计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(10):312−318 [doi: 10.15888/j.cnki.csa.008113]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

软件多分支开发代码漏合问题及解决途径①

王晓龙, 董玉雪

(中国海洋大学 基础教学中心, 青岛 266100)
通讯作者: 董玉雪, E-mail: 853483095@qq.com

摘　要: 代码管理是软件开发过程的一个重要的环节, 随着软件需求和功能的日益复杂, 多分支开发场景越来越普遍, 使得代码管理的难度直线上升, 分支间代码漏合问题也应运而生, 严重影响了开发的效率和版本的交付质量. 本文根据个人的代码管理经验, 对常用的分支管理模式进行了研究与分析, 并主要基于分支开发、分支发布模式对多分支代码漏合问题进行了解决途径的研究. 实践应用结果表明可以有效避免分支代码漏合问题, 优化代码管理过程, 提高整体代码管理的效率.

关键词: 代码管理; 分支管理模型; 多分支; 并行开发

引用格式: 王晓龙,董玉雪.软件多分支开发代码漏合问题及解决途径.计算机系统应用,2021,30(10):312–318. http://www.c-s-a.org.cn/1003-3254/8113.html

Problem and Solution to Code Leakage in Multi-Branch Software Development

WANG Xiao-Long, DONG Yu-Xue

(Teaching Center of Fundamental Courses, Ocean University of China, Qingdao 266100, China)

Abstract: Code management is an important part of software development. With the increasing complexity of software requirements and functions, multi-branch development scenarios become more common, which makes the difficulty of code management soar. Also, code leakage between branches arises, seriously affecting the development efficiency and version delivery quality. According to personal code management experience, this study analyzes commonly used branch management modes and explores the solution to the multi-branch code leakage mainly from perspectives of branch development and branch release mode. The practical application results show that the proposed solution can avoid the branch code leakage, optimize the code management process, and improve the overall code management efficiency.

Key words: code management; branch management model; multi-branch; parallel development

代码管理在软件发展过程中扮演者至关重要的角色, 也是衡量软件开发过程的一个重要指标, 已经被越来越多的企业所重视.随着软件需求和功能越来越复杂, 对于代码管理的要求也越来越高, 良好的代码管理可以提高整体开发效率, 及时暴露问题[1,2].

分支管理是代码管理过程中最常用的操作, 也是最容易出错的环节[3,4], 目前的代码分支管理更多的是利用 Git 或者 SVN 等成熟的版本控制工具进行维护, 随着软件的功能和版本越来越复杂, 代码分支的数量日益增多, 开发人员在代码提交过程中难以保证分支代码的一致性, 日趋复杂的软件开发活动给代码分支管理带来巨大的挑战, 多分支代码漏合[5,6] 问题也应运而生, 严重影响了软件开发效率[2,7]. 为了更好的推动代码分支管理, 支撑软件开发过程的快速有序进行, 越来越多的公司已经对代码分支管理进行了深入的研究, 伴随着出现了许多代码分支管理模型[3], 例如 FaceBook

① 收稿时间: 2021-01-05; 修改时间: 2021-02-03; 采用时间: 2021-02-08

采用的主干开发, 主干发布模型, 以单一主干分支支撑开发过程; Qunar、猪齿鱼、阿里云效平台采用的分支开发, 主干发布模型, 将代码的开发与发布过程通过分支进行划分. 代码分支管理模型制定了代码管理规范, 优化了代码提交过程, 对于提高软件开发效率具有重要的作用.

虽然已有的代码分支管理模型可以提高软件开发的效率, 但如果在实际应用中缺乏系统的管理, 依然会存在很多问题. 本文以 Git 版本管理工具作为研究对象[8], 对开发过程中常用的代码分支管理模式进行了研究分析, 并基于分支开发、分支发布模式对多分支开发过程中的分支代码漏合问题进行了解决途径研究.本文的研究问题来源于实际应用场景, 解决措施经过了实践的验证, 可以为多分支代码管理的研究提供一定的参考.

