【华为OD机考 统一考试机试C卷】模拟目录管理功能（C++ Java JavaScript Python）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

实现一个模拟目录管理功能的软件，输入一个命令序列，输出最后一条命令运行结果。

支持命令：

- 创建目录命令：mkdir 目录名称，如 mkdir abc 为在当前目录创建abc目录，如果已存在同名目录则不执行任何操作。此命令无输出。
- 进入目录命令：cd 目录名称，如 cd abc 为进入abc目录，特别地，cd .. 为返回上级目录，如果目录不存在则不执行任何操作。此命令无输出。
- 查看当前所在路径命令：pwd，输出当前路径字符串。

约束：

- 目录名称仅支持小写字母；mkdir 和 cd 命令的参数仅支持单个目录，如：mkdir abc 和 cd abc；不支持嵌套路径和绝对路径，如 mkdir abc/efg，cd abc/efg，mkdir /abc/efg，cd /abc/efg 是不支持的。
- 目录符号为/，根目录/作为初始目录。
- 任何不符合上述定义的无效命令不做任何处理并且无输出。

输入描述

输入 N 行字符串，每一行字符串是一条命令。

命令行数限制100行以内，目录名称限制10个字符以内。

输出描述

输出最后一条命令运行结果字符串。

用例

输入

```
mkdir abc
cd abc
pwd
```

输出

```
/abc/
```

说明

> 在根目录创建一个abc的目录并进入abc目录中查看当前目录路径，输出当前路径/abc/。

解题思路

1. 定义一个节点类（Node），用于表示文件系统中的每个目录。该类包含路径信息和一个映射，映射存储子目录和对应的节点对象。
2. 创建一个根节点实例，代表文件系统的根目录。根目录没有父目录。
3. 读取用户输入，根据输入的命令和参数执行相应的操作。
    - 如果输入的是创建目录的命令（例如，“mkdir”），检查目录名是否有效，然后在当前节点下创建新的子目录节点。
    - 如果输入的是切换目录的命令（例如，“cd”），检查目标目录是否存在，如果存在，则更新当前节点为目标节点。
    - 如果输入的是打印当前目录路径的命令（例如，“pwd”），则输出当前节点的路径信息。
4. 循环读取输入直到结束，并在结束时输出最后的路径信息。

C++

```cpp
#include <iostream>
#include <string>
#include <unordered_map>
#include <sstream>

using namespace std;
// 定义一个类Node，用于表示文件系统中的每个目录
class Node {
public:
    string path; // 目录的路径
    unordered_map<string, Node*> next; // 存储当前目录下的子目录，键为目录名，值为对应的Node指针

    // Node类的构造方法
    Node(string path, Node* parent) : path(path) {
        // 如果存在父目录，则在子目录映射中添加一个指向父目录的条目
        if (parent != nullptr) {
            this->next[".."] = parent;
        }
    }
};

// 检查目录名是否有效的函数，目录名只能包含小写字母
bool isValidDirectoryName(const string& name) {
    for (char c : name) {
        if (c < 'a' || c > 'z') {
            return false; // 如果目录名中包含非小写字母的字符，则返回false
        }
    }
    return true; // 如果目录名全部由小写字母组成，则返回true
}

// 检查是否可以切换到指定的目录的函数，目录名要么是有效的，要么是".."表示上级目录
bool isValidChangeDirectory(const string& name) {
    return name == ".." || isValidDirectoryName(name); // 如果是".."或者是有效的目录名，则返回true
}

int main() {
    Node* root = new Node("/", nullptr); // 创建根目录节点，根目录没有父目录，所以第二个参数为nullptr

