

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
王雅雯	2023120240	17838829131@163.com

目 录

1 实验要求.....	1
2 实验内容 1 发送 pull request.....	2
2.1 fork 项目.....	2
2.2 git 操作命令.....	2
2.3 代码修改.....	3
2.4 测试类代码.....	4
2.5 测试通过截图.....	6
3 实验内容 2 接受 pull request.....	7
4 实验内容 3 github 辅助工具.....	9
4.1 熟悉 GoodFirstIssue 工具.....	9
4.2 安装并使用 Hypercrx.....	10
4.3 利用 OpenLeaderboard 工具.....	12
5 小结.....	16

1 实验要求

1. 实验内容 1 发送 pull request

解决编程题目 solution2，题目中存在不定数量的 bug。

- 在 github 上 fork 题目仓库到个人账号中
- Clone github 上 fork 后的仓库到本地
- 创建 fix 分支，在 fix 分支上修改所分配题目代码中的所有错误，编写测试类，测试通过无误后，提交到本地仓库并推送到 github 上
- 在 github 上向编程题目所在仓库提交 pullrequest

注意事项

- 代码修改 - 直接在代码原文件中进行修改
- 测试类注意事项 - 测试类命名规则“L_学号_X_Test.java”，其中 X 为所分配题号的编号(1-20) - 使用单元测试技术 - 测试类最开始要给出测试用例设计的总体原则，如等价类划分原则等 - 各个测试方法前面给出方法的测试目的、用到的测试用例
- PR 注意事项 - PRtitle 中注明修改的题目号 - PR 描述的最开始要注明个人学号 - PR 描述中给出修改的思路

2. 实验内容 2 接受 pull request

- 创建一个新的仓库，将实验内容 1 中建立的代码仓库中所有内容（包含代码修改成功后的文件）复制到新仓库中，任选一位其他同学，相互 fork 对方在实验内容 2 中建立的新仓库。
- 评审对方的程序代码和测试用例，对其中的文件进行修改（修改内容不强制要求：修改代码、添加注释、添加评论等均可），然后提交 PR
 - 通过 PR 进行交流
- 检查无误后接受 PR

3. 实验内容 3 github 辅助工具

- 熟悉 <https://goodfirstissue.dev>
 - 熟悉

网站的使用

<https://open-leaderboard.x-lab.info/>

阅读下面 3 篇文章

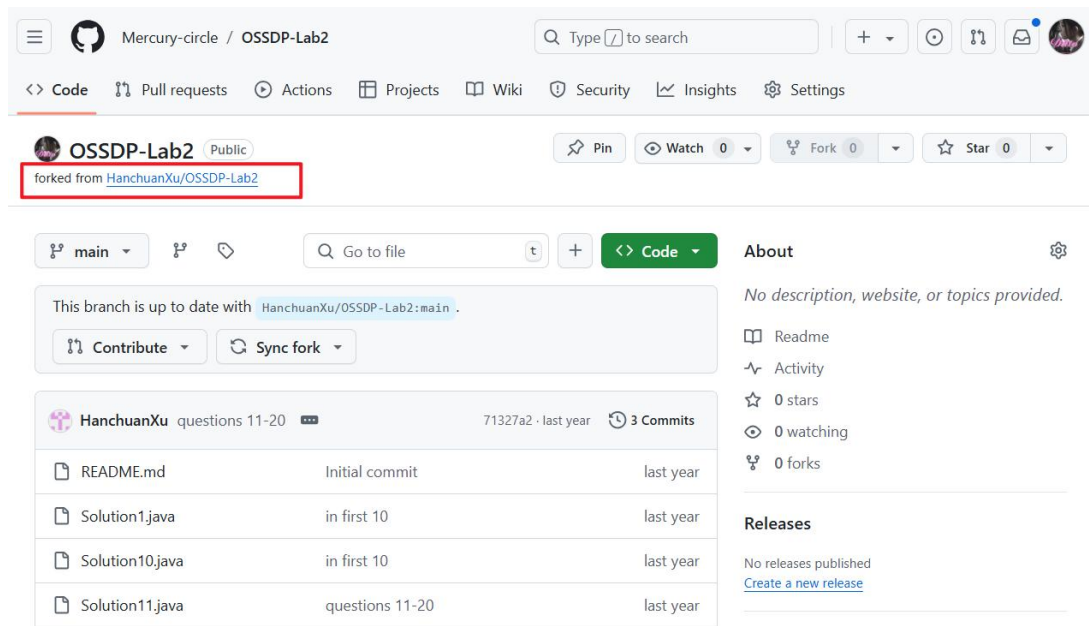
- 活跃度指标: http://blog.frankzhao.cn/how_to_measure_open_source_1/
- 影响力指标: http://blog.frankzhao.cn/how_to_measure_open_source_2/
- 价值流网络: http://blog.frankzhao.cn/how_to_measure_open_source_3/
 - 安装并使用

<https://github.com/hypertrons/hypertrons-crx>

2 实验内容 1 发送 pull request

2.1 fork 项目

在 github 上 fork 题目仓库到个人账号中, fork 后截图如下:



2.2 git 操作命令

Clone github 上 fork 后的仓库到本地

```
PS E:\A开源软件开发实践> git clone https://github.com/Mercury-circle/OSSDP-Lab2.git
Cloning into 'OSSDP-Lab2'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (21/21), done.
Receiving objects: 88% (24/27) | 20.04 KiB | 86.00 KiB/s, done.
Receiving objects: 100% (27/27), 20.04 KiB | 86.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
PS E:\A开源软件开发实践> |
```

创建 fix 分支

```
PS E:\A开源软件开发实践> cd .\OSSDP-Lab2\
PS E:\A开源软件开发实践\OSSDP-Lab2> git checkout -b fix
Switched to a new branch 'fix'
PS E:\A开源软件开发实践\OSSDP-Lab2> |
```