1 多分支代码漏合问题

多分支代码管理是软件开发过程中的重要活动, 对于提高软件开发效率和软件质量有重要的作用[9], 做好代码分支管理是软件开发过程正常进行的有效保障[1,6]. 由于代码开发过程的日趋复杂, 多分支并行开发过程容易出现分支间代码漏合问题且不易排查, 代码漏合问题主要是由如下几个方面导致的.

1.1 分支操作权限混乱

代码分支的操作权限缺乏统一的管理, 分支的创建、删除、合并等权限如果没有按照等级分类进行分配, 开发人员在自主创建代码分支时容易导致多个重复分支的出现, 致使分支管理过程中出现代码相互覆盖、分支操作混乱等问题[5,10], 最终导致未经验证的代码直接流入生产环境.

1.2 分支管理缺乏规范

分支管理缺乏规范的指导, 包括分支命名存在二义性、分支与代码版本缺少对应关系, 容易导致代码提交过程混乱, 出现代码提交分支错误[6,10] 和无法对历史代码及版本进行追溯等问题, 增加了分支维护的难度[2,4,11].

1.3 分支合并流程混乱

分支代码漏合主要是多分支并行开发过程中对于分支合并操作缺乏规范的操作流程, 导致代码合并时出现遗漏的现象, 致使发布到生产环境中的软件功能不全. 常见的多分支开发场景及分支合并流程如图 1 所示, 代码开发过程中新分支的创建可能来源于 master 分支上, 也可能来源于已有的 feature 分支, 每个分支负责一个 feature 功能开发, 进行版本发布时需要将该分支的代码合入到 master 分支中, 版本发布后需要将 master 分支的最新代码合入到其他的 feature 分支中, 以保证 feature 分支中的代码最全, 可以避免后续分支版本发布时的代码遗漏现象.

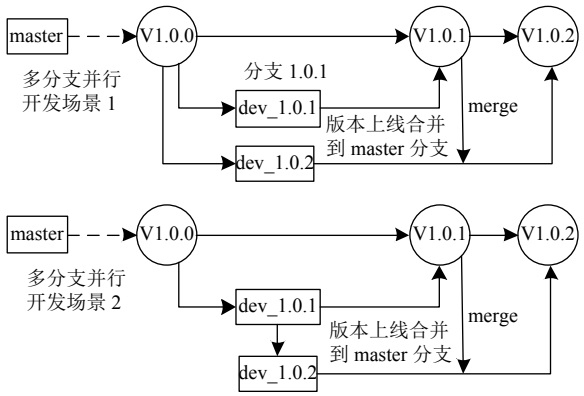

图 1 多分支并行开发场景

2 代码分支管理模式

软件开发过程中的代码管理一般是基于代码分支管理模式进行的, 代码分支管理模式可以规范代码分支的管理过程, 在一定程度上减少分支管理和维护的成本, 有利于保证代码分支管理的有序进行.

不同的软件开发场景所采用的代码分支管理模式不尽相同, 本文就以下几种常用的代码分支管理模式进行了调研和分析, 并阐述了各个模式的优缺点和适用的开发场景.

2.1 主干开发, 主干发布: M-M

主干开发、主干发布模式只有一条主干分支, 所有的代码提交和 Bug 修复都在主干分支上进行, 提交的内容可以从主干分支上直接发布到生产环境, 如图 2 所示.

主干提交、主干发布的模式只有一个分支, 在实际开发实践中存在如下几点优势: 1) 迭代极快, 随时可发布; 2) 没有分支合并的麻烦, 不用担心分支合并出错. 同时该模式的缺点也比较明显: 1) 版本迭代过程中开发人员的协同难度大; 2) 版本发布频率取决于主干分支的代码质量和稳定性; 3) 频繁的代码提交使得主干分支的压力较大, 需要具备较为完备的代码提交标准.