```

```
    Node* currentNode = root; // 初始化当前目录为根目录
    string lastOutput; // 用于存储最后输出的路径

    // 循环读取用户输入的命令
    string input;
    while (getline(cin, input)) {
        istringstream iss(input);
        string command, arg;
        iss >> command;

        if (command == "mkdir") {
            iss >> arg;
            if (isValidDirectoryName(arg)) {
                // 如果目录名有效并且不存在，则创建一个新的目录节点，并将其添加到当前目录的子目录映射中
                if (currentNode->next.find(arg) == currentNode->next.end()) {
                    currentNode->next[arg] = new Node(currentNode->path + arg + "/", currentNode);
                }
            }
        } else if (command == "cd") {
            iss >> arg;
            if (isValidChangeDirectory(arg)) {
                // 处理cd命令，用于改变当前目录
                auto it = currentNode->next.find(arg);
                if (it != currentNode->next.end()) {
                    currentNode = it->second; // 如果目录存在，则将当前目录切换为该目录
                }
            }
        } else if (command == "pwd") {
            // 处理pwd命令，用于打印当前目录的路径
            lastOutput = currentNode->path; // 将当前目录的路径保存到lastOutput变量中
        }
    }

    cout << lastOutput << endl; // 循环结束后，打印最后保存的路径

    // 清理动态分配的内存
    // 注意：这里需要实现一个递归删除函数来正确释放所有分配的Node
    // 为简洁起见，这里省略了实现细节
```

```
        return 0;
    }
```

Java

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    // 定义一个内部类Node，用于表示文件系统中的每个目录
    static class Node {
        String path; // 目录的路径
        Map<String, Node> next = new HashMap<>(); // 存储当前目录下的子目录，键为目录名，值为对应的Node对象

        // Node类的构造方法ac
        Node(String path, Node parent) {
            this.path = path; // 设置当前节点的路径
            // 如果存在父目录，则在子目录映射中添加一个指向父目录的条目
            if (parent != null) {
                this.next.put("..", parent);
            }
        }
    }

    // 程序的主入口点
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // 创建Scanner对象来读取用户的输入
        Node root = new Node("/", null); // 创建根目录节点，根目录没有父目录，所以第二个参数为null
        Node currentNode = root; // 初始化当前目录为根目录
        String lastOutput = ""; // 用于存储最后输出的路径

        // 循环读取用户输入的命令
        while (scanner.hasNextLine()) {
            String input = scanner.nextLine().trim(); // 读取一行输入并去除前后空格
            if (input.isEmpty()) break; // 如果输入为空，则退出循环

            String[] parts = input.split(" "); // 将输入的命令按空格分割为命令和参数
            String command = parts[0]; // 获取命令部分

```

```
            // 处理mkdir命令，用于创建新的子目录
            if ("mkdir".equals(command) && parts.length == 2 && isValidDirectoryName(parts[1])) {
                // 如果目录名有效并且不存在，则创建一个新的目录节点，并将其添加到当前目录的子目录映射中
                currentNode.next.putIfAbsent(parts[1], new Node(currentNode.path + parts[1] + "/", currentNode));
            } else if ("cd".equals(command) && parts.length == 2 && isValidChangeDirectory(parts[1])) {
                // 处理cd命令，用于改变当前目录
                Node nextNode = currentNode.next.get(parts[1]); // 从子目录映射中获取要切换的目录节点
                if (nextNode != null) {
                    currentNode = nextNode; // 如果目录存在，则将当前目录切换为该目录
                }
            } else if ("pwd".equals(command) && parts.length == 1) {
                // 处理pwd命令，用于打印当前目录的路径
                lastOutput = currentNode.path; // 将当前目录的路径保存到lastOutput变量中
            }
        }

        System.out.println(lastOutput); // 循环结束后，打印最后保存的路径
    }

    // 检查目录名是否有效的方法，目录名只能包含小写字母
    private static boolean isValidDirectoryName(String name) {
        for (char c : name.toCharArray()) {
            if (c < 'a' || c > 'z') {
                return false; // 如果目录名中包含非小写字母的字符，则返回false
            }
        }
        return true; // 如果目录名全部由小写字母组成，则返回true
    }

    // 检查是否可以切换到指定的目录的方法，目录名要么是有效的，要么是".."表示上级目录
    private static boolean isValidChangeDirectory(String name) {
        return "..".equals(name) || isValidDirectoryName(name); // 如果是".."或者是有效的目录名，则返回true
    }
}
```

javaScript

```
const readline = require('readline');

// 定义一个类Node，用于表示文件系统中的每个目录
```

```
class Node {
    constructor(path, parent) {
        this.path = path; // 目录的路径
        this.next = {}; // 存储当前目录下的子目录，键为目录名，值为对应的Node对象
        if (parent) {
            this.next['..'] = parent; // 如果存在父目录，则在子目录映射中添加一个指向父目录的条目
        }
    }
}

