

# 法律声明

---

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。



关注 小象学院

---

# 第十课 C语言工程与学习建议

林沐

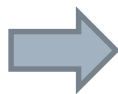
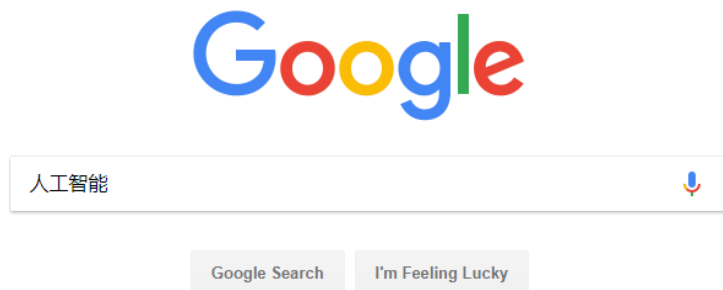
# 内容概述

---

- 1.工程背景概述：搜索引擎
- 2.工程背景概述：网页收录与计算
- 3.工程背景概述：网页与HTML解析
- 4.C语言工程：开源库gumbo解析网页
- 5.gumbo编译
- 6.使用htmlparser获取网页标题
- 7.整体程序
- 8.gumbo数据结构使用与算法
- 9.标题文本提取
- 10.编译与测试
- 11.一些建议:关于学习
- 12.一些建议:关于算法之路
- 13.一些建议:关于算法之路
- 14.一些建议:关于面试
- 15.一些建议:关于实习
- 16.一些建议:关于求职谈薪资
- 17.一些建议:关于职场
- 18.一些建议:大小公司的比较
- 19.一些建议:关于人工智能

# 工程背景概述：搜索引擎

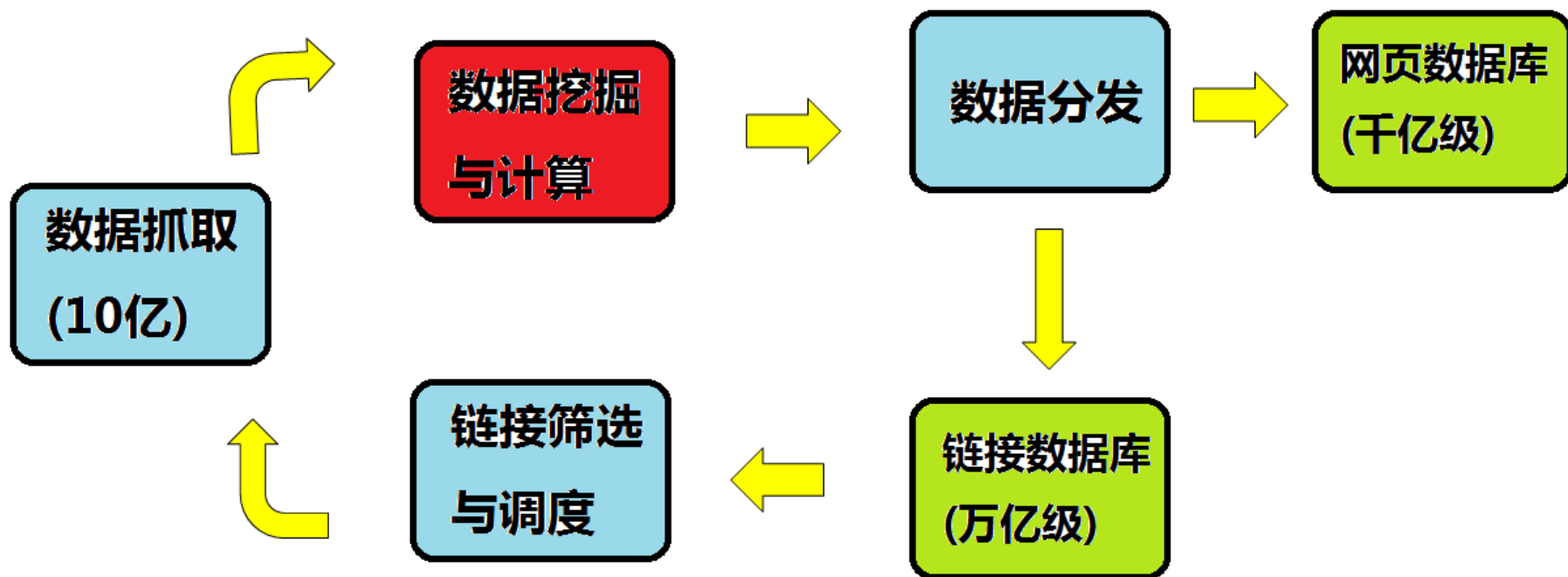
一个搜索事件：



无论**搜索引擎**如何升级与迭代，无论用**何种手段**，搜索引擎就是在做**三件事**：  
**理解用户行为**：语音、文字、图片等等输入  
**收录并理解**互联网数据：网页  
将用户行为与互联网数据**建立联系**，并推荐结果

# 工程背景概述：网页收录与计算

搜索引擎最基础的系统:spider数据收录与数据挖掘系统，一个**全网搜索引擎**(google、baidu、sogou)的**spider系统**每日至少**调度抓取**十亿级链接，整个spider数据库会收录存储**千亿级网页**与**万亿级链接**，它是一个**环状系统**。