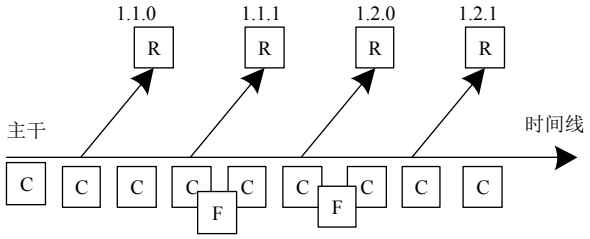

图 2 主干开发、主干发布模式

该模式对于分支的维护较为简洁, 可以减少多分支管理的成本, 避免分支间代码漏合的现象, 但对于开发人员的能力要求较高, 开发人员本身要具备保证分支版本一致性的能力和意识.

2.2 主干开发, 分支发布: M-B

主干开发、分支发布模式要求项目团队都在一个主干上进行版本开发, 只有在需要发布时拉取新的分支, 并在新分支上进行缺陷的修复, 如图 3 所示.

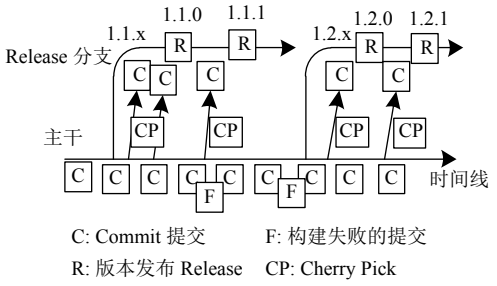

图 3 主干开发、分支发布模式

主干开发, 分支发布模式是以发布为目的创建短期分支, 新创建的分支仅仅用来版本发布, 生命周期比较短, 发布结束之后都要回归主干分支. 其优势主要有: 1) 只有一条主干分支, 保证了代码的唯一性; 2) 主干分支永久存在, 开发过程比较平稳; 3) 版本发布过程以版本号来规范分支创建, 便于统计维护. 同时该模式也存在如下缺点: 1) 对主干分支要求很高, 主干分支不稳定容易导致发布分支难产; 2) 主干分支质量决定版本发布频率; 3) 频繁的代码提交容易导致版本发布的节点不易把控.

M-B 模式可以看作是 M-M 模式的延伸, 可以保证主干开发的延续性和版本的正常发布, 适用于频发发布为主的代码开发模式.

2.3 分支开发, 主干发布: B-M

分支开发, 主干发布模式允许系统工程师根据系统功能创建分支, 并在功能开发完成后合并到主分支, 最后将特性分支删除, 如图 4 所示.

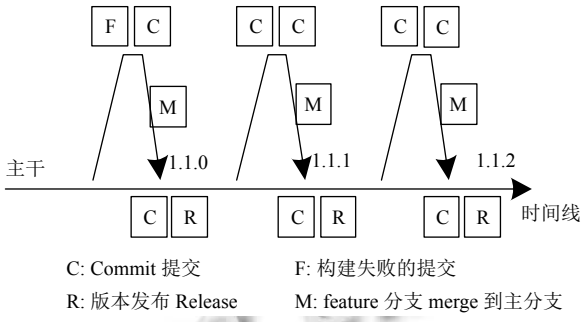

图 4 分支开发、主干发布模式

分支开发, 主干发布模式的优势主要有: 1) 特性分支相互独立, 开发过程互不干扰; 2) 分支和功能特性相对应, 版本功能迭代方便, 便于维护和追溯. 其缺点主要有: 1) 分支按照功能特性创建, 一个特性只关联一个分支, 考验开发人员的特性拆分能力; 2) 分支命名规范严格, 否则不利于分支的统一管理.

B-M 管理模式是基于软件的功能特性, 以快速交付为目的, 适用于版本迭代快速的敏捷开发场景.

2.4 分支开发, 分支发布: B-B

分支开发, 分支发布模式中开发人员在特性分支上进行功能开发, 并从中发布版本, 然后在发布后将代码合并回主线分支, 如图 5 所示.

