哈尔滨工业大学 计算学部

2024 年秋季学期《开源软件开发实践》

Lab 2：开源软件开发协作流程

姓名	学号	联系方式
李璇	2023120261	L1xxx59@163.com/15193215452

目 录

1 实验要求

1.1 实验目标

(1)了解和掌握基于代码托管平台的开源软件协作开发过程：
将通过实际操作熟悉如何在 GitHub 等平台上进行协作开发。
(2)掌握基于 GitHub 的软件项目协作开发命令和方法：
包括使用 Git 指令完成任务，如克隆仓库、创建分支、提交更改、发起 Pull Request (PR)等。
(3)熟悉几个 GitHub 中常用开源软件开发工具：
探索并学习一些辅助工具和服务，例如用于寻找入门级问题的网站、评估贡献者活跃度和影响力的指标，以及安装和使用某些扩展程序来提高效率。

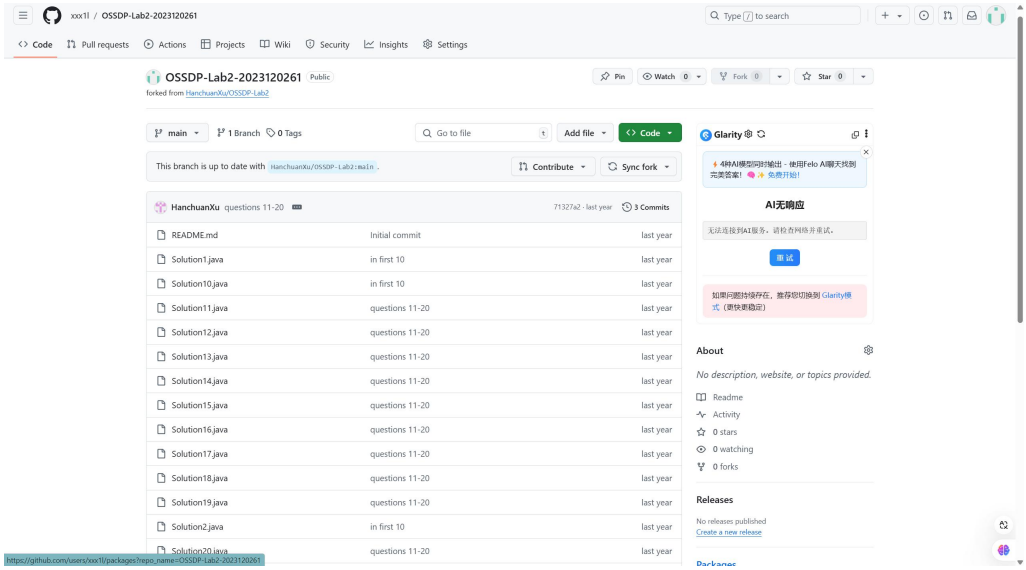
1.2 实验过程

实验内容 1：发送 Pull Request
实验内容 2：接受 Pull Request
实验内容 3：GitHub 辅助工具

2 实验内容 1 发送 pull request

2.1 fork 项目

fork 后的个人仓库中项目的界面截图



2.2 git 操作命令

①git clone

```
D:\Study\homework\opensource\lab2-2023120261>git clone https://github.com/xxx1l/OSSDP-Lab2-2023120261.git
Cloning into 'OSSDP-Lab2-2023120261'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (21/21), done.
Receiving objects:  96% (26/27)
Receiving objects: 100% (27/27), 20.04 KiB | 2.50 MiB/s, done.
Resolving deltas: 100% (1/1), done.

D:\Study\homework\opensource\lab2-2023120261>
```

② 创建 fix 分支:git checkout -b fix

```
D:\Study\homework\opensource\lab2-2023120261\OSSDP-Lab2-2023120261>git checkout -b fix
Switched to a new branch 'fix'

D:\Study\homework\opensource\lab2-2023120261\OSSDP-Lab2-2023120261>
```

③修改完毕提交 git add . git commit -m“solution 18”

```
D:\Study\homework\opensource\lab2\OSSDP-Lab2-2023120261>git add .

D:\Study\homework\opensource\lab2\OSSDP-Lab2-2023120261>git commit -m"solution 18"
[fix e956f86] solution 18
 9 files changed, 210 insertions(+), 3 deletions(-)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/checkstyle-idea.xml
 create mode 100644 .idea/compiler.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 L2023120261_18_Test.java
 create mode 100644 OSSDP-Lab2-2023120261.iml
```

④推送 git push --set-upstream origin fix

```
D:\Study\homework\opensource\lab2\OSSDP-Lab2-2023120261>git push --set-upstream origin fix
git: 'credential-manager-core' is not a git command. See 'git --help'.
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 20 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 3.60 KiB | 1.80 MiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'fix' on GitHub by visiting:
remote:      https://github.com/xxx1l/OSSDP-Lab2-2023120261/pull/new/fix
remote:
To https://github.com/xxx1l/OSSDP-Lab2-2023120261.git
 * [new branch]      fix -> fix
branch 'fix' set up to track 'origin/fix'.
```

2.3 代码修改

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int length = nums.length;
        if (length == 0) {
            return new int[0];
        }
        // L 和 R 分别表示左右两侧的乘积列表
        int[] L = new int[length];
        int[] R = new int[length];

        int[] answer = new int[length];

