

数据结构第一讲

一、自我介绍

- 1> 姓名：张鹏鹏
- 2> 产地：安徽阜阳
- 3> TEL：13141518517
- 4> 所带课程：C基础、数据结构、C++&QT

二、课前绪论

- 1> 数据结构这门课，没有新的知识点，都是在C语言的基础上，进行逻辑上的整合，将一些常用的操作进行封装
- 2> 学习起来比较轻松，主要注重逻辑思维的培养
- 3> 由于是线上，课程进度可能会稍慢一些，有助于大家充分消化吸收
- 4> 上课过程中，大家尽可能的积极互动，一方面可以增强自己的记忆力，另一方面有助于我看清大家接收程度
- 5> 写代码过程中，最好准备一个纸和笔，绘图使用，只要图像心中过，优异成绩有把握
- 6> 在学习过程中，有问题一定要当场提出了，当场解决，不要堆积问题

三、课程安排

- day1> 数据结构理论+顺序表
- day2> 单向链表、单向循环链表
- day3> 双向链表、栈
- day4> 队列、二叉树
- day5> 算法、图

四、数据理论概念

4.1 数据的定义

计算机中数据的定义：能够被计算机识别、存储、处理的、用于描述客观事物的 **符号**

4.2 数据分类

- 1> 数值数据：二进制数据、整数、小数
- 2> 非数值数据：文字、图像、视频、音频

4.3 数据元素

所谓数据元素，就是用于描述完整事物的数据**基本单位**
例如：华清远见上海中心的每个学生，就是一个数据元素

4.4 数据项

用于组成数据元素、具有独立且不可分割的**最小单位**
例如：每个学生的姓名、年龄、学号、专业等都是一个数据项

4.5 数据对象

由多个性质相同的数据元素组成的集合
例如：一个班级的所有学生组成数据对象

4.6 数据关系的总结

数据项 < 数据元素 < 数据对象 < 数据

五、结构的基本概念

数据结构的结构主要由两部：逻辑结构、存储结构

5.1 逻辑结构

所谓逻辑结构，就是数据元素之间的逻辑关系，用来描述数据元素之间的关联情况的

- 1> 集合结构：所有数据元素只是存在一个集合中，任意两个元素之间没有关系。例如：一个果篮中的蔬果
- 2> 线性结构：数据元素之间存在一对一的关系。例如：排队做核酸
- 3> 树形结构：数据元素之间存在一对多的关系。例如：族谱
- 4> 图形结构：数据元素之间存在多对多的关系。例如：朋友关系

5.2 存储结构

所谓存储结构，就是数据元素逻辑结构映射在计算机中的存储关系

- 1> 顺序存储：是用一段连续的存储单元，来存储逻辑上相邻的元素。特点：逻辑上相邻，物理上也相邻
- 2> 链式存储：使用任意一个存储单元，来存储逻辑上相邻的元素。特点：逻辑

上相邻，物理上不一定相邻

3> 索引存储：存储数据时，多加一张索引表进行定位的存储方式（了解）

4> 散列存储：也称哈希存储，存储数据的物理位置跟数据元素关键字有关（了解）

六、数据结构的概念

所谓数据结构，其实就是指互相之间存在一种或多种特定关系的数据元素的集合

七、顺序表

7.1 线性表

线性表是由多个相同特数据元素组成的线性结构

7.2 线性表的分类

1> 顺序表：顺序存储的线性表

2> 链表：链式存储的线性表

3> 栈：只能在一端进行插入和删除的线性表

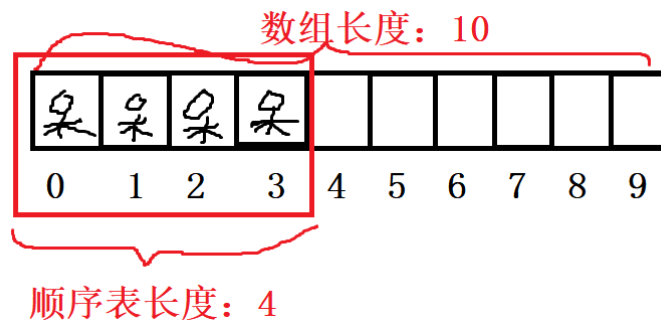
4> 队列：只能在一端进行插入、另一端进行删除的线性表



7.3 顺序表概念

一般使用数组来存储数据元素（实现连续存储），并且附加一个表的长度来实现

所以顺序表的实现，是一个构造数据类型，包括两部分：存储顺序表的数组+顺序表的长度



7.4 顺序表结构体

```

1  #define MAX 20                //数组长度
2  typedef int datatype;
3
4  typedef struct
5  {
6      datatype    data[MAX];    //存储顺序表的数组
7      int    len;              //记录顺序表的表长
8  }seqList;

