|  |  |
| --- | --- |
| **P-No.** | EX2019.001 |

|  |
| --- |
| **在线考试系统**  **结构设计书** |

第1.0版

|  |  |  |
| --- | --- | --- |
| 作成 |  |  |
| 翻訳 |  |  |
| 承認 |  |  |

**南京富士通南大軟件技術有限公司技術開発部**

All Rights Reserved, Copyright ©FNST 2005

|  |  |  |  |
| --- | --- | --- | --- |
| **修正履歴** | | **ページ** | 1 |
| **仕様書名** | 在线考试系统结构设计书 | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版数** | **修正日** | **修正概要** | **修正者** | **承認者** |
| 0.1 | 2005-07-26 | 1. 新规作成 |  |  |
| 0.2 | 2005-08-02 | 1. [1.1]背景，[4.3.4]分析文件，[4.4]外部接口设计的内容追加 2. [图2-1]关系描述修正 3. [2.2.1]，[3.2]流程图修正 4. [3.3] lpstrFlolder类型描述修正 5. [3.4] 函数filemain描述错误修正 6. [4.3.1]删除“使用opsrch函数时，只要将该表的首地址Optab以及表中元素个数作为第二第三参数传入opsrch函数即可。” |  |  |
| 1.0 |  | 1. [图3-3] 函数调用修正 2. 板式调整 |  |  |
| 2.0 |  | 1. [5.3]追加外部接口设计 2. [5.4]增加内部函数GetClassLink（），ReleaseClassLink（），ReleaseRelationLink（） |  |  |

**目 录**

[1. 功能概要 5](#_Toc110913296)

[2. 界面 7](#_Toc110913297)

[2.1 类的设计 7](#_Toc110913298)

[2.1.1 ClassAnalysisDlg类 7](#_Toc110913299)

[2.1.2 CAlarmMgr类 10](#_Toc110913300)

[2.2 事件触发流程图 12](#_Toc110913301)

[2.2.1 选择目录 12](#_Toc110913302)

[2.2.2 分析目录 13](#_Toc110913303)

[2.2.3 导出结果 14](#_Toc110913304)

[2.2.4 关于 16](#_Toc110913305)

[2.2.5 退出 17](#_Toc110913306)

[3. 文件处理 18](#_Toc110913307)

[3.1数据结构定义 18](#_Toc110913308)

[3.2 模块流程图 19](#_Toc110913309)

[3.3 外部接口设计 20](#_Toc110913310)

[3.3.1 输入 20](#_Toc110913311)

[3.3.2 输出 20](#_Toc110913312)

[3.4 内部函数设计 20](#_Toc110913313)

[3.4.1 自定义的函数 20](#_Toc110913314)

[3.4.2 调用的API函数 22](#_Toc110913315)

[4. 词法分析 23](#_Toc110913316)

[4.1数据结构定义 23](#_Toc110913317)

[4.1.1 被识别单词的保存 23](#_Toc110913318)

[4.1.2 标识符的定义 24](#_Toc110913319)

[4.1.3 输出定义 26](#_Toc110913320)

[4.2 模块流程图 28](#_Toc110913321)

[4.3 内部函数设计 29](#_Toc110913322)

[4.3.1 操作符识别 29](#_Toc110913323)

[4.3.2 单词识别 32](#_Toc110913324)

[4.3.3 单链表生成 35](#_Toc110913325)

[4.3.4 分析文件 35](#_Toc110913326)

[4.4 外部接口设计 36](#_Toc110913327)

[5. 逻辑分析 37](#_Toc110913328)

[5.1数据结构定义 37](#_Toc110913329)

[5.2 模块流程图 39](#_Toc110913330)

[5.3 外部接口设计 39](#_Toc110913331)

[5.4内部函数设计 40](#_Toc110913332)

# **功能概要**

## **1.1 功能概述**

在公司的招新活动中，传统的现场笔试效率低下，筛选力一般，线上笔试能够克服传统方法的不足。本期项目将开发一个在线笔试系统，用于公司的招新工作。面试官在系统中录入笔试题，笔试者进入系统时将会看到自己笔试题的URL链接，点击后进入笔试，面试官能够实时的观看笔试者的答题过程，答题完毕后也可回放该过程。

## **1.2 功能点**

基于以上的基本功能描述，可以得到如下的功能点一览表：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点描述 | 备注 |
| 1 | 系统登入身份判断 |  |
| 2 | 笔试题信息的录入 |  |
| 3 | 考生状态的判断 |  |
| 4 | 考生获取URL链接进入笔试 |  |
| 5 | 笔试过程的实时观看与回放 |  |

表1-1 功能点一览表

## **1.3 基本模型**

依据以上四个功能点，可以将整个软件分为以下模块：

1. 界面

界面分为面试官界面、考生界面和登入界面。面试官界面可即时查看各登入考生的状态，给等待笔试的考生提供考题的URL链接，实时观看正在进行笔试的考生答题过程，查看笔试结束的考生的答题回放；考生界面在没有收到URL链接时即为等待笔试，收到链接后可进入笔试界面，开始答题后开始计时。登入界面根据登入信息决定登入的是考生界面还是面试官界面。

1. 笔试题信息的录入

用于新建笔试题，在面试官界面操作，录入的题目将存入数据库

1. 考生状态判断

考生状态有三种，等待笔试、笔试进行、笔试结束，即时显示在面试官界面

1. 考生获取URL链接进入笔试

考生收到来自面试官提供的URL链接，通过链接进入笔试界面

1. 笔试过程的实时观看与回放

笔试过程中，面试官可进行实时观看，笔试结束后，面试官可以回看考生的笔试过程

各模块之间之间的关系如下图所示：

登入界面

考生界面

考生状态判断

获取URL进入笔试

笔试过程

实时观看与回放

面试官界面

试题录入

图 1-1 在线笔试系统的基本模型

对于开发者而言，系统则可以分为前端界面和后端控制系统。

# **系统架构图**

## **Springboot搭建SSM框架**

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。

从最根本上来讲，Spring Boot就是一些库的集合，它能够被任意项目的构建系统所使用。

要进行打包和分发的工程会依赖于像Maven或Gradle这样的构建系统。为了简化依赖图，Boot的功能是模块化的，通过导入Boot所谓的“starter”模块，可以将许多的依赖添加到工程之中。为了更容易地管理依赖版本和使用默认配置，框架提供了一个parent POM，工程可以继承它。

## **SSM框架**

SSM（Spring+SpringMVC+MyBatis）框架集由Spring、MyBatis两个开源框架整合而成（SpringMVC是Spring中的部分内容）。常作为数据源较简单的web项目的框架。

**Spring**

Spring就像是整个项目中装配bean的大工厂，在配置文件中可以指定使用特定的参数去调用实体类的构造方法来实例化对象。也可以称之为项目中的粘合剂。

Spring的核心思想是IoC（控制反转），即不再需要程序员去显式地`new`一个对象，而是让Spring框架帮你来完成这一切。

**SpringMVC**

SpringMVC在项目中拦截用户请求，它的核心Servlet即DispatcherServlet承担中介或是前台这样的职责，将用户请求通过HandlerMapping去匹配Controller，Controller就是具体对应请求所执行的操作。SpringMVC相当于SSH框架中struts。

**mybatis**

mybatis是对jdbc的封装，它让数据库底层操作变的透明。mybatis的操作都是围绕一个sqlSessionFactory实例展开的。mybatis通过配置文件关联到各实体类的Mapper文件，Mapper文件中配置了每个类对数据库所需进行的sql语句映射。在每次与数据库交互时，通过sqlSessionFactory拿到一个sqlSession，再执行sql命令。

页面发送请求给控制器，控制器调用业务层处理逻辑，逻辑层向持久层发送请求，持久层与数据库交互，后将结果返回给业务层，业务层将处理逻辑发送给控制器，控制器再调用视图展现数据。项目架构图如图2-1.