提交

```
PS E:\A开源软件开发实践\Lab2-2023120240> git add .
warning: in the working copy of '.idea/workspace.xml', LF will be replaced by CRLF the next time Git touches it
PS E:\A开源软件开发实践\Lab2-2023120240> git commit -m"update souldtion2"
[main (root-commit) 6478f96] update souldtion2
15 files changed, 568 insertions(+)
create mode 100644 .idea/compiler.xml
create mode 100644 .idea/misc.xml
create mode 100644 .idea/modules.xml
create mode 100644 .idea/uiDesigner.xml
```

远程推送

```
PS E:\A开源软件开发实践\OSSDP-Lab2> git push origin fix
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 2.63 KiB | 1.32 MiB/s, done.
Total 12 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Mercury-circle/OSSDP-Lab2.git
c96e06b..c9600a7 fix -> fix
```

2.3 代码修改

修改完善后的源代码如下：

```
/**
 * @description:
 *
 * 给你一个字符串 s，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证 返回结果的字典序最小（要求不能打乱其他字符的相对位置）。
 *
 *
 * 示例 1:
 *
 * 输入: s = "bcabc"
 * 输出: "abc"
 * 示例 2:
 *
 * 输入: s = "cbacdcbc"
 * 输出: "acdb"
 *
 * 1 <= s.length <= 104
 * s 由小写英文字母组成
 */
class Solution2 {
    public String removeDuplicateLetters(String s) {
        boolean[] vis = new boolean[26];
        int[] num = new int[26];
        for (int i = 0; i < s.length(); i++) {
            num[s.charAt(i) - 'a']++;
        }

        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (!vis[ch - 'a']) {
                while (sb.length() > 0 && sb.charAt(sb.length() - 1) > ch) {
                    if (num[sb.charAt(sb.length() - 1) - 'a'] > 0) {
                        vis[sb.charAt(sb.length() - 1) - 'a'] = false;
                        sb.deleteCharAt(sb.length() - 1);
                    } else {
                        break;
                    }
                }
                vis[ch - 'a'] = true;
                sb.append(ch);
            }
        }
    }
}
```

```
        num[ch - 'a'] -= 1;
    }
    return sb.toString();
}
}
```

2.4 测试类代码

将测试类代码的原文件粘贴在此处

```
import org.junit.Test;
import static org.junit.Assert.*;

public class L2023120240_2_Test {

    private Solution2 solution2 = new Solution2();

    // 等价类划分原则：空字符串是一个独立的等价类
    @Test
    public void testEmptyString() {
        String input = "";
        String expected = "";
        assertEquals(expected, solution2.removeDuplicateLetters(input));
    }

    // 等价类划分原则：单个字符的字符串是一个独立的等价类
    @Test
    public void testSingleCharacterString() {
        String input = "a";
        String expected = "a";
        assertEquals(expected, solution2.removeDuplicateLetters(input));
    }

    // 等价类划分原则：所有字符都是唯一的字符串是一个独立的等价类
    @Test
    public void testAllUniqueCharacters() {
        String input = "abc";
        String expected = "abc";
        assertEquals(expected, solution2.removeDuplicateLetters(input));
    }

    // 等价类划分原则：所有字符都相同的字符串是一个独立的等价类
    @Test
    public void testAllSameCharacters() {
        String input = "aaaaa";
        String expected = "a";
        assertEquals(expected, solution2.removeDuplicateLetters(input));
    }

    // 等价类划分原则：包含多个重复字符的字符串是一个独立的等价类
    @Test
    public void testMultipleDuplicates() {
        String input = "bcabc";
        String expected = "abc";
        assertEquals(expected, solution2.removeDuplicateLetters(input));
    }
}
```

```
// 等价类划分原则：包含复杂重复字符的字符串是一个独立的等价类
@Test
public void testComplexDuplicates() {
    String input = "cbacdcbc";
    String expected = "acdb";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 边界值分析：字符串长度为 1 的情况是一个边界条件
@Test
public void testBoundaryCase() {
    String input = "z";
    String expected = "z";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：包含所有小写字母的字符串是一个独立的等价类
@Test
public void testLongString() {
    String input = "abcdefghijklmnopqrstuvwxyz";
    String expected = "abcdefghijklmnopqrstuvwxyz";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：包含所有小写字母且有重复的字符串是一个独立的等价类
@Test
public void testLongStringWithDuplicates() {
    String input = "aabbccddeeffgghhiijjkllmmnnnooppqrrssttuuvvwwxxyyzz";
    String expected = "abcdefghijklmnopqrstuvwxyz";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：逆序字符串是一个独立的等价类
@Test
public void testReversedString() {
    String input = "zyxwvutsrqponmlkjihgfedcba";
    String expected = "zyxwvutsrqponmlkjihgfedcba";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：逆序字符串且有重复的字符串是一个独立的等价类
@Test
public void testReversedStringWithDuplicates() {
    String input = "zzyyxxwvutsrqponmlkjihgfedcba";
    String expected = "zyxwvutsrqponmlkjihgfedcba";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：包含重复模式的字符串是一个独立的等价类
@Test
public void testRepeatedPattern() {
    String input = "abcabcabcabc";
    String expected = "abc";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 等价类划分原则：包含复杂模式的字符串是一个独立的等价类
```

```
@Test
public void testComplexPattern() {
    String input = "abacb";
    String expected = "abc";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}

// 边界值分析：可能有多个解决方案的字符串是一个边界条件
@Test
public void testEdgeCaseWithMultipleSolutions() {
    String input = "abacb";
    String expected = "abc";
    assertEquals(expected, solution2.removeDuplicateLetters(input));
}
}
```