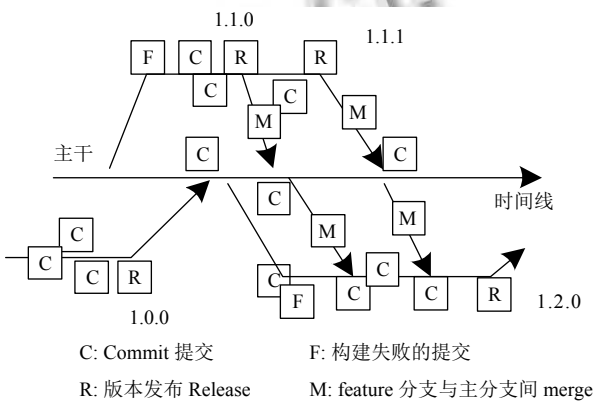

图 5 分支开发、分支发布模式

分支开发、分支发布模式的优势包括: 1) 主干分支长期稳定存在, 代码维护较为统一; 2) 特定分支单独维护, 不同分支的版本发布相对独立.其缺点为: 1) 特性分支过多, 不同分支的版本维护比较混乱, 容易出现分支漏合的现象; 2) 各个分支单独维护, 很难做到分支的统一管理.

B-B 模式中主干分支用于代码同步, 多条特性分支并存, 不同分支的代码迭代相对独立, 适用于多项目、多版本的并行开发场景.

2.5 分支管理模式选择

每种分支管理模式都有其特点和应用场景, 就通用特点而言, M-M 模式适用于需求功能较为简单且版本发布有先后顺序开发场景; M-B 模式适用于单一功能版本的串行开发场景; B-M 模式适用于功能紧凑、版本迭代快速的敏捷开发场景; B-B 模式适用于功能相对独立的多版本的并行开发场景.

相对于其他 3 种分支管理模式而言, B-B 模式中分支单独维护, 相互间的影响较小, 分支的生命周期维护清晰明确, 对于版本并行开发和发布以及功能的追溯比较容易实现, 可以更好的支持多项目、多版本的并行开发[8], 所以本文主要基于 B-B 模式对多分支代码漏合问题进行了解决途径的研究.

3 多分支代码漏合的解决途径

本文结合了企业工作中的代码管理经验, 基于分支开发、分支发布管理模式对分支代码漏合问题进行了解决途径研究, 主要包括分支操作权限规范、分支管理规范和多分支代码合并规范 3 个方面的内容.

B-B 模式中分支管理过程如图 6 所示, 所有分支的创建都以 master 分支为来源, 不同分支单独开发维护, 当 master 分支有新版本代码合入时需要同步将 maser 分支最新代码 merge 到其他特性分支中, 以保证特性分支的代码是最新的, 可以避免后续的版本发布过程出现代码遗漏现象; 任意特性分支在版本发布之后将当前分支代码合入到 master 分支中并将当前分支删除, 保证 master 分支始终包含所有的功能代码; master 分支代码更新之后同步将当前最新代码 merge 到其他的特性分支中, 并继续开展后续的版本开发工作.