```

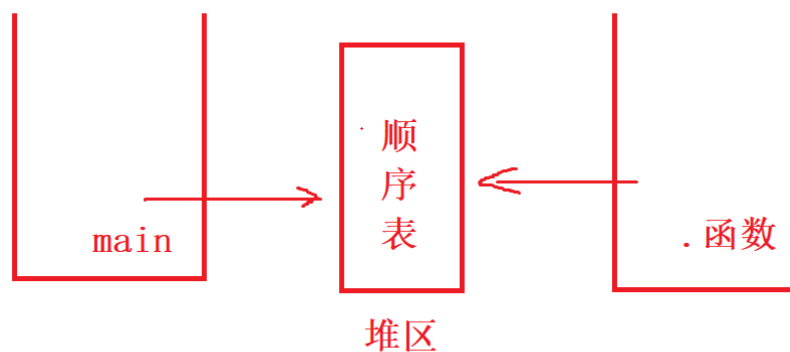
注:

- 1> 数组长度表示的是存储数据的最大上限
- 2> 顺序表的长度表示当前数组所存放数据元素的个数
- 3> 顺序表的长度 \leq 数组的长度
- 4> 顺序表长度的作用有两个: 第一可以表示顺序表总的长度、第二可以表示数组中第一个没有存放数据的下标

7.5 顺序表创建

在堆区申请空间, 有利我们人为进行申请和释放。有两种函数可以实现, 第一种作为参考, 主要使用第二种

在函数申请完空间并初始化后, 讲申请空间的地址作为返回值返回给主调函数



```

1  //创建顺序表的实现方式1, 不常使用
2  /*

```

```

3 void list_create(seqList **L)
4 {
5     *L = (seqList*)malloc(sizeof(seqList));
6 }
7 */
8
9 //创建函数是返回堆区申请空间的地址
10 seqList *list_create()
11 {
12     seqList *L = (seqList*)malloc(sizeof(seqList));
13     if(NULL == L)
14     {
15         printf("创建失败\n");
16         return NULL;
17     }
18
19     //对顺序表初始化
20     memset(L->data, 0, sizeof(L->data));
21
22     L->len = 0;           //顺序表初始长度为0
23
24     printf("创建成功\n");
25
26     return L;           //将顺序表返回
27 }

```

7.6 顺序表判空、判满

```

1 //判空
2 int list_empty(seqList *L)
3 {
4     return L->len==0 ? 1:0;    //1表示空，0表示非空
5 }
6
7
8 //判满
9 int list_full(seqList *L)
10 {
11     return L->len==MAX ? 1:0;    //1表示满，0表示非满
12 }

```

7.7 顺序表添加元素

- 1> 判断表是否合法、判断表是否满了
- 2> 在表的第一个没有存放数据的位置，将要添加的数据放入
- 3> 表长要自增

```
1 //添加元素
2 int list_add(seqList *L, datatype e)
3 {
4     //判断逻辑
5     if(NULL==L || list_full(L))
6     {
7         printf("添加失败\n");
8         return -1;
9     }
10
11     //添加逻辑：在表中第一个没有存放数据的元素中将e放入
12     L->data[L->len] = e;
13
14     //表的变化
15     L->len++;
16
17     printf("添加成功\n");
18     return 0;
19 }
```

7.8 顺序表遍历

所谓顺序表遍历，实质是是对一维数组的遍历，但是要注意，遍历的结束条件应该是表的长度，而不是数组的长度，因为数组不一定存满

```
1 //定义遍历函数
2 void list_show(seqList *L)
3 {
4     //判断逻辑
5     if(NULL==L || list_empty(L))
6     {
7         printf("遍历失败\n");
8         return;
9     }
10
11     //遍历逻辑
12     printf("目前表中元素分别是: ");
13     for(int i=0; i<L->len; i++)
```

