**金融软件工程——程序编码**

161278039 薛岚天

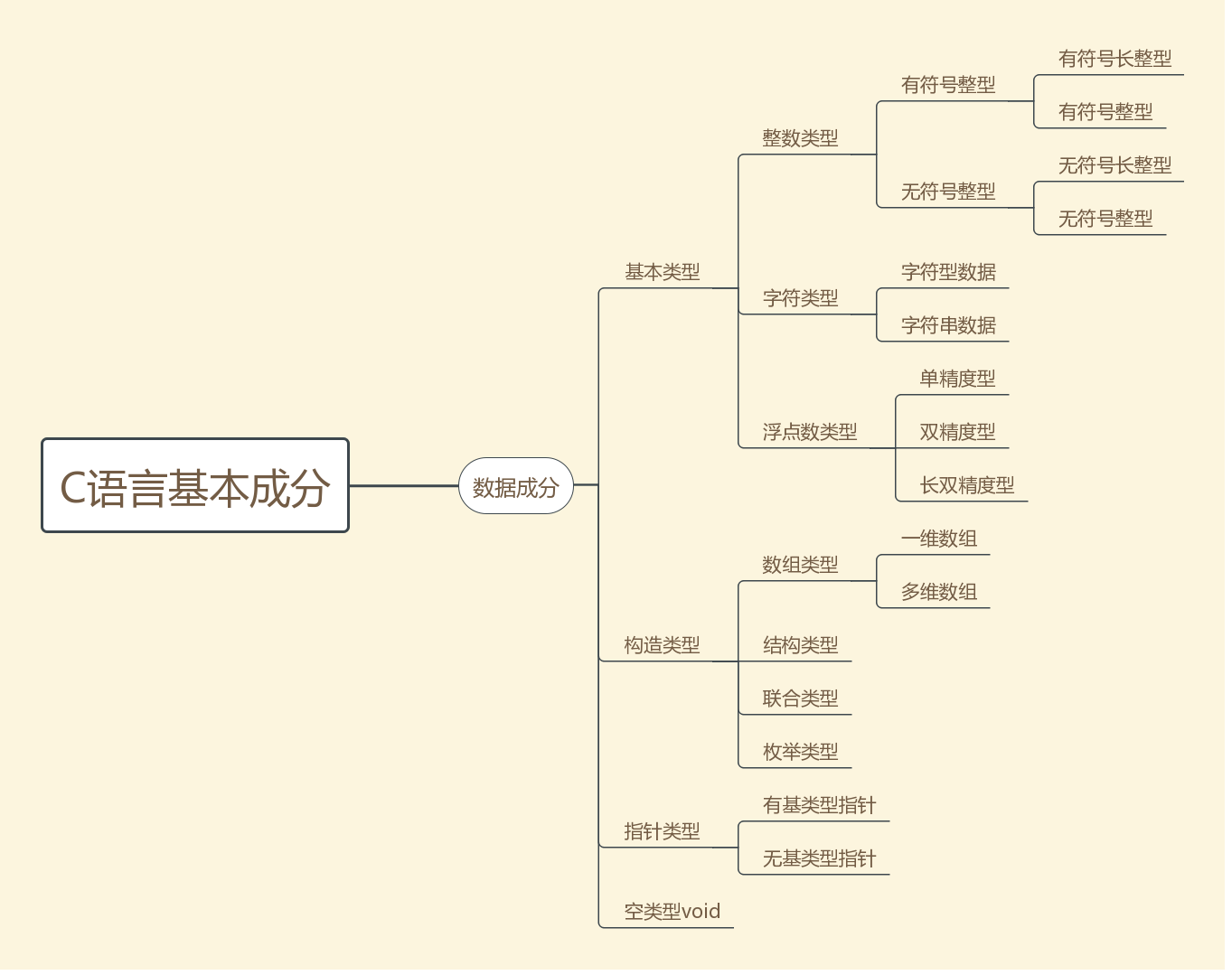
*题目：对照本章内容，对自己熟悉的一门程序设计语言，尝试分析和总结其基本成分和语言特性，要求实验对象为包含输入输出、数据运算、注释的程序，长度不小于500行。*