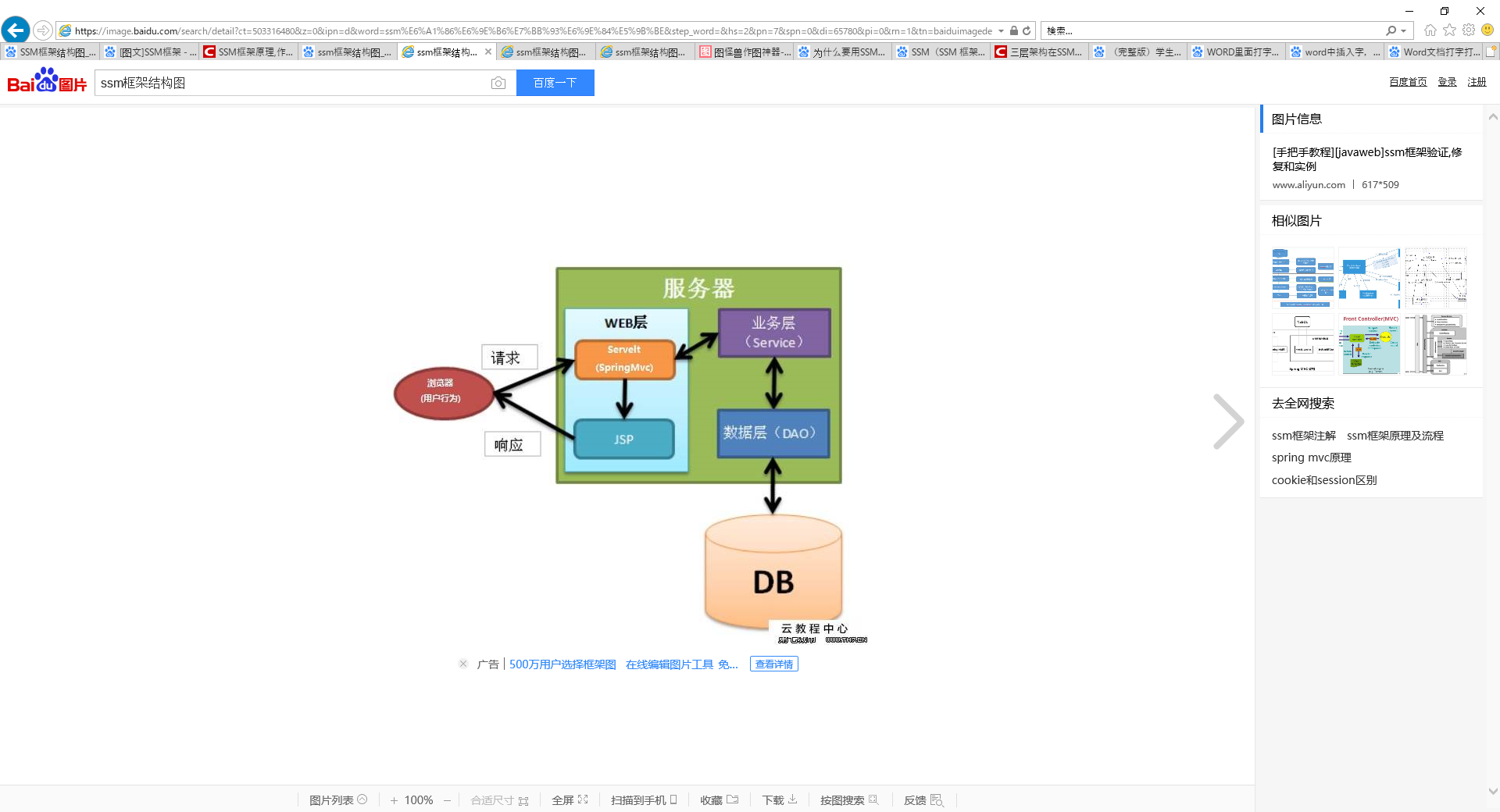


图 2-1 SSM架构图

## **数据库**

数据库采用mysql进行搭建。建表如图2-2.

