

Design In-Memory File System

Design an in-memory file system to simulate the following functions:

ls : Given a path in string format. If it is a file path, return a list that only contains this file's name. If it is a directory path, return the list of file and directory names **in this directory**. Your output (file and directory names together) should in **lexicographic order**.

mkdir : Given a **directory path** that does not exist, you should make a new directory according to the path. If the middle directories in the path don't exist either, you should create them as well. This function has void return type.

addContentToFile : Given a **file path** and **file content** in string format. If the file doesn't exist, you need to create that file containing given content. If the file already exists, you need to **append** given content to original content. This function has void return type.

readContentFromFile : Given a **file path**, return its **content** in string format.

Example:

Input:

```
["FileSystem","ls","mkdir","addContentToFile","ls","readContentFromFile"]  
[[],["/"],["/a/b/c"],["/a/b/c/d","hello"],["/"],["/a/b/c/d"]]
```

Output:

```
[null,[],null,null,["a"],"hello"]
```

Explanation:

Operation	Output	Explanation
FileSystem fs = new FileSystem()	null	The constructor returns nothing.
fs.ls("/")	[]	Initially, directory <code>/</code> has nothing. So return empty list.
fs.mkdir("/a/b/c")	null	Create directory <code>a</code> in directory <code>/</code> . Then create directory <code>b</code> in directory <code>a</code> . Finally, create directory <code>c</code> in directory <code>b</code> .
fs.addContentToFile("/a/b/c/d","hello")	null	Create a file named <code>d</code> with content <code>"hello"</code> in directory <code>/a/b/c</code> .
fs.ls("/")	["a"]	Only directory <code>a</code> is in directory <code>/</code> .
fs.readContentFromFile("/a/b/c/d")	"hello"	Output the file content.

Note:

1. You can assume all file or directory paths are absolute paths which begin with `/` and do not end with `/` except that the path is just `/`.
2. You can assume that all operations will be passed valid parameters and users will not attempt to retrieve file content or list a directory or file that does not exist.
3. You can assume that all directory names and file names only contain lower-case letters, and same names won't exist in the same directory.

Solution 1

```
public class FileSystem {
    class File {
        boolean isFile = false;
        Map<String, File> children = new HashMap<>();
        String content = "";
    }

    File root = null;

    public FileSystem() {
        root = new File();
    }

    public List<String> ls(String path) {
        String[] dirs = path.split("/");
        File node = root;
        List<String> result = new ArrayList<>();
        String name = "";
        for (String dir : dirs) {
            if (dir.length() == 0) continue;
            if (!node.children.containsKey(dir)) {
                return result;
            }
            node = node.children.get(dir);
            name = dir;
        }

        if (node.isFile) {
            result.add(name);
        }
        else {
            for (String key : node.children.keySet()) {
                result.add(key);
            }
        }

        Collections.sort(result);

        return result;
    }

    public void mkdir(String path) {
        String[] dirs = path.split("/");
        File node = root;
        for (String dir : dirs) {
            if (dir.length() == 0) continue;
            if (!node.children.containsKey(dir)) {
                File file = new File();
                node.children.put(dir, file);
            }
            node = node.children.get(dir);
        }
    }

    public void addContentToFile(String filePath, String content) {
```

```

        public void addContentFromFile(String filePath, String content) {
            String[] dirs = filePath.split("/");
            File node = root;
            for (String dir : dirs) {
                if (dir.length() == 0) continue;
                if (!node.children.containsKey(dir)) {
                    File file = new File();
                    node.children.put(dir, file);
                }
                node = node.children.get(dir);
            }
            node.isFile = true;
            node.content += content;
        }

        public String readContentFromFile(String filePath) {
            String[] dirs = filePath.split("/");
            File node = root;
            for (String dir : dirs) {
                if (dir.length() == 0) continue;
                if (!node.children.containsKey(dir)) {
                    File file = new File();
                    node.children.put(dir, file);
                }
                node = node.children.get(dir);
            }

            return node.content;
        }
    }
}