        // L[i] 为索引 i 左侧所有元素的乘积
        // 对于索引为 '0' 的元素，因为左侧没有元素，所以 L[0] = 1
        L[0] = 1;
        for (int i = 1; i < length; i++) {
            L[i] = nums[i - 1] * L[i - 1];
        }

        // R[i] 为索引 i 右侧所有元素的乘积
        // 对于索引为 'length-1' 的元素，因为右侧没有元素，所以 R[length-1] = 1
        R[length - 1] = 1;
        for (int i = length - 2; i >= 0; i--) {
            R[i] = nums[i + 1] * R[i + 1];
        }

        // 对于索引 i，除 nums[i] 之外其余各元素的乘积就是左侧所有元素的乘积乘以右侧所有元素的乘积
        for (int i = 0; i < length; i++) {
            answer[i] = L[i] * R[i];
        }
```

```
        return answer;
    }
}
```

2.4 测试类代码

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertArrayEquals;

/**
 *
 * 测试用例设计的总体原则:
 * 1. 等价类划分: 将输入数据划分为几个等价类，选择有代表性的值作为测试用例。
 * 2. 边界值分析: 考虑到数组的边界情况，如空数组、单元素数组、最大最小可能的数组长度。
 * 3. 错误推测法: 根据经验预测容易出错的情况，如数组中有 0 的情况，以及所有元素都相同的情况。
 */
public class L2023120261_18_Test {
    private Solution solution;

    @BeforeEach
    public void setUp() {
        solution = new Solution();
    }

    /**
     * 测试目的: 验证常规情况下算法是否能正确计算出除了自身以外数组的乘积。
     * 测试策略: 使用等价类划分，选取一组正整数进行测试。
     * 测试用例: 输入一个包含不同正整数的数组 {1, 2, 3, 4}。
     */
    @Test
    public void testProductExceptSelfNormalCase() {
        int[] input = {1, 2, 3, 4};
        int[] expected = {24, 12, 8, 6};
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "常规情况下的输出不正确");
    }

    /**
```