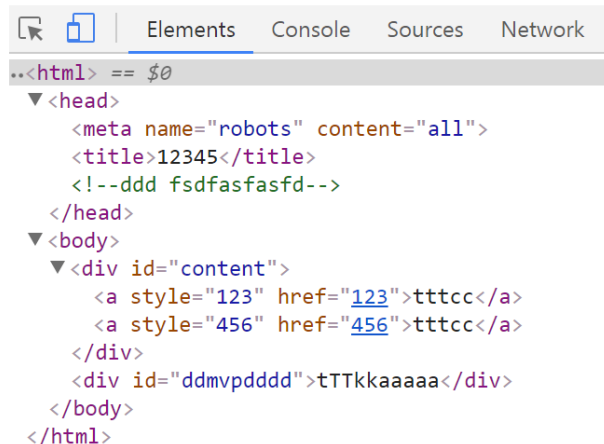


# 工程背景概述：网页与HTML解析

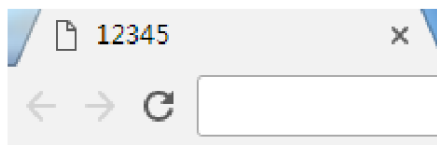
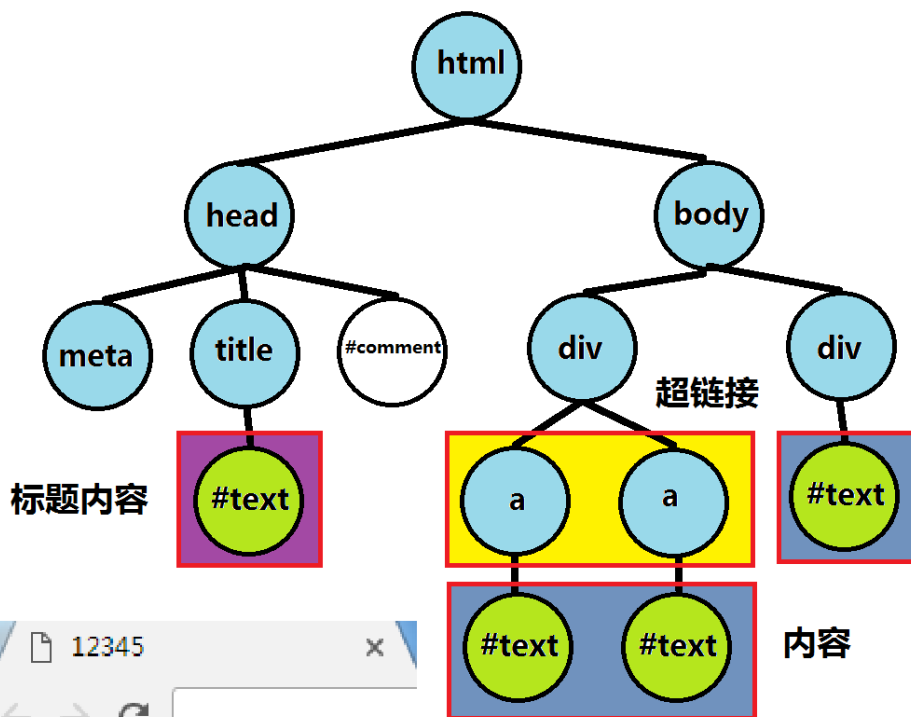
spider最为基础的环节就是**网页(HTML)解析**，**网页解析**即将网页源码**HTML字符串**转为计算机可以理解的数据结构，**HTML树**，即dom树。

如下图，蓝色的是**标签节点(Element)**，绿色的是**文本节点(Text)**。标签节点标识了HTML文档的**逻辑结构**，文本节点是HTML文档的**内容**，内容可以**显示**在浏览器中。在HTML文档中，有很多重要的内容用于分析网页属性，例如HTML**文档标题**，HTML文档中的**超链接**，HTML文档中的**文本内容**。

```
1 <html>
2 <head>
3   <meta name="robots" content="all" />
4   <title>12345</title>
5   <!--ddd fsdfasfasfd-->
6 </head>
7 <body>
8   <div id="content">
9     <a style="123" href="123" >tttcc</a>
10    <a style="456" href="456" >tttcc</a>
11  </div>
12  <div id="ddmvpddd">tTTkkaaaaa</div>
13 </body>
14 </html>
```



The screenshot shows a web browser's developer console with the 'Elements' tab selected. It displays the DOM tree structure for the provided HTML code. The root is <html>, which branches into <head> and <body>. <head> contains <meta> and <title>. <body> contains a <div id="content"> with two <a> tags and another <div id="ddmvpddd"> with text. The text content of the <a> tags is highlighted in blue, and the text content of the <div id="ddmvpddd"> is highlighted in green.