由于本人较为熟悉的程序设计语言为C语言，下面将以其为例分析基本成分、语言特性，之后参考其他人的C语言项目分析其注释风格；

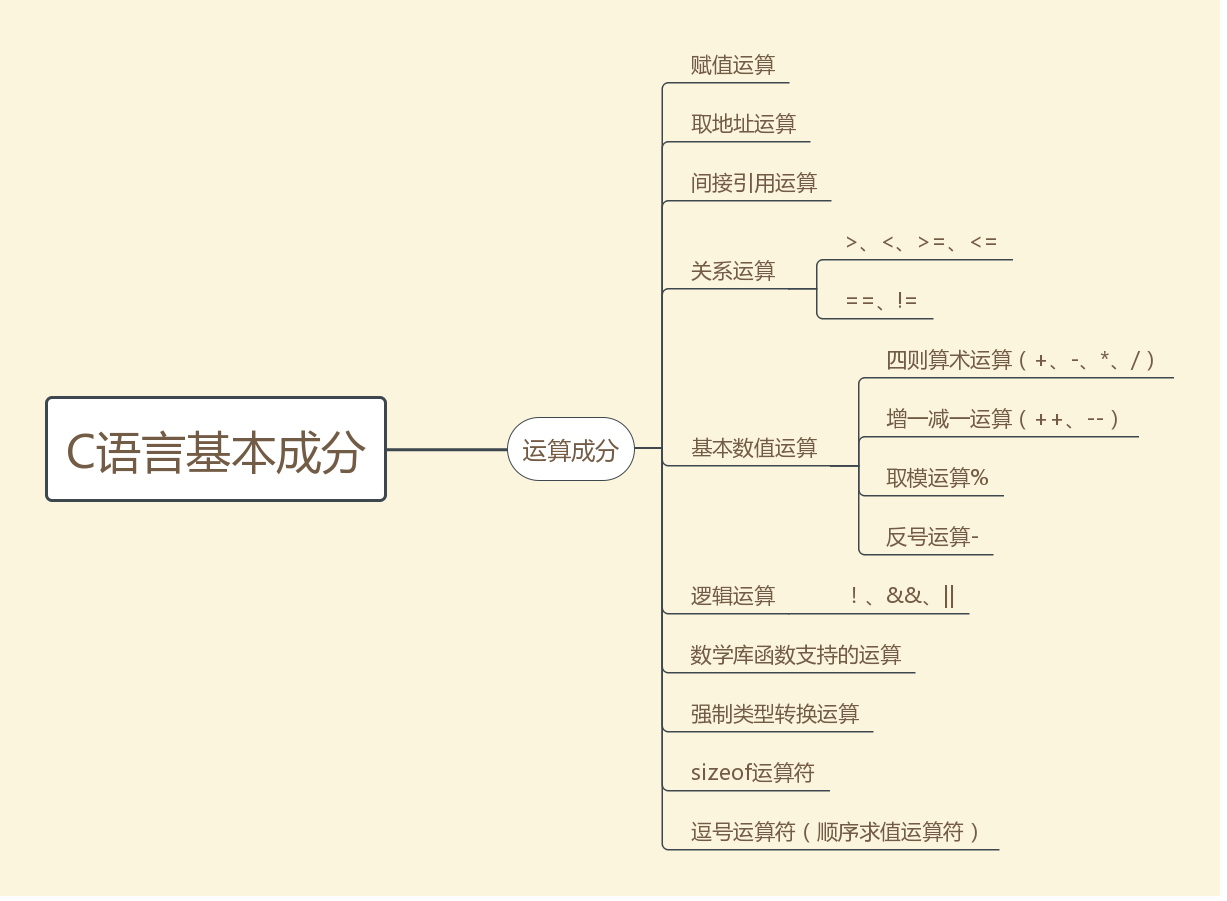
1. C语言基本成分

将C语言数据成分、运算成分、控制成分、传输成分总结如下。

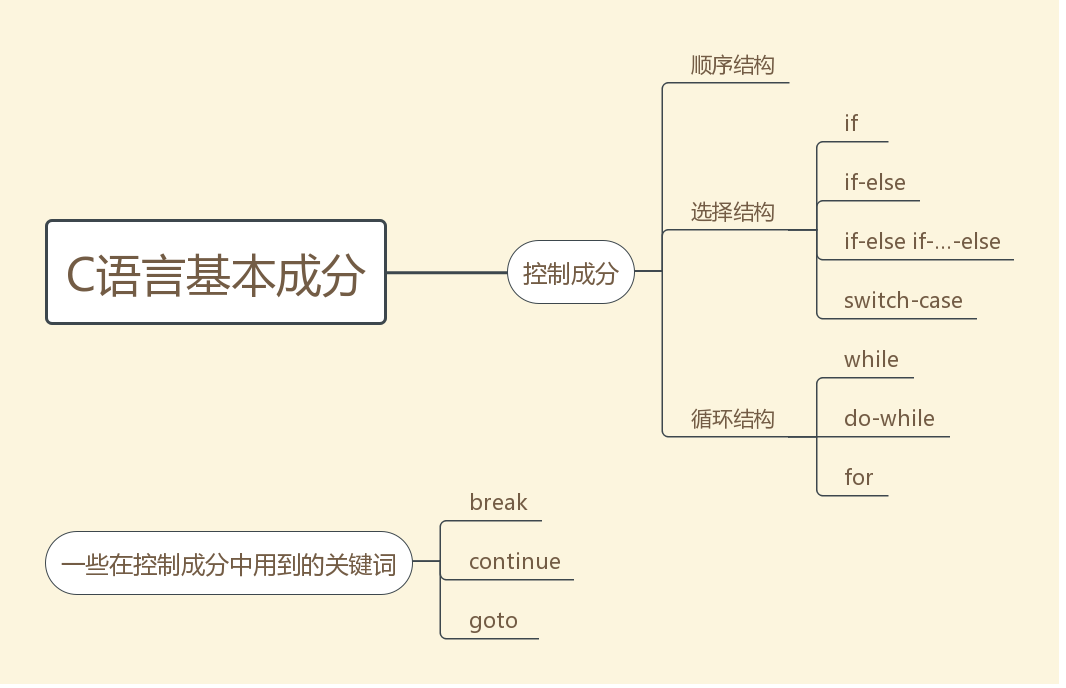
数据成分：



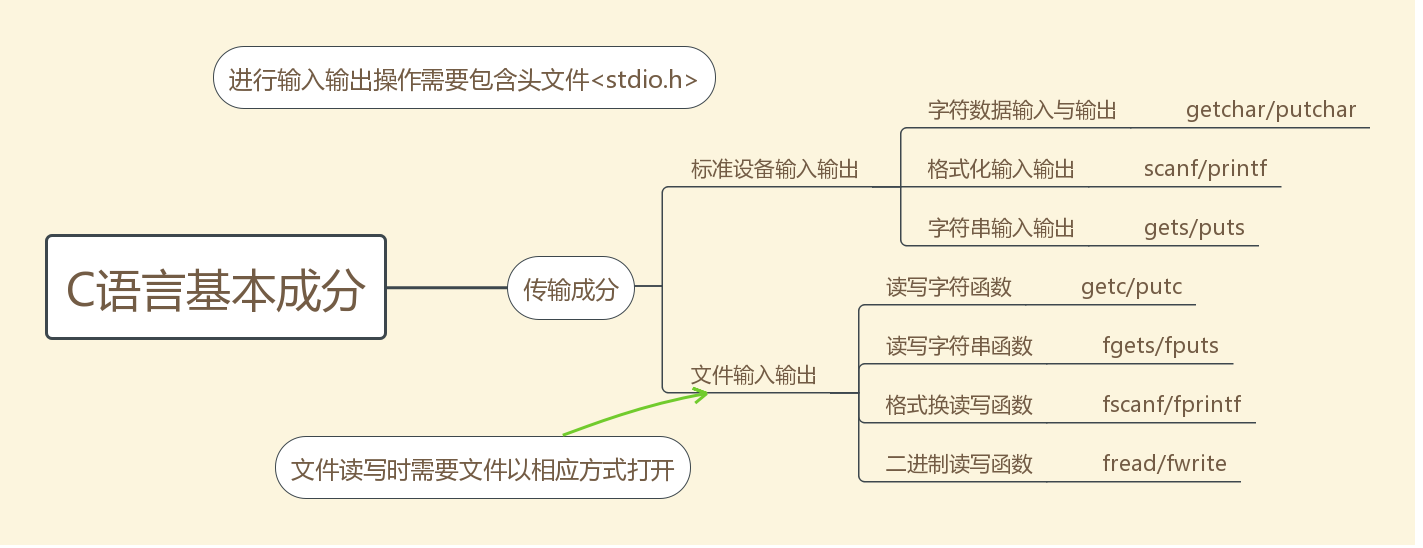
运算成分：



控制成分：



传输成分：



1. C语言语言特性
2. **心理特性**
3. 紧致性高。程序员需记忆的与编码有关的信息总量相对不大。其一共只有32个关键字，9种控制语句，简洁紧凑，灵活方便。
4. 局部性高。C语言是结构式语言，程序的各个部分除必要的信息交流外彼此独立。C语言中提供很多方便的库函数，并具有多种循环、条件语句可控制程序流向。
5. 不影响人们学习新语种的积极性，相反，C语言可以为学习其他语言打下良好基础。
