北京大学

硕士研究生学位论文

题目：基于相似检测的交互式缺陷报告工具的设计与实现

姓　　名：谢佳亮

学　　号：1101213952

院　　系：信息科学技术学院

专　　业：计算机软件与理论

研究方向：软件工业化生产技术与系统

导师姓名：周明辉　副教授

二〇一四 年 五 月

版权声明

摘 要

高质量软件的开发需要及时、高效的缺陷收集和处理机制。在开源项目中，缺陷追踪系统（Issue-tracking System）被广泛用于记录和追踪用户反馈，特别是软件缺陷。因为开源项目中存在许多缺乏技术经验的用户，因此项目中存在相当数量的、由他们提交的低质量缺陷报告[1]。其中，信息缺失和重复报告是报告质量低下的主要原因。为了完善缺失的信息，项目成员（包括审核者、开发者和管理者）和报告者需要花费大量时间和精力进行交流。这不但降低了报告修复的效率，还损害了报告者的体验。重复报告很可能导致项目成员重复的工作，而发现、检测重复报告的工作本身也耗时耗力。项目成员的时间极为宝贵，如果他们被过多的低质量缺陷报告干扰，就无法及时高效地修复软件缺陷。

目前，许多研究工作致力于提高报告的质量，主要分为重复检测和质量反馈两个方面。前者主要采用自然语言分析等方法，计算缺陷报告间的相似度，并由此检测重复报告。这部分工作大都假定被检测的报告具有完整的信息，然而这个假设对于大量存在的、信息缺失的报告并不成立（它们往往由缺乏经验的报告者提交）。后者主要探索影响报告质量的因素，通过对缺陷报告质量进行建模，指导报告者完善缺陷报告。这部分工作主要针对报告中存在的基本的、具有共性的问题，指导报告者完善报告（例如，提示报告者添加相关截屏）。然而，缺陷往往有各自的特殊性，报告者需要更有针对性的建议来完善报告信息。

针对上述问题，本文通过挖掘和分析 Gnome、Mozilla 等项目的近 941k 条缺陷报告[2]，总结现有的缺陷处理流程，分析低质量报告的特点，并设计和实现了一个基于相似检测的交互式报告工具 Intereport。Intereport 以挖掘项目的缺陷历史为基础，采用交互的方式，根据提交报告的描述，提示报告者提交报告所需要补充的信息，并检测重复报告，从而减少信息缺失的报告及重复报告的提交。本文的贡献在于：1、提出了一个基于相似检测的交互式报告完善方法。对新提交报告，该方

[1] 为了简洁，本文中的“报告”均指“缺陷报告”。后同。

[2] 包括 Gnome 社区的 2000 年至 2010 年间提交的 419k 条报告，以及 Mozilla 社区的 2000 年至 2010 年间提交的 513k 条报告。

法根据自然语言相似度及堆栈相似度检测相似报告及重复报告，根据相似度分布判断信息是否缺失，并利用最大信息增益原理分析相似报告，进而生成建议，指导报告者完善报告。2、基于上述设计，本文使用 MongoDB、Flask 及 Bootstrap 实现了基于 B/S 架构的交互式缺陷报告工具 Intereport。作为实验评估，本文从 Gnome 项目中随机抽取了 5 条报告，使用 Intereport 检测它们在项目中的重复报告。结果显示，Intereport 检测出的重复报告列表（最大长度为 7）中，有 50%～60%的列表包含了真正的重复报告。此外，本文还提供了 3 个案例，展示了报告者如何通过 Intereport 的指导，逐步完善报告的过程。

关键词：软件仓库数据挖掘，缺陷追踪系统，报告工具，相似检测，交互式

Design and Implementation of a Similarity-Detection Based Interactive Issue Reporting Tool

Jialiang Xie (Computer Software and Theory)
Directed by Associate Professor Minghui Zhou

Abstract

In OSS (Open Source Software) projects, Issue-tracking system (ITS) is widely used to keep track of user feedbacks, in particular, new features and software defects, and to acquire high quality software. Due to the existence of unskilled users, a considerable number of reported issues are in low quality and can not be fixed. Incompletion and duplication are the two main reasons of low-quality issues. Completing the issues requires exhaustive communication between reporters and community contributors, and thus, the user experience as well as the efficiency of issue fixing are impaired. Duplicate issues may result in contributors' duplicate effort and, it is also a time consuming task to search for the duplicate issues. Meanwhile, there are relatively few contributors in the community. If they are overwhelmed by the low-quality issues, they can hardly focus on fixing critical problems.