## **界面的设计**

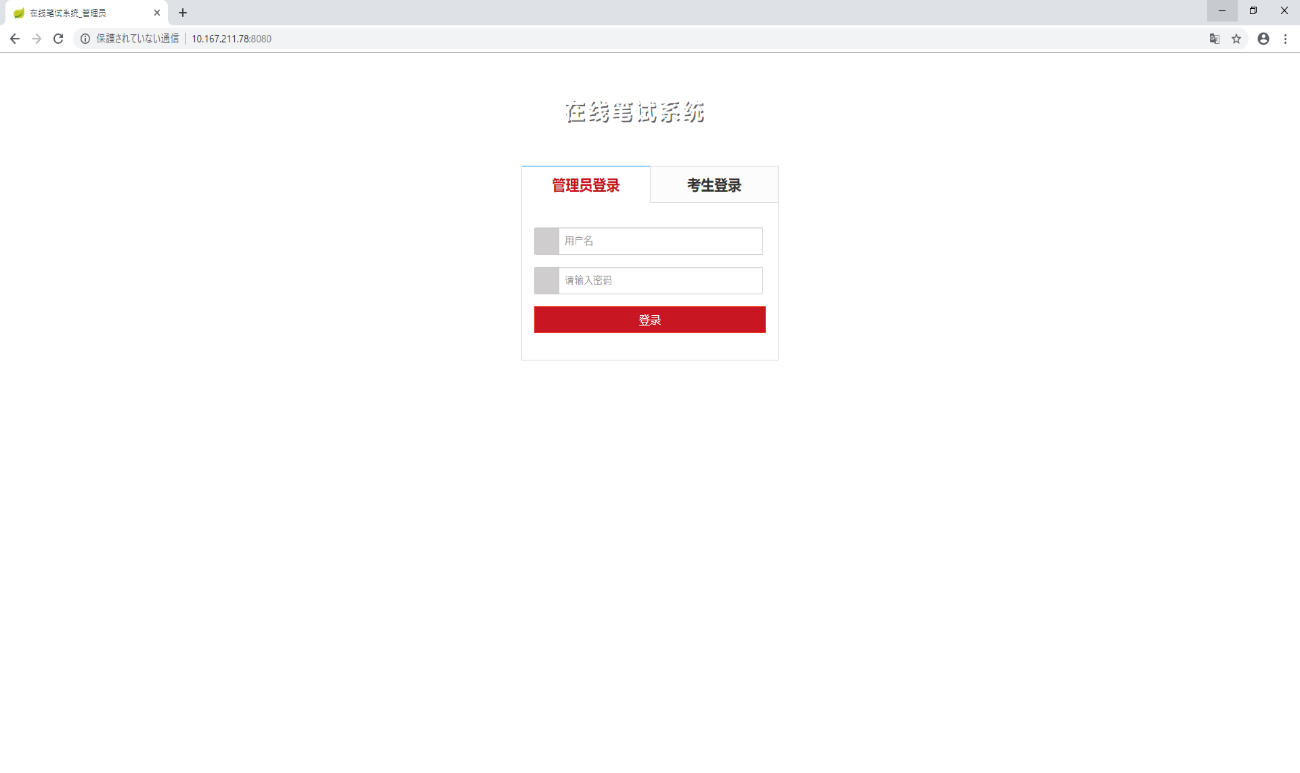


图 2-1 在线笔试系统登录界面

用户选择不同的权限进入网站，管理者进入后台管理界面，考生进入考试界面进行考试。

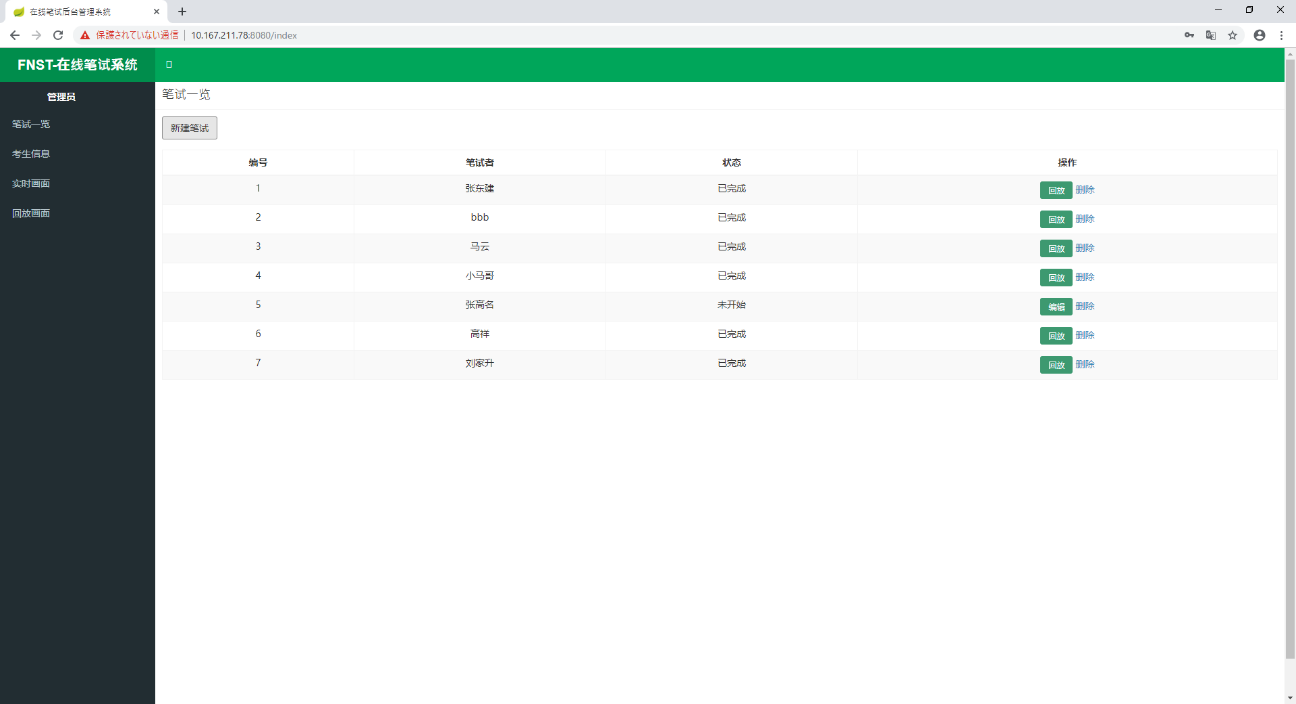


图 2-2 管理员界面-笔试一览模块

管理员在笔试一览界面可以查看考生的考试情况。未开考前，可以点击“编辑”修改考生题目；考试时，可以“直播”实时查看考生考试的电脑界面；考试结束后，还可以点击“回放”查看已经结束的考试信息。