6. **工程特性**
7. 将设计转换为代码的便利程度高。C语言支持结构化部件，同时具有丰富 的数据类型及运算符，便于设计到代码的转换。
8. 编译器效率高。C语言可以直接在机器上运行，其既具有高级语言的特点，又具有汇编语言的特点，程序执行效率高，一般只比汇编程序生成的目标代码低10%~20%。
9. 配套开发工具丰富。C语言具有众多开发工具，如VC,VS,DEV,eclipse等等，这些开发工具的功能都较为完备优秀。
10. 源代码可移植性好。由于标准的存在，C代码可以用于多种操作系统，如windows、DOS、UNIX等，也适用于多种机型，代码移植时只需做较少修改或不做修改。
11. 可维护性较好。C语言提供注释手段，使得用户在编写代码时可以做相应的备注与解释，这有助于提高源程序的可读性。
12. **应用特性**
13. 底层开发，如开发操作系统等。这是由于C语言简洁紧凑，运行效率较高。
14. 嵌入式系统开发。嵌入式系统开发一般都具有相应的资源限制，实时性要求也较高，因此选用C语言开发可以较好地满足这些要求。
15. 与硬件相关的开发。C语言相较其他语言更贴近编译语言，因此能较好地保证一旦编译完成后许多特性可以保持不变，最大程度不依赖随机因素与编译器。
16. 单片机开发。C语言开发的程序具有坚固性，能较好避免单片机运行中间非异步的破坏，同时C语言可移植性较强，编译效率也较高，这使得其较适合单片机的开发场合。

3. 经典C语言项目开源项目分析

本次选择分析的项目一为**Webbench。**这是是一个在linux下使用的非常简单的网站压测工具。它使用fork()模拟多个客户端同时访问我们设定的URL，测试网站在压力下工作的性能，最多可以模拟3万个并发连接去测试网站的负载能力。代码行数总共约600行。

本项目主要由两个C文件构成，其中webbbench.c作为主要文件，socket.c作为支撑接口文件。下面逐一分析这两个文件中的注释情况。对每项注释，也根据其所在位置做出了相应的功能性阐述。

