SudaOJ 企画

Peterlits Zo

2021 年 1 月 26 日

目录

1 面向 Skokatt

这是一个实现 Skokatt 的企画书，我们把 Skokatt 定义为一个高扩展的内容平台，提供我们称之为 node 的最基本的抽象。

和其他的专门用来做 OJ 的平台不同，我们希望能够提供一个有着丰富语义的抽象层，它可以保证我们能够得到的不仅仅是一个 OJ 系统，而更是一个内容系统：

- 分布式系统，没有客户端和服务端之分，但是我们选择一个特殊的分布式系统节点作为其他节点的上游。更多关于分布式的内容可以查阅 **??** 章。
- 记录所有历史，我们在最基本的抽象 node 中提供了以 commit object 为基础的轻量级历史系统。在基本概念上和 git 中最基本的概念保持一致。
- 高扩展性，可以保证后面围绕着 Skokatt 上不同类型的 node 的快速开发，正因为一开始设计时便确保了其构架的松耦合，才能确保越来越复杂的横向功能的基础上依然能够确保开发的井井有条。
- 向后兼容性，可以确保后期 Skokatt 即使发生了比较大的改变依然能够向后兼容，这正是因为所有的 node 都保留了 Skokatt 的版本信息，可以防止因为版本更改而产生的错误，也可以支持后期的大幅度更改。

我们需要支持个人页面，题面，题集，和提交页面等等。而这些基本措施正是基于 node 上进行开发的。

我们基本使用 JavaScript 来进行开发，需要先设计出能运行在桌面的 Electron 应用，再争取使得一些 node 可以在脱离操作系统 API 的情况下运行，从而得到能够运行在浏览器上的网页。

在后端我们需要使用 Java 进行开发，但不是最重要的地方，我们在基于分布式身份验证的基础上进行存储身份信息，在保证能够确认身份的情况下储存一系列基本用户信息而已即可。

基本的时间表应该如下：

2021/1/22 - 2021/1/27	学习 Java 和 Vue.js 基本语法
2021/1/28 - 2021/2/10	相关的 Electron 开发
2021/2/10 - 2021/2/20	相关的 Java 后端开发
2021/2/20 - 2021/2/30	在浏览器中运行子集

2 分布式

为了支持一个高扩展的 OJ 系统，我们需要支持分布式系统，提高所有节点的参与度，总体来说采用基于分布式的原因如下：

- 所有用户可以（甚至是离线的情况下！）对内容进行修改（但是上游是否采用需要通过一系列决策：包括拉取申请的提交者的 IP 是否在封闭期间？提交的新内容的文件格式是否指定需要审查？指定文件是否默认不可更改？）

2.1 分布式系统

SudaOJ 应该建立在一个分布式系统上。为了实现分布式存储的需求，有两种不同的存储方案：

- 类似 Git 的分布式存储，基于操作系统的文件系统，使用相对路径完成定位。同时所有的节点地位相同，但是可以选取一个作为上游进行集中式存储。
- 使用类似 Ceph 的分布式数据库储存系统，可以支持大规模扩展和较高性能。

综合考虑我们选择第一个，这是因为以下的这两点原因：

- 易于实现和维护，开发者和维护者只需要了解 SudaOJ 的基本概念和编程语言所需要的访问操作系统的文件系统 API 的相关知识即可。
- 更高的兼容性。为了应对每个用户可能不一样的操作系统，使用文件系统来进行存储可以避免数据库软件在不同的操作系统上分布式数据库的兼容性不同而带来的一系列问题。

2.2 全局唯一编码

我们将所有的数据都抽象为一个 node，提出在 SudaOJ 中“一切皆 node”，在此基础上我们需要保证所有的 node 有着全局唯一的编码，以避免冲突。我们有两种不同的 node 编码方式：

- 类似 Git 的哈希值，由内容决定编码。
- 使用标准 UUID，可以保证所有信息的编码两两互不相同。

我们采取第二种编码，这是因为和 Git 不一样，我们不可能在本地储存下所有的信息，其他的 node 我们需要通过 UUID 向上游请求拉取，而因此 node 之间的互相联系的需求，就决定了即使内容发生变化，编码也不能发生改动。

但是我们的 UUID 只是用来进行标记 node 的，node 中的具体内容并不需要一定要容纳在文件中，我们把 node 对应的文件内容存储在 `objects` 对象库里，对象库里大体上分为三种对象：分别为提交对象（commit object），树对象（tree object）和块对象（blob object）。

而对象的哈希我们则采用第一种方法，这种好处完全不言自喻。

2.3 node 格式

考虑到向后兼容等一系列要求，我们让 node 的格式作为一个文件夹暴露出来，举一个例子：

```
.SudaOJ
    +- nodes
    |   +- xx
    |       +- xxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
    +- objects
    |   +- xx1
    |   |   +- xxxxxxxxxxxxxx1
    |   +- xx2
    |       +- xxxxxxxxxxxxxx2
    +- ...
```

在数据根文件夹 `.SudaOJ` 下储存着 node 库 `nodes`，每一个 node 都由一个全局唯一的 UUID 码标识，比如 UUID 码 `xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx` 标识着其对应的对象，可以通过对应的路径进行访问。而每一个对象对应的文件内容应该如下：

```
1
SudaOJ:MD:1
xx1xxxxxxxxxxxxxx1
```