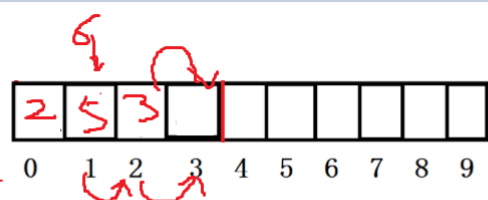
```

14     {
15         printf("%d\t", L->data[i]);
16     }
17     printf("\n");
18 }

```

7.9 顺序表任意位置插入（重点掌握）

- 1> 判断条件：是否合法、表是否满了、位置是否合法
- 2> 腾空，从顺序表的最后一个元素开始，到要插入的位置为止，所有元素后移一格
- 3> 将要插入的数据放入腾好的空里
- 4> 表长自增



$len = 3$

```

for(i=len-1; i>=pos; i--)
{
    data[i+1] = data[i];
}

```

腾空

```
data[pos] = e;
```

占空

```
len++;
```

官宣

```

1 //任意位置插入元素
2 int list_insert_pos(seqList *L, int pos, datatype e)
3 {
4     //判断逻辑
5     if(NULL==L || list_full(L) || pos<0 || pos>L->len)
6     {
7         printf("插入失败\n");
8         return -1;
9     }
10
11     //腾空逻辑
12     for(int i=L->len-1; i>=pos; i--)
13     {
14         L->data[i+1] = L->data[i];    //将当前元素后移
15     }
16

```

```

17     // 占空
18     L->data[pos] = e;
19
20     // 表的变化
21     L->len++;
22
23     printf("插入成功\n");
24     return 0;
25 }

```

7.10 按任意位置删除

- 1> 判断逻辑：判断表的合法性、判断是否空、判断删除位置是否合法
- 2> 删除逻辑：将从要删除的位置的下一个元素开始，到最后一个元素位置为止，所有元素向前移动一格
- 3> 表的变化

```

1  //定义任意位置删除函数
2  int list_delete_pos(seqList *L, int pos)
3  {
4      //判断逻辑
5      if(NULL==L || list_empty(L) || pos<0 || pos>=L->len)
6      {
7          printf("删除失败\n");
8          return -1;
9      }
10
11     //删除逻辑
12     for(int i=pos; i<L->len; i++)
13     {
14         L->data[i] = L->data[i+1];    //将元素前移动
15     }
16
17     //表的变化
18     L->len--;
19
20     printf("删除成功\n");
21     return 0;
22 }

```


7.11 按位置进行修改

- 1> 判断逻辑：判断表合法性、表是否空、位置是否合法
- 2> 将当前下标对应的值进行修改

```
1 //按位置进行修改
2 int list_update_pos(seqList *L, int pos, datatype new_e)
3 {
4     //判断逻辑
5     if(NULL==L || list_empty(L) || pos>=L->len || pos<0)
6     {
7         printf("修改失败\n");
8         return -1;
9     }
10
11     //修改逻辑
12     L->data[pos] = new_e;
13
14     printf("修改成功\n");
15     return 0;
16 }
```

7.12 按位置进行查找并返回该位置上的值

- 1> 判断逻辑：判断表合法性、判断表是否为空、判断查找位置是否合法
- 2> 将该位置的值返回

```
1 datatype list_search_pos(seqList *L, int pos)
2 {
3     //判断逻辑
4     if(NULL==L || list_empty(L) || pos>=L->len || pos<0)
5     {
6         printf("查找失败\n");
7         return -1;
8     }
9
10    //将值返回
11    return L->data[pos];
12 }
```

7.13 按值进行修改

- 1> 判断逻辑：表的合法性、表空
- 2> 将所有元素进行遍历，跟要修改的旧值进行比较，如果相等则修改成新值
- 3> 如果全部都不相等，还是修改失败

```
1  int list_update_value(seqList *L, datatype old_e, datatype
    new_e)
2  {
3      //判断逻辑
4      if(NULL==L || list_empty(L))
5      {
6          printf("修改失败\n");
7          return -1;
8      }
9
10     //修改逻辑
11     int flag = 0;                //判断是否修改
12     for(int i=0; i<L->len; i++)
13     {
14         if(L->data[i] == old_e)    //判断原值是否在表中
15         {
16             L->data[i] = new_e;    //修改
17             flag = 1;
18             // break;              //只修改第一个找到值
19         }
20     }
21
22     //判断flag
23     if(flag == 0)
24     {
25         printf("未找到要修改的值\n");
26         return -2;
27     }
28
29     printf("修改成功\n");
30     return 0;
31
32 }
```