2.5 测试通过截图

单元测试通过后的截图如下：

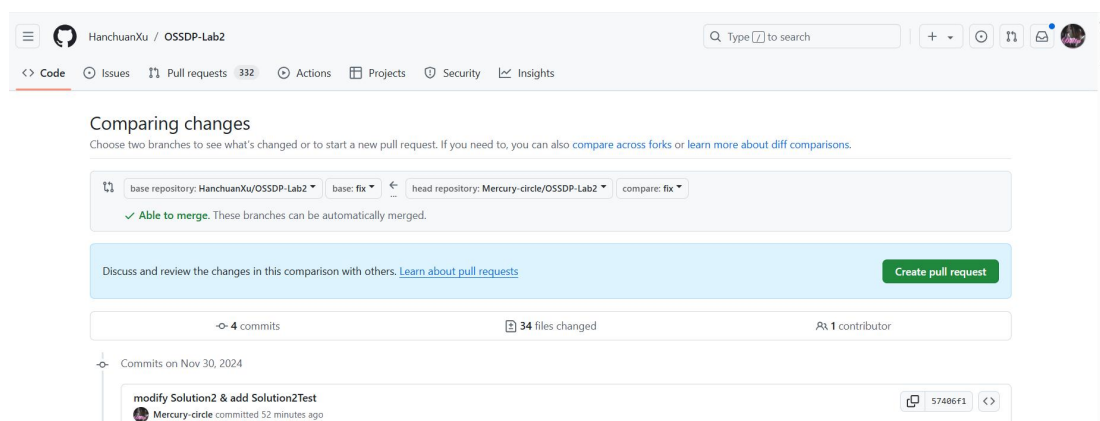


发送 PR 过程如下：

首先需要链接最上层的仓库：<https://github.com/HanchuanXu/OSSDP-Lab2>

```
PS E:\A开源软件开发实践\OSSDP-Lab2> git remote add upstream https://github.com/HanchuanXu/OSSDP-Lab2.git
PS E:\A开源软件开发实践\OSSDP-Lab2> git remote -v
origin https://github.com/Mercury-circle/OSSDP-Lab2.git (fetch)
origin https://github.com/Mercury-circle/OSSDP-Lab2.git (push)
upstream https://github.com/HanchuanXu/OSSDP-Lab2.git (fetch)
upstream https://github.com/HanchuanXu/OSSDP-Lab2.git (push)
```

进入被 fork 的上层仓库，然后 create pull request



添加 title 和 description

base repository: HanchuanXu/OSSDP-Lab2

base: fix

head repository: Mercury-circle/OSSDP-Lab2

compare: fix

✓ **Able to merge.** These branches can be automatically merged.

Add a title

solution2

Helpful resources

[GitHub Community Guidelines](#)

Add a description

WritePreview

HBI≡<>🔗⋮⋮⋮🔗@🔄🔗

2023120240-王雅雯
修改的思路如下:
①修正字符索引:
将 num[s.charAt(i) - ''] 和 vis[s.charAt(i) - ''] 改为 num[s.charAt(i) - 'a'] 和 vis[s.charAt(i) - 'a'], 以正确地索引字母数组。
②调整数组大小
将 boolean[] vis = new boolean[25]; 和 int[] num = new int[25]; 改为 boolean[] vis = new boolean[26]; 和 int[] num = new int[26];, 以覆盖全部 26 个字母。
③修正循环条件
将 for (int i = 0; i < s.length()+1; i++) 改为 for (int i = 0; i < s.length(); i++), 避免数组越界。
④修正计数
将 num[ch - 'a'] += 1; 改为 num[ch - 'a'] -= 1; , 以正确地减少每个字符的计数

Markdown is supported

Paste, drop, or click to add files

Allow edits by maintainers

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

3 实验内容 2 接受 pull request


首先 fork 学号为 2023120244 的同学

Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Required fields are marked with an asterisk (*).


Owner *

 Mercury-circle ▾

 /

Repository name *

fork-Lab2-2023120244

 fork-Lab2-2023120244 is available.

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)


Lab2-2023120244 created by GitHub Classroom

☒ Copy the

main

 branch only

Contribute back to OSSDP/Lab2-2023120244 by adding your own branch. [Learn more.](#)

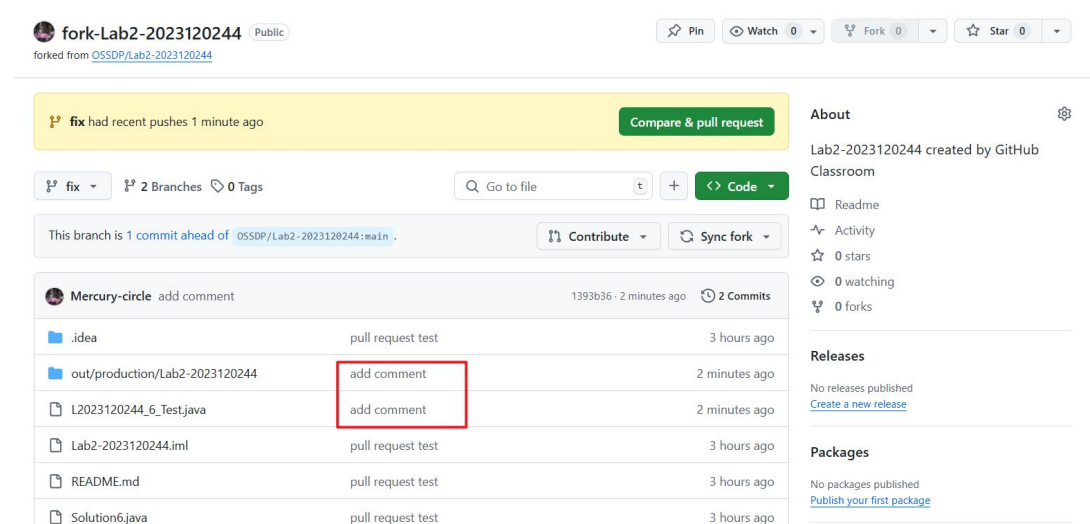
 You are creating a fork in your personal account.

