1.题目名称

树的应用

2.代码行数

134行

3.算法思想

处理json数据，根节点为key，子节点为value。遇到'{'则循环到'}'对内部的字符串调用自身函数，执行相同的操作，返回处理过后的根节点即可。

寻找的时候，按','进行分割记作cp,cq，如果cq为空，直接返回值，否则调用自身传入object为cp的节点以及对应cq字符串。

4.主要/核心函数分析

Deal

```
Tree* Deal(string json,string ct){          //处理Json数据
    Tree *root= new Tree;
    root->data=ct;
    int k=0;
    if(json[0]=='{'){
        json.erase(json.begin());
    }
    if(json[json.size()-1]=='}'){
        json.erase(json.end()-1);
    }
    Tree *p=new Tree;
    while(k<json.size()){
        if(json[k]=='{'){
            string temp="";
            k+=1;
            while(json[k]!='}'){
                temp+=json[k];
                k++;
            }
            k++;
            root->Child.push_back(Deal(temp,p->data));  //进行递归,遍历该json
            p=new Tree;
        }
        if(json[k]=='"'){
            k+=1;
            while(json[k]!='"'){
                if(json[k]=='\\'){
                    k++;
                    p->data+=json[k];
                }else
                    p->data+=json[k];
                k++;
            }
            k++;
```

```
        }
        if(json[k]==':'){
            k+=2;
            Tree *q=new Tree;
            if(json[k]=='"'){
                k+=1;
                while(json[k]!='"'){
                    if(json[k]=='\\'){
                        k++;
                        q->data+=json[k];
                    }else
                        q->data+=json[k];
                    k++;
                }
                k++;
                p->Child.push_back(q);
                root->Child.push_back(p);
                p=new Tree;
            }
        }

        if(json[k]==' ' || json[k]==','){
            k++;
            continue;
        }

    }
    return root;
}
```

处理json数据，根节点为key，子节点为value。遇到'{'则循环到'}'对内部的字符串调用自身函数，执行相同的操作，返回处理过后的根节点即可。还有就是要单独处理转义字符。

findValue

```
void findValue(Tree* root, const std::string& ct) { //查找相关值
    string cp = "", cq = "";
    int flag=0;
    for (int j = 0; j < ct.size(); j++) {
        if (ct[j] == '.') {
            flag = 1;
            continue;
        }
        if (!flag) {
            cp += ct[j];
        } else {
            cq += ct[j];
        }
    }
    for (int j = 0; j < root->Child.size(); j++) {
        if (root->Child[j]->data == cp) {
            if (cq == "") {
                if (root->Child[j]->Child[0]->Child.size() > 0) {
                    std::cout << "OBJECT" << std::endl;
                    return;
                } else {
```

```
                    std::cout << "STRING" << " " << root->Child[j]->Child[0]->data << std::endl;
                    return;
                }
            } else {
                findValue(root->Child[j], cq);  //进入Json寻找
                return;
            }
        }
    }
    cout<<"NOTEXIST"<<endl;
}

```

寻找的时候，按'.'进行分割记作cp,cq，如果cq为空，直接返回值，否则调用自身传入object为cp的节点以及对应cq字符串。

5.测试数据(规模,测试次数)

规模:　字符串 (string)&&对象 (object)

测试次数:1

测试用例:见测试文件

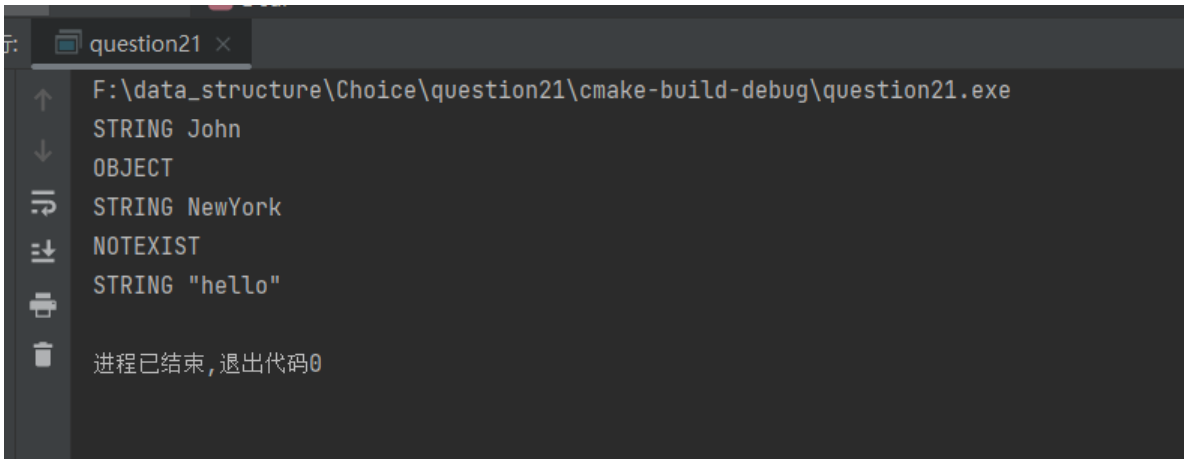
6.运行结果

```
F:\data_structure\Choice\question21\cmake-build-debug\question21.exe
STRING John
OBJECT
STRING NewYork
NOTEXIST
STRING "hello"

进程已结束,退出代码0

```

7.时间复杂度分析

时间复杂度取决于表达式的长度，因此为O(n)。

8.结果截屏图片



9.心得体会

对于Json的处理及其value的查找更加了解与熟悉。