7.14 按值进行查找返回位置

- 1> 判断逻辑：表的合法性、表空
- 2> 将所有元素进行遍历，跟要查找的值进行比较，如果相等返回位置
- 3> 如果全部都不相等，还是修改失败返回负数

```
1  int list_search_value(seqList *L, datatype e)
2  {
3      //判断逻辑
4      if(NULL==L || list_empty(L))
5      {
6          printf("修改失败\n");
7          return -1;
8      }
9
10     //查找逻辑
11     for(int i=0; i<L->len; i++)
12     {
13         if(L->data[i] == e)    //判断原值是否在表中
14         {
15             return i;        //将下标返回
16         }
17     }
18
19     printf("没有该元素，查找失败\n");
20
21     return -2;
22
23 }
```

7.15 顺序表释放

注意：释放函数中，将堆区空间释放后，主调函数和被调函数中会出现两个野指针，所有在被调函数调用free函数后，将其赋值为NULL，主调函数调用被调函数结束后，也要将主调函数中的指针设置为NULL

```
1  void list_free(seqList *L)
2  {
3      //判断逻辑
4      if(NULL == L)
5      {
6          printf("释放失败\n");
7          return;
8      }
```

```

9
10     free(L);           //将空间释放
11     L = NULL;          //防止野指针
12
13     printf("释放成功\n");
14 }

```

八、顺序表的优缺点

优点:

- 1> 对数据的查找和修改比较方便，可以通过下标直接定位到元素 $O(1)$
- 2> 顺序表的操作本质上就是对数组的操作，比较好理解，易接受

缺点:

- 1> 对数据的插入和删除比较麻烦，因为需要移动大量的元素 $O(n)$
- 2> 致命缺点：存储数据有上限

九、全部代码

1> seqlist.h头文件

```

1  #ifndef __SEQLIST_H__
2  #define __SEQLIST_H__
3
4  #define MAX 20           //数组长度
5  typedef int datatype;
6
7  typedef struct
8  {
9      datatype    data[MAX];    //存储顺序表的数组
10     int    len;                //记录顺序表的表长
11 }seqList;
12
13 //函数声明
14 //创建函数是返回堆区申请空间的地址
15 seqList *list_create();
16
17 //判空
18 int list_empty(seqList *L);
19
20 //判满
21 int list_full(seqList *L);
22

```

```

23 //添加元素
24 int list_add(seqList *L, datatype e);
25
26 //定义遍历函数
27 void list_show(seqList *L);
28
29 //任意位置插入元素
30 int list_insert_pos(seqList *L, int pos, datatype e);
31
32 //定义任意位置删除函数
33 int list_delete_pos(seqList *L, int pos);
34
35 //按位置进行修改
36 int list_update_pos(seqList *L, int pos, datatype new_e);
37
38 //按位置进行查找
39 datatype list_search_pos(seqList *L, int pos);
40
41 //按值进行修改
42 int list_update_value(seqList *L, datatype old_e, datatype
    new_e);
43
44 //定义按值查找函数并返回位置
45 int list_search_value(seqList *L, datatype e);
46
47 //定义释放表函数
48 void list_free(seqList *L);
49
50 #endif

```

2> seqlist.c函数文件

```

1 #include"seqlist.h"
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<string.h>
5
6 //函数定义
7 //创建顺序表的实现方式1，不常使用
8 /*
9 void list_create(seqList **L)
10 {
11     *L = (seqList*)malloc(sizeof(seqList));
12 }
13 */