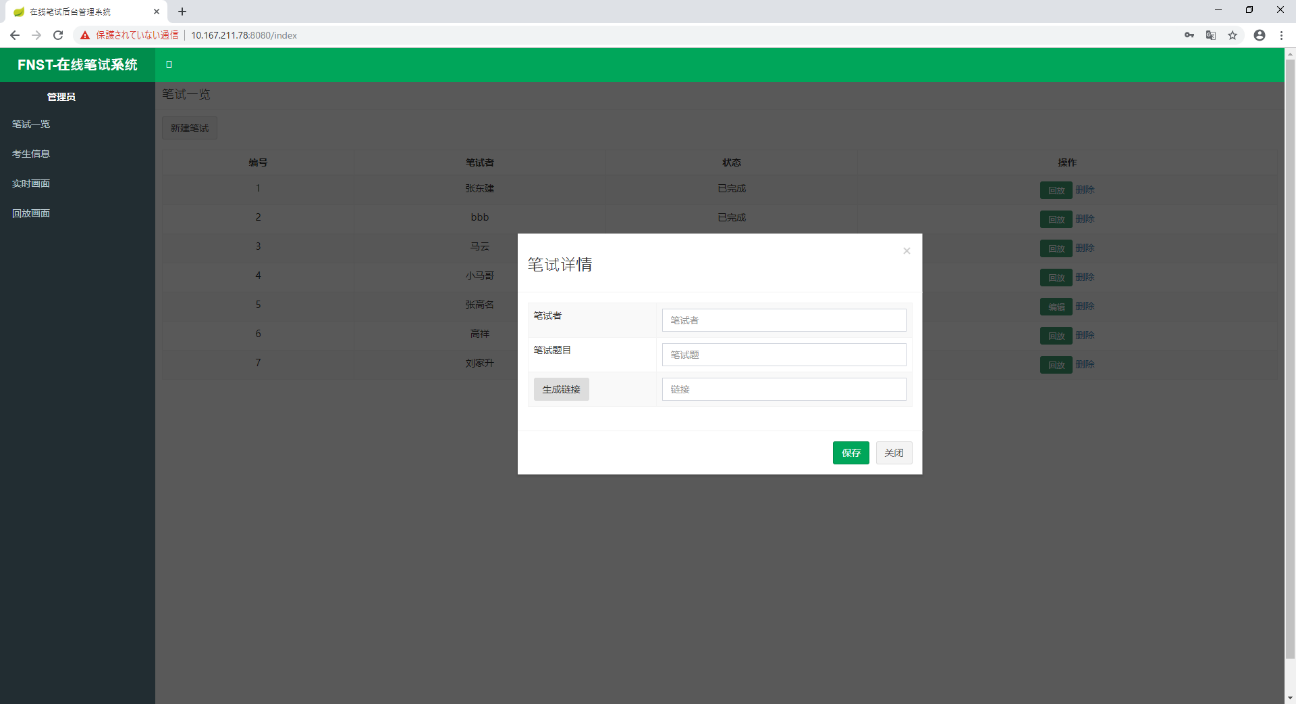


图 2-3 新建笔试界面

管理员可以点击“新建笔试”按钮来将考生与试题进行绑定，即建立考试，生成url考试链接。

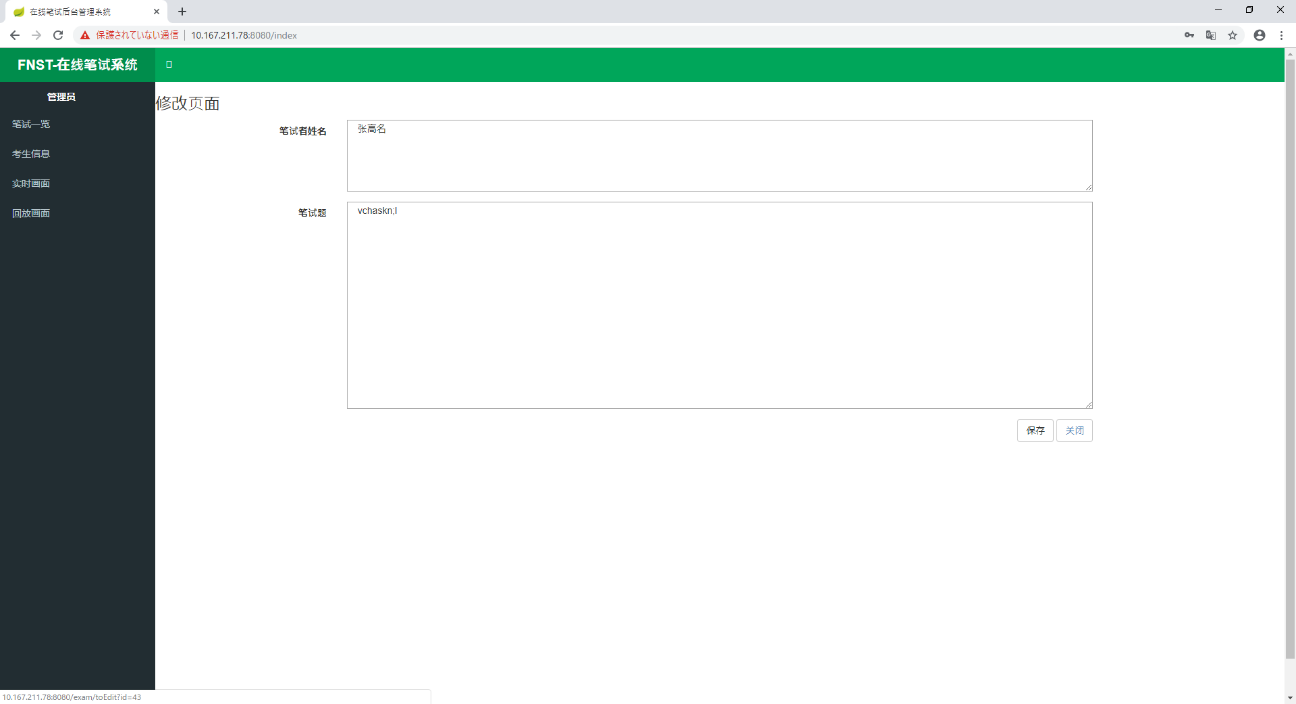


图 2-4 修改笔试界面

在考试未开始前，管理员可以点击“编辑”按钮，对已考试试题进行修改。

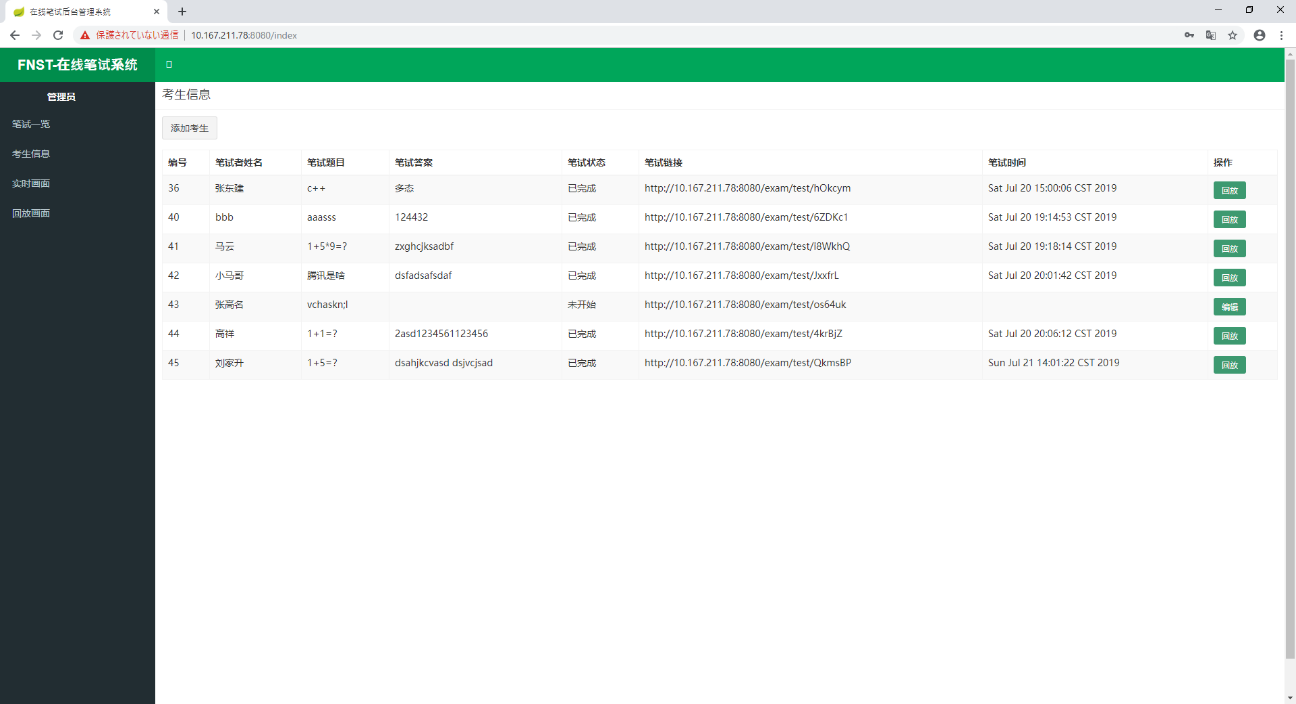


图 2-5考生信息界面

管理员也可以在考生信息界面查看考生的详细信息，包括：考生姓名，试题，答案，考试状态，url链接，考试完成时间以及查看考试视频。

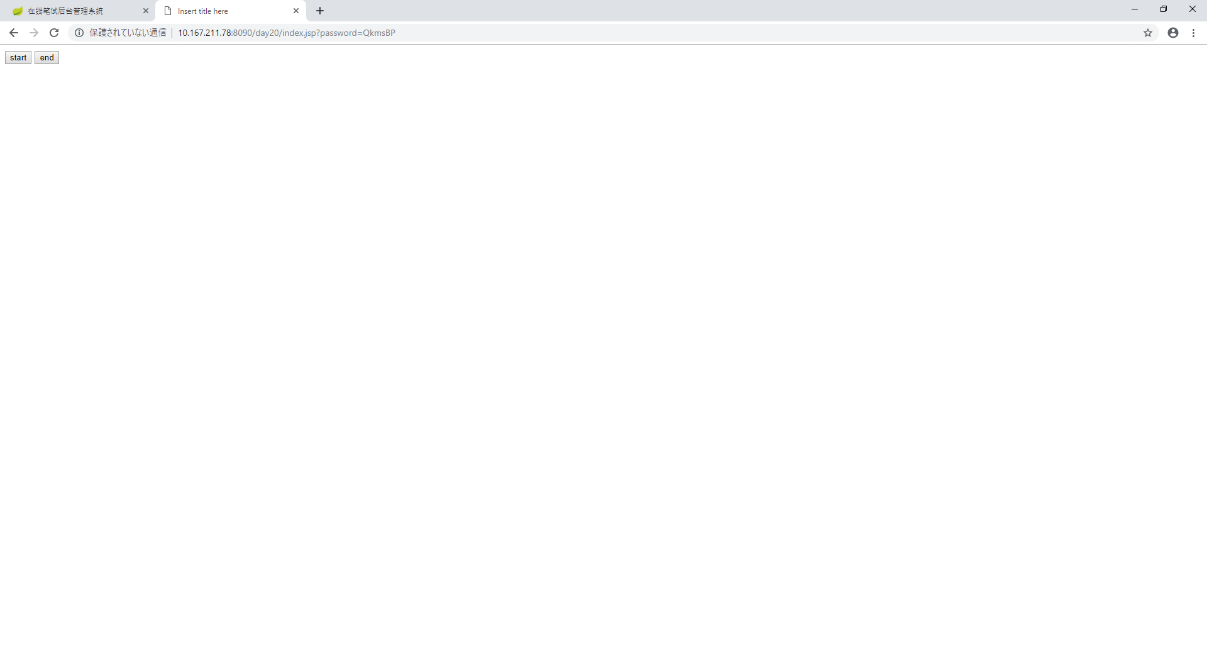


图 2-6考生开始界面

考生点击“start”按钮，进入答题界面，答题结束后，回到此界面，点击“end”结束考试和录屏。

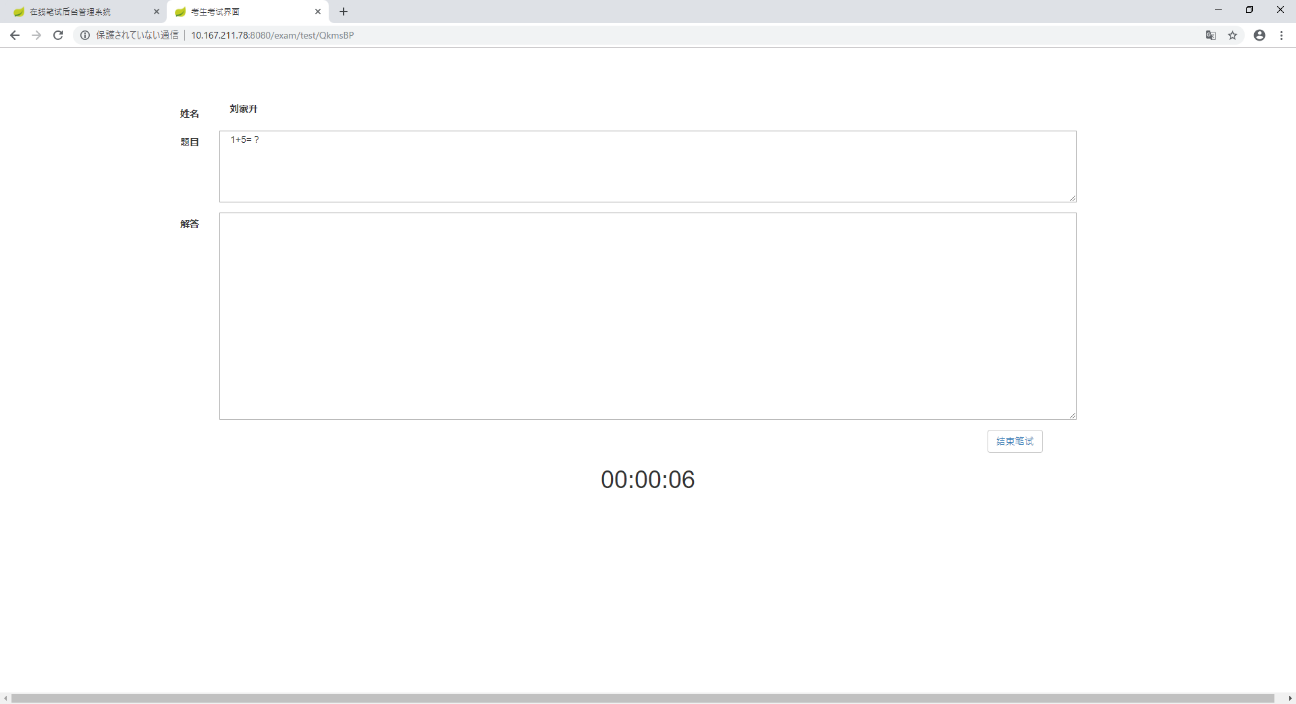


图 2-7考生考试界面

考生进入考试界面，开始考试，页面下方有正计时器，同时录屏开始。考试结束后，点击“结束笔试” 按钮，回到图2-6界面，点击“end”按钮，结束录屏和考试，退出系统。

