软件架构调研——Git

肖江

对于一个正常的版本管理系统来说，有以下三个主要的功能需求:

- 存储项目内容
- 存储版本变更历史
- 将项目内容和版本变更历史分发给所有的协作者

而 Git 是目前流行的一款分布式版本管理系统，这篇报告主要介绍 Git 所采用的软件架构，并剖析这些架构对 Git 本身的功能实现产生的影响。

1. 架构设计

由于 Git 本身是一个版本管理系统，它的数据结构需要解决两个问题:

- 如何表示不同版本的项目内容
- 如何表示版本变更历史

对于这两点，Git 给出的答案都是 DAG(有向无环图)，也就是用有向无环图来表示项目内容，用有向无环图来表示版本变更历史。而 Git 的有向无环图中一共有四种可能的节点元素，分别是:

- Tree:表示一个文件夹，它的子元素可以有另一个 Tree，也可以有 Blob
- Blob:表示一个文件
- Commit: 表示某一个版本，它的子元素必然是一个 Tree，且该 Tree 代表了当前版本的项目的根文件夹内容
- Tag: 类似 Commit，也表示一个版本，有自己的名字，子元素必然是一个 Commit

这其中的每个节点元素，在一个 Git 仓库中都有自己的唯一标识符 SHA 码，两个元素相同当且仅当 SHA 码相同。

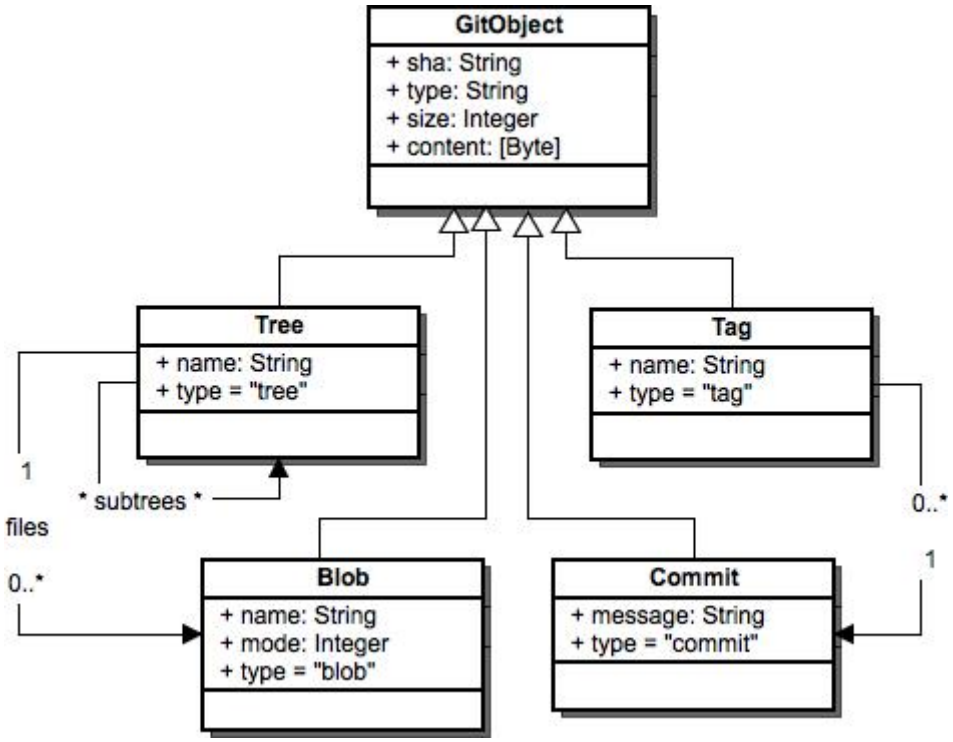

图 1：节点元素

基于这四种可能的节点元素，首先介绍一下 Git 如何表示不同版本的项目内容，所谓的项目内容就是文件，对于 Git 来说，天然就可以使用一个 Commit 来表示一个版本的项目内容，Commit 保存了版本信息，而 Commit 的子元素 Tree 保存了项目的文件信息。

