20220422-书&C++

1.过程描述

1.1 Complications—书

> In the end, it is practical, everyday medicine that most interests me—what happens when the simplicities of science come up against the complexities of individual lives.

简单的科学遇到复杂的个体与生命

> Every day, surgeons are faced with uncertainties. Information is inadequate; the science is ambiguous; one's knowledge and abilities are never perfect. Even with the simplest operation, it cannot be taken for granted that a patient will come through better off—or even alive.

医生的知识也是有限的。医学是一门不完美的科学。

> the most important way in which innate factors play a role may be in one's willingness to engage in sustained trainin

坚持长期持续的训练

> I still have no idea what I did differently that day. But from then on, my lines went in. Practice is funny that way. For days and days, you make out only the fragments of what to do. And then one day you’ve got the thing whole. Conscious learning becomes unconscious knowledge, and you cannot say precisely how.

刻意的学习变成无意间收获的知识

> “There should be no learning curve as far as patient safety is concerned.” But that is entirely wishful thinking.

医生有时候不得不通过在病人身上实践来积累经验。固然所有的病人都理应得到最好的治疗，但这只不过是美好的幻想。

1.2 JSON解析器实现

leptjson.h C++ 复制代码

```cpp
#pragma once
#ifndef LEPTJSON_H_
#define LEPTJSON_H_

/*
JSON是一种树状结构，包含六种数据类型，把ture和false当作两个类型则有七种
*/
typedef enum
{
    LEPT_NULL,
    LEPT_FALSE,
    LEPT_TRUE,
    LEPT_NUMBER,
    LEPT_STRING,
    LEPT_ARRAY,
    LEPT_OBJECT
}lept_type;

/*
我们最终需要实现一个树的数据结构，
每个节点使用lept_value结构体表示，称它为一个JSON值
*/
typedef struct
{
    lept_type type;
}lept_value;

/*
此单元使用的JSON语法子集
JSON-text=ws value ws
ws=*(%x20 / %x09 / %x0A / %x0D)
value=null/false/true
null="null"
false="false"
true="true"
第一行的意思是：JSON文本由3部分组成，首先是whitespace，然后是值，最后是whitespace
第二行告诉我们：所谓空白，是由零或多个空格符、制表符、换行符、回车符所组成
%xhh表示以十六进制表示的字符，/是多选一，*是零或多个，()用于分组
第三行是说，我们现在的值只可以是null，false，或者true
我们的解析器应能判断输入是否是一个合法的JSON，如果输入的JSON不合法，我们要产生对应的错误码
在这个JSON语法子集下，我们定义3中错误码：
1）若一个JSON只含有空白，传回LEPT_PARSE_EXPECT_VALUE；
2）若一个值后，在空白之后还有其它字符，传回LEPT_PARSE_ROOT_NOT_SINGULAR；
3）若值不是那三种字面值，传回LEPT_PARSE_INVALID_VALUE.
```

```
45  */
46
47  enum
48  {
49      LEPT_PARSE_OK = 0,
50      LEPT_PARSE_EXPECT_VALUE,
51      LEPT_PARSE_INVALID_VALUE,
52      LEPT_PARSE_ROOT_NOT_SINGULAR
53  };
54  /*
55  API函数:
56  1.解析JSON,传入的JSON文本是一个C字符串，传入的根节点指针v由使用方负责分配
57  一般用法是:
58  lept_value v;
59  const char json[]=...;
60  int ret=lept_parse(&v,json);
61  返回值是上面的枚举值，无错误会返回LEPT_PARSR_OK
62
63  2.一个访问结果的函数，获取其类型
64  */
65  int lept_parse(lept_value* v, const char* json);
66  lept_type lept_get_type(const lept_value* v);
67
68  #endif
```