|  |  |  |
| --- | --- | --- |
| **程序名** | **来源** | **程序语言** |
| Webbench.c | Github | C |
| **序言性注释** | | |
| （1-17行） | | |
| |  | | --- | | /\* | |  | \* (C) Radim Kolar 1997-2004 | |  | \* This is free software, see GNU Public License version 2 for | |  | \* details. | |  | \* | |  | \* Simple forking WWW Server benchmark: | |  | \* | |  | \* Usage: | |  | \* webbench --help | |  | \* | |  | \* Return codes: | |  | \* 0 - sucess | |  | \* 1 - benchmark failed (server is not on-line) | |  | \* 2 - bad param | |  | \* 3 - internal error, fork failed | |  | \* | |  | \*/ |   这里的注释申明了程序使用的语言—c语言、作者及程序编写时间，说明程序的主体内容和用途—用fork方法测试网站的抗压能力，同时阐明程序运行后的一些返回值的含义。 | | |
| **功能性注释** | | |
| 28行：/\* values \*/  其下的代码块定义了一些将使用的变量 | | |
| 34行：/\* gloabals \*/  说明之后定义的变量为全局变量 | | |
| 35行：/\* 0 - http/0.9, 1 - http/1.0, 2 - http/1.1 \*/  说明不同HTTP协议版本号 | | |
| 36行：/\* Allow: GET, HEAD, OPTIONS, TRACE \*/  允许服务器连接的几个方法 | | |
| 50行：/\* internal \*/  说明之后定义的变量为内部变量 | | |
| 75行：/\* prototypes \*/  说明其后定义的函数将作为蓝本使用 | | |
| 166行：/\* Copyright \*/  说明之下输出的内容为申明版权作用 | | |
| 173行：//print request info, do it in function build\_request | | |
| 260行：/\* protocol/host delimiter \*/  此处开始程序主要操作对象分界 | | |
| 270行：/\*get port from hostname \*/  说明其下程序的作用为从主机处获得接口号 | | |
| 324行：/\* vraci system rc error kod \*/  说明其下函数的作用为创建管道和子进程，对http请求进行测试 | | |
| 331行：/\* check avaibility of target server \*/  说明其后尝试为查看是否有代理服务器 | | |
| 339行：/\* create pipe \*/  说明其后工作是开始创建管道 | | |
| 354行：/\*fork childs \*/  说明其后工作为开始创建子进程 | | |
| 360行：/\* child process or error \*/  说明如果创建子进程过于缓慢或出现错误进入该分支 | | |
| 361行: /\*make childs faster \*/  说明之前操作的作用是加快创建子进程进程 | | |
| 375行：/\* I am a child \*/  说明其下操作建立在成功创建的子进程上 | | |
| 381行：/\* write results to pipe \*/  说明其后操作为将结果写入管道 | | |
| 445行：/\* setup alarm signal handler \*/  说明其下操作的作用 | | |
| 451行：//after benchtime, then exit  承接之前setup的代码块，当到达指定时间后，退出进程 | | |
| 473行：/\* read all available data from socket \*/  解释之下循环的作用是从接口中读取所有信息 | | |

在本程序的注释中我们看到，其同时具有序言性注释和功能性注释。序言性注释是对整个软件情况的一个整体概括。而功能性注释一般用于阐述随后模块中程序的功能。在每个具有不同功能的模块开头处，编写者都添加了相应的注解性注释，同时，在模块内部一些较难理解的程序或函数后，编写者也做了相应注释。除这两种注释外，程序中还具有编写者注释掉的一些代码块，应该是调试过程中尝试的不同方法，便于以后代码的修改和维护。

|  |  |  |
| --- | --- | --- |
| **程序名** | **程序来源** | **程序语言** |
| Socket.c | github | C |
| **序言性注释** | | |
| 1-14行 | | |
| /\* $Id: socket.c 1.1 1995/01/01 07:11:14 cthuang Exp $   |  |  | | --- | --- | |  | \* This module has been modified by Radim Kolar for OS/2 emx | |  | \*/ | |  |  | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |  | module: socket.c | |  | program: popclient | |  | SCCS ID: @(#)socket.c 1.5 4/1/94 | |  | programmer: Virginia Tech Computing Center | |  | compiler: DEC RISC C compiler (Ultrix 4.1) | |  | environment: DEC Ultrix 4.3 | |  | description: UNIX sockets code. | |  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   在序言性注释中，编写人员说明了本模块的作用、使用的编译器、编写人员及编程环境，这些信息为后续程序的阅读者提供了更多关于代码的信息，有助于后续维护更新 | | |
| **功能性注释** | | |
| 本程序中仅有一个函数，仅60行，作为一个编程接口出现，无功能性注释 | | |