## **2.2 事件触发流程图**

本模块由事件驱动，由用户点击按钮后触发各种事件的发生，以下将详细介绍各个按钮的功能以及基本流程。

### **登录界面流程图**

-

图2-8 登录流程图

在这个事件触发过程中，功能的实现主要在“点击登录”一部分，在该部分中，使用click(function (){})事件处理函数，将输入框得到的账户与密码与数据库保存的考生信息进行比对，若存在，即可进入考试界面，否则无法登录。

|  |
| --- |
| $(document).ready(function () {  $("#login").click(function () {  var stuName = $("#username").val();  var password = $("#password").val();  $.ajax({  url: "/exam/loginStu",  data: {stuName:stuName,password:password},  success: function (data) {  alert(data)  if(data=="登陆成功"){  window.location.href="http://10.167.211.78:8090/day20/index.jsp?"+"password="+password;  }else{  }  }  });  });  }); |

表2-1 登录验证代码

### **2.2.2 笔试界面一览流程图**

“笔试一览”界面是在线考试系统后端的最关键部分，实现的功能包括：新建笔试、修改笔试、查看考生考试进度以及删除考生信息的功能。以下我们主要介绍一下管理员查看考试进度的具体流程图：



图2-8 管理员后台功能流程图

# **文件处理**

## **3.1数据结构定义**

文件处理模块定义了FileName结构，用来存储文件名链表，并把这个链表传给词法分析作进一步的工作。结构如下所示：

struct FileName

{

char \* data;

FileName \*next;

};

|  |  |
| --- | --- |
| 成员名 | 作用 |
| data | .h文件名 |
| next | 指向下一个FileName类型的结点 |

表3-1 FileName结构中的成员的说明

## **3.2 模块流程图**

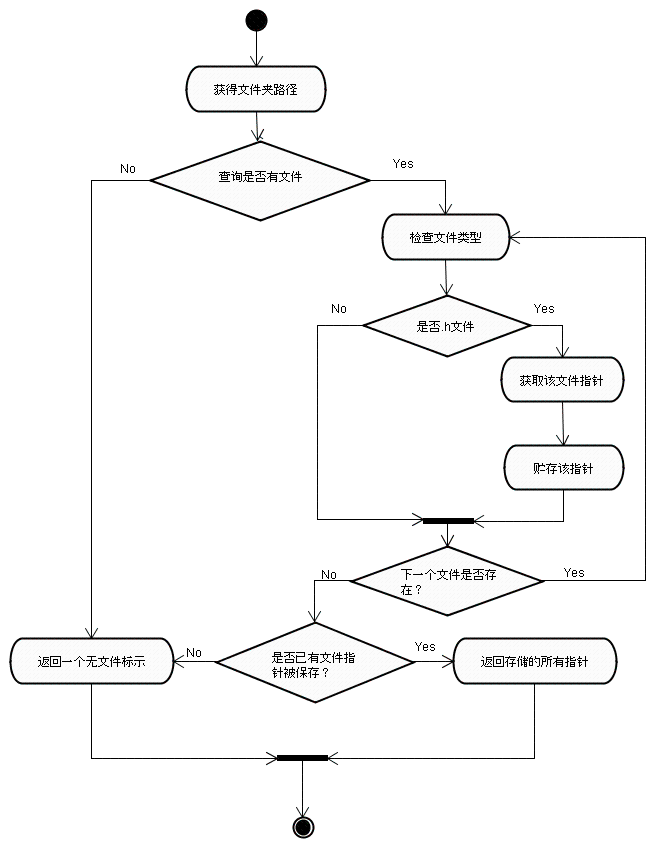


图 3-1 文件处理模块的流程图

## **3.3 外部接口设计**

### **3.3.1 输入**

|  |  |
| --- | --- |
| 函数签名 | int fileMain(string path) |
| 输入参数 | path ---- 用户选择文件夹的绝对路径 |
| 返回值 | 如果操作成功，返回0，如果操作失败，返回-1 |
| 功能 | 文件处理模块的主函数，查找当前目录和子目录下的.h文件 |

表3-2 fileMain函数中的说明

fileMain

directorySearch

fileSearch

图3-2 fileMain函数调用流程图

### **3.3.2 输出**

|  |  |
| --- | --- |
| 成员名 | 作用 |
| FileName \*head | 链表的头指针，存放了.h文件的文件名。作为词法分析的输入 |

表3-3 head头指针的说明

## **3.4 内部函数设计**

### **3.4.1 自定义的函数**

|  |  |
| --- | --- |
| 函数签名 | FileName \* GetNewNode(char \*identifier) |
| 输入参数 | identifier ---- char型指针，指向识别的字符串 |
| 返回值 | 返回一个FileName指针 |
| 功能 | 利用identifier生成一个FileName结点 |

表3-4 GetNewNode函数中的说明

|  |  |
| --- | --- |
| 函数签名 | FileName \* InsertNode(FileName \*head, FileName \*NewNode) |
| 输入参数 | head ---- FileName型指针，以FileName型单链表的头指针  NewNode ---- FileName型指针，表示带插入的结点 |
| 返回值 | 返回一个FileName指针，是经过更新的单链表的头结点 |
| 功能 | 对已有的单链表从尾部插入一个新结点 |

表3-5 InsertNode函数中的说明

|  |  |
| --- | --- |
| 函数签名 | void fileSearch(string filePath) |
| 输入参数 | filePath ---- 目录或子目录的绝对路径 |
| 返回值 | 无 |
| 功能 | 查找filePath目录下的所有.h文件 |

表3-6 fileSearch函数中的说明

|  |  |
| --- | --- |
| 函数签名 | Void directorySearch(string directoryPath,WIN32\_FIND\_DATA FindDirectoryData) |
| 输入参数 | filePath ---- 给定目录下文件的绝对路径  FindDirectoryData ----用来收集查找到的文件或目录的WIN32\_FIND\_DATA结构指针 |
| 返回值 | 无 |
| 功能 | 判断从fileMain传来的文件名是目录还是文件。如果是目录，就调用fileSearch函数查找目录下是否有.h文件 |

表3-7 directorySearch函数中的说明

fileSearch

GetNewNode

InsertNode

.h文件的文件名

返回-1

directorySearch

图3-3 directorySearch函数调用流程图

### **3.4.2 调用的API函数**

调用的API函数如表所示：

|  |  |
| --- | --- |
| 函数签名 | HANDLE FindFirstFile(LPCTSTR lpFileName,LPWIN32\_FIND\_DATA lpFindFileData) |
| 输入参数 | lpFileName ---- 目录的绝对路径  lpFindFileData ----用来收集查找到的文件或目录的WIN32\_FIND\_DATA结构指针 |
| 返回值 | 如果操作成功，返回可以被FindNextFile或FindClose调用的handle，如果操作失败，返回INVALID\_HANDLE\_VALUE |
| 功能 | 查找目录下的第一个文件 |

表3-8 FindFirstFile函数中的说明

|  |  |
| --- | --- |
| 函数签名 | BOOL FindNextFile(HANDLE hFindFile,LPWIN32\_FIND\_DATA lpFindFileData) |
| 输入参数 | hFindFile ---- 调用FindFirstFile返回的handle  lpFindFileData ----用来收集查找到的文件或目录的WIN32\_FIND\_DATA结构指针 |
| 返回值 | 如果操作成功，返回非0值，如果操作失败，返回0 |
| 功能 | 调用了FindFirstFile后用的，功能是接着查找下一个文件 |

表3-9 FindNextFile函数中的说明

|  |  |
| --- | --- |
| 函数签名 | BOOL FindClose(HANDLE hFindFile) |
| 输入参数 | hFindFile ----调用FindFirstFile返回的handle |
| 返回值 | 如果操作成功，返回非0值，如果操作失败，返回0 |
| 功能 | 关闭查找处理，是在查找结束后调用的 |

表3-10 FindClose函数中的说明

# **词法分析**

由于逻辑分析器要对类的关系进行分析，所以词法分析器必须能够对.h文件中的单词进行识别，从而将自定义的类名辨别出来赋予特定标识符，从而为逻辑分析器识别自定义类提供方便。

## **4.1数据结构定义**

### **4.1.1 被识别单词的保存**

如果希望词法分析器能够识别各个单词，实现以下两类功能：首先要对文件中的字符串进行分割；其次要能对分割出来的字符串进行识别，比如识别出它是关键字，操作符，变量还是自定义的结构或类。

因此，可以在词法分析器中定义一个二元组，使之如下所示，前面一个元素记录该字符串，后面的标识符用来标识该字符串的类型：

<单词， 标识符>

因此，可以将二元组定义为如下形式

struct OPSYMSTRUCT

{

char \*name;

int val;

};

struct KWRDSTRUCT

{

char \*name;

int val;

};