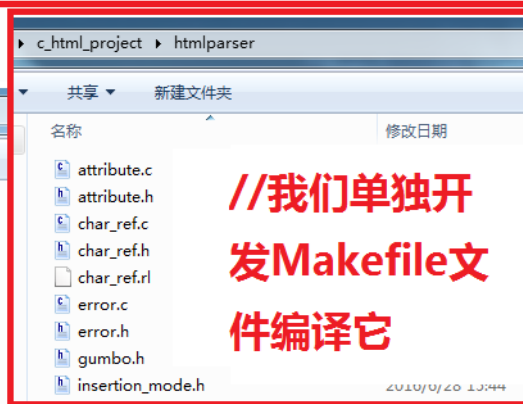
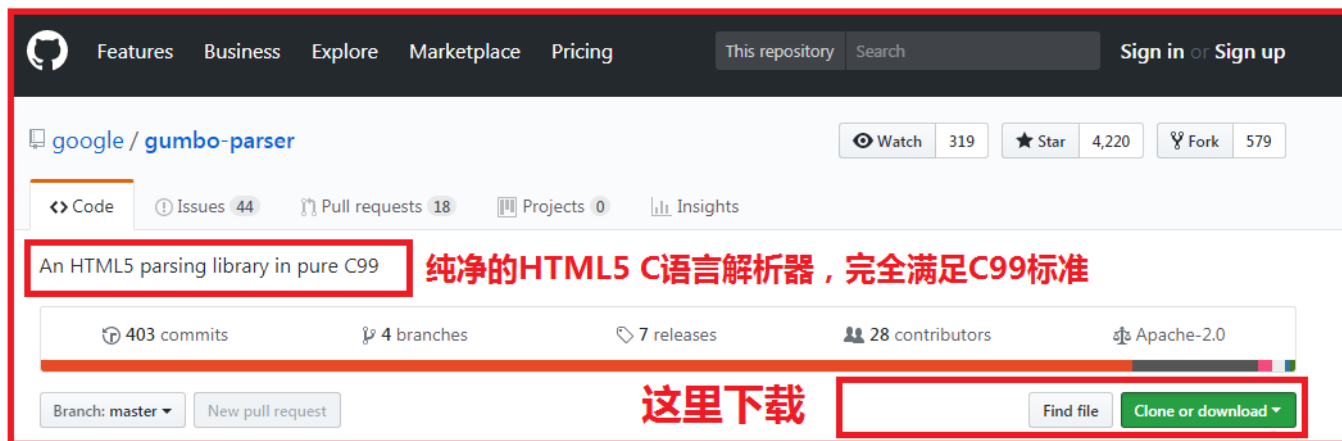
tttcc tttcc  
tTTkkaaaaa

# C语言工程：开源库gumbo解析网页

gumbo是Google的研发团队开发的开源的C语言HTML解析器，它完全符合C99标准(我们的课程是基于C89讲解的)，它完全不依赖其他任何第三方库，且支持最新HTML5标准，经过了Google搜索引擎索引的25亿的网页测试，是不可多得的HTML解析器！

源码下载地址: <https://github.com/google/gumbo-parser>，完成下载后，将src中的源码单独拷贝到另外的文件夹，我们单独开发Makefile文件编译它(不使用Google自带编译脚本)。

<https://github.com/google/gumbo-parser>



# gumbo编译

在htmlparser目录(保存src文件夹的所有内容)下, 单独开发Makefile文件编译gumbo, 生成htmlparser解析器。发布内容output包括两部分, 头文件output/include目录, 与静态链接库output/lib目录, 将全部.h文件拷贝至output/include目录, 将静态库libhtmlparser.a拷贝至output/lib。

## //编译静态库 libhtmlparser.a

```
$ make
gcc -std=c99 -Wall -I./ -c attribute.c
gcc -std=c99 -Wall -I./ -c char_ref.c
gcc -std=c99 -Wall -I./ -c error.c
gcc -std=c99 -Wall -I./ -c parser.c
gcc -std=c99 -Wall -I./ -c string_buffer.c
gcc -std=c99 -Wall -I./ -c string_piece.c
gcc -std=c99 -Wall -I./ -c tag.c
In file included from tag.c:30:
tag_sizes.h:4:456: warning: no newline at end of file
gcc -std=c99 -Wall -I./ -c tokenizer.c
gcc -std=c99 -Wall -I./ -c utf8.c
gcc -std=c99 -Wall -I./ -c util.c
gcc -std=c99 -Wall -I./ -c vector.c
ar rc libhtmlparser.a attribute.o char_ref.o error.o
parser.o string_buffer.o string_piece.o tokenizer.o
utf8.o util.o vector.o
```

## //生成库文件与头文件

```
$ make output
mkdir -p output
mkdir -p output/lib
mkdir -p output/include
cp libhtmlparser.a output/lib
cp *.h output/include
```

```
$ ll output/
total 8
drwxrwxr-x 4096 Feb  8 14:19 include
drwxrwxr-x 4096 Feb  8 14:19 lib

$ ll output/include/
total 108
-rwxr--r-- 1045 Feb  8 14:19 attribute.h
-rwxr--r-- 2214 Feb  8 14:19 char_ref.h
-rwxr--r-- 7935 Feb  8 14:19 error.h
-rwxr--r-- 23760 Feb  8 14:19 gumbo.h
-rwxr--r-- 1897 Feb  8 14:19 insertion_mode.h
-rwxr--r-- 2049 Feb  8 14:19 parser.h
-rwxr--r-- 3063 Feb  8 14:19 string_buffer.h
-rwxr--r-- 1210 Feb  8 14:19 string_piece.h
-rwxr--r-- 2603 Feb  8 14:19 tag_enum.h
-rwxr--r-- 6907 Feb  8 14:19 tag_gperf.h
-rwxr--r-- 558 Feb  8 14:19 tag_sizes.h
-rwxr--r-- 1403 Feb  8 14:19 tag_strings.h
-rwxr--r-- 4621 Feb  8 14:19 tokenizer.h
-rwxr--r-- 3737 Feb  8 14:19 tokenizer_states.h
-rwxr--r-- 1109 Feb  8 14:19 token_type.h
-rwxr--r-- 5004 Feb  8 14:19 utf8.h
-rwxr--r-- 2029 Feb  8 14:19 util.h
-rwxr--r-- 2398 Feb  8 14:19 vector.h

$ ll output/lib/
total 620
-rw-rw-r-- 628530 Feb  8 14:19 libhtmlparser.a
```

## //产出文件

```
Makefile
CC := gcc
INCLUDE := -I./ //注意添加C99标准
LDFLAGS := -L./
CFLAGS = -std=c99 -Wall $(INCLUDE)
LIB := libhtmlparser.a
```

```
src := attribute.c \
char_ref.c \
error.c \
parser.c \
string_buffer.c \
string_piece.c \
tag.c \
tokenizer.c \
utf8.c \
util.c \
vector.c
```