总而言之，该项目的注释风格比较简洁，但也提供了理解代码所需的足够信息。在每个源文件开头都有关于本模块功能及相关信息的较为详细的介绍，对源文件中的子模块开头一般也具有对该模块功能的简单注释，模块中较为复杂的功能性语句，程序也对其作了注解。

从该项目的注释风格中我了解到，在编写项目时，对于负责不同功能的每个源代码文件都应当做出相应注释以方便理解；同时文件内部不同模块间最好用注释+空白行分隔，便于读者对源程序的阅读理解。

选择分析的另一个项目为**tinyhttpd,**它是一个超轻量型Http Server，使用C语言开发，全部代码共500多行。

本项目的主要文件为httpd.c，下面详细分析其注释情况，同样对每项注释，根据其所在位置做出了相应的解读：

|  |  |  |
| --- | --- | --- |
| **程序名** | **程序来源** | **程序语言** |
| http.c | Github | C |
| **序言性注释** | | |
| 1-14行 | | |
| |  |  | | --- | --- | |  | /\* J. David's webserver \*/  /\* This is a simple webserver. | |  | \* Created November 1999 by J. David Blackstone. | |  | \* CSE 4344 (Network concepts), Prof. Zeigler | |  | \* University of Texas at Arlington | |  | \*/ | |  | /\* This program compiles for Sparc Solaris 2.6. | |  | \* To compile for Linux: | |  | \* 1) Comment out the #include <pthread.h> line. | |  | \* 2) Comment out the line that defines the variable newthread. | |  | \* 3) Comment out the two lines that run pthread\_create(). | |  | \* 4) Uncomment the line that runs accept\_request(). | |  | \* 5) Remove -lsocket from the Makefile. | |  | \*/ |   该注释说明了程序的责任人，程序的用途，编写时间以及当在linux系统上运行时一些必要的修改操作。这段注释使读者能对该程序有一些必要的了解，同时也提高了程序的可维护性。 | | |
| **功能性注释** | | |
| 50-54行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* A request has caused a call to accept() on the server port to |  | |  | \* return. Process the request appropriately. |  | |  | \* Parameters: the socket connected to the client \*/ |  | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |  |   说明此下代码块的主要作用 | | |
| 65-66行  /\* become true if server decides this is a CGI program\*/  说明定义的cgi标识变量取非0值的条件 | | |
| 115行  /\*read & discard headers\*/  说明之前表达式的作用，使程序更容易理解 | | |
| 136-139行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |  | /\* Inform the client that a request it has made has a problem. | |  | \* Parameters: client socket \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   注明其后函数的作用为报错通知，指出所用的参数 | | |
| 156-162行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Put the entire contents of a file out on a socket. This function | |  | \* is named after the UNIX "cat" command, because it might have been | |  | \* easier just to do something like pipe, fork, and exec("cat"). | |  | \* Parameters: the client socket descriptor | |  | \* FILE pointer for the file to cat \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   表明下方代码块的作用 ，同时指出函数命名原因，指出函数中所用参数的意义 | | |
| 175-178行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Inform the client that a CGI script could not be executed. | |  | \* Parameter: the client socket descriptor. \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   同样指出下方函数的作用，阐明所用的关键参数 | | |
| 193-197行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Print out an error message with perror() (for system errors; based | |  | \* on value of errno, which indicates system call errors) and exit the | |  | \* program indicating an error. \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   指出下方函数代码块的作用为报错同时退出执行 | | |
| 204-209行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Execute a CGI script. Will need to set environment variables as | |  | \* appropriate. | |  | \* Parameters: client socket descriptor | |  | \* path to the CGI script \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   指出下方代码块的作用，同时指明执行条件，阐明关键参数意义 | | |
| 242行  /\*head or other\*/  阐明进入某一分支的条件 | | |
| 262行  /\* child: CGI script \*/  阐明之前条件标识的情况 | | |
| 278行  /\* post\*/  指明在何种情形下进入这一分支 | | |
| 284 行  /\*parent\*/  同样指明进入某一分支的情况 | | |
| 301-313行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Get a line from a socket, whether the line ends in a newline, | |  | \* carriage return, or a CRLF combination. Terminates the string read | |  | \* with a null character. If no newline indicator is found before the | |  | \* end of the buffer, the string is terminated with a null. If any of | |  | \* the above three line terminators is read, the last character of the | |  | \* string will be a linefeed and the string will be terminated with a | |  | \* null character. | |  | \* Parameters: the socket descriptor | |  | \* the buffer to save the data in | |  | \* the size of the buffer | |  | \* Returns: the number of bytes stored (excluding null) \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   指出下方所用函数块的作用，阐明所用参数的意义，同时指出返回值的具体含义 | | |
| 346-350行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Return the informational HTTP headers about a file. \*/ | |  | /\* Parameters: the socket to print the headers on | |  | \* the name of the file \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   指出下方函数块的作用，阐明所用参数的具体含义 | | |
| 354行  /\* could use filename to determine file type \*/  指出所申明变量的作用 | | |
| 366-368行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Give a client a 404 not found status message. \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   分隔代码块，指出下方函数块的作用 | | |
| 393-399行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Send a regular file to the client. Use headers, and report | |  | \* errors to client if they occur. | |  | \* Parameters: a pointer to a file structure produced from the socket | |  | \* file descriptor | |  | \* the name of the file to serve \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   分隔代码块，指出下方函数块的作用及所用参数意义 | | |
| 421-428行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* This function starts the process of listening for web connections | |  | \* on a specified port. If the port is 0, then dynamically allocate a | |  | \* port and modify the original port variable to reflect the actual | |  | \* port. | |  | \* Parameters: pointer to variable containing the port to connect on | |  | \* Returns: the socket \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   分隔代码块，指出下方函数块的作用，指出参数及返回值含义 | | |
| 460-464行   |  |  | | --- | --- | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ | |  | /\* Inform the client that the requested web method has not been | |  | \* implemented. | |  | \* Parameter: the client socket \*/ | |  | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |   分隔代码块，指出下方函数块的作用，指出函数参数意义 | | |
| 487行  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  分隔主函数与其他函数代码块 | | |