其中，OPSYMSTRUCT用来存储操作符的二元组，KWRDSTRUCT用来存储关键字的二元组，由于只是作用不同，其存储本质是相同的，所以它们的结构成员也是相同的，其定义如下：

|  |  |  |
| --- | --- | --- |
| 成员 | 类型 | 定义 |
| name | 字符指针 | 指向识别出的单词 |
| val | 整型 | 单词的标识符 |

表4-1 二元组成员定义

为了操作的方便，需要将二元组串结起来以便于查找更新，可以基于上述的二元组定义一个结构体，以用来将整个文件中识别出的单词连成一条链，方便逻辑分析器的操作。

struct OUTSTRUCT

{

char name[30]；

int val；

OUTSTRUCT\* next；

}；

其中该结构体中的成员定义如下

|  |  |  |
| --- | --- | --- |
| 成员 | 类型 | 定义 |
| name | 字符数组 | 记录单个单词 |
| val | 整型 | 单词的标识符 |
| next | OUTSTRUCT指针 | 指向下一个OUTSTRUCT结点 |

表4-2 OUTSTRUCT结构成员定义

当一个文件中的单词被分析完，就可以生成一条以OUTSTRUCT结构为结点的单链表。后面所有的分析操作，都可以转化为对链表的操作。

### **4.1.2 标识符的定义**

在词法分析的过程中，对单词进行类型识别是非常重要的一部分，因为一个单词有可能是关键字，操作符，自定义的结构或类，还有可能是非法字符。在[4.1.1]定义的结构中，实现了用一个整型数来标示单词的功能，因此需要对用来进行标示的数字进行定义，为了方便起见，可以通过定义常量来间接给单词进行标示。标识符的定义如下：

|  |  |  |
| --- | --- | --- |
| 常量名 | 值 | 说明（所表示的关键字或操作符） |
| \_EOI | 0 | 输入结束的标记 |
| NAME | 1 | 变量 |
| STRING | 2 | 字符串型常量 |
| ICON | 3 | 整型或字符型常量 |
| FCON | 4 | 浮点型常量 |
| PLUS | 5 | + |
| 常量名 | 值 | 说明（所表示的关键字或操作符） |
| MINUS | 6 | － |
| STAR | 7 | ＊ |
| AND | 8 | & |
| QUEST | 9 | ? |
| COLON | 10 | : |
| ANDAND | 11 | && |
| OROR | 12 | || |
| RELOP | 13 | < <= > >= |
| EQUOP | 14 | == != |
| DIVOP | 15 | / |
| OR | 16 | | |
| XOR | 17 | ^ |
| SHIFTOP | 18 | >> << |
| INCOP | 19 | ++ -- |
| UNOP | 20 | ! ~ |
| STRUCTOP | 21 | . -> |
| TYPE | 22 | int,long,等 |
| ETC | 23 | extern,typedef等 |
| STRUCT | 24 | struct union |
| RETURN | 25 | return |
| GOTO | 26 | goto |
| IF | 27 | if |
| ELSE | 28 | else |
| SWITCH | 29 | swich |
| BREAK | 30 | break |
| CONTINUE | 31 | continue |
| WHILE | 32 | while |
| DO | 33 | do |
| FOR | 34 | for |
| DEFAULT | 35 | default |
| CASE | 36 | case |
| SIZEOF | 37 | sizeof |
| LP | 38 | ( |
| RP | 39 | ) |
| 常量名 | 值 | 说明（所表示的关键字或操作符） |
| LC | 40 | { |
| RC | 41 | } |
| LB | 42 | [ |
| RB | 43 | ] |
| COMMA | 44 | ， |
| SEMI | 45 | ； |
| SQUOMRK | 46 | ＇ |
| DQUOMRK | 47 | ＂ |
| EQUAL | 48 | ＝ |
| ASSIGNOP | 49 | ＋＝，－＝等 |
| NUMBER | 50 | 数字常量 |
| JINGHAO | 51 | ＃ |
| INCLUDE | 52 | include |
| DEFINE | 53 | define |
| IFDEF | 54 | ifdef |
| IFNDEF | 55 | ifndef |
| ENDIF | 56 | endif |
| CONTROL | 57 | public,private,protected |
| FRIEND | 58 | friend |
| CLASS | 59 | class |
| CNAME | 60 | 自定义的类名 |
| STATIC | 61 | static |
| VIRTUAL | 62 | virtual |
| WHITE | 100 | \t,\n等非输出字符 |
| ERRORCHAR | －1 | 错误字符 |

表4-3 标识符的定义

### **4.1.3 输出定义**

由于词法分析器要对一个文件进行分析,从而输出一个如[4.1.1]中所述的二元组单链表.将该模块的输出结构定义为一个包含OUTSTRUCT类型单链表的结构体。这样，当有多个文件输入时，就可以得到一个以二元组链表为结点的单链表，因而输出结点的定义如下：

struct OUTPUT

{

OUTSTRUCT\* link;

int cCount;

OUTPUT\* next;

};

|  |  |  |
| --- | --- | --- |
| 成员 | 类型 | 定义 |
| link | OUTSTRUCT指针 | 一个以OUTSTRUCT结构为结点的单链表的头结点 |
| cCount | 整型 | 记录一条链中自定义类名的个数[[1]](#footnote-1) |
| next | OUTPUT指针 | 指向下一个OUTPUT结点 |

表4-4 OUTPUT结构成员定义

## **4.2 模块流程图**

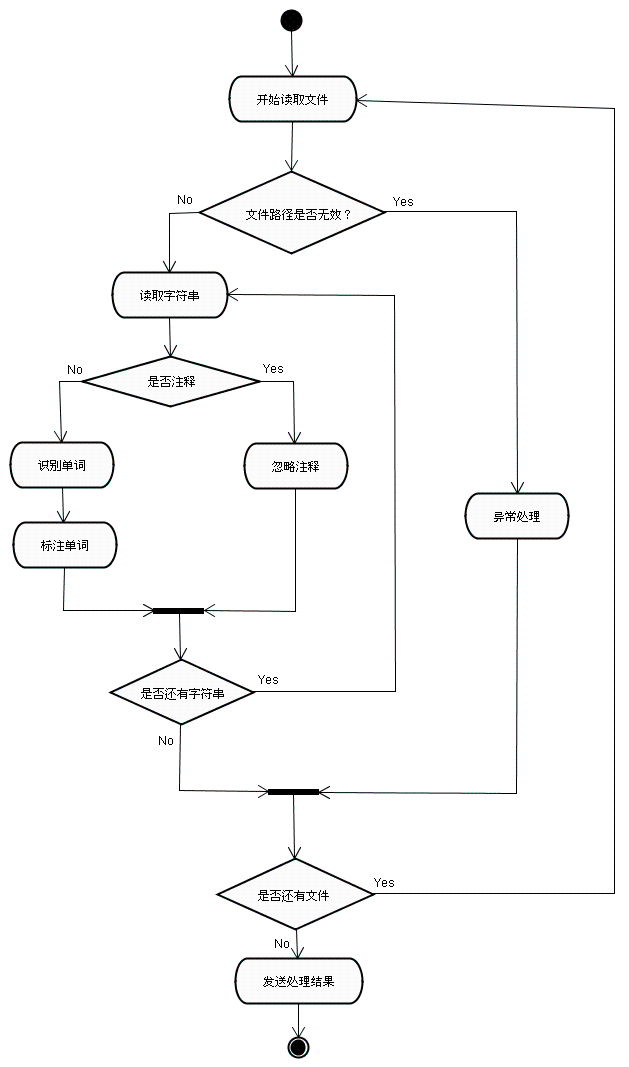


图4-1 词法分析流程图

## **4.3 内部函数设计**

### **4.3.1 操作符识别**

下面的几个函数实现了对文件中操作符的识别：

|  |  |
| --- | --- |
| 函数签名 | int opsym(char\*lx) |
| 输入参数 | lx ---- char型指针，指向识别出的特殊字符串 |
| 返回值 | 返回一个int值，对应的是lx所代表的操作符的标识值 |
| 功能 | 识别传入函数的特殊字符串，并将其对应的标识符返回出来 |

表4-5 opsym函数设计

|  |  |
| --- | --- |
| 函数签名 | OPSYMSTRUCT \*opsrch ( OPSYMSTRUCT \*a, OPSYMSTRUCT \*tab, int number) |
| 输入参数 | a ---- OPSYMSTRUCT型指针，指向包含操作符的二元组结构  tab ---- 操作符的参照表，存储了所有操作符所对应的标识值  number ---- 参照表的元素个数 |
| 返回值 | 返回一个OPSYMSTRUCT指针，得到操作符的标识值后填入a指向的结构实例，并返回该实例 |
| 功能 | 将二元组内包含的操作符的对应标示值求出，并填入二元组 |
| 说明 | 函数opsrch由函数opsym调用来具体实现操作符的识别。 |