## // II后, 将源文件名添加上

## //目标.c源文件 展开为.o文件

```
obj := $(src:%.c=%.o)
```

```
%.o : %.c
$(CC) $(CFLAGS) -c $^ //将.c文件编译为.o文件
```

```
$(LIB) : $(obj)
ar rc $@ $^ //打包.o文件为.a文件
```

```
output : $(LIB)
mkdir -p output
mkdir -p output/lib
mkdir -p output/include
cp $(LIB) output/lib
cp *.h output/include //发布.h头文件与
静态库libhtmlparser.a
```

```
clean :
rm -rf *.o $(LIB) //清空编译内容
rm -rf output
```



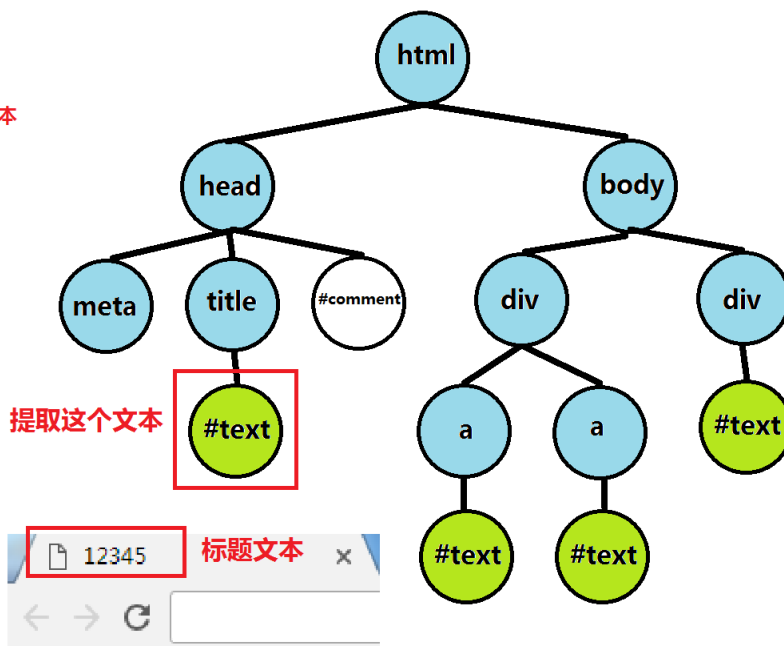
# 使用htmlparser获取网页标题

完成gumbo代码编译后，使用发布的output/lib静态库libhtmlparser.a与对应的头文件output/include，开发一个提取网页标题的程序parser\_title.c。已知标题标签是<title>，它一定是<head>标签的孩子，<head>标签一定是<html>标签的孩子。

思考：

1. 如何将网页读入，并输入给gumbo的解析接口？
2. 如何获取title标签？又如何获取title标签中的文本？
3. 完成代码开发后，如利用Makefile编译该工具？

```
1 <html>
2 <head>
3 <meta name="robots" content="all" />
4 <title>12345</title> //标题标签与对应的文本
5 <!--ddd fsdfasfasfd-->
6 </head>
7 <body>
8 <div id="content">
9 <a style="123" href="123" >tttcc</a>
10 <a style="456" href="456" >tttcc</a>
11 </div>
12 <div id="ddmvpddd">tTTkkaaaaa</div>
13 </body>
14 </html>
```



提取这个文本

12345

标题文本

tttcc tttcc

tTTkkaaaaa

```
$ ll
total 128
-rwxr--r-- 232 Feb 8 15:32 Makefile
drwxr-xr-x 4096 Feb 8 16:16 output

//完成编译的静态库
//开发一个提取title的源文件
-rwxr--r-- 1822 Feb 8 16:04 parser_title.c
-rwxr--r-- 109656 Feb 8 15:34 test-github.html
-rwxr--r-- 279 Feb 8 14:33 test.html

//test测试数据

$ make parser_title //编译
gcc -Wall -I./output/include -o parser_title
parser_title.c -L./output/lib -lhtmlparser

$ ./parser_title test.html //运行
title = [12345]
```

# 整体程序，课堂练习

```
#include <stdio.h>
#include <string.h>
```

```
#include "gumbo.h" //包含gumbo头文件
```

```
void parser_title(const GumboOutput *output, char title[]){
    //传入gumbo解析结果，传出title，存储在字符数组title中
}
```

```
char g_page[2 * 1024 * 1024] = {0}; //存储网页与网页长度的全局变量
int g_page_len = 0;
```

```
int read_page(const char *file_name){
    FILE *fp = fopen(file_name, 1);
    if (!fp){
        return -1; //将文件file_name的内容读取并存储至g_page
    } //中，g_page_len代表g_page中的有效字符数
    int ch;
    ch = fgetc(fp); //文件打开失败返回-1，成功返回0
    while( 2 ){
        3
        ch = fgetc(fp);
    }
    4
    return 0;
}
```

```
int main(int argc, char *argv[]){
    if ( 5 ){
        printf("usage : ./parser_title html\n");
        return -1;
    }
    if (read_page(argv[1]) != 0){
        printf("cannot open file %s.\n", argv[1]);
        return -1;
    } //调用gumbo解析器
```

```
GumboOutput* output = gumbo_parse_with_options(
    &kGumboDefaultOptions, g_page, g_page_len);
```

```
char title[10240] = {0};
parser_title(output, title);
printf("title = [%s]\n", title); //打印解析出的title
return 0;
```

```
}
```