Create fork

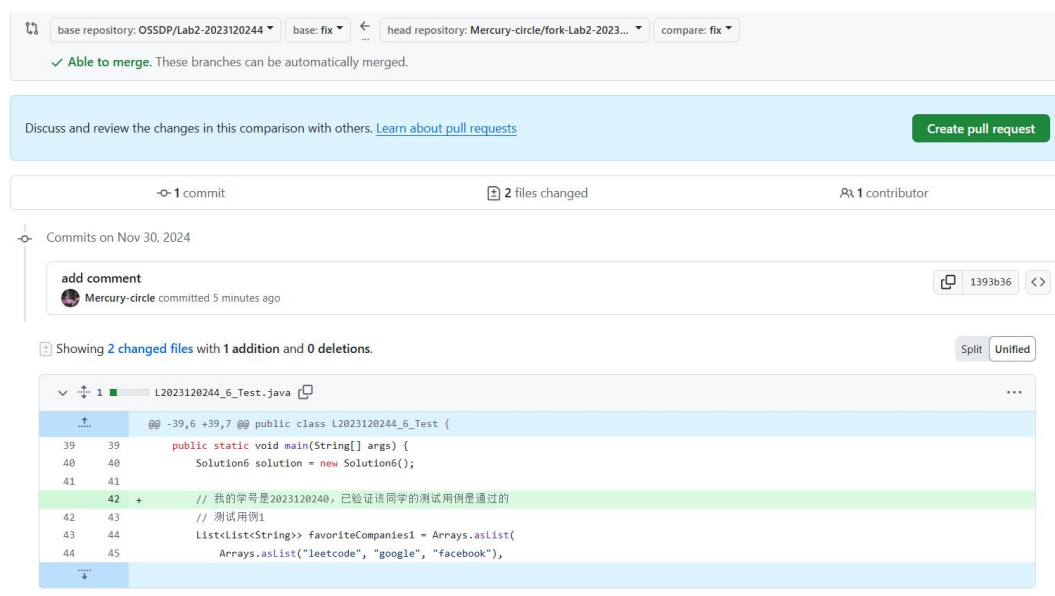
拉取仓库内容并在本地建立分支 fix

```
PS E:\A开源软件开发实践> git clone https://github.com/Mercury-circle/fork-Lab2-2023120244.git
Cloning into 'fork-Lab2-2023120244'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 16 (delta 2), reused 16 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (16/16), 4.82 KiB | 821.00 KiB/s, done.
Resolving deltas: 100% (2/2), done.
PS E:\A开源软件开发实践> git checkout -b fix
fatal: not a git repository (or any of the parent directories): .git
PS E:\A开源软件开发实践> cd .\fork-Lab2-2023120244\
PS E:\A开源软件开发实践\fork-Lab2-2023120244> git checkout -b fix
Switched to a new branch 'fix'
PS E:\A开源软件开发实践\fork-Lab2-2023120244> |
```