```

```
14
15 //创建函数是返回堆区申请空间的地址
16 seqList *list_create()
17 {
18     seqList *L = (seqList*)malloc(sizeof(seqList));
19     if(NULL == L)
20     {
21         printf("创建失败\n");
22         return NULL;
23     }
24
25     //对顺序表初始化
26     memset(L->data, 0, sizeof(L->data));
27
28     L->len = 0;           //顺序表初始长度为0
29
30     printf("创建成功\n");
31
32     return L;           //将顺序表返回
33 }
34
35 //判空
36 int list_empty(seqList *L)
37 {
38     return L->len==0 ? 1:0;    //1表示空，0表示非空
39 }
40
41
42 //判满
43 int list_full(seqList *L)
44 {
45     return L->len==MAX ? 1:0;    //1表示满，0表示非满
46 }
47
48 //添加元素
49 int list_add(seqList *L, datatype e)
50 {
51     //判断逻辑
52     if(NULL==L || list_full(L))
53     {
54         printf("添加失败\n");
55         return -1;
56     }
57
58     //添加逻辑：在表中第一个没有存放数据的元素中将e放入
```

```

59     L->data[L->len] = e;
60
61     //表的变化
62     L->len++;
63
64     printf("添加成功\n");
65     return 0;
66 }
67
68 //定义遍历函数
69 void list_show(seqList *L)
70 {
71     //判断逻辑
72     if(NULL==L || list_empty(L))
73     {
74         printf("遍历失败\n");
75         return;
76     }
77
78     //遍历逻辑
79     printf("目前表中元素分别是: ");
80     for(int i=0; i<L->len; i++)
81     {
82         printf("%d\t", L->data[i]);
83     }
84     printf("\n");
85 }
86
87 //任意位置插入元素
88 int list_insert_pos(seqList *L, int pos, datatype e)
89 {
90     //判断逻辑
91     if(NULL==L || list_full(L) || pos<0 || pos>L->len)
92     {
93         printf("插入失败\n");
94         return -1;
95     }
96
97     //腾空逻辑
98     for(int i=L->len-1; i>=pos; i--)
99     {
100         L->data[i+1] = L->data[i];    //将当前元素后移
101     }
102
103     //占空

```

```

104     L->data[pos] = e;
105
106     //表的变化
107     L->len++;
108
109     printf("插入成功\n");
110     return 0;
111 }
112
113
114 //定义任意位置删除函数
115 int list_delete_pos(seqList *L, int pos)
116 {
117     //判断逻辑
118     if(NULL==L || list_empty(L) || pos<0 || pos>=L->len)
119     {
120         printf("删除失败\n");
121         return -1;
122     }
123
124     //删除逻辑
125     for(int i=pos+1; i<L->len; i++)
126     {
127         L->data[i-1] = L->data[i];    //将元素前移动
128     }
129
130     //表的变化
131     L->len--;
132
133     printf("删除成功\n");
134     return 0;
135 }
136
137 //按位置进行修改
138 int list_update_pos(seqList *L, int pos, datatype new_e)
139 {
140     //判断逻辑
141     if(NULL==L || list_empty(L) || pos>=L->len || pos<0)
142     {
143         printf("修改失败\n");
144         return -1;
145     }
146
147     //修改逻辑
148     L->data[pos] = new_e;

```



```

149
150     printf("修改成功\n");
151     return 0;
152
153 }
154
155 //按位置进行查找
156 datatype list_search_pos(seqList *L, int pos)
157 {
158     //判断逻辑
159     if(NULL==L || list_empty(L) || pos>=L->len || pos<0)
160     {
161         printf("查找失败\n");
162         return -1;
163     }
164
165     //将值返回
166     return L->data[pos];
167 }
168
169 //按值进行修改
170 int list_update_value(seqList *L, datatype old_e, datatype
new_e)
171 {
172     //判断逻辑
173     if(NULL==L || list_empty(L))
174     {
175         printf("修改失败\n");
176         return -1;
177     }
178
179     //修改逻辑
180     int flag = 0; //判断是否修改
181     for(int i=0; i<L->len; i++)
182     {
183         if(L->data[i] == old_e) //判断原值是否在表中
184         {
185             L->data[i] = new_e; //修改
186             flag = 1;
187             // break; //只修改第一个找到值
188         }
189     }
190
191     //判断flag
192     if(flag == 0)

```

```

193     {
194         printf("未找到要修改的值\n");
195         return -2;
196     }
197
198     printf("修改成功\n");
199     return 0;
200
201 }
202
203 //定义按值查找函数并返回位置
204 int list_search_value(seqList *L, datatype e)
205 {
206     //判断逻辑
207     if(NULL==L || list_empty(L))
208     {
209         printf("修改失败\n");
210         return -1;
211     }
212
213     //查找逻辑
214     for(int i=0; i<L->len; i++)
215     {
216         if(L->data[i] == e) //判断原值是否在表中
217         {
218             return i; //将下标返回
219         }
220     }
221
222     printf("没有该元素，查找失败\n");
223
224     return -2;
225
226 }
227
228 //定义释放表函数
229 void list_free(seqList *L)
230 {
231     //判断逻辑
232     if(NULL == L)
233     {
234         printf("释放失败\n");
235         return;
236     }
237