表4-6 opsrch函数设计

在使用函数opsrch之前，需要预先创建一张表tab来作为识别标识符的参照，而该表实质上就是一个OPSYMSTRUCT的结构数组，因此，对表tab可以做如下定义：

OPSYMSTRUCT OPtab[]= //set operational symbol table

{

{ "+" , PLUS },

{ "-" , MINUS },

{ "\*" , STAR },

{ "&" , AND },

{ "?" , QUEST },

{ ":" , COLON },

{ "&&" , ANDAND },

{ "||" , OROR },

{ "<" , RELOP },

{ "<=" , RELOP },

{ ">" , RELOP },

{ ">=" , RELOP },

{ "==" , EQUOP },

{ "!=" , EQUOP },

{ "/" , DIVOP },

{ "%" , DIVOP },

{ "|" , OR },

{ "^" , XOR },

{ ">>" , SHIFTOP },

{ "<<" , SHIFTOP },

{ "++" , INCOP },

{ "--" , INCOP },

{ "!" , UNOP },

{ "~" , UNOP },

{ "." , STRUCTOP },

{ "->" , STRUCTOP },

{ "(" , LP },

{ ")" , RP },

{ "{" , LC },

{ "}" , RC },

{ "[" , LB },

{ "]" , RB },

{ "," , COMMA },

{ "\'" , SQUOMRK },

{ "\"" , DQUOMRK },

{ ";" , SEMI },

{ "=" , EQUAL },

{ "#" , JINGHAO },

{ "+=" , ASSIGNOP },

{ "-=" , ASSIGNOP },

{ "/=" , ASSIGNOP },

{ "\*=" , ASSIGNOP }

};

|  |  |
| --- | --- |
| 函数签名 | bool is\_op\_not(char \*a) |
| 输入参数 | a ---- char型指针，指向识别出的特殊字符串 |
| 返回值 | 返回一个bool值.为真时,表明a是一个操作符;为假时,则不是 |
| 功能 | 判断传入的字符串是否是一个操作符 |

表4-7 is\_op\_not函数设计

因此，对操作符识别功能的实现可以通过以下函数调用关系来表示：

opsym

opsrch

Is\_op\_not

其他动作

图4-2 操作符识别函数调用关系

### **4.3.2 单词识别**

|  |  |
| --- | --- |
| 函数签名 | int id\_or\_keyword(char\*lx) |
| 输入参数 | lx ---- char型指针，指向识别的字符串 |
| 返回值 | 返回一个int值，代表了输入单词所对应的标识符 |
| 功能 | 对单词进行检验，以判明它是个关键字还是个变量，并将对应的标识符值返回 |

表4-8 id\_or\_keyword函数设计

|  |  |
| --- | --- |
| 函数签名 | KWRDSTRUCT \* kwsrch (KWRDSTRUCT \*a, KWRDSTRUCT \*tab, int number) |
| 输入参数 | a ---- KWRDSTRUCT型指针，指向识别的字符串  tab ---- KWRDSTRUCT型指针，指向一个操作符的参照表  number ---- int值，是参照表中元素的个数 |
| 返回值 | 返回一个KWRDSTRUCT指针，指向参照表中的一个元素 |
| 功能 | 对单词进行检验，以判明它是个关键字还是个变量，返回关于该字符串的二元组 |
| 说明 | 该函数在id\_or\_keyword函数判断出字符串是关键字时调用 |

表4-9 kwsrch函数设计

在使用函数kwsrch之前，需要预先创建一张表tab来作为识别标识符的参照，从而对已识别单词的类型进行识别，而该表实质上就是一个KWRDSTRUCT的结构数组，因此，对表tab可以做如下定义：

KWRDSTRUCT Ktab[]= //set keyword table

{

{ "auto" , ETC },

{ "break" , BREAK },

{ "case" , CASE },

{ "char" , TYPE },

{ "class" , CLASS },

{ "continue" , CONTINUE },

{ "default" , DEFAULT },

{ "do" , DO },

{ "double" , TYPE },

{ "else" , ELSE },

{ "extern" , ETC },

{ "float" , TYPE },

{ "for" , FOR },

{ "friend" , FRIEND },

{ "goto" , GOTO },

{ "if" , IF },

{ "int" , TYPE },

{ "long" , TYPE },

{ "private" , CONTROL },

{ "protected" , CONTROL },

{ "public" , CONTROL },

{ "register" , ETC },

{ "return" , RETURN },

{ "short" , TYPE },

{ "sizeof" , SIZEOF },

{ "static" , STATIC },

{ "struct" , STRUCT }

{ "switch" , SWITCH },

{ "typedef" , ETC },

{ "union" , STRUCT },

{ "unsigned" , TYPE },

{ "virtual" , VIRTUAL },

{ "void" , TYPE },

{ "while" , WHILE },

{ "include" , INCLUDE },

{ "define" , DEFINE },

{ "ifdef" , IFDEF },

{ "ifndef" , IFNDEF },

{ "endif" , ENDIF }

};

该部分函数的调用关系如下：

kwsrch

Id\_or\_keyword

图4-3 单词识别函数调用关系

### **4.3.3 单链表生成**

|  |  |
| --- | --- |
| 函数签名 | OUTSTRUCT \* GetNewNode(char \*identifier, int intID) |
| 输入参数 | identifier ---- char型指针，指向识别的字符串  intID ---- int值，是identifier对应的标识符 |
| 返回值 | 返回一个OUTSTRUCT指针 |
| 功能 | 利用identifier和intID生成一个OUTSTRUCT结点 |

表4-10 GetNewNode函数设计

|  |  |
| --- | --- |
| 函数签名 | OUTSTRUCT \* InsertNode (OUTSTRUCT \*head, OUTSTRUCT \*NewNode) |
| 输入参数 | head ---- OUTSTRUCT型指针，以OUTSTRUCT型单链表的头指针  NewNode ---- OUTSTRUCT型指针，表示带插入的结点 |
| 返回值 | 返回一个OUTSTRUCT指针，是经过更新的单链表的头结点 |
| 功能 | 对已有的单链表从尾部插入一个新结点 |

表4-11 InsertNode函数设计

### **4.3.4 分析文件**

这个功能点只要求对一个文件做词法分析，将识别出的单词和操作符生成一个单链表，因此，可以按如下设计这里的函数

|  |  |
| --- | --- |
| 函数签名 | OUTSTRUCT \* scanner(FILE \*fp) |
| 输入参数 | fp ---- FILE型指针，指向待分析的文件 |
| 返回值 | 返回一个OUTSTRUCT指针，是经过分析后形成单链表的头结点 |
| 功能 | 对一个文件进行此法分析，并生成一条包含了单词和标识符的单链表 |

表4-12 scanner函数设计

## **4.4 外部接口设计**

|  |  |
| --- | --- |
| 函数签名 | OUTPUT \* FilesAnalysis(FileName \*nameLink) |
| 输入参数 | nameLink ---- FileName型指针，传入一个包含所需要的文件路径的单链表 |
| 返回值 | 返回一个OUTPUT指针，对应每个文件生成一条OUTSTRUCT链表，并将这些链表作为结点生成一条单链表 |
| 功能 | 对输入的多个指定文件进行分析，生成多条<单词,标识符>链 |

表4-13 外部接口设计

# **逻辑分析**

对词法分析器的输出结果（二元组<单词，标识符>形成的链----单词链），进行类以及类关系的识别。识别的依据是类定义中的特殊关键字及界限符（如“{” “}”）。

本模块是通过两次遍历词法分析器输出的结果---单词链来实现的。第一次遍历实现从单词链中，过滤出所有的类名并将所有的类名形成一条链表即类链。单词按不同含义划分成不同的类别（变量统一为一类，运算符为一类等等），不同的类别使用不同的标识符进行标识。由于单词链中单词为类名的单词并未使用统一标识符标识。在单词链中，跟在关键字“class”后面的类名的标识符是CNAME，其他情况下，类名的标识符与变量的标识符是一致的。这将不便使用通用的方法进行类关系的识别。所以在进行第二次遍历前，必须将单词链中的所有类名的标识符转换成CNAME。然后将类链和单词链一并传送到第二次遍历模块中，进行类关系的识别。最后，将生成的链释放。