leptjson.cpp C++ 复制代码

```cpp
#include "leptjson.h"
#include <stdlib.h>
#include <assert.h>

#define EXPECT(c,ch) do{assert(*c->json==(ch));c->json++;} while(0)

typedef struct
{
    const char* json;
}lept_context;


//C语言用static声明函数，表示该函数名除了对该函数声明的文件可见外，其它文件都无法访问
//即只能被本文件中的函数调用，而不能被同一程序中的其它文件访问


static void lept_parse_whitespace(lept_context* c)
{
    const char* p = c->json;
    while (*p == ' ' || *p == '\t' || *p == '\n' || *p == '\r')
        p++;
    c->json = p;
    //这里主要是检查如果出现空白（可能由空格符、制表符、换行符、回车符组成）
    //则将指针一直加到不为上面的那些符号，并将指针赋给c->json
}

static int lept_parse_null(lept_context* c, lept_value* v)
{
    EXPECT(c, 'n');
    if (c->json[0] != 'u' || c->json[1] != 'l' || c->json[2] != 'l')
        return LEPT_PARSE_INVALID_VALUE;
    c->json += 3;
    v->type = LEPT_NULL;
    return LEPT_PARSE_OK;//当传入的文本为null时，则表示解析通过
}


static int lept_parse_value(lept_context* c, lept_value* v)
{
    switch (*c->json)
    {
    case '\n': return lept_parse_null(c,v);
    case '\0': return LEPT_PARSE_EXPECT_VALUE;
    default:   return LEPT_PARSE_INVALID_VALUE;
    }
```

```
46  }
47
48  int lept_parse(lept_value* v, const char* json)
49  {
50      lept_context c;
51      assert(v != NULL);
52      c.json = json;
53      v->type = LEPT_NULL;
54      lept_parse_whitespace(&c);
55      return lept_parse_value(&c, v);
56  }
57
58
59  lept_type lept_get_type(const lept_value* v)
60  {
61      assert(v != NULL);
62      //assert是C语言中常用的防御式编程方式，用于减少编程错误
63      //当程序以release配置编译时，assert不会做检测；
64      //当程序以debug配置编译时，则会在运行时检测assert(con)是否为真，断言失败会直接
    令程序崩溃
65      return v->type;
66      //这里会返回json的类型（h文件中的7种）
67  }
```

test.cpp C++ 复制代码

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "leptjson.h"

static int main_ret = 0;
static int test_count = 0;
static int test_pass = 0;

#define EXPECT_EQ_BASE(equality,expect,actual,format) \
    do{\
        test_count++;\
        if(equality)\
            test_pass++;\
        else{\
            fprintf(stderr,"%s:%d: expect: " format " actual: " format "\n",__FILE__,__LINE__,expect,actual);\
            main_ret=1;\
        }\
    } while(0)

#define EXPECT_EQ_INT(expect,actual) EXPECT_EQ_BASE((expect)==(actual),expect,actual,"%d")

static void test_parse_null()
{
    lept_value v;
    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_OK, lept_parse(&v, "null"));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));
}

static void test_parse_expect_value()
{
    lept_value v;
    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_EXPECT_VALUE, lept_parse(&v, ""));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));

    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_EXPECT_VALUE, lept_parse(&v, " "));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));
}

static void test_parse_invalid_value()
```

```
{
    lept_value v;
    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_INVALID_VALUE, lept_parse(&v, "nul"));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));

    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_INVALID_VALUE, lept_parse(&v, "?"));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));
}

static void test_parse_root_not_singular()
{
    lept_value v;
    v.type = LEPT_FALSE;
    EXPECT_EQ_INT(LEPT_PARSE_ROOT_NOT_SINGULAR, lept_parse(&v,"null x"));
    EXPECT_EQ_INT(LEPT_NULL, lept_get_type(&v));
}


static void test_parse()
{
    test_parse_null();
    test_parse_expect_value();
    test_parse_invalid_value();
    test_parse_root_not_singular();
}

int main()
{
    test_parse();
    printf("%d/%d (%3.2f%%) passed\n", test_pass, test_count, test_pass * 100.0 / test_count);
    return main_ret;
}

```

2.结果输出

今天上午看了complication一书，是一个医生写的关于自己行医生涯的一些启示，感觉初读下来还挺有意思的，后面估计会继续读下去。下午看了用C++实现ML一书，只大概摸索了以下Eigen库的一些使用，还没深入下去；此外还看了一点C++ API设计原则一书，也只是刚读完关于API的优缺点的一点Introduction。书附带的一些示例代码估计挺值得学习的，后面应该也会接着读下去。晚上突然想起有

个挺有意思的JSON解析器教程还没看，便又又把tutorial01完成了下，没完全理解。感觉还是老毛病，对啥都挺有兴趣的，但鲜少能坚持下来把一件事情做完整。