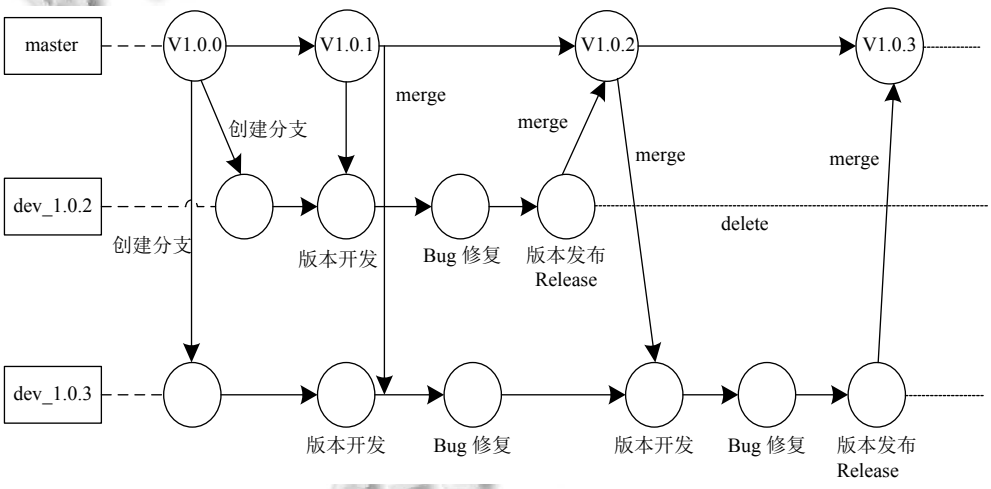

图 6 分支开发、主干发布管理模式

3.1 规范分支操作权限

对于分支操作的权限分配应该由配置管理团队统一管理, 不同级别的开发人员具有不同的分支操作权限, 普通开发者只有分支代码提交和代码评审权限, 系统工程师负责分支的具体维护, 包括分支的创建、删除、合并等操作[12].

分支操作权限的分配可以规范开发人员的代码操作过程, 权限的合理分配可以减少分支合并过程中的问题, 保证分支管理的有序进行.

3.2 规范分支管理过程

分支管理规范的制定可以标准化分支管理过程, 降低分支维护的难度, 主要包括分支命名、分支创建来源和分支生命周期维护等几个方面.

1) 分支命名. 分支命名采用版本化处理, 与代码版本进行逻辑绑定, 确保分支与代码功能的可追溯性. 分支的命名可以采用“分支类型_版本号”的形式, 例如版本 1.0.0 的开发分支可命名为: dev_1.0.0, 既可以实现分支和版本的对应关系, 又使得分支维护更加清晰.

2) 分支创建来源. 根据 B-B 分支管理模式, 所有的分支的创建都来源于 master 分支, 并且需要系统工程师根据版本功能进行分支整体规划, 以保证分支代码来源的一致性和规范性, 避免后续分支代码合入的过

程中出现混乱情况[13]. 以 master 分支创建特性分支 dev_1.0.0 为例, 具体实现过程如下:

```
//将 project 项目 clone 到本地
git clone project
//切换到项目的 master 分支
git checkout master
//创建本地 dev_1.0.0 分支
git branch dev_1.0.0
//将本地分支推送到远程仓库中
git push -u origin dev_1.0.0
```

3) 分支生命周期维护. 代码开发之前系统工程师提前规划代码版本并创建特性分支, 将代码提交分支通知到开发人员, 避免代码提交时分支选择错误; 版本发布时系统工程师进行分支合并及 tag 版本发布; 代码发布上线之后, 系统工程师将对应的特性分支删除, 避免历史遗留分支影响代码提交过程.

3.3 规范多分支代码合并过程

代码分支间漏合问题主要是分支合并时操作不规范导致分支代码遗漏, 在 B-B 模式中主要的漏合场景是 master 分支已经合入了新的功能代码, 而未将 master 分支最新代码 merge 到其他特性分支中, 导致后续特性分支在进行版本发布时代码不全, 出现分支代码漏合现象, 主要的解决措施有如下几个.

3.3.1 增加分支漏合检测机制

在 B-B 模式中, 由于所有的特性分支都来源于 master 分支, 所以针对代码分支间漏合问题, 增加代码一致性检测机制, 在分支代码开发过程中通过命令 git merge-base 判断当前分支与 master 分支的最新代码是否同步.

1) 分支同步检测机制

开发人员进行提交代码时, 触发分支代码同步检测算法, 检测当前分支的来源 commitid 是否和 master 分支最新的 commitid 保持同步, 如果不同步则代表 master 分支的代码优先于当前开发分支, 即 master 分支合入了新的代码, 需要将 master 分支的代码 merge 到当前分支之后再进行后续的开发, 否则版本发布时容易出现代码漏合现象.