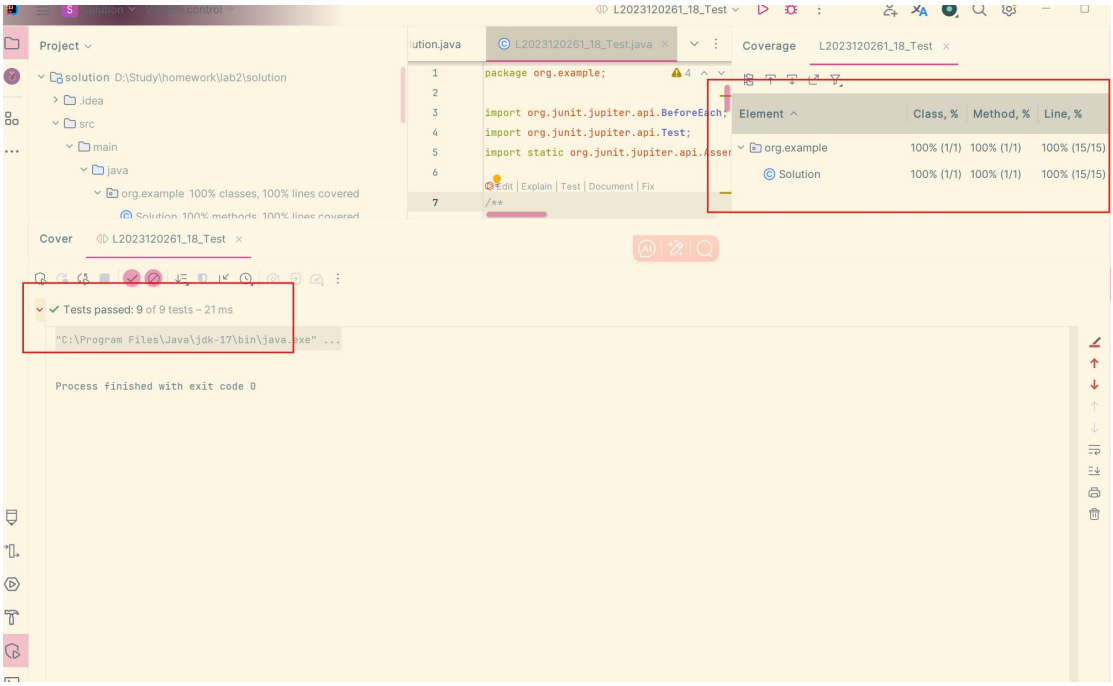
```
     * 测试目的: 检查当输入数组包含零时，函数是否能正确处理并返回预期结果。
     * 测试策略: 使用边界值分析，考虑至少存在一个 0 的情况。
     * 测试用例: 输入一个包含至少一个 0 的数组 {-1, 1, 0, -3, 3}。
     */
    @Test
    public void testProductExceptSelfWithZero() {
        int[] input = {-1, 1, 0, -3, 3};
        int[] expected = {0, 0, 9, 0, 0};
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "包含零的情况下的输出不正确");
    }

    /**
     * 测试目的: 确认单元素数组的处理是否符合预期，即返回 [1]。
     * 测试策略: 使用边界值分析，考虑最短的有效数组。
     * 测试用例: 输入一个只包含一个非零元素的数组 {5}。
     */
    @Test
    public void testProductExceptSelfSingleElement() {
        int[] input = {5};
        int[] expected = {1}; // 注意：这里的预期结果取决于题目要求
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "单个元素数组的输出不正确");
    }

    /**
     * 测试目的: 测试空数组作为输入的情况，应返回空数组。
     * 测试策略: 使用边界值分析，考虑最极端的数组长度。
     * 测试用例: 输入一个空数组 {}。
     */
    @Test
    public void testProductExceptSelfEmptyArray() {
        int[] input = {};
        int[] expected = {};
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "空数组输入的输出不正确");
    }

    /**
     * 测试目的: 检查所有元素相同的数组，验证是否能正确计算出除了自身以外数组的乘积。
     * 测试策略: 使用等价类划分，考虑所有元素相等但非零的情况。
     * 测试用例: 输入一个所有元素都相同的数组 {2, 2, 2, 2}。
```

```
     */
    @Test
    public void testProductExceptSelfAllElementsSame() {
        int[] input = {2, 2, 2, 2};
        int[] expected = {8, 8, 8, 8};
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "所有元素相同情况下的输出不正确
");
    }

    /**
     * 测试目的: 检查当输入数组包含负数时，函数是否能正确处理并返回预
期结果。
     * 测试策略: 使用等价类划分，考虑含有负数的数组。
     * 测试用例: 输入一个包含负数的数组 {-1, 2, -3, 4}。
     */
    @Test
    public void testProductExceptSelfWithNegativeNumbers() {
        int[] input = {-1, 2, -3, 4};
        int[] expected = {-24, 12, -8, 6};  // 负数乘积的正确结果
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "包含负数的情况下的输出不正确");
    }

    /**
     * 测试目的: 检查当输入数组中包含多个零时，函数是否能正确处理并返
回预期结果。
     * 测试策略: 使用错误推测法，考虑存在多个 0 的情况。
     * 测试用例: 输入一个包含多个零的数组 {0, 2, 0, 4}。
     */
    @Test
    public void testProductExceptSelfWithMultipleZeros() {
        int[] input = {0, 2, 0, 4};
        int[] expected = {0, 0, 0, 0};  // 任何包含 0 的乘积结果应为 0
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "多个零的情况下的输出不正确");
    }

    /**
     * 测试目的: 验证极大数值的情况下算法是否能正确计算出除了自身以外
数组的乘积。
     * 测试策略: 使用边界值分析，考虑数值达到整型最大值的情况。
     * 测试用例: 输入一个包含极大数值的数组 {Integer.MAX_VALUE,
Integer.MAX_VALUE, Integer.MAX_VALUE}。
```

```
     */
    @Test
    public void testProductExceptSelfWithLargeNumbers() {
        int[] input = {Integer.MAX_VALUE, Integer.MAX_VALUE,
Integer.MAX_VALUE};
        // 注意: 这里可能会导致溢出，需要根据实际情况调整测试用例或方法
实现
        int[] expected = {Integer.MAX_VALUE * Integer.MAX_VALUE,
Integer.MAX_VALUE * Integer.MAX_VALUE, Integer.MAX_VALUE *
Integer.MAX_VALUE};
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "极大数值的情况下的输出不正确");
    }

    /**
     * 测试目的: 测试更大规模的数组以确保算法在较大输入下依然有效。
     * 测试策略: 使用边界值分析，考虑较长的数组长度。
     * 测试用例: 输入一个包含 100 个元素的数组，所有元素均为 1。
     */
    @Test
    public void testProductExceptSelfLargeArray() {
        int[] input = new int[100];
        for (int i = 0; i < 100; i++) {
            input[i] = 1;
        }
        int[] expected = new int[100];
        for (int i = 0; i < 100; i++) {
            expected[i] = 1;
        }
        int[] actual = solution.productExceptSelf(input);
        assertArrayEquals(expected, actual, "大规模数组的输出不正确");
    }
}
```