该项目的注释风格较第一个项目而言更加详尽，同时除了第一个项目所具有的特点——都有序言性注释和功能性注释，且序言性注释用于阐明程序的整体用途和责任人及项目相关事项外，该项目对每个函数代码块都使用注释进行分隔，且分隔使用更为引人注意的/\*\*\*\*\*\*\*\*\*\*\*\*/方式，在每个函数前，除了说明其功能外，该项目还会对函数所用的参数及返回值做出一定的解释，使得代码具有更好的可读性。

总结这两个项目，我认为良好的代码注释应当做到以下几点：

1. 兼具序言性注释和功能性注释
2. 序言性注视中应当说明程序的主要作用，有关人员等开发者信息以及给出一切可能对程序运行产生影响的信息。
3. 功能性注释应当为每个程序段做注解，将不同程序段分隔开来，提高程序的可阅读性。
4. 注释不要求很多，但对关键步骤或难以理解的代码段应当提供注释。

4. 对自己项目代码的修改

在进行以上分析后，我对我的项目程序进行了一定的修改。

修改后的项目代码以文件夹形式附上，命名为study。在该文件夹中，我在readme.txt中对项目情况作出了整体说明。在注释方面，在每个代码文件前加上了相应的序言性注释，同一代码文件中的不同功能模块作了分隔和功能性注释，对于相对比较复杂和作了特殊处理的代码 ，在其后也有相应的功能性注释。在程序风格上，调整了部分语句的写法，使整个项目更容易理解。

以其中的一个源文件index.php为例，修改后的代码大致如下，注释部分以灰色标出，更多代码的修改详见文件夹：

<!--用户主界面，

提供通往父子模块管理添加页接口；

显示笔记组织层次；

提供通往子模块详情页接口；-->

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>学习笔记管理</title>

<link href="inc/style.css" rel="stylesheet" type="text/css">

</head>

<body>

<?php

include "inc/mysql.inc";

include "inc/myfunction.inc";

include "inc/head.php";

$aa=new mysql;

$bb=new myfunction;

$aa->link("");