3分钟，填写代码  
，有问题提出！

# 整体程序，实现

```
#include <stdio.h>
#include <string.h>
#include "gumbo.h" //包含gumbo头文件

void parser_title(const GumboOutput *output, char title[]){
    //传入gumbo解析结果，传出title，存储在字符数组title中
}

char g_page[2 * 1024 * 1024] = {0}; //存储网页与网页长度的全局变量
int g_page_len = 0;

int read_page(const char *file_name){
    FILE *fp = fopen(file_name, "r");
    if (!fp){
        return -1; //将文件file_name的内容读取并存储至g_page
    } //中，g_page_len代表g_page中的有效字符数
    int ch;
    ch = fgetc(fp); //文件打开失败返回-1，成功返回0
    while( ch != EOF ){
        g_page[g_page_len++] = ch;
        ch = fgetc(fp);
    }
    fclose(fp);
    return 0;
}

int main(int argc, char *argv[]){
    if ( argc != 2 ){
        printf("usage : ./parser_title html\n");
        return -1;
    }
    if (read_page(argv[1]) != 0){
        printf("cannot open file %s.\n", argv[1]);
        return -1;
    } //调用gumbo解析器
    GumboOutput* output = gumbo_parse_with_options(
        &kGumboDefaultOptions, g_page, g_page_len);
    char title[10240] = {0};
    parser_title(output, title);
    printf("title = [%s]\n", title); //打印解析出的title
    return 0;
}
```

# gumbo数据结构使用与算法

找到**根节点**(<html>标签)孩子中的<head>标签节点，再找到<head>标签孩子节点中的<title>标签节点，<title>标签节点中的**第一个孩子**即为最终需要提取的**文本节点**，提取该文本节点中的文本。

```
typedef struct GumboInternalOutput {
    GumboNode* document;
    GumboNode* root; //根节点
    GumboVector errors;
} GumboOutput;

typedef enum {
    GUMBO_NODE_DOCUMENT,
    GUMBO_NODE_ELEMENT, //标签节点
    GUMBO_NODE_TEXT,
    GUMBO_NODE_CDATA,
    GUMBO_NODE_COMMENT,
    GUMBO_NODE_WHITESPACE,
    GUMBO_NODE_TEMPLATE
} GumboNodeType;

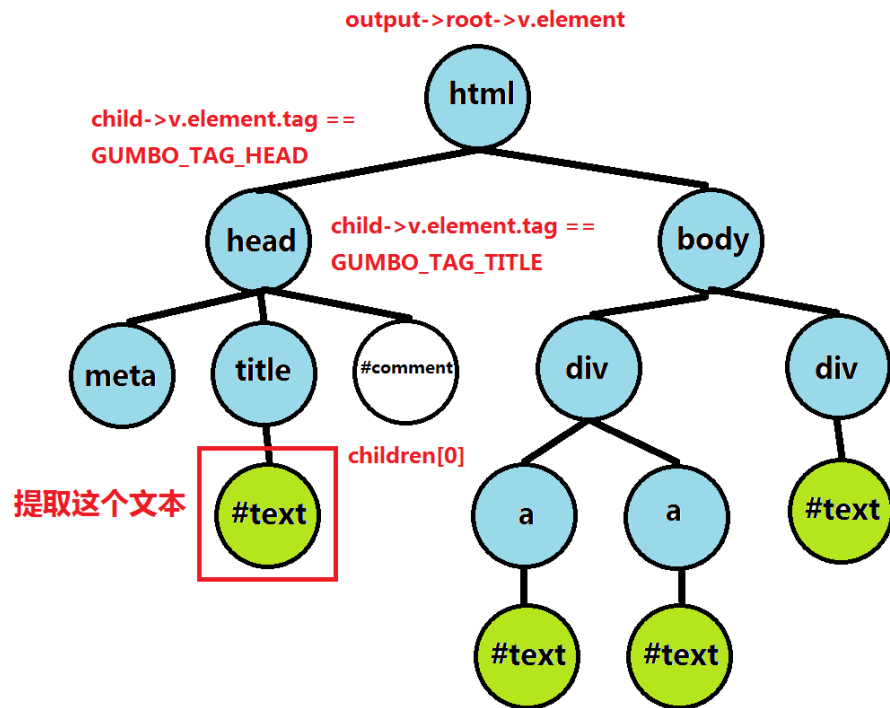
struct GumboInternalNode {
    GumboNodeType type;
    GumboNode* parent;
    size_t index_within_parent;
    GumboParseFlags parse_flags;
    union {
        GumboDocument document;
        GumboElement element; //标签或文本
        GumboText text;
    } v;
};

typedef struct GumboInternalNode GumboNode;

typedef struct {
    void** data;
    unsigned int length;
    unsigned int capacity;
} GumboVector;

//孩子数组
typedef struct {
    GumboVector children;
    GumboTag tag;
    GumboNamespaceEnum tag_namespace;
    GumboStringPiece original_tag;
    GumboStringPiece original_end_tag;
    GumboSourcePosition start_pos;
    GumboSourcePosition end_pos;
    GumboVector attributes;
} GumboElement;

//最终存储的文本
typedef struct {
    const char* text;
    GumboStringPiece original_text;
    GumboSourcePosition start_pos;
} GumboText;
```