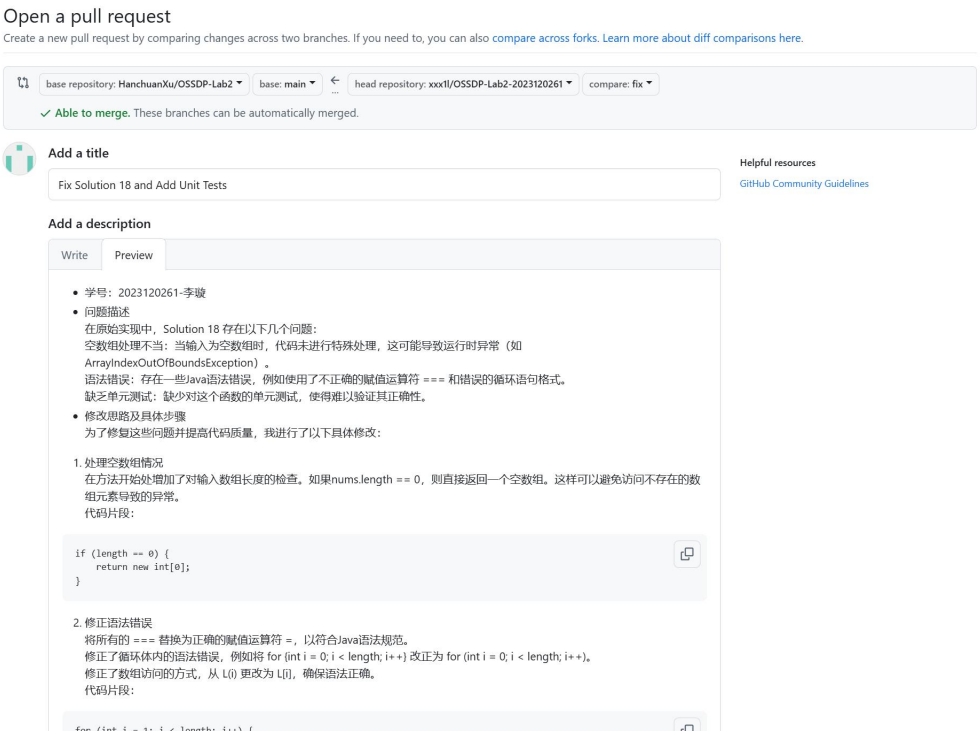
2.5 测试通过截图

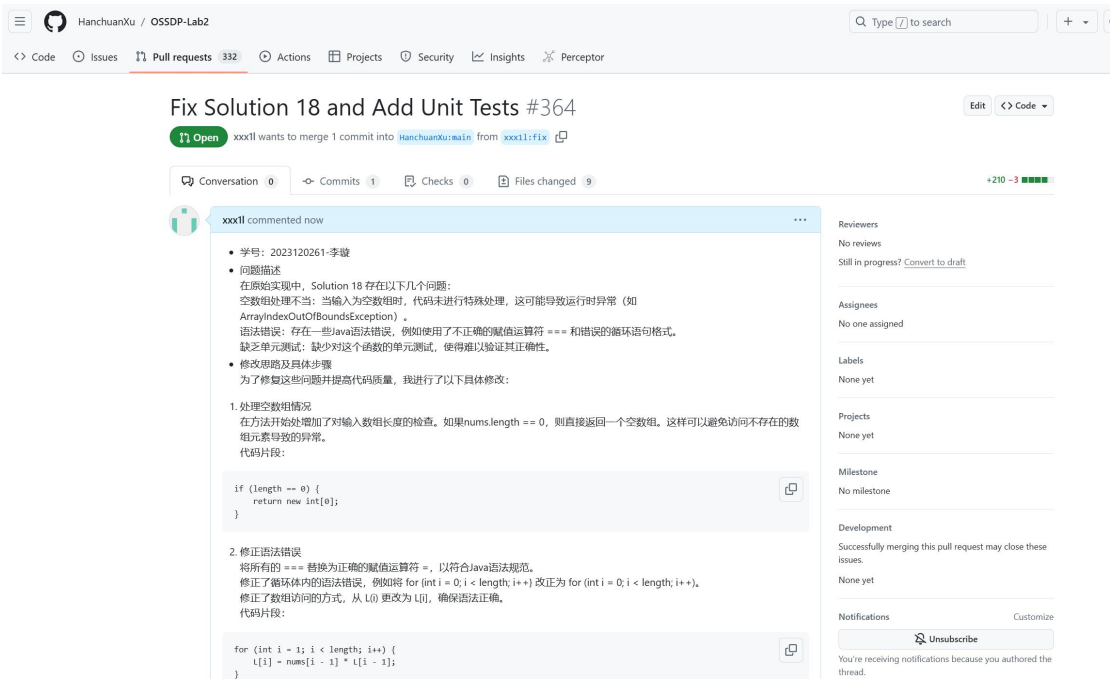
单元测试结果：9 个测试用例全部通过，覆盖率 100%

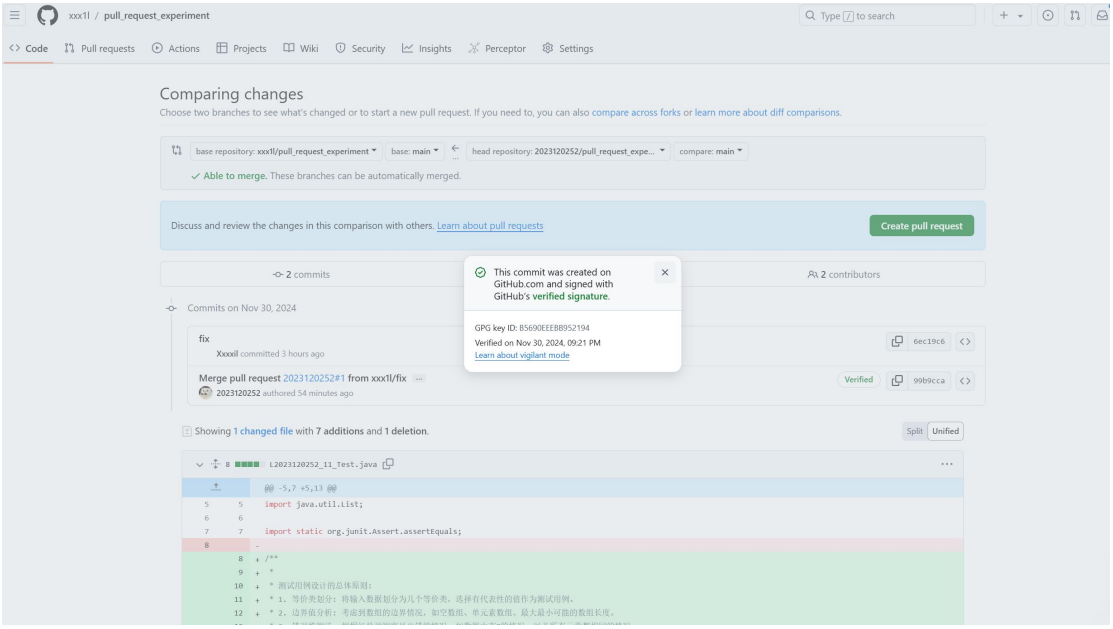


3 实验内容 2 接受 pull request

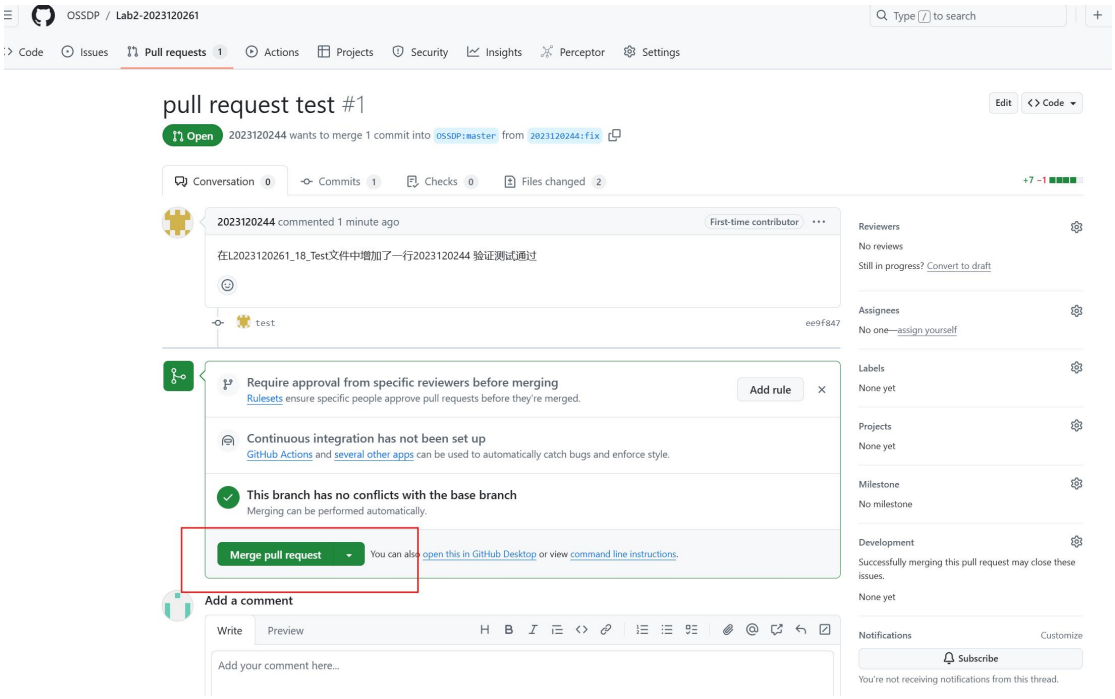
(1) create pull request

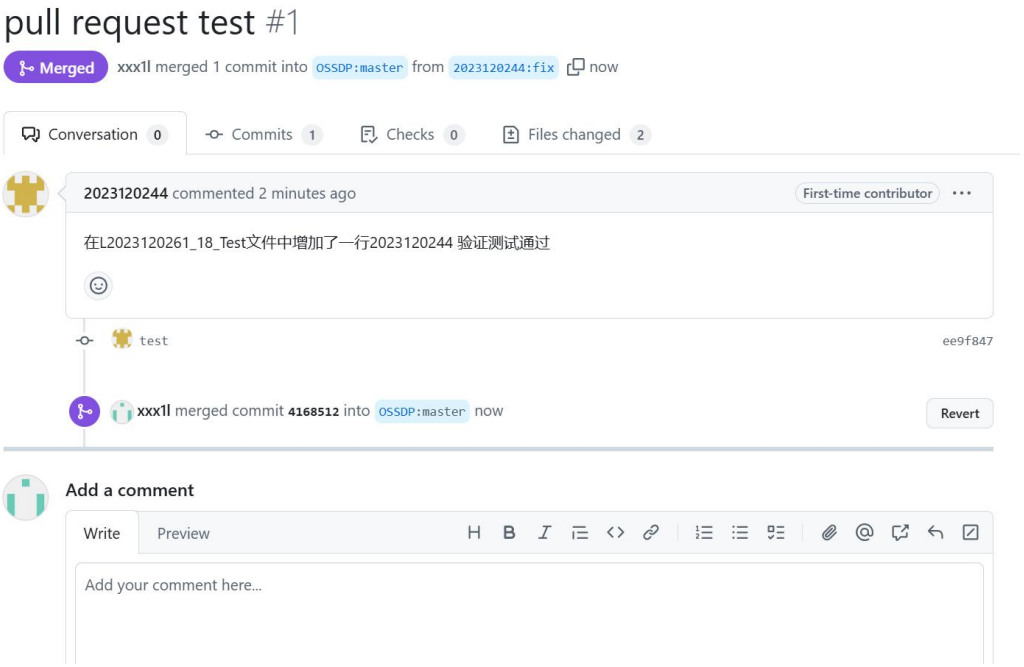


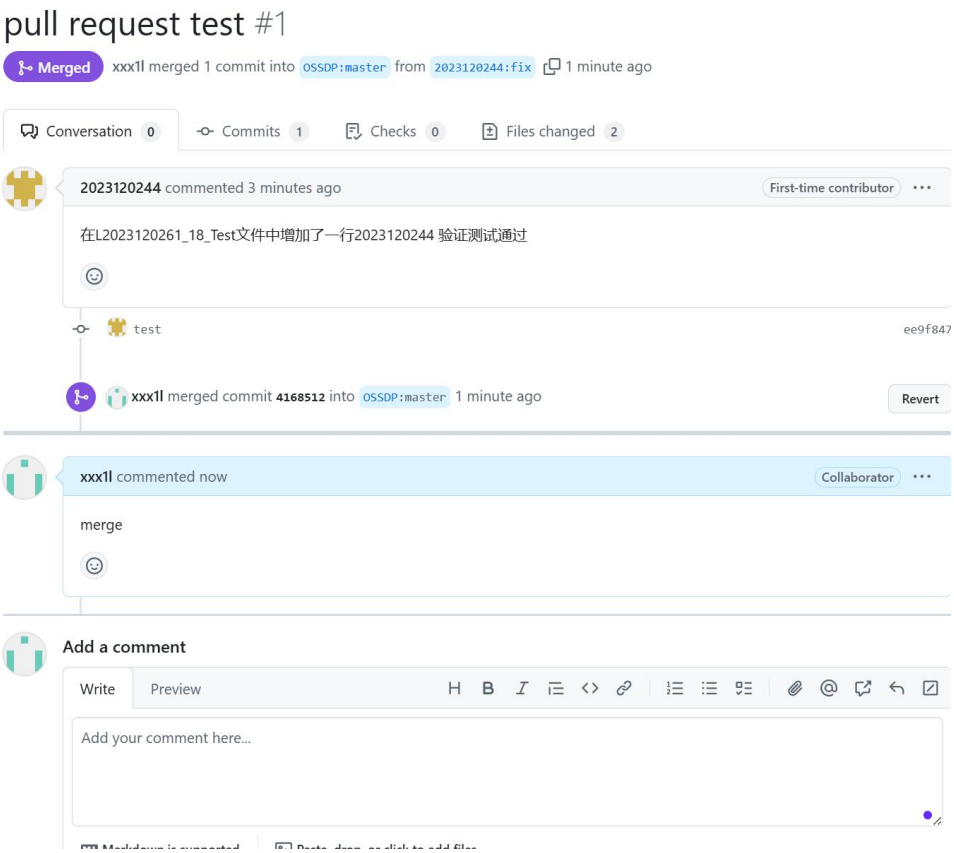




(2) merge pull request


OSSDP / Lab2-2023120261
Code
Issues
Pull requests 1
Actions
Projects
Security
Insights
Perceptor
Settings
pull request test #1
Open
2023120244 wants to merge 1 commit into OSSDP:master from 2023120244:fix
Edit
Code
Conversation 0
Commits 1
Checks 0
Files changed 2
+7 −1
2023120244 commented 1 minute ago
First-time contributor
在L2023120261_18_Test文件中增加了一行2023120244 验证测试通过
test
ee9f847
Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.
Add rule
Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Add a comment
Write
Preview
Add your comment here...
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Subscribe
You're not receiving notifications from this thread.


pull request test #1
Merged
xxx1l merged 1 commit into OSSDP:master from 2023120244:fix now
Conversation 0
Commits 1
Checks 0
Files changed 2
2023120244 commented 2 minutes ago
First-time contributor
在L2023120261_18_Test文件中增加了一行2023120244 验证测试通过
test
ee9f847
xxx1l merged commit 4168512 into OSSDP:master now
Revert
Add a comment
Write
Preview
Add your comment here...