而对于版本变更历史来说，Git 采用了 Commit 的有向无环图来表示版本变更历史，每个 Commit 可以拥有零到多个父节点，父节点为当前版本的前驱版本，拥有多个父节点表明当前版本是通过多个前驱版本合并而来的，零个父节点表明当前版本为根版本或者遗弃版本。

这样一来，Git 对不同版本的项目内容和版本更新历史的表示方式就明确了。

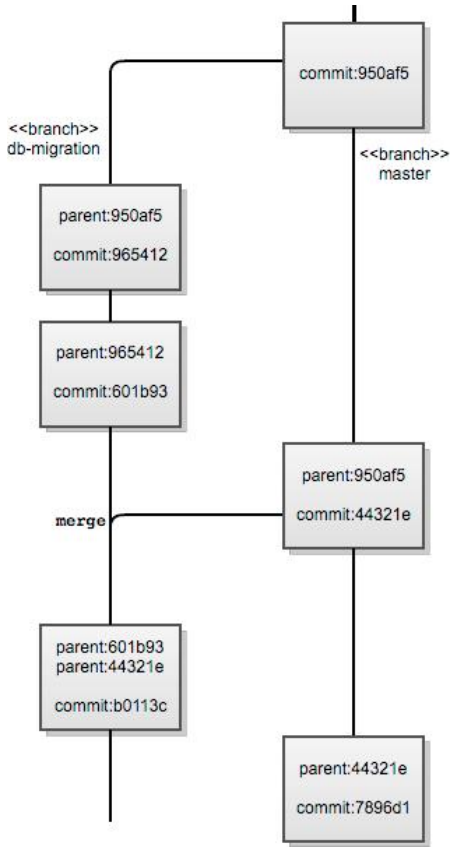

图 2: 版本变更历史

2. 这些架构设计对 Git 功能实现的影响

2.1. SHA 码

由于 SHA 码和节点元素具有一一对应的关系，这导致在进行不同版本的比较时，Git 可以更加轻松地进行比较工作，也就是在 SHA 码相同的情况下，可以直接跳过整个文件或者整个文件夹的比较工作，只在 SHA 码不同的情况下进行比较，相比于传统的使用 delta 修改集的版本管理工具，这样的比较显然更有效率。

但是，与此同时 SHA 码也导致 Git 的存储开销较大，每次修改文件，都需要重新存储一份文件内容，同时，一旦被修改的文件处在深层文件夹中，就需要对所有的上层文件夹创建新的 tree 对象并赋予新的 SHA 值，Git 通过压缩文件的方式解决存储开销过大的问题，将所有的节点元素分成两部分存储，一部分是实

际上的压缩后的合集文件，另一部分是一份索引文件，保存了不同的 SHA 值对应的节点元素在合集文件中的位置。

2.2. 有向无环图表示的版本变更历史

Git 深受开发者喜爱的一个重要原因是“Git 天然支持分支”，而其中所谓的“天然”，指的就是在软件架构设计时，Git 采用了有向无环图来表示版本变更历史，这样的设计使得分支特性变得自然。

在传统的使用线性数据结构来表示版本变更历史的版本控制系统中，分支特性变得难以维护，比较显著的问题是，由于线性数据结构中，一个版本最多只能拥有一个父版本，无法拥有多个父版本，一旦分支发生了合并操作，也就是将其中一个分支的变更历史应用到另一个分支上以后，“合并用到的两个分支”这一信息就丢失了，这样的版本控制系统无法回答“当前分支合入了哪些分支”这个关键问题，进而无法确定某一分支的变更在当前分支是否有效。

但是对于 Git 来说，这是很简单就可以处理的问题，通过在合入时产生一个拥有两个父节点的合并节点，就可以保留分支的合入信息。

3. 总结

Git 通过设计版本变更相关的数据结构，巧妙实现了灵活的分支版本变更管理，充分展现了软件架构设计在软件最终特性上的决定性作用。