27 Code Review的正确姿势

更新时间：2020-06-03 15:50:52

> 我们有力的道德就是通过奋斗取得物质上的成功；这种道德既适用于国家，也适用于个人。——罗素

1. 前言

相信很多从事 Java 开发行业的同学都听说过代码审查 (Code Review)。那么是否思考过下面几个问题？

- 代码审查是什么？为啥要代码审查？
- 代码审查审查什么？谁来审查？
- 如何进行代码审查呢？

2.What? Why?Who?

2.1 什么是代码审查？

> 代码审查也叫代码复查，即通过阅读代码的方式来检查代码是否符合要求。

通俗来讲，我们代码开发完成以后，项目发布之前，需要让其他人阅读我们的代码，看是否符合要求，是否存在隐患等。

代码审查的目标主要有：

- 保证每一次上线时，至少会有两个人能够理解该代码的逻辑。
- 保证每次代码审查后的结果都会落到实处，有跟进有解决。

2.2 为什么要代码审查？

代码审查是提高代码质量，提前发现 BUG ，统一团队代码规范的一个重要途径。

俗话说：“不识庐山真面目，只缘身在此山中”。

由于第一人称视角和个人的习惯难改，经验有限的缘故，我们通常很难发现自己编码中存在的问题。

因此团队中开发经验丰富的其他成员对代码进行把关，更容易发现代码中存在的功能和性能问题，能够提前发现一些隐患。

通过代码审查可以检查团队成员是否按照团队的编码规范进行开发。

同时代码审查也是团队成员相互学习的一种重要方式，作为代码审查人员可以学习团队其他成员的编程技巧，设计方案等，是学习进阶的一个极佳方式。

2.3 谁来审查？

每个公司和团队的规定可能有所不同。

常见的代码审查人员为团队的技术主管，团队中有丰富开发经验的老员工。

也有些团队对于日常的小需求会让开发人员自己找其他熟悉相关背景的同事进行代码审查。

这里分享一个**非常重要的经验**，我们在开发过程中就可以将自己的开发分支和 master 分支的最新代码进行比对，按照代码审查的内容对自己代码进行审查，这样能够及时发现问题，尽可能避免将问题遗留在 “正式的代码审查阶段”。

2.4 代码审查，审查什么？

2.4.1 审查设计规范

- 代码是否符合面向对象的五大原则原则？是否符合领域设计的原则？

> **面向对象的五大原则**：单一职责原则，开闭原则，里氏替换原则，接口隔离原则，依赖倒置原则。
>
> 如一个函数干了两件不相干的事情，应该拆分为两个函数。
>
> 如嵌套超过 3 层可以使用卫语句、策略模式、状态模式等。

- 是否使用了某种设计模式？该设计模式使用是否恰当？
- 新的代码是否符合团队的代码规范？
- 代码位置是否正确？如用户相关的代码是否放到了用户服务中？
- 新代码是否重用了已有代码？新代码是否提供了可重用的代码？结合前面章节学到的重构原则，是否需要重构？
- 新代码是否有过度设计的现象？是否违背 YAGN 原则？

> **YAGN** 即 “You aren’t gonna need it”。只有在需要的时候程序员才应该编写相应的代码。1

2.4.2 审查可读性和可维护性

- 属性、变量、参数、方法类的命名是否能够反映出其要表达的真实含义？
- 是否能够理解自己阅读的代码？
- 是否能够理解测试代码？
- 测试是否覆盖了所有分支？测试是否覆盖了正常和异常情况？是否有一些未考虑的情况？
- 异常信息是否容易理解？
- 代码片段、文档、注释是否有令人困惑的部分？

2.4.3 审查是否有错误

代码是否符合需求？ 测试代码编写的是否正确？

代码中是否有潜在的 BUG？ 是否校验错了参数？判断条件是否有误？

代码是否有安全问题？如 SQL 注入。

代码是否存在性能问题？是否有不必要的数据库调用、IO 读写操作和远程调用？

编写代码的开发人员是否更新了相关文档？

是否有明显的错误？如是否会偶发性指向测试库？是否需要调用真实服务时，却没有调用真实的服务，而采用了硬编码方式 mock 了接口？

2.4.4 审查测试代码

源码作者是否为新的代码或修复的代码编写了测试代码？

测试是否覆盖到了容易出错的部分或者复杂的部分？

测试是否验证了代码的正确性？

是否能够想出没有被当前测试代码覆盖的场景？

是否测试了代码的限制？如批量查询接口只支持 100 个元素，是否被测试到了？

是否涵盖了安全方面的测试？

是否有性能测试？

2.4.5 审查性能方面

- 新的代码中某些函数是否有性能要求？是否验证了该函数的性能？
- 新的代码是否降低了之前的性能？
- 是否有不必要的外部服务调用？如不必要的调用数据库，不必要的外部服务调用。
- 新代码是否有可能引起内存泄露的情况？
- 是否有可能引起内存占用无限增长的情况？
- 代码中使用的资源是否正确关闭？如创建的连接或者流。
- 资源池的配置是否正确？