4 实验内容 3 github 辅助工具

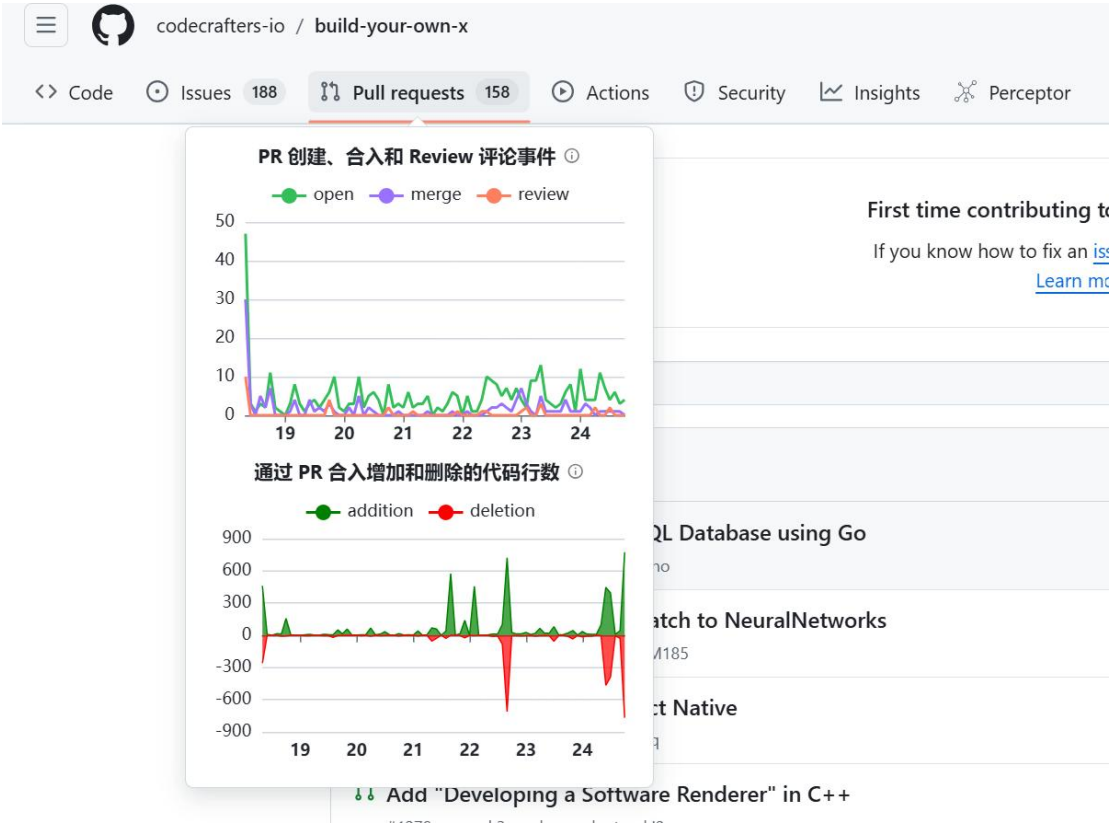
4.1 熟悉 GoodFirstIssue 工具

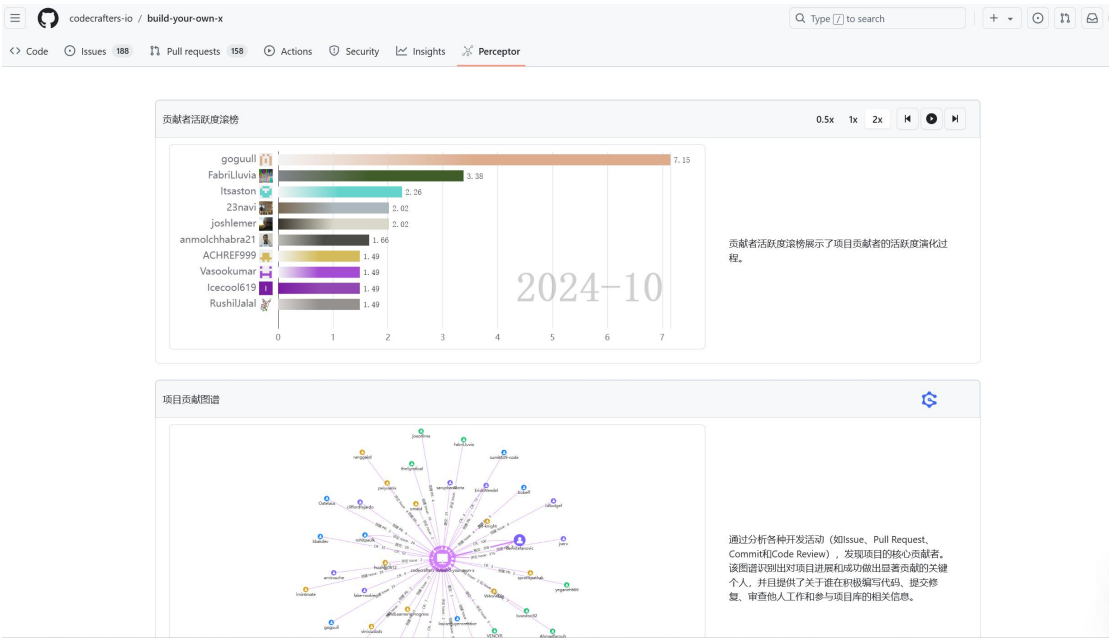
1. 确保项目托管在支持的平台上
Good First Issue 主要抓取来自 GitHub 的项目数据。因此，确保项目托管在 GitHub 上，并且已经公开可见。
2. 创建易于上手的问题
在项目中创建一些简单易懂的任务或问题，这些问题是专门为新贡献者设计的，以便他们可以轻松地上手并做出贡献。
这些任务应该有足够的指导信息，包括详细的描述、预期的行为和任何必要的上下文。
3. 标记为 "good first issue"
当创建了合适的新手任务后，在 GitHub 上将这些问题标记为 good first issue。这可以通过给 GitHub Issue 添加标签来完成。
如果已经在使用其他的标签系统（例如 beginner-friendly 或 help wanted），考虑添加 good first issue 标签以兼容更多的工具和服务。
4. 撰写清晰的贡献指南
编写一份详尽的 CONTRIBUTING.md 文件，其中包含如何设置开发环境、提交更改的流程以及其他有助于新人开始贡献的信息。

提供明确的指引可以帮助减少潜在贡献者的困惑，并鼓励更多的人参与进来。
5. 保持活跃和支持性
积极响应社区成员的问题和支持请求，营造一个友好且欢迎新人加入的氛围。定期检查是否有新的贡献者完成了他们的第一个 Pull Request，并及时给予反馈。
6. 提交项目到 Good First Issue
虽然 Good First Issue 会自动抓取带有 good first issue 标签的问题，但可以在 Good First Issue 的 GitHub 仓库中提交一个 Issue 或者 Pull Request，要求将您的项目加入到其数据库中。
7. 遵循最佳实践
确保项目遵循良好的开源实践，比如定期更新 README 文档、维护高质量的代码库等。高质量的项目更容易吸引贡献者。