# 标题文本提取，课堂练习

```
void parser_title(const GumboOutput *output, char title[]){
    const GumboVector *children = &(output->root->v.element.children);
    GumboNode* head = NULL;
    int i;
    for (i = 0; i < 1; i++) {
        GumboNode* child = 2
        if (child->type == GUMBO_NODE_ELEMENT &&
            3 == GUMBO_TAG_HEAD) {
            head = child;
            break;
        }
    }
    children = 4
    for (i = 0; i < 1; i++) {
        GumboNode* child = 2
        if (child->type == GUMBO_NODE_ELEMENT &&
            3 == GUMBO_TAG_TITLE) {
            GumboNode *title_text = child->v.element.children.data[0];
            if (5) {
                strcpy(title, title_text->v.text.text);
            }
        }
    }
}
```

3分钟，填写代码  
，有问题提出！

```
typedef struct GumboInternalOutput {
    GumboNode* document;
    GumboNode* root; //根节点
    GumboVector errors;
} GumboOutput;

typedef enum {
    GUMBO_NODE_DOCUMENT,
    GUMBO_NODE_ELEMENT; //标签节点
    GUMBO_NODE_TEXT,
    GUMBO_NODE_CDATA,
    GUMBO_NODE_COMMENT,
    GUMBO_NODE_WHITESPACE,
    GUMBO_NODE_TEMPLATE
} GumboNodeType;

struct GumboInternalNode {
    GumboNodeType type;
    GumboNode* parent;
    size_t index_within_parent;
    GumboParseFlags parse_flags;
    union {
        GumboDocument document;
        GumboElement element;
        GumboText text; //标签或文本
    } v;
};

typedef struct GumboInternalNode GumboNode;

typedef struct {
    void** data;
    unsigned int length;
    unsigned int capacity;
} GumboVector;

//孩子数组
typedef struct {
    GumboVector children;
    GumboTag tag;
    GumboNamespaceEnum tag_namespace;
    GumboStringPiece original_tag;
    GumboStringPiece original_end_tag;
    GumboSourcePosition start_pos;
    GumboSourcePosition end_pos;
    GumboVector attributes;
} GumboElement;

//最终存储的文本
typedef struct {
    const char* text;
    GumboStringPiece original_text;
    GumboSourcePosition start_pos;
} GumboText;
```

# 标题文本提取，实现

```
void parser_title(const GumboOutput *output, char title[]){
    const GumboVector *children = &(output->root->v.element.children);
    GumboNode* head = NULL;
    int i;

    for (i = 0; i < children->length; i++) {
        GumboNode* child = children->data[i];
        if (child->type == GUMBO_NODE_ELEMENT &&
            child->v.element.tag == GUMBO_TAG_HEAD) {
            head = child;
            break;
        }
    }
    children = &(head->v.element.children);

    for (i = 0; i < children->length; i++) {
        GumboNode* child = children->data[i];
        if (child->type == GUMBO_NODE_ELEMENT &&
            child->v.element.tag == GUMBO_TAG_TITLE) {
            GumboNode *title_text = child->v.element.children.data[0];
            if (title_text->v.text.text){
                strcpy(title, title_text->v.text.text);
            }
        }
    }
}
```

```
typedef struct { //最终存储的文本
    const char* text;
    GumboStringPiece original_text;
    GumboSourcePosition start_pos;
} GumboText;
```

```
typedef struct GumboInternalOutput {
    GumboNode* document;
    GumboNode* root; //根节点
    GumboVector errors;
} GumboOutput;

typedef enum {
    GUMBO_NODE_DOCUMENT,
    GUMBO_NODE_ELEMENT, //标签节点
    GUMBO_NODE_TEXT,
    GUMBO_NODE_CDATA,
    GUMBO_NODE_COMMENT,
    GUMBO_NODE_WHITESPACE,
    GUMBO_NODE_TEMPLATE
} GumboNodeType;

struct GumboInternalNode {
    GumboNodeType type;
    GumboNode* parent;
    size_t index_within_parent;
    GumboParseFlags parse_flags;
    union {
        GumboDocument document;
        GumboElement element; //标签或文本
        GumboText text;
    } v;
};

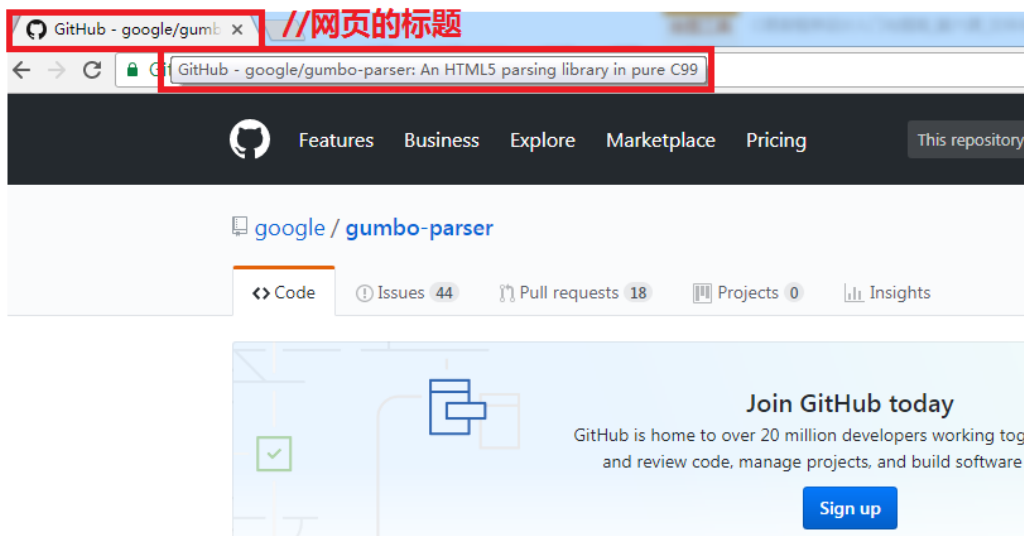
typedef struct GumboInternalNode GumboNode;

typedef struct {
    void** data;
    unsigned int length;
    unsigned int capacity;
} GumboVector;

typedef struct { //孩子数组
    GumboVector children;
    GumboTag tag;
    GumboNamespaceEnum tag_namespace;
    GumboStringPiece original_tag;
    GumboStringPiece original_end_tag;
    GumboSourcePosition start_pos;
    GumboSourcePosition end_pos;
    GumboVector attributes;
} GumboElement;
```