对 fork 的文件增加了评价，然后提交如下：

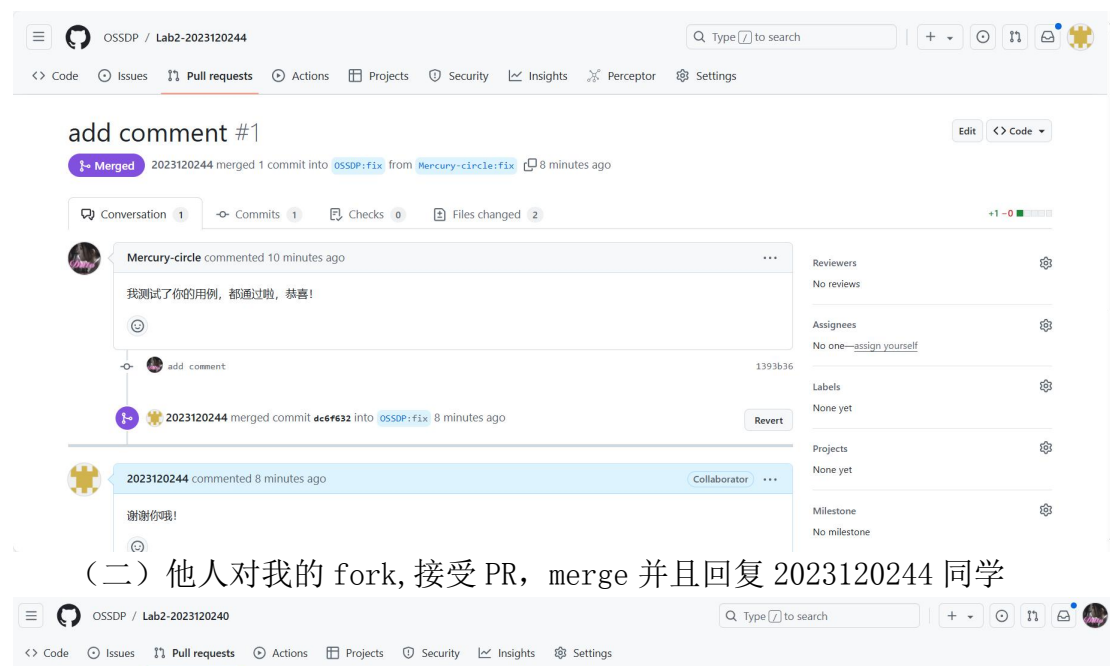


然后 create pull request



双方通过 github 上 PR 交流的记录截图如下：

（一）我对他人的 fork, 发送 PR, 收到 2023120244 同学的回复：



4 实验内容 3 github 辅助工具

4.1 熟悉 GoodFirstIssue 工具

如何使自己的开源项目被此网站收录

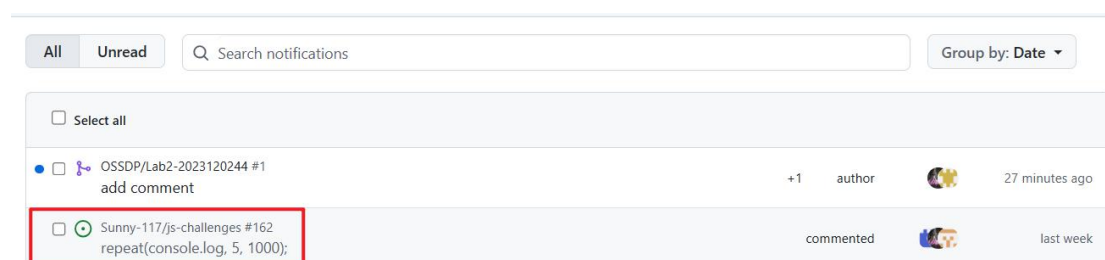
在 Good First Issue 中添加新项目，遵循这些简单的步骤：

1. 为了保持 Good First Issue 项目的质量，请确保您的 GitHub 存储库符合以下标准

- 至少有三个 issue 带有 good first issue 的标签。默认情况下，这个标签已经存在于所有的仓库中。

- 至少有 10 个贡献者。
 - 需要有一个 README 文件包含项目的详细启动说明，以及一个 contribute.md 文件可以为新的贡献者提供参与贡献的指导说明。
 - 需要积极维护这个开源项目
2. 在 data/repository.toml 中添加仓库的路径
 3. 创建一个新的拉取请求。请在 PR 描述中添加到存储库问题页面的链接。
- 一旦合并了 pull 请求，更改将在 goodfirstissue.dev 上实时显示。

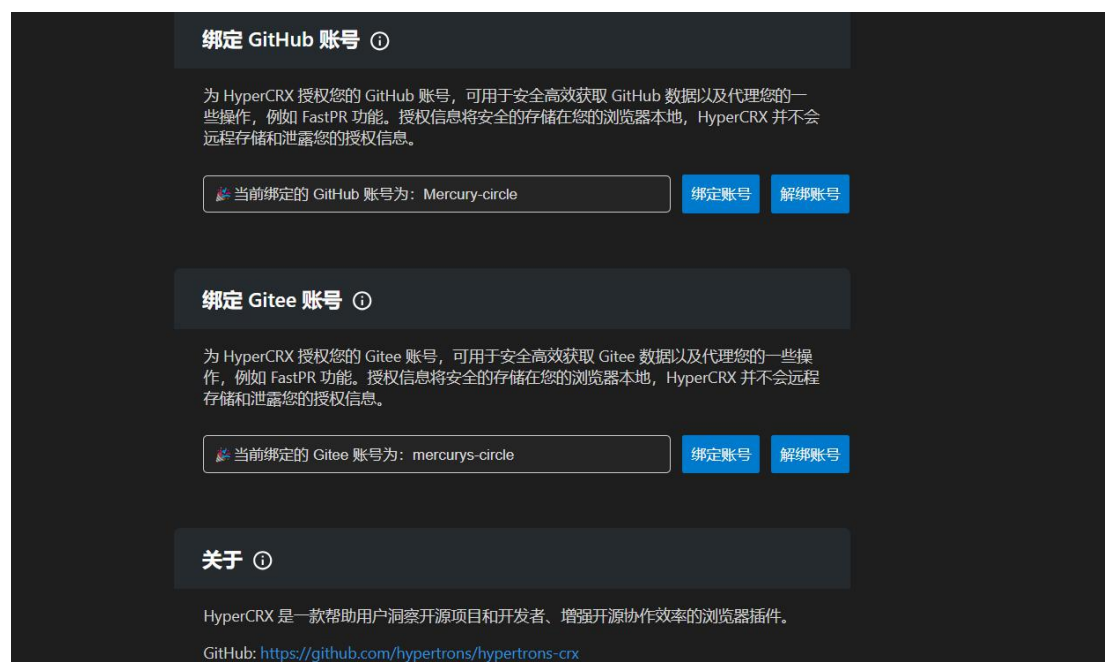
我曾参与的开源项目如下：



4.2 安装并使用 Hypercrx

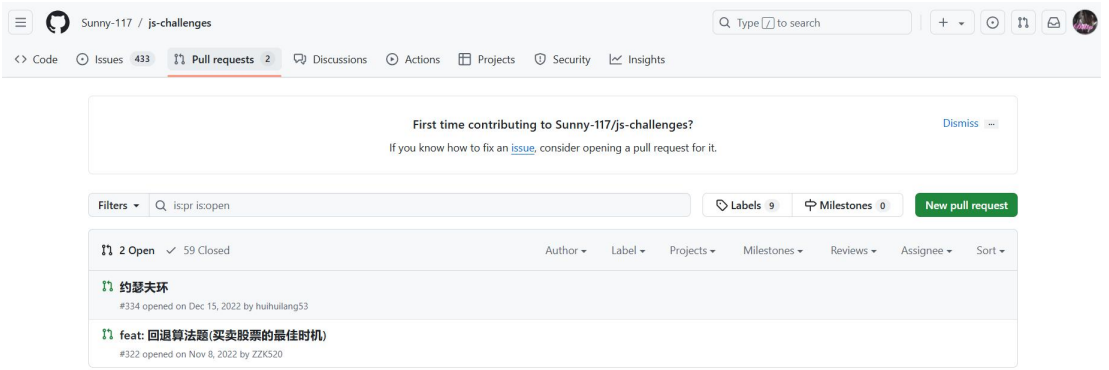
访问此网站 <https://github.com/hypertrons/hypertrons-crx>