其中第一行为版本号，之后分别是解析格式和入口提交，每一行都使用 ASCII 码中的回车符 `\n` 进行分割。

其中第一行为十六进制数字，标识着系统版本号，截止本文时默认为 1；解析格式应该约定俗成地使用符号 ':' 将各个命名空间隔开，最后一部分约定为格式版本，比如说所有 SudaOJ 自定义的格式都是位于命名空间 SudaOJ 中的，当前的文件使用 SudaOJ:MD:1 规则来进行解析生成对应 HTML 文件。

node 应该支持以下格式：

- SudaOJ:MD:1 。
- SudaOJ:Question:1 。

node 文件中除了前两行，其余的代表了所有的提交对象（ commit object ，储存在 objects ），

3 node 对象模型

我们在 ?? 节中提到了 node 如何储存（即 node 的所有入口储存在 nodes 文件夹中，而所有的 node 的子组成部分，即各种对象则储存在 objects 中。我们必须要解释 node 中三大对象，即提交对象（ commit object ），树对象（ tree object ）和块对象（ blob object ）中的具体作用。

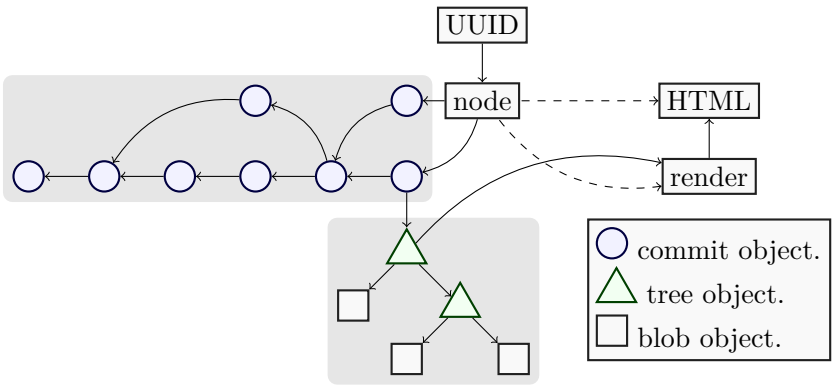


图 1: node 对象模型

而具体的实现方式可以参考样例图 ?? 。

正如 ?? 图所指出的一样，我们通过 UUID 可以查找到 node 文件，它指定了它的文件版本，渲染器（ render ，比如说 SudaOJ:MD:1 ），和以提交对象索引所表示的历史。

提交历史使用提交对象构成的一个有向无环图来进行表示，使用内容来进行哈希，可以有效保证了 node 历史，避免无法溯源的情况，也可以更加灵活地完成 node 的快速开发 [1] 。

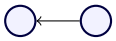
3.1 示例

举一个例子，爱丽丝想要新建一个 node ，用来编辑她的关于某道 SudaOJ 上的博弈论题目，她使用全局唯一的 UUID 生成了一个全新的 node ，没有任何历史提交，除了一个空文件夹。她遵守着 SudaOJ:MD:1 的渲染规则，在空文件夹下新建了一个文件 index.md ，并进行了相关的编辑：

```
1 关于 SudaOJ:1024 题的题解
2 ================================================================================
3
4 我认为有手就行。
```

[1] 这种想法完全借鉴于 git 如何管理项目历史，并经过了多个大型开源项目的开发的检验，比如著名的、以复杂度和高可靠性和高效性所闻名的 Linux 系统，我相信这种基于有向无环图的历史抽象模型能够同样作用于本项目。

她将这次修改分为两次提交（对应地生成了两个提交对象，值得注意的是第一个提交对象没有父节点）并上传到了 SudaOJ 的平台上，现在有版本历史有：



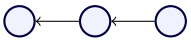
鲍勃也注意到了爱丽丝的提交，他为她添加上了关于这道题的链接：

```
关于 SudaOJ:1024 题的题解
================================================================================

[原链接](oj.suda.edu.cn/node?UUID=xxx)

我认为有手就行。
```

鲍勃进行了提交，现在的版本历史为：

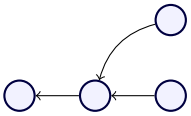


与此同时，爱丽丝觉得不太礼貌，修改了文档：

```
关于 SudaOJ:1024 题的题解
================================================================================

我认为有手就行（不是瞧不起没有手的残疾人的意思）。
```

目前的版本历史为：



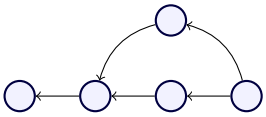
爱丽丝提交上去了之后发现她的 node 提示了多分支警告，爱丽丝知道 node 不应该有多个分支，所以她更倾向于将它们进行合并：

```
关于 SudaOJ:1024 题的题解
================================================================================

[原链接](oj.suda.edu.cn/node?UUID=xxx)

我认为有手就行（不是瞧不起没有手的残疾人的意思）。
```

这个提交和之前的提交都不一样，它同时有两个（甚至有时候会有更多个!）父节点。我们可以根据这个来进行作图表示当前的版本历史：



4 SudaOJ 子集

基于前端都是通过 Skogkatt 来实现的，SudaOJ 子集应该实现的是一个可访问的服务。

4.1 SudaOJ Judger

我们实现一个用来判题的 Judger[2]，它需要接受以下参数，并返回运行值：

[2]可以参考 C 语言版本的 Judger - https://github.com/QingdaoU/Judger

max_cpu_time	CPU 最大运行时间
max_real_time	最大实际运行时间
max_memory	最大使用内存

需要注意的是,测量进程运行时间离不开操作系统的支持,而我们需要用 C/C++ 编写基于 Linux 的接口，并在外面利用 JNI（ Java Native Interface ）包装为一个 Judger 类。

Judger 类需要接受参数，调用 JNI 并返回相关的数据。

4.1.1 运行原理

TODO