- 是否使用了在多线程场景下可能引发性能问题的类？

> 正如《手册》并发处理章节 第 15 条 Random 如果被多线程共享，虽然线程安全，但是因为竞争同一个 seed 会导致性能下降 [^3]。

- 主要可能引发性能问题的常见场景，如果用到了反射，要思考是否必须要用反射？要思考提供的接口如果超时会怎样？超时时间设置为多少更合理？是否使用了并行特性？是否在没必要的地方使用了并行特性？

2.4.6 审查正确性

- 多线程环境下数据结构是否选择正确？

> 如在多线程环境下共享变量使用非线程安全的类，就可能出现线程安全问题。
>
> 如可以使用 `Collections.sychronizedList()` 来创建线程安全的 `List`. 如果读的频率远大于写的频率，可以使用 `CopyOnWriteArrayList`.
>
> 再如多线程条件下， `SimpleDateFormat` 存在线程安全问题，为了避免线程安全问题可以使用 `ThreadLocal` 的方式。

代码中锁的使用是否正确？

代码使用的数据结构是否正确？如需要随机访问，却使用了 `LinkedList`，需要保证唯一性却使用了 List.

日志的级别设置是否正确？

2.4.7 审查安全性

- 是否缺乏安全校验？如外部服务调用的结果没有判空直接使用导致空指针错误。
- 批量查询接口是否有对查询数量进行限制？
- 敏感操作是否做了权限校验？
- 在使用平台资源时，是否做了限制？如短信、邮件是否做了数量限制、疲劳控制？下单、支付是否做了幂等校验？
- 是否存在 SQL 注入的可能性？
- 日志是否脱敏？
- 发帖、评论、发送即时消息等用户生成内容的场景是否实现了防刷，风控？
- 开发人员是否开发了一些危险后门？

更多具体案例大家可以参考 Upsource - Docs & Demos

2.5 如何审查

2.5.1 代码审查工具

最简单直接的代码审查工具就是 GIT ，git 提供了强大的 diff 功能，通常代码审查人员会将所要审查的分支和 master 分支进行对比。

也可以使用类似 Upsource （JetBrains 出品的一款知名的代码审查和项目分析工具）。

建议在代码审查前自己预先使用 findbugs 等静态检查工具，检查出一些潜在的风险提前修改。

2.5.2 代码审查的形式

代码审查的形式每个公司和团队也千差万别，这里介绍常见的利用简单的代码审查形式。

代码审查的形式

有些团队的代码审查人员通过需求文档了解项目的需求，通过技术文档了解对方的设计，然后拉取对方的开发分支和 master 进行比较，按照上述的审查内容进行审查，对发现的问题进行记录，并约时间双方进行沟通。

也有些团队可能将审查人员和开发人员约在一起，开发人员负责介绍项目的需求、技术文档，然后介绍自己代码修改点，自己代码的核心实现。由代码审查人员随时提问，开发人员现场解答，并对待修改的地方做记录，根据代码审查人员的建议做出修改。代码审查后要修复审查中遇到的问题，尽量当天完成。下一次代码审查时，需先检查之前的问题修复情况。再次记录此次代码审查的问题。

具体原则

小项目代码审查：只要有新写的或者修改的代码都可以拿出来审查；代码审查至少两人在场，场地不限，时长尽量短；聚焦核心代码、核心业务逻辑，关键点。

大项目的代码审查：尽量找个会议室，更多人参与其中，和 master 全量 diff。

3. 总结

本节主要讲述了代码审查的概念，重点解答了：为什么要代码审查？代码审查审查什么内容？如何进行代码审查。

希望大家不要坐等别人对自己代码进行审查，可以养成在开发过程中对自己代码进行自我审查的习惯，多将自己的开发分支代码和最新的 master 代码进行比对，避免将太多问题暴露到软件生命周期的靠后的阶段，降低修改的成本。

下一节我们将学习思维导图的相关知识。

参考资料

> You aren’t gonna need it □□

}