// 检查目录名是否有效的函数，目录名只能包含小写字母
function isValidDirectoryName(name) {
    return /^[a-z]+$/.test(name); // 如果目录名全部由小写字母组成，则返回true
}

// 检查是否可以切换到指定的目录的函数，目录名要么是有效的，要么是".."表示上级目录
function isValidChangeDirectory(name) {
    return name === '..' || isValidDirectoryName(name); // 如果是".."或者是有效的目录名，则返回true
}

// 创建readline接口实例
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const root = new Node('/', null); // 创建根目录节点，根目录没有父目录，所以第二个参数为null
let currentNode = root; // 初始化当前目录为根目录
let lastOutput = ''; // 用于存储最后输出的路径

// 逐行读取输入
rl.on('line', (input) => {
    const parts = input.trim().split(' '); // 将输入的命令按空格分割为命令和参数
    const command = parts[0]; // 获取命令部分

    if (command === 'mkdir' && parts.length === 2 && isValidDirectoryName(parts[1])) {
        // 处理mkdir命令，用于创建新的子目录
        if (!currentNode.next[parts[1]]) {
            currentNode.next[parts[1]] = new Node(currentNode.path + parts[1] + '/', currentNode);
        }
```

```
    } else if (command === 'cd' && parts.length === 2 && isValidChangeDirectory(parts[1])) {
        // 处理cd命令，用于改变当前目录
        const nextNode = currentNode.next[parts[1]];
        if (nextNode) {
            currentNode = nextNode; // 如果目录存在，则将当前目录切换为该目录
        }
    } else if (command === 'pwd' && parts.length === 1) {
        // 处理pwd命令，用于打印当前目录的路径
        lastOutput = currentNode.path; // 将当前目录的路径保存到lastOutput变量中
    }
}).on('close', () => {
    console.log(lastOutput); // 当输入流关闭时，打印最后保存的路径
});
```

Python

```
# 定义一个类Node，用于表示文件系统中的每个目录
class Node:
    def __init__(self, path, parent):
        self.path = path  # 目录的路径
        self.next = {}  # 存储当前目录下的子目录，键为目录名，值为对应的Node对象
        if parent:
            self.next['..'] = parent  # 如果存在父目录，则在子目录映射中添加一个指向父目录的条目

# 检查目录名是否有效的函数，目录名只能包含小写字母
def is_valid_directory_name(name):
    return name.islower() and name.isalpha()  # 如果目录名全部由小写字母组成，则返回true

# 检查是否可以切换到指定的目录的函数，目录名要么是有效的，要么是".."表示上级目录
def is_valid_change_directory(name):
    return name == '..' or is_valid_directory_name(name)  # 如果是".."或者是有效的目录名，则返回true

root = Node('/', None)  # 创建根目录节点，根目录没有父目录，所以第二个参数为None
current_node = root  # 初始化当前目录为根目录
last_output = ''  # 用于存储最后输出的路径

# 循环读取用户输入的命令
try:
    while True:
        input_command = input().strip()  # 读取一行输入并去除前后空格
```

```
        if not input_command:
            break
        parts = input_command.split(' ')  # 将输入的命令按空格分割为命令和参数
        command = parts[0]  # 获取命令部分

        if command == 'mkdir' and len(parts) == 2 and is_valid_directory_name(parts[1]):
            # 处理mkdir命令，用于创建新的子目录
            if parts[1] not in current_node.next:
                current_node.next[parts[1]] = Node(current_node.path + parts[1] + '/', current_node)
        elif command == 'cd' and len(parts) == 2 and is_valid_change_directory(parts[1]):
            # 处理cd命令，用于改变当前目录
            next_node = current_node.next.get(parts[1])
            if next_node:
                current_node = next_node  # 如果目录存在，则将当前目录切换为该目录
        elif command == 'pwd' and len(parts) == 1:
            # 处理pwd命令，用于打印当前目录的路径
            last_output = current_node.path  # 将当前目录的路径保存到last_output变量中
except EOFError:
    pass

print(last_output)  # 打印最后保存的路径
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师