# 编译与测试

```
$ ./parser_title test-github.html
title = [GitHub - google/gumbo-parser: An HTML5 parsing library in pure C99]
```



```
7 <!DOCTYPE html>
8 <html lang="en">
9   <head>
10     <meta charset="utf-8">
11     <link rel="dns-prefetch" href="https://assets-cdn.github.com">
12     <link rel="dns-prefetch" href="https://avatars0.githubusercontent.com">
13     <link rel="dns-prefetch" href="https://avatars1.githubusercontent.com">
14     <link rel="dns-prefetch" href="https://avatars2.githubusercontent.com">
15     <link rel="dns-prefetch" href="https://avatars3.githubusercontent.com">
16     <link rel="dns-prefetch" href="https://github-cloud.s3.amazonaws.com">
17     <link rel="dns-prefetch" href="https://user-images.githubusercontent.com/">
18
19
20
21     <link crossorigin="anonymous" href="https://assets-cdn.github.com/assets/frameworks-521cbf980c80.css"
22     Uqy/mAyAx1HfserumjQ85ASp0k5bJt2Ay03pMeixIXtVhLEm5S+N4uOHWFdGhwS9Yx4bGyviXWGPCZeIffqYcNA==" media="all" ">
23     <link crossorigin="anonymous" href="https://assets-cdn.github.com/assets/github-b028b0565d07.css" inr
24     sCiW10HopKw2sd/QLCwI+iRbJHnjEXp/sYZCPCru4gLinlY/xDUGcpWznQGGDB4I2QpqxTepNP0sVDm45N11A==" media="all" ">
25
26     <link crossorigin="anonymous" href="https://assets-cdn.github.com/assets/site-e1e1bc98a53e.css" inte
27     4eG8mKU+R9QAnJMQwSIgbo24hS+nUXxSuUs5G5LMQw+5wjC1TSKfgxj2j61PYa94+wlwN10T7r7MF5rbdUkve==" media="all" ">
28
29
30     <meta name="viewport" content="width=device-width">
31
32     <title>GitHub - google/gumbo-parser: An HTML5 parsing library in pure C99</title>
```

//标题标签与对应的文本

## Makefile

```
1
2 CC := gcc
3 INCLUDE := -I./output/include
4 LDFLAGS := -L./output/lib -lhtmlparser
5 CFLAGS := -Wall $(INCLUDE)
6
7 all: parser_title
8
9 parser_title: parser_title.c
10     $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
11
12 clean:
13     rm parser_title
14
```

# 课间休息10分钟！

---

## 有问题提出！



# 一些建议:关于学习

---

1. 掌握一门**编译语言**
2. 掌握**算法与数据结构**
3. 掌握一门**脚本语言**
4. 掌握**开发环境**
5. 丰富其他**前沿知识**

# 一些建议:关于算法之路1

---

阶段0(**初学乍练,不足挂齿**): 学习了**基础知识**(如经典算法、数据结构等), 一般为掌握知识或通过考试在纸上的练习写代码, 好一点的**主动编写**一些小程序, 全国50%以上的计算机专业的同学可能都在这阶段**挣扎**。

阶段1(**粗懂皮毛,半生不熟**): 基本的算法各类题型都已经见过, 简单题或是难题加一起至少也刷过**200多道**, 看到原题、熟题、模板题都可以解决, 但是一到考察想法、或者很复杂的题目, 单靠自己可能想不出来, 需要看**解题报告**才可解决, 反正照着他的思路写出来没问题。

阶段2(**已有小成,融会贯通**): 一般难度的题目(leetcode 80%以上的题目应该都是一般难度的、ACM区域赛的部分题也是)都可以**独立解决**, 虽然有的做的快点, 有的做的慢点, 但是这种题死磕一下肯定都通过, 有了不错的**举一反三**能力。一般计算机专业的每个班都能有一两个同学达到这高度。

达到阶段2的时候, 那么恭喜, 只要再加上**系统性的训练**, 什么BAT微软亚马逊的**算法面试**不会难倒你的! :) )

# 一些建议:关于算法之路2

---

阶段3(**炉火纯青, 出类拔萃**): 刷leetcode这种非竞赛OJ早已经没感觉了, 刷OJ也只刷中等偏上或者难题了, 这样才能提高自己。这一阶段更重要的是培养**临场竞赛能力**, Topcoder SRM, odeforce, 谷歌GCJ, 百度之星之类的算法竞赛, 有时间参加一定会玩一玩, 状态好的时候打个前几名, 状态差点反正差不到哪里去。Topcoder、Codeforce黄名或者红名的阶段。如果还有两个志同道合的好队友, ACM打个金奖也没什么问题。

阶段4(**登峰造极, 举世无双**): Topcoder红色带靶心, 世界上到这阶段的选手其实**屈指可数**, ACM world final的奖牌是曾经的回忆, 退役后有空也会玩一下, 不小心就是某个比赛的**第一名**。题目早就不会再去刷了, **无招胜有招**。其实想到这阶段, 光凭努力一般是不行的。最重要的还是**天赋**, 就是天才那就是1%的灵感加上99%的汗水中的那1%的灵感。