安装 Hypercrx 插件，绑定账号如下：

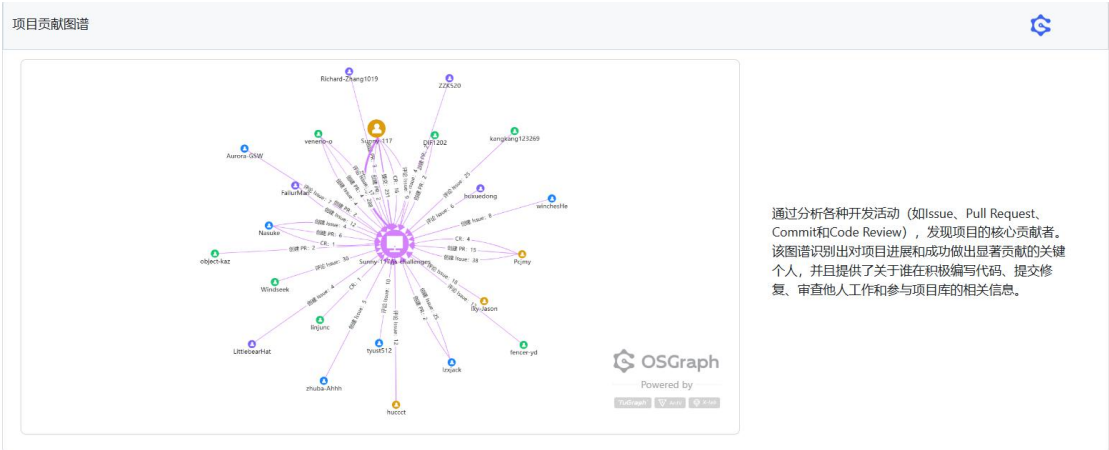
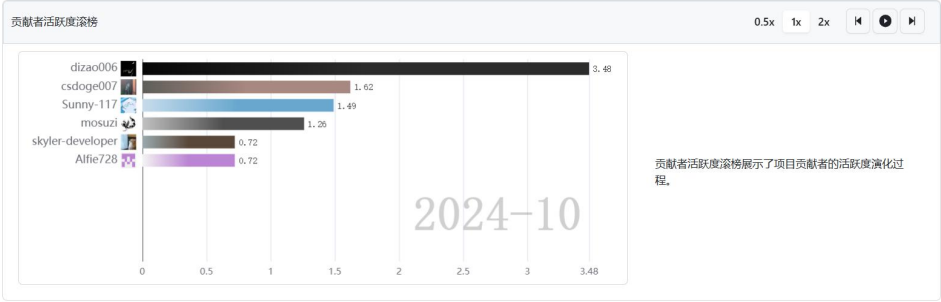
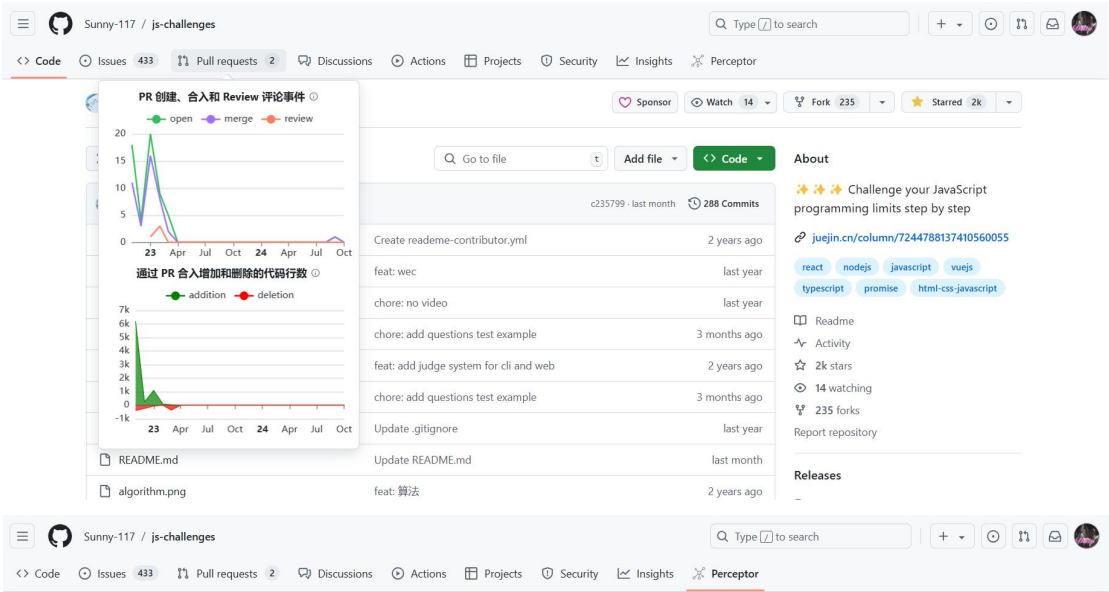