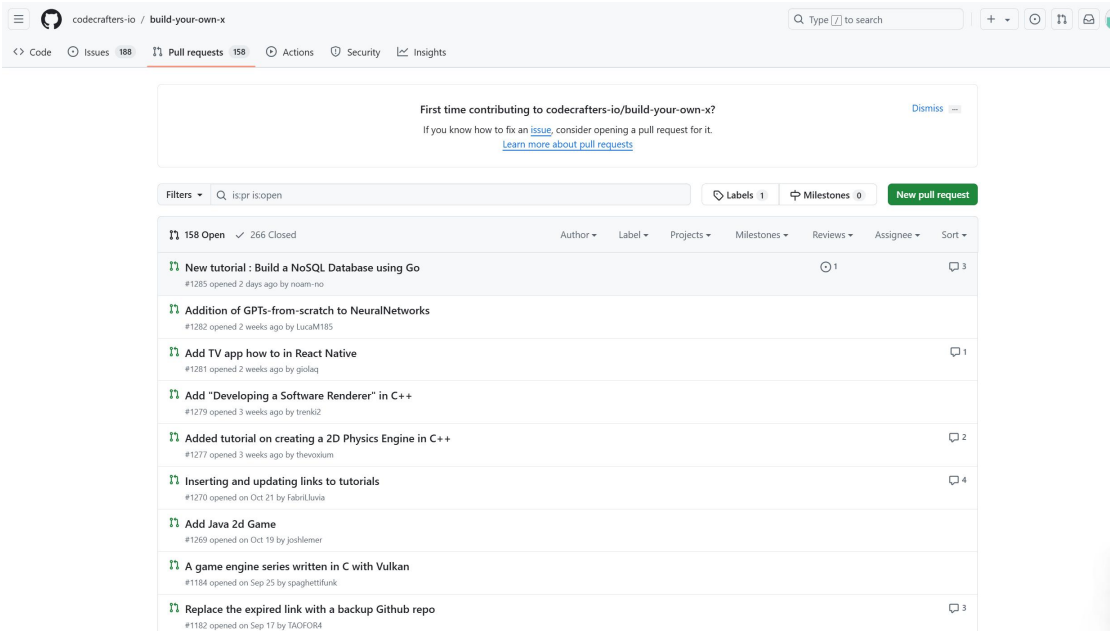
4.2 安装并使用 Hypercrx

安装插件后：





安装插件前：



4.3 利用 OpenLeaderboard 工具

1. 开源项目活跃度指标

活跃度指标衡量的是一个开源项目的活跃程度，通常包括以下几个方面：

提交次数（Commits）：项目的提交频率可以反映出开发者的活跃程度。频繁的提交意味着项目的活跃度较高。

问题和拉取请求（Issues & Pull Requests）：关注项目中的问题报告和拉取请求的数量及其处理速度。大量未处理的拉取请求可能意味着项目的维护不活跃。

参与者数量：参与贡献的开发者数量也是衡量活跃度的重要标准。如果有很多开发者参与提交和讨论，项目的活跃度较高。

参考文章：
活跃度指标文章详细介绍了如何通过这些数据来衡量一个开源项目的活跃情况。
2. 开源项目影响力指标
影响力指标用于衡量项目在开源社区的受欢迎程度和外部影响力，包括：
Stars 数量：GitHub 上的 Stars 是衡量项目受欢迎程度的一个重要指标。更多的 Stars 表示项目对开发者的吸引力更强。
Forks 数量：项目被其他开发者 Fork 的数量可以反映出该项目在其他项目中被采用的频率。
贡献者的数量和质量：除了开发者的数量外，开发者的贡献质量也很重要，比如贡献代码的质量和影响力。
3. 开源项目价值流网络
价值流网络旨在通过分析项目中的核心贡献者和他们的合作关系，描绘出开源社区中的价值流动。这种方法关注：
贡献者的角色：分析项目中的关键贡献者，了解他们如何影响项目的方向。
贡献者之间的关系：识别开发者之间的合作和交互，帮助分析社区的合作效率。
项目的可持续性：高效的价值流网络有助于项目的长期可持续发展。
4. OpenRank 的计算原理
OpenRank 是一个基于开源项目数据的排名工具，它根据开源项目的不同维度（如活跃度、影响力等）来评估项目的排名。OpenRank 的计算考虑了：
项目的受欢迎程度（如 Stars 数量）
贡献者的数量与活跃度
项目的处理效率（如拉取请求和问题的响应速度）

5 小结

本次实验经历了从理论到实践的转变，通过实际操作 GitHub 上的项目协作开发，对开源软件开发有了更深刻的理解。具体来说：
(1)理解协作流程：实验帮助我熟悉了如何在一个团队环境中使用 Git 进行代码管理与协作，包括创建分支、解决冲突、提交 Pull Request 等。
(2)掌握工具和技术：不仅限于 Git 命令行工具，还涉及到了一些辅助工具和服务（如 goodfirstissue.dev），这些都极大地提高了工作效率。
(3)问题解决能力：面对编程题目中的 bug 修复任务，增强了我的问题分析能力和解决问题的能力。
对本次实验的收获
(1)技术技能提升：
熟练掌握了基于 GitHub 的软件项目协作开发流程，学会了如何有效地利用版本控制系统来维护代码库。
学会了编写单元测试以确保代码质量，并理解了测试的重要性及其对软件稳定性的影响。
(2)团队合作意识：
体验了真实的代码审查流程，了解到沟通交流对于团队协作的重要性。
在与其他同学相互评审对方的 PR 时，学到了不同的编程风格和技术观点，促进了知识共享和个人成长。

(3)开源社区参与感：
了解到了如何让自己的项目更容易被新手开发者接受，以及如何为开源社区做贡献。
意识到良好的文档和支持机制是吸引和保留贡献者的必要条件。
(4)个人责任感：
深刻体会到了每个贡献者的工作都是整个项目的一部分，需要对自己负责的部分认真对待。
提交高质量的代码不仅是对项目的尊重，也是对其他贡献者的尊重。