?>

<style type="text/css">

a{ text-decoration:none}

a:hover{ text-decoration:underline}

a:visited{color:#00F}

#choice {

font-size: 19px;

font-family:Georgia, "Times New Roman", Times, serif;

}

</style>

<!--父子模块管理页布局-->

<table width="98%" align="center" cellpadding="0" cellspacing="1" rules="cols" height="30">

<tr>

<td width="25%" align="center" valign="middle"><a href=fmodule\_add.php><img src='pic/new.png' width='20px' style='vertical-align:middle' alt='add'><span id="choice">&nbsp;添加父模块</span></a></td>

<td width="25%" align="center" valign="middle"><a href=fmodule\_list.php><img src='pic/manage1.png' width='20px' style='vertical-align:middle' ><span id="choice">&nbsp;管理父模块</span></a></td>

<td width="25%" align="center" valign="middle"><a href=smodule\_add.php><img src='pic/new.png' width='20px' style='vertical-align:middle' alt='add'><span id="choice">&nbsp;添加子模块</span></a></td>

<td width="25%" align="center" valign="middle"><a href=smodule\_list.php><img src='pic/manage1.png' width='20px' style='vertical-align:middle'><span id="choice">&nbsp;管理子模块</span></a></td>

</tr>

</table>

<!--笔记结构展示区布局-->

<table width="98%" border="0" align="center" cellpadding="0" cellspacing="1" bgcolor="#ffffff">

<tr>

<td width="60%" height="25" align="center" valign="middle" bgcolor="#CCCCCC"><font face="宋体" size="+1" >主题</font></td>

<td width="10%" align="center" valign="middle" bgcolor="#CCCCCC"><font face="宋体" size="+1">笔记</font></td>

<td width="30%" align="center" valign="middle" bgcolor="#CCCCCC"><font face="宋体" size="+1">最新笔记及修改时间</font></td>

</tr>

<tr>

<td colspan="5">

<!--从数据库中获取相应信息-->

<?php

$query="select \* from father\_module\_info order by show\_order";

$result=$aa->excu($query);

while($father\_module=mysqli\_fetch\_array($result)){//当存在父模块时循环

?>

<style type="text/css">

#ftitle {

font-size: 19px;

font-family:"微软雅黑";

}

</style>

<table width="100%" border="0" cellspacing="0" cellpadding="0">

<tr>

<td height="30" colspan="6" bgcolor="#CCFFFF"><b><span id="ftitle">&nbsp;&nbsp;&nbsp;<?php echo $father\_module["module\_name"]?></span></b></td><!--输出父模块名称-->

</tr>

<?php

$query2="select \* from son\_module\_info where father\_module\_id='$father\_module[id]' order by id";

$result2=$aa->excu($query2);

while($son\_module=mysqli\_fetch\_array($result2)){

?>

<tr>

<td width="15%" height="40" align="center" valign="middle"></td>

<td width="45%" align="center" valign="middle">

<?php

echo "<img src='pic/star.png' width='20px' style='vertical-align:middle'>&nbsp;&nbsp;章节:&nbsp;&nbsp;<b><a href=module\_list.php?module\_id=".$son\_module['id']."><font color=0000ff>".$son\_module["module\_name"]."</font></a></b><br>";

echo "主要内容:&nbsp;&nbsp;".$son\_module["module\_cont"];//输出子模块标题及简介

?>

</td>

<td width="10%" align="center" valign="middle" > <?php echo $bb->son\_module\_id\_tonum($son\_module["id"]);?> </td><!--输出模块中的笔记数-->

<td width="30%" align="left" valign="middle"> <?php echo $bb->son\_module\_idtolast($son\_module["id"]);?> </td><!--输出最新笔记-->

</tr>

<?php } ?>

</table>

<?php } ?>

</td>

</tr>

<tr>

<table style='position:absolute; bottom:40px;' width="100%" align='center' border="0" cellspacing="0" cellpadding="0">

<tr>

<td height="25px" bgcolor="CCFFFF" align='center' width='98%'></td>

</tr>

</table>

<tr>

</table>

<?php

include "inc/foot.php";

?>

</body>

</html>