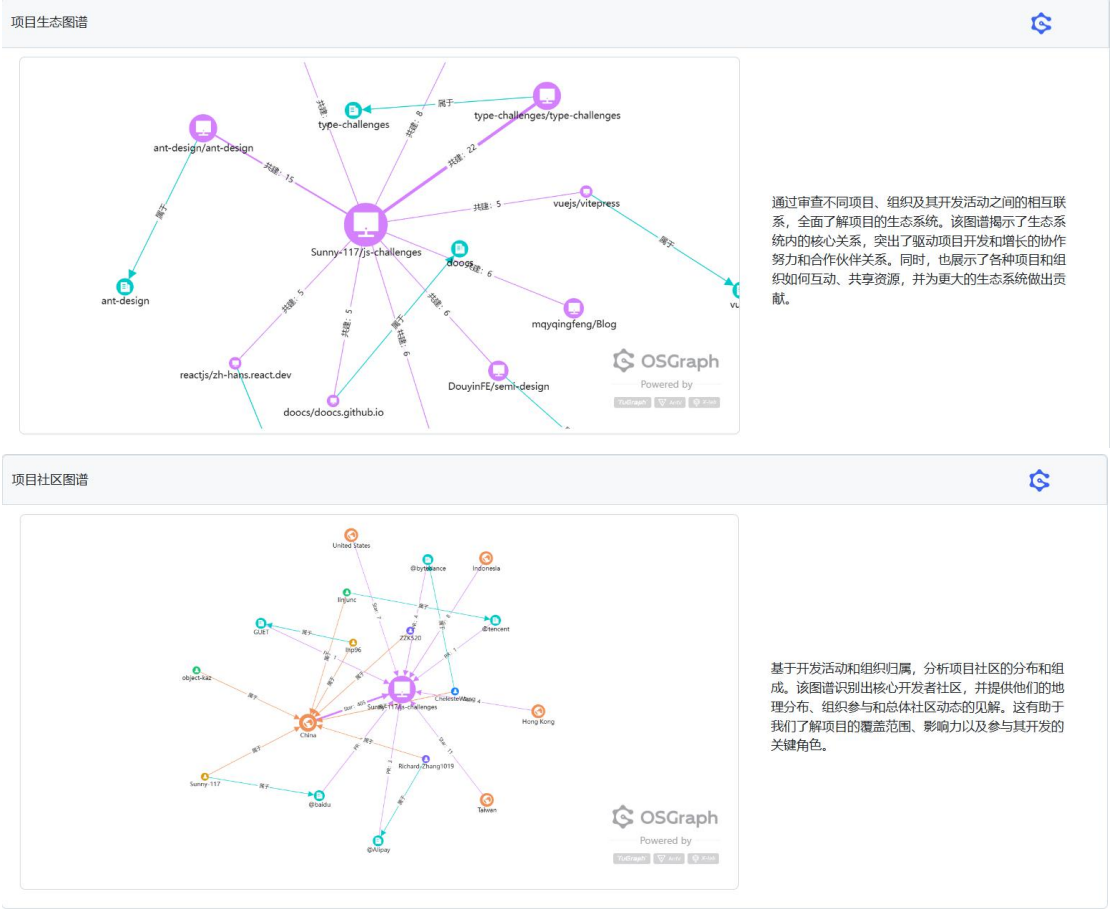
下面是某个 star 在 100+ 的项目。如下图：

使用插件前：



利用插件查询项目状态的截图如下：





4.3 利用 OpenLeaderboard 工具

（一）OpenLeaderboard 工具

OpenLeaderboard 是一个用于评估和展示 GitHub 上开源项目统计数据工具。它结合了活跃度、影响力以及价值流网络等多种度量指标，帮助用户全面了解开源项目的健康状况和发展潜力

使用步骤：

1. 访问 OpenLeaderboard 平台

打开浏览器，访问 OpenLeaderboard 的官网或 GitHub 仓库。
确保有足够的权限来访问相关数据和报告。

2. 选择度量指标

OpenLeaderboard 提供了多种度量指标，如活跃度、影响力、关注度等。
选择感兴趣的指标，例如活跃度或协作影响力

3. 筛选项目

可以通过关键词搜索、分类筛选（如编程语言、项目类型等）来找到感兴趣的项目，也可以选择特定的时间段来查看项目在该时间段内的统计情况。

4. 查看项目统计报告

选定项目后，OpenLeaderboard 会生成详细的统计报告，包括该项目的活跃度、影响力、关注度等指标。

报告中通常会包含以下内容：

活跃度：开发者的活跃行为统计，如提交代码、提出和评论 Issue、提交和 Review PR 等。

影响力：基于开源协作网络的计算结果，展示项目的影响力排名和变化趋势。

关注度：项目的 star、fork、watch 等单向关注行为的统计。

贡献者：项目的贡献者列表和他们的活跃度。

项目依赖关系：项目之间的依赖关系和使用情况。

5. 对比分析

可以在 OpenLeaderboard 中选择多个项目进行对比分析，了解它们在不同度量指标下的表现。

对比分析可以帮助你发现项目的优劣，以及项目之间的协作关系。

6. 参与讨论和反馈

OpenLeaderboard 通常会有一个社区讨论区或反馈渠道，可以参与讨论，提出自己的看法或改进建议。

社区的反馈可以帮助工具的开发者优先完善和优化度量方法。

（二）评价一个开源项目

衡量开源项目的活跃度的方法是考虑不同行为的权重，例如提交 issue、发起 pull request 等，然后计算开发者的活跃度。评价一个开源项目需要考虑开发者在不同项目间的协作情况，以评估项目的协作影响力。这种活跃度指标虽然有效，但是未能充分利用项目间的关系。改进方法是通过开源协作网络，使用调和平均方法计算项目间的协作关联度，并应用 PageRank 算法计算项目的协作影响力，这种方法有助于更全面地理解开源项目的价值和影响力。评价开源项目还可以用价值流网络，构建一个高度可扩展的数学模型来衡量开源软件的社会价

值。这个网络模型考虑了开发者与项目之间的关系，包括活跃度、关注度、依赖关系等，以衡量从生产端到消费端的价值流动。这个方法不仅有助于理解开源项目的价值，还能为构建完整开源经济生态体系提供基础。

1. 开源项目活跃度 (Activity Degree, Ad)

定义：活跃度是指开发者在特定时间段内在 GitHub 上的活跃程度。通过统计开发者在项目中的各种行为（如提交代码、提出 Issue、评论、Review 代码等），并赋予不同的权重，可以量化开发者的活跃度。活跃度不仅反映了项目的活跃程度，还体现了开发者对项目的贡献。

计算方法：

$$Ad = \sum w_i c_i$$

- Ad: 开发者的活跃度。
- c_i : 开发者某项行为的次数。
- Issue 评论: 1 分
- 提出 Issue: 2 分
- 提交 PR: 3 分
- Review PR 代码: 4 分
- 合并 PR: 5 分

项目活跃度 (Project Activity Degree, Ar):

$$Ar = \sum \sqrt{Ad}$$

Ar: 项目的活跃度。

开方操作是为了降低核心开发者过高的活跃度带来的影响，体现了项目的整体参与度。

2. 开源项目影响力 (Collaboration Influence, CI)

定义：影响力是指项目在整个开源生态中的重要性，不仅考虑了项目的活跃度，还考虑了项目之间的协作关联。通过构建开源协作网络，利用图分析算法（如 PageRank）计算项目的影响力。

计算方法：

- 开源协作网络：项目和项目之间的关联通过共同活跃的开发者来建立，每个

开发者对两个项目协作关联度的贡献为：

$$\frac{Ad_{p1} \cdot Ad_{p2}}{Ad_{p1} + Ad_{p2}}$$

- 这里使用了调和平均的计算方法，即只有开发者在两个项目上都非常活跃时，才会对这两个项目的关联度产生较大影响。

PageRank 算法：通过项目之间的协作网络，利用 PageRank 算法计算每个项目的协作影响力。

$$CI = \text{PageRank}(\text{协作网络})$$

PageRank 算法原理：一个高质量的项目会被更多的项目引用，而高质量的项目引用的其他项目质量也较高。通过迭代计算，最终得到每个项目的影响力。

3. 价值流网络 (Value Flow Network, VFN)