```

written by [shawngao](#) original link [here](#)

Solution 2

I uses a dictionary of recursing dictionaries to model the file system tree. There's quite a bit of repetition in the code for traversing the file directories but it should still be quite straightforward.

```
def path_split(path):
    return [frag for frag in path.split('/') if frag.strip() != '']

class FileSystem(object):

    def __init__(self):
        self.fs = {}

    def ls(self, path):
        """
        :type path: str
        :rtype: List[str]
        """
        curr = self.fs
        frags = path_split(path)
        for frag in frags:
            if frag not in curr:
                curr[frag] = {}
            curr = curr[frag]
            if type(curr) == unicode:
                return [frags[-1]]
        return sorted(curr.keys())

    def mkdir(self, path):
        """
        :type path: str
        :rtype: void
        """
        curr = self.fs
        frags = path_split(path)
        for frag in frags:
            if frag not in curr:
                curr[frag] = {}
            curr = curr[frag]

    def addContentToFile(self, filePath, content):
        """
        :type filePath: str
        :type content: str
        :rtype: void
        """
        curr = self.fs
        frags = path_split(filePath)
        for frag in frags[:-1]:
            if frag not in curr:
                curr[frag] = {}
            curr = curr[frag]
        file_name = frags[-1]
        if file_name not in curr:
            curr[file_name] = ''
```

```
curr[file_name] =  
curr[file_name] += content
```

```
def readContentFromFile(self, filePath):  
    """  
    :type filePath: str  
    :rtype: str  
    """  
  
    curr = self.fs  
    frags = path_split(filePath)  
    for frag in frags[:-1]:  
        if frag not in curr:  
            curr[frag] = {}  
        curr = curr[frag]  
    file_name = frags[-1]  
    return curr[file_name]
```

```
# Your FileSystem object will be instantiated and called as such:  
# obj = FileSystem()  
# param_1 = obj.ls(path)  
# obj.mkdir(path)  
# obj.addContentToFile(filePath,content)  
# param_4 = obj.readContentFromFile(filePath)
```

written by [yangshun](#) original link [here](#)

Solution 3

We'll keep two structures, `self.fs` being a Trie, and `self.fileinfo` being a dictionary mapping filepaths to the string content in their files. For convenience, we can use a nested defaultdict structure instead of a proper Trie object. This means we should exercise caution as our call to `TrieNode.__getitem__(child)` has potential side effects if `child` is not in the node, but otherwise our code is very similar.

```
Trie = lambda: collections.defaultdict(Trie)

class FileSystem(object):
    def __init__(self):
        self.fs = Trie()
        self.fileinfo = collections.defaultdict(str)

    def ls(self, path):
        if path in self.fileinfo:
            return path.split('/')[1:]

        cur = self.fs
        for token in path.split('/'):
            if token in cur:
                cur = cur[token]
            elif token:
                return []

        return sorted(cur.keys())

    def mkdir(self, path):
        cur = self.fs
        for token in path.split('/'):
            if token: cur = cur[token]

    def addContentToFile(self, filePath, content):
        self.mkdir(filePath)
        self.fileinfo[filePath] += content

    def readContentFromFile(self, filePath):
        return self.fileinfo[filePath]
```

Here is the same solution using a TrieNode object.

```

class Node(object):
    def __init__(self):
        self.children = {}

    def setdefault(self, token):
        return self.children.setdefault(token, Node())

    def get(self, token):
        return self.children.get(token, None)

class FileSystem(object):
    def __init__(self):
        self.root = Node()
        self.fileinfo = collections.defaultdict(str)

    def ls(self, path):
        if path in self.fileinfo:
            return path.split('/')[1:]

        cur = self.root
        for token in path.split('/'):
            if cur and token:
                cur = cur.get(token)

        return sorted(cur.children.keys()) if cur else []

    def mkdir(self, path):
        cur = self.root
        for token in path.split('/'):
            if token: cur = cur.setdefault(token)

```

written by [awice](#) original link [here](#)

From [LeetCoder](#).