```

```
238     free(L);           //将空间释放
239     L = NULL;          //防止野指针
240
241     printf("释放成功\n");
242 }
```

3> main.c测试文件

```
1  #include"seqlist.h"
2  #include<stdio.h>
3  int main(int argc, const char *argv[])
4  {
5      seqList *L = list_create();           //在栈区定义一个变量
6      if(NULL==L)
7      {
8          return -1;
9      }
10
11     //调用添加函数
12     list_add(L, 2);
13     list_add(L, 5);
14     list_add(L, 3);
15
16     //调用遍历函数
17     list_show(L);
18
19     //调用任意位置插入函数
20     list_insert_pos(L, 3, 10);
21     list_show(L);
22
23     //调用任意位置删除函数
24     list_delete_pos(L, 1);
25     list_show(L);
26
27     //调用修改函数
28     list_update_pos(L, 2, 100);
29     list_show(L);
30
31     //测试按位置查找函数
32     datatype res = list_search_pos(L, 1);
33     if(res != -1)
34     {
35         printf("下标为1的元素值为: %d\n", res);
36     }
37 }
```

```

38
39 //调用按值修改函数
40 list_update_value(L, 20, 5);
41 list_show(L);
42
43 //调用按值查找函数
44 int res1 = list_search_value(L, 7);
45 if(res1>=0)
46 {
47     printf("查找成功，您要找的是第%d个元素\n", res1+1);
48 }
49
50 //调用释放函数
51 list_free(L);
52 L = NULL; //防止主函数中L野指针
53
54 list_show(L);
55
56 return 0;
57 }

```

作业：

作业1：实现顺序表的冒泡排序

```

1 void list_sort(seqList *L)
2 {
3     //判断逻辑
4     if(NULL==L || list_empty(L))
5     {
6         printf("排序失败\n");
7         return ;
8     }
9
10    //排序
11    int i,j; //循环变量
12    datatype temp; //交换变量
13
14    for(i=1; i<L->len; i++) //控制趟数
15    {
16        for(j=0; j<L->len-i; j++) //元素及比较次数
17        {
18            if(L->data[j]>L->data[j+1]) //大升小降
19            {
20                //交换三部曲

```

```

21         temp = L->data[j];
22         L->data[j] = L->data[j+1];
23         L->data[j+1] = temp;
24     }
25 }
26 }
27 printf("排序成功\n");
28 }

```

作业2：实现顺序表去重 例如顺序表元素：1 2 3 3 3 4 2 调用去重函数后：1 2 3 4

```

1 void list_del_repeat(seqList *L)
2 {
3     //判断逻辑
4     if(NULL==L || L->len<=1)
5     {
6         printf("去重失败\n");
7         return;
8     }
9
10    //去重逻辑
11    int i,j;           //循环变量
12    for(i=0; i<L->len; i++) //遍历所有元素
13    {
14        for(j=i+1; j<L->len; j++) //从当前元素下一个到最后，遍历一
15        遍
16        {
17            if(L->data[i] == L->data[j])
18            {
19                list_delete_pos(L, j); //删除下标为j的元素
20                j--; //防止漏删
21            }
22        }
23    }
24    printf("去重成功\n");
25 }

```

作业3：实现顺序表按值删除函数（可以使用已经实现的函数完成，按值找到位置，按位置删除元素）

```

1 int list_del_value(seqList *L, datatype e)
2 {
3     //判断逻辑

```

```
4     if(NULL==L || list_empty(L))
5     {
6         printf("删除失败\n");
7         return -1;
8     }
9     //删除逻辑
10    int pos = list_search_value(L, e);
11    if(pos >= 0)
12    {
13        list_delete_pos(L, pos);
14        return 0;
15    }else
16    {
17        printf("删除失败\n");
18        return -2;
19    }
20
21 }
```

作业4：使用Xmind思维导图，将今天所学内容画出来，尽可能细致