2) 分支同步检测算法

以 master 分支和 dev_1.0.0 分支为例, 分支同步检测算法的执行过程如下所示.

```
git checkout master
commit_master = 'git rev-parse HEAD'
git checkout dev_1.0.0
commit_dev = 'git rev-parse HEAD'
commit_id = 'git merge-base commit_dev commit_master'
if [ commit_id == commit_master ]
then
  echo "sync"
else
  echo "out of sync"
fi
```

分支同步检测过程分为 3 个步骤:

(1) 执行命令 git rev-parse HEAD 分别获取 dev_1.0.0 分支和 master 分支最新的 commitid.

(2) 利用 git merge-base 命令获取分支同步检测结果. 例如: dev_1.0.0 分支最新的 commtid 是 f5c67f88fc, master 分支最新的 commtid 是 a1e2dee286, 执行命令 git merge-base f5c67f88fc a1e2dee286 查看返回结果.

(3) 判断分支代码是否同步, 如果 git merge-base 命令返回的是 master 的 commitid: a1e2dee286, 则表示当前分支已经同步了 master 分支的最新代码, 否则代表分之间代码尚未同步.

3.3.2 规范多分支并行开发流程

代码开发过程中由于版本迭代频繁, 开发人员通常需要在多个分支上进行功能开发, 且每次代码提交的 Patch 在 review 阶段需要一定的周期, 根据实际的开发经验, 多分支开发过程可以按照如图 7 所示的流程进行, 以避免分支代码提交过程混乱.

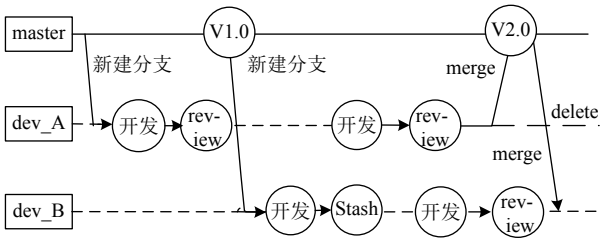

图 7 多分支开发流程图

1) master 分支只用来同步分支代码, 开发过程中不进行任何代码提交操作.

2) 版本功能的开发在特性分支上完成, 分支 A 进行 A 功能的开发, 分支 B 进行 B 功能的开发.

3) 分支 A 进行代码 review 时暂停 A 功能的开发,

然后同步在 B 分支上进行功能 B 的开发; A 分支代码 review 结束之后, 暂存 B 分支的开发任务, 结合分支 A 的 review 结果进行代码修改及后续的代码开发; 同样的在 B 分支进行代码 review 时, 暂停 B 功能的开发并同步进行 A 分支的代码开发工作, 以此不断交替进行.

4) 当分支 A 代码合入到 master 分支后即可删掉, 同步更新 master 分支最新代码到 B 分支上.

5) 以此循环往复不断重复以上步骤.

4 解决途径结果验证

多分支代码漏合问题的解决措施已经基于分支开发、分支发布模式应用到了实践中, 本文主要就分支操作权限和多分支漏合检测机制两个方面进行了验证结果的相关描述.

4.1 分支操作权限验证

分支权限管理依托 Gerrit 开源工具的支持, 所有代码及分支的操作权限都通过 Gerrit 权限机制实现. 如图 8 所示的部分权限规划内容, 通过对不同的人员进行分组, 然后对组实现权限划分. 分支创建权限交由 Project Owners 群组; 代码 review 权限按照操作权限等级分配给不同的开发者, 从而达到权限控制的效果.

图 8 部分权限规划内容

以代码 review 操作为例, 普通的开发者只有提交代码和代码 review+1 的权限, 代码+2 和分支合并等权限由更高级别的系统工程师完成, Gerrit 中的实现效果如图 9 所示.

4.2 多分支漏合检测机制验证

分支漏合检测机制在每次代码提交之后执行, 可以及早的发现问题, 缩小问题的影响范围. 本文的分支漏合检测机制通过 Jenkins 调度执行, Jenkins 的 CI 流程通过监听 Git 仓库的代码变更, 然后从远程仓库中克隆代码并执行对应的检测算法. Jenkins 的具体执行效果如图 10 所示, 分别获取当前分支和 master 分支的 commit_id, 然后通过判断 git merge-base 的检测结果来验证分支代码是否同步.

