pos例句
std::string	getTranslation(const std::string &word, const int &pos) const	返回word的第pos释义
std::string	getAllWord() const	返回所有释义
std::string	getAllExample(const std::string &word) const	返回word所有例句
Word	getSearchWord(const std::string &word) const	返回搜索word的对象
Word	getSpecificWord(const int &pos) const	返回第pos个word的对象

int	getDegree(const std::string &word) const	返回word难度
int	getNumOfExample(const std::string &word) const	返回word例句个数
int	getDictSize() const	返回单词个数
bool	importDict(const std::string &filename)	导入词典
bool	importExample(const std::string &filename)	导入例句
bool	exportDict(const std::string &filename)	导出词典
bool	exportExample(const std::string &filename)	导出例句
bool	isExisted(const std::string &word)	判断word是否存在
bool	addWord(const std::string &word, const std::string &meaning, const int &degree = 0)	添加单词
bool	addExample(const std::string &word, const std::string &example, const std::string &translation)	添加例句
bool	modifyWord(const std::string &word, const std::string &new*meaning, const int &new*degree = 0)	更改单词
bool	setDegree(const std::string &word, const int &new_degree = 0)	设置word难度

- Recite类
  - Public Member Functions

返回值	名称和参数	说明
	Recite()	构造函数
	virtual ~Recite()	析构函数
std::vector(Word)	makeReciteList()	制作测试列表，返回待测试内容

- TestList类
  - Public Member Functions

返回值	名称和参数	说明
	TestList(int num*ETOC, int num*CTOE, int num_SPEL, Dictionary *dict)	构造函数
	virtual ~TestList()	析构函数
std::vector(Test) *	getTestList() const	返回测试列表
bool	makeTestList()	生成测试列表
bool	clearList()	清空测试列表

- Translation类
  - Public Member Functions

返回值	名称和参数	说明
	Translation(Dictionary *dict))	构造函数
	virtual ~Translation()	析构函数
std::vector(std::string)	getSplitWord() const	分词去空格，生成vector
std::vector(std::string)	getSplitMeaning () const	以vector返回各个意思
std::string	getRawString() const	返回待处理字段
std::string	getOutString() const	返回已处理字段
bool	importFile(const std::string &filename)	导入文件
bool	setRawString(const std::string &rawstring)	设置待处理字段
bool	processRawString()	处理字段

- Random类
  - Public Member Functions

返回值	名称和参数	说明
int	getSingleRand(int l, int r)	返回一个随机值
int*	getMultiRand(int num, int l, int r)	返回多个随机值

6.3 View类

类名	功能
Login	登录界面处理器
Processer	主界面处理器，提供指向各处理器的指针
Handler	所有字典处理器的虚基类
HistoryHandler	展示及编辑历史记录处理器
ManageHandler	修改词典词条总处理器，提供指向各编辑处理器的指针
AddHandler	向词典中添加单词处理器
ExampleHandler	添加例句处理器
ModifyHandler	修改词条处理器
MyWordListHandler	记录并编辑用户生词本处理器
ReciteHandler	单词背诵处理器
SearchHandler	查询单词处理器
TestHandler	测试界面总处理器，提供指向各测试处理器的指针
ChineseChoiceTestHandler	中->英选择题测试处理器
EnglishChoiceTestHandler	英->中选择题测试处理器
SpellTestHandler	默写题测试处理器
TranslationHandler	文本翻译处理器

- Login类
  - Public Member Functions

返回值	名称和参数	说明
	Login()	构造函数
	~Login()	析构函数
void	CallHelper()	输出指示信息
void	GotoProcessor()	创建Processor指针

- Processer类
  - Public Member Functions

返回值	名称和参数	说明
	Processer(const std::string &username)	构造函数
	~Processer()	析构函数
void	CallHelper()	输出指示信息
void	GotoHandler()	处理用户选项进入相应处理器

- Handler类
  - Public Member Functions

返回值	名称和参数	说明
	Handler(Dictionary * dict, History * hist, History * mydic)	构造函数
	~Handler()	析构函数
void	CallHelper()	输出指示信息
void	MainFunction()	从上层处理器跳转的接口
bool	OwnFunction()	各处理器自身特定功能

- 其余Handler类 **均继承Handler类，实现各自的Callhelper(), Ownfunction功能()**