# 一些建议:关于面试

---

1. 听清并**搞懂**问题

2. **冷静思考**

3. 努力解决，**不**轻易**放弃**

4. 自信并**谦虚**

# 一些建议:关于实习

---

1. 实习是拿到BAT offer的捷径
2. 实习的岗位有许多:RD、QA、FE、OP、PM等
3. 尽量在一个实习岗位坚持半年
4. 实习还有不菲的薪水喔~
5. 关于学校课堂与上课, 自己把握

# 一些建议:关于求职谈薪资

---

1. **表达对工作、职位、技术的热爱**: 在谈薪资时, 不管满意与否, 也不管是否手里还有其他offer, 一定要对招聘方表达你对这个岗位工作的兴趣, 因为站在招聘者的角度, 只有有热情的候选人将来才能做好工作。

2. 表达对期望薪资的**客观理由**:

- a. 自身素质, 既然已经通过了各项技术面试, 那就说明已经胜任该工作岗位, 所以再表达一下也是应该的。
- b. 竞品公司类似岗位, 在谈薪资前, 要对竞品公司的同职位的薪资有充分的了解, 才能与更合理的与HR argue薪资。
- c. 应聘公司的相应级别或岗位的一般待遇, 因为同是职称(title), 由于候选人个人素质不同, 给的一般也不一样, 要对该职称有整体了解。

3. **话不说死**:

在这一阶段, 无论HR如何施压(不马上签就没了), 也不要马上答应或说死(不要马上对HR表达是否接受offer), 要留有余地, 这个潜台词是我可能还在看别的机会, 但是对于这个岗位我仍然保有很大兴趣。

4. **接offer**:

无论怎样, offer还是要接的, 一般来讲, HR会给到他最终能给到的最高点(他也希望你入职, 不然他本次的沟通就是无用的), 但是不代表一定会入职, 如果真不太满意, 可以拿着这个offer再去找别家, 骑着骡子找马嘛。反正入职时间可以往后拖一拖。

5. **态度: 积极、乐观、谦虚、自信**: 无论你们对该职位满意与否, 装也得装的这样。: )

# 一些建议:关于职场

---

1. 正面看待工作中的各种问题，**积极面对**，让工作变得快乐
2. 遇到问题，**独立思考**，认真听取他人的建议，自己做决定
3. 技术与产品的发展在于创新，**敢于创新**，即使创新可能带来失败
4. 培养owner的精神，**勇于承担**，得到同事与上级的信任

# 一些建议:大小公司的比较

---

大公司(优势):流程规范, 最初成长快速, 资源丰富, 可以镀金增加将来的**应聘竞争力**。建议校园招聘直接进入大公司。

大公司(劣势):**竞争激烈**, 绩效严格, 加班严重, 等级比较严格。

小公司(优势):**直接与公司管理者对话**, 工作灵活, 一般竞争不会太激烈, 老板们也会较为珍惜人才

小公司(劣势):大公司优势的**反面**。



# 一些建议:关于人工智能

---

1. 人工智能**时代**的来临: PC桌面软件->互联网->移动互联网->**人工智能**
2. 人工智能的**学习**:打牢**基础**, 接地气的解决问题, 尝试新方法
3. 人工智能的**就业**:首选BAT等互联网行业巨头, 不断**积累**, 最终尝试创造新事物

# 广告: 算法与数据结构入门经典

---

## 系统课 算法与数据结构 入门经典



 老师: 林沐、林清

 开课日期: 2018-06-23

# 结束

---

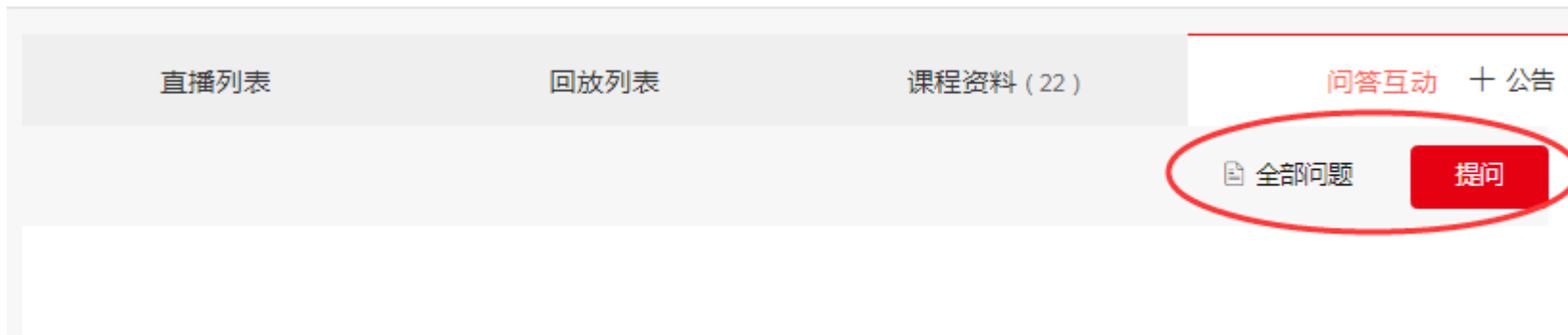
非常感谢大家！

林沐

# 问答互动

在所报课的课程页面，

- 1、点击“全部问题”显示本课程所有学员提问的问题。
- 2、点击“提问”即可向该课程的老师 and 助教提问问题。



# 联系我们

---

## 小象学院：互联网新技术在线教育领航者

— 微信公众号：**小象学院**