图 9 代码 review 权限划分效果

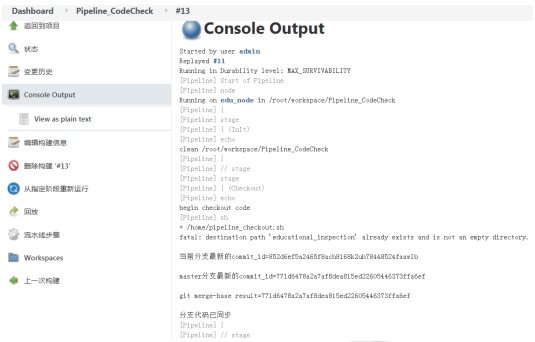

图 10 Jenkins 分支合并检测

5 小结

本文针对常见的代码分支管理模式进行了研究与分析, 并基于分支开发、分支发布模型对多分支代码漏合问题进行了解决途径的探讨和研究. 本文探讨的问题源于实践开发经验, 具有实际的解决意义, 同时解决措施也进行了实践的验证, 可以为代码分支管理的研究提供一定的参考; 但相对于分支管理功能较为完善的大公司来说, 尚缺乏统一直观的管理平台对分支进行跟踪与维护, 仍需要进行不断的完善和改进.

参考文献

1 Krusche S, Berisha M, Bruegge B. Teaching code review management using branch based workflows. Proceedings of the 38th International Conference on Software Engineering. New York: Association for Computing Machinery, 2016. 384–393.

2 廖鹏举, 高文东, 王慧. 浅谈企业软件代码质量管理. 信息系统工程, 2020, (5): 114–116. [doi: 10.3969/j.issn.1001-2362.2020.05.051]
3 包永红. Git 在 JavaWeb 技术实践教学课程中的应用与探索. 内蒙古农业大学学报 (社会科学版), 2019, 21(5): 32–36.
4 徐娅. Git 版本控制工具在团队协作项目中的应用. 智能计算机与应用, 2019, 9(5): 341–343.
5 Le Nguyen H, Ignat CL. An analysis of merge conflicts and resolutions in git-based open source projects. Computer Supported Cooperative Work, 2018, 27(3–6): 741–765. [doi: 10.1007/s10606-018-9323-3]
6 张宇嘉, 张啸川, 庞建民. 代码混淆技术研究综述. 信息工程大学学报, 2017, 18(5): 635–640. [doi: 10.3969/j.issn.1671-0673.2017.05.023]
7 朱超. 基于管理信息系统的代码自动生成技术分析. 微型电脑应用, 2017, 33(2): 78–80. [doi: 10.3969/j.issn.1007-757X.2017.02.020]
8 关婷婷, 任洪敏, 江金莲. 基于 Git 的跨版本迁移的软件评审设计与实现. 现代计算机: 专业版, 2018, (19): 70–73, 77.
9 刘志伟, 邢永旭, 于澔, 等. 企业级海量代码的检索与管理技术. 软件学报, 2019, 30(5): 1498–1509. [doi: 10.13328/j.cnki.jos.005718]
10 耿普, 祝跃飞. 一种基于分支条件混淆的代码加密技术. 计算机研究与发展, 2019, 56(10): 2183–2192. [doi: 10.7544/issn1000-1239.2019.20190368]
11 German DM, Adams B, Stewart K. Cregit: Token-level blame information in git version control repositories. Empirical Software Engineering, 2019, 24(4): 2725–2763. [doi: 10.1007/s10664-019-09704-x]
12 王真. 版本控制工具在软件开发项目管理中的应用——以 GIT 为例. 项目管理技术, 2020, 18(6): 131–134. [doi: 10.3969/j.issn.1672-4313.2020.06.028]
13 苏小红, 张凡龙. 面向管理的克隆代码研究综述. 计算机学报, 2018, 41(3): 628–651. [doi: 10.11897/SP.J.1016.2018.00628]