## **5.1数据结构定义**

以下结构是保存整个程序中出现的类的名称。

struct CLASSNAME

{

char \*cname；

CLASSNAME \* next；

}；

其中该结构体中的成员定义如下：

|  |  |
| --- | --- |
| 成员 | 定义 |
| cname | 类的名称 |
| next | 指向下一个结点的指针 |

表5-1 CLASSNAME结构中的成员的说明

以下结构是利用C++中的类模板vector生成二维矩阵，二维矩阵用于存放类之间的关系。由于预先不知道将出现类的个数（不能确定数组的大小），所以不能使用普通数组结构来存取类之间的关系。二维矩阵行和列都是代表类名（行代表被解析的类，列代表与被解析类有可能产生关系的所有类），一行代表一个类名，一列也代表一个类名，行/列类名的含义与类链中各结点的类名的含义是一致的。利用上述一一对应关系，就可以通过一个映射函数，返回一个确定的类名在类链中的次序。两个类就可以形成一个次序对，从而定位关系矩阵中的一个单元。该单元可以存放该单元行号所代表类和列号所代表的类之间的关系。

vector< vector<int> >Row;

vector<int>Column;

假设类链如下：

AAA

NULL

DDD

CCC

BBB

关系矩阵如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类名 | AAA | BBB | CCC | DDD |
| AAA | 0 | 0 | 0 | 0 |
| BBB | 1 | 0 | 2 | 0 |
| CCC | 1 | 3 | 0 | 4 |
| DDD | 0 | 0 | 0 | 0 |

表5-2 类关系的描述

其中，各个数字的表达意义如下：

0——类间无关系　　　　1——列类继承于行类

2——列类组合行类　　　3——列类和行类是友元类

4——列类嵌套于行类

list<int>CnStack;

使用类模板list来实现堆栈的作用，用来存放待解析的类。

list<char>MarkStack;

使用类模板list来实现堆栈的作用，用来存放大括号。

## **5.2 模块流程图**

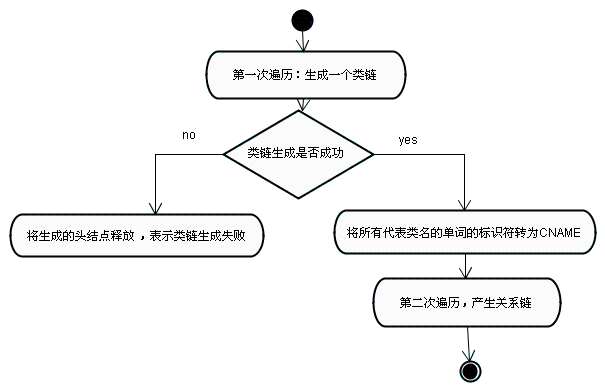


图 5-1 模块主流程

## **5.3 外部接口设计**

|  |  |
| --- | --- |
| 函数签名 | int LogicAnalysis(OUTSTRUCT \* wordLink, Classname \*\* classLink, Relation \*\* rLink) |
| 输入参数 | wordLink ---- OUTSTRUCT型指针，词法分析器输出的结果，单词链的指针  classLink ---- Classname \*\*类型，用于返回类链的指针  rLink ---- Relation \*\*类型，用于返回关系链的指针 |
| 返回值 | 返回一个int型变量，代表类链中类的个数 |
| 功能 | 完成类及类链的识别 |

表5-3 LogicAnalysis函数的说明

## **5.****4 内部函数设计**

**5．4．1 类链的生成**

该函数是用来生成一个类链，它将调用match函数进行类名与类链中的类名进行匹配。

|  |  |
| --- | --- |
| 函数签名 | int GetClassLink(OUTSTRUCT \*SData,Classname \*\*cname) |
| 输入参数 | Sdata ---- OUTSTRUCT型指针，单词链的指针  cname ---- Classname类型，用于返回类链的指针 |
| 返回值 | 返回一个int型变量，代表类链的个数 |
| 功能 | 识别类，并将类连成一条链 |

表5-4 GetClassLink函数的说明



图 5-2 类链生成算法

**5．4．2 类关系的识别**

该函数是用于识别类关系。

|  |  |
| --- | --- |
| 函数签名 | Relation \* SecTraversal(int num, OUTSTRUCT \*SourceString,Classname \*cname) |
| 输入参数 | num ---- int型变量，类的个数  SourceString ---- OUTSTRUCT型指针，单词链的指针  cname ----类链的指针 |
| 返回值 | 返回一个Relation型的指针变量，即关系链 |
| 功能 | 识别类之间的关系并将关系矩阵转换成关系链返回 |

表5-5 SecTraversal函数的说明

在识别类关系中，只对类名和特殊的符号进行相应得处理， 处理的依据是词法分析器的结果<单词，标识符> 中的标识符。

首先，判断类链是否识别完成，如果没有，则读取一个类链中的一个结点，根据结点中的标示符的值，对应表5-6进行相应的处理。处理完一个结点后，继续处理下一个结点，直到所有结点处理完成。最后将本次处理结果和类链传给界面模块。

|  |  |
| --- | --- |
| 标识符 | 处理方法 |
| CLASS | 如果友元标识符为false,将其后面出现的类名对应类链中的次序号（通过MAP函数获得）压栈，表明一个类的解析的开始,否则表明calss是申明友元的一部分，不做任何操作。 |
| COLON | 将类关系标识符设为继承关系，表明以后出现的类名是被解析类的父类。 |
| LC | 左大括号入栈，以大括号来区分可能出现的不同类层次，并判断括号栈和类栈数目是否一样多，如果不一样多，则表明类定义中出现了逻辑块而非嵌套类，需在类栈中添加一个特殊的值，以作标识。并将关系标识符设为组合关系。 |
| RC | 栈顶的左大括号出栈，栈顶的类（或是前面所定义的特殊值）出栈，如果栈未出空，而且刚出栈的不是一个特殊值，则表明出现了类的嵌套。 |
| FRIEND | 表明以后出现的类是友元类。 |

表5-6 类关系识别的依据及方法

该函数是一个转换函数，存在于第一次遍历和第二次遍历之间。是第二次遍历的前提。

|  |  |
| --- | --- |
| 函数签名 | void ConvertVar2Class (OUTSTRUCT \* wordLink, Classname \* classLink) |
| 输入参数 | wordLink ---- OUTSTRUCT型的指针，词法分析器的结果  classLink ---- 类链的指针名 |
| 返回值 | 无 |
| 功能 | 将词法分析器的分析结果中的二元组中，类别为类名的标示符改成CNAME |

表5-7 ConvertVar2Class函数的说明

|  |  |
| --- | --- |
| 函数签名 | Match(char \* name, CLASSNAME \*classname) |
| 输入参数 | name ---- 待匹配的类名  classname ---- 类链的指针 |
| 返回值 | 返回一个int型变量，有匹配返回1，没有返回0 |
| 功能 | 字符串匹配 |

表5-8 Match函数的说明

|  |  |
| --- | --- |
| 函数签名 | Map(char \*name，CLASSNAME \*classname) |
| 输入参数 | name ---- 待定位的类名  classname ---- 类链的指针 |
| 返回值 | 返回一个 int 型变量，表示类在类链中的位置 |
| 功能 | 返回指定的类在类链中的位置，方便填写类关系表 |

表5-9 Map函数的说明

|  |  |
| --- | --- |
| 函数签名 | void InsertClass(Classname \* newNode, Classname \*\* classLink) |
| 输入参数 | newNode ---- 待插入类链中的新类名  classLink ---- 类链的指针名 |
| 返回值 | 无 |
| 功能 | 在类链中插入新类 |

表5-10 InsertClass函数的说明

|  |  |
| --- | --- |
| 函数签名 | int CompareName (char \* varName, Classname \* classLink) |
| 输入参数 | varName ---- char型的指针,待判断的类名  classLink ---- 类链的指针名 |
| 返回值 | 返回一个int变量，1代表类名存在于类链中，0代表类名不存在类链中 |
| 功能 | 判断指定的类名是否存在于类链中 |

表5-11 CompareName函数的说明

|  |  |
| --- | --- |
| 函数签名 | void ReleaseClassLink(Classname \*\* cLink) |
| 输入参数 | cLink ---- Classname型的指针，待释放类链的指针名 |
| 返回值 | 无 |
| 功能 | 将类链释放 |

表5-12 ReleaseClassLink函数的说明

|  |  |
| --- | --- |
| 函数签名 | void ReleaseRelationLink(Relation \*\* rLink) |
| 输入参数 | rLink ---- Relation型的指针，待释放关系链的指针名 |
| 返回值 | 无 |
| 功能 | 将关系链释放 |

表5-13 ReleaseRelationLink函数的说明

1. 记录链中自定义类名的个数以便于逻辑分析器的识别和分析操作，可以直接读取类的个数从而减轻了对类名识别的依赖 [↑](#footnote-ref-1)