定义：价值流网络是从开源软件产生的社会价值的角度进行分析，通过构建一个包含开发者、项目、软件、公司、基金会、投资机构、用户等多实体的价值网络，评估每个实体的社会价值。

计算方法：

网络结构：项目和开发者之间通过活跃度和关注度连接，项目之间通过依赖关系和使用关系连接，开发者之间通过关注关系连接，公司和基金会通过拥有关系连接，投资机构通过投资关系连接，用户通过使用关系连接。

价值传递：每个节点会将一部分价值传递给其他节点，具体传递方式如下：

- 开发者通过活跃度、关注度和对其他开发者的关注将价值传递给项目、软件和开发者。
- 项目通过依赖关系和使用关系将价值传递给其他项目和软件。
- 公司通过拥有关系和投资关系将价值传递给项目和投资机构。
- 基金会通过拥有关系和赞助关系将价值传递给项目和公司。
- 投资机构通过投资关系将价值传递给公司。
- 用户通过活跃度和关注关系将价值传递给项目和软件。
- 价值流网络模型的稳态解：

为了确保模型的稳态解，需要满足两个条件：

随机矩阵：转移矩阵的每一行之和为 1，表示价值的总和不变。

素矩阵或本原矩阵：转移矩阵需满足不可约和非周期的条件，保证系统能够收敛到一个稳态。

4. OpenRank

定义：OpenRank 是在价值流网络的基础上，通过一系列数学和业务模型的结合，评估开源项目和开发者的综合价值。OpenRank 不仅考虑了项目的活跃度和影响力，还引入了更多维度的数据，如消费侧的数据（项目被用户使用的情况）、生产侧的数据（开发者的能力和贡献）等。

计算原理：

- 业务模型：定义开源生态中各个实体之间的关系和价值传递机制。
- 数学模型：基于高维异质信息网络，利用随机过程的理论（如 PageRank 算法的变种）计算每个节点的价值。
- 价值传递：通过多种关系（如活跃度、关注度、依赖关系、使用关系等）进行价值传递，形成一个复杂的价值流网络。

优点：

- 可扩展性：业务模型和数学模型解耦，可以方便地引入新的数据和关系。
- 综合评估：从多个维度评估开源项目的综合价值，包括生产侧和消费侧。
- 稳定性：通过数学模型的约束，确保价值评估结果的稳定性和鲁棒性。

局限性：

- 数据获取：需要大量的数据，部分数据难以获取和关联。
- 模型设计：业务模型设计人员需要有一定的数学知识，确保模型可以收敛到稳态解。
- 计算复杂度：由于涉及多个实体和关系，计算复杂度较高，需要高效的算法和计算资源。

5 小结

实验内容 1：发送 pull request

在实验内容 1 中，我面临的主要任务是对一个编程题目进行修复，并提交一

个 pull request。首先，我在 GitHub 上 fork 了题目仓库到自己的账号中，然后通过 `git clone` 命令将仓库克隆到了本地。然后我创建了一个名为 `fix` 的分支，在这个分支上对所分配的题目代码进行了修改。这个题目中确实存在几个 bug，难度不大，有在力扣做过这道题，比较轻松的修复了这些问题。为了确保代码修复的正确性和稳定性，我编写了一个测试类。我在测试类的最开始部分给出了测试用例设计的总体原则，比如等价类划分原则，并在每个测试方法前注明了测试目的和用到的测试用例。

完成所有修改和测试后，我将本地的代码提交到了 `fix` 分支，并推送到 GitHub 上。最后，我向编程题目所在仓库提交了一个 pull request。在 PR 的标题中注明了修改的题目号，并在 PR 的描述中写明了我的学号和修改思路。

实验内容 2: 接受 pull request

在实验内容 2 中，我需要创建一个新的仓库，将实验内容 1 中建立的代码仓库中所有内容复制到新仓库中。我选择了另一位同学的新仓库，fork 了她的仓库到我的账号中。我评审了对方的程序代码和测试用例，我进行了一些修改，添加了一些评论，然后提交了一个 pull request。通过 PR，我和对方进行了交流。检查无误后，我接受了对方的 PR，而对方也接受了我的 PR。这个过程让我体会到了代码评审的重要性和团队协作的价值。

实验内容 3: GitHub 辅助工具

在实验内容 3 中，我熟悉了几个 GitHub 辅助工具和网站。

<https://goodfirstissue.dev> 这个网站帮助我找到了一些适合初学者的项目和 issue，这对于刚刚接触开源项目的我来说非常有帮助。然后，我访问了 <https://open-leaderboard.x-lab.info/>，并阅读了以下三篇文章：

活跃度指标：这篇文章介绍了如何通过活跃度指标来衡量开源项目的健康发展。活跃度指标包括提交数量、参与者数量等。

影响力指标：这篇文章讨论了如何通过影响力指标来评估开源项目的影响，如 star 数量、fork 数量、issue 关闭速度等。

价值流网络：这篇文章介绍了价值流网络的概念，以及如何通过价值流网络来分析项目的贡献者和贡献路径。

最后，我安装并使用了 <https://github.com/hypertrons/hypertrons-crx>。

这个 Chrome 扩展程序可以帮助我更方便地管理 GitHub 上的项目和 issue，提供了一些实用的功能，让我能更清楚的看到数据以及开源项目中各贡献值的协作关系。

通过这次实验，我不仅学习了技术知识，还提升了团队协作和项目管理能力，感觉收获颇丰。希望未来能有更多这样的实验机会，让我们在实践中不断进步。对于这个实验，我希望未来能有更多真实的开源项目供我们参与，这样可以更好地将理论知识应用到实践中。