There has been a substential amount of work trying to improve issue quality, which primarily focuses on duplicate detection and quality feedback. Duplicate detection uses various ways (e.g. Natural Language Processing technologies) to compare the similarities of issue reports and identify the duplicate issues. Most of the work assumed that the issues being detected have complete information. An inexperienced reporter, however, may probably submit incomplete issues, which prevent her/him from benefiting from duplicate detection. The work on quality feedback models and predicts the quality of new issues, and provides recommendations to assist reporters in completing issues. The work gives some abstract tips, e.g. "Could you add a screenshot?" Facing a specific problem, however, reporters may require more targeted recommendations.

To address these problems, we mine 341k+ issues in Gnome and Mozilla. Based on the understanding on the existing issue-processing pipeline and the features of low-quality issues, we implement an interactive issue reporting tool based on similarity-detection, i.e., Intereport. Intereport mines the history of ITS data and interactively helps reporters complete their reports by presenting what additional information the issue needs.

Thus it improves the issue quality by reducing the submitting of incomplete and duplicate issues. The main contributions of this work include: 1. Designing a similarity-detection based interactive issue reporting tool. Different from existing issue reporting process, the tool helps reporters complete the issues they are reporting and identify duplicate issues in iterations. In each iteration, it uses NLP method and execution information to determine similar and duplicate issues, and measures the completeness of the reported issue with the number of its similar issues. If the issue is incomplete, the tool uses the theory of maximum information gain to analyze similar issues and generate targeted recommendations. 2. Implementing a novel issue reporting tool, Intereport, with B/S architecture. As an evaluation, we randomly picked 60 issues from Gnome ITS and use Intereport to find their duplicates in the repository. The result shows that 50%~60% of the duplicate-issue lists, which are generated by Intereport, contains the real duplicate issues. We further present 3 cases demonstrating how Intereport helps reporters complete their issues.

Keywords: Mining Software Repository, Issue-Tracking System, Issue Reporting Tool, Similarity Detection, Interactive

目 录

表目录

图目录

代码目录

1 引言

高质量软件的开发需要有及时、高效的缺陷收集、处理机制。大型开源软件项目广泛使用缺陷追踪系统（Issue-tracking System）进行用户反馈（特别是软件缺陷）的记录和追踪。例如开源缺陷追踪系统 Bugzilla，被 Mozilla、Gnome 等多个社区使用。项目成员通过缺陷追踪系统进行合作，完成软件缺陷的报告、讨论、修复、归档。使用该系统，缺陷报告者（Reporter）将软件缺陷描述及相关信息提供于缺陷报告（Issue Report）中。项目成员对提交的报告进行审核、重演，并在必要的情况下与报告者交流以进一步获取缺陷信息。然后由项目开发者对审核后的报告进行修复。

因为有许多缺乏技术经验的用户报告缺陷，缺陷追踪系统中存在相当数量的低质量报告。低质量的报告主要包括信息缺失的报告（Incomplete Report）和重复报告（Duplicate Report）[3]。信息缺失的报告是指由于缺乏必要的信息而无法重演、修复的报告。为了获取缺失信息，项目成员需要耗费大量时间精力与报告者交流，这不但影响缺陷修复的效率，还损害报告者的体验。[4]更严重的是，一部分报告者在提交报告后便永久离开缺陷追踪系统，导致该报告由于信息不足而被迫关闭（Resolved as Incomplete）。Just 等人发现，缺陷追踪系统迫切需要“提示缺乏经验的用户应该在报告中提供什么信息”[1]的工具。重复报告是指多个报告反映了相同的缺陷。Wang 等人在[4]中指出，“未能及时发现的重复报告有可能导致开发人员的重复劳动”，而“搜寻重复的报告也是一项耗时耗力的工作”。项目成员的时间极为宝贵，如果他们被过多的低质量报告干扰，则无法及时高效地修复软件缺陷。由此可见，低质量的报告不仅影响了缺陷修复效率，也危害了软件的质量。因此，如何给报告者指导以避免报告信息缺失，如何高效地识别重复报告以降低其不良影响，是提高缺陷修复效率、提升软件质量一个关键问题。

[3] Mozilla 社区 2000 年至 2010 年间提交的 513k 条缺陷报告中，有 65%的低质量报告，因信息缺失而被迫关闭的报告占报告总数的 5%, 重复报告占 31%。Gnome 社区 2000 年至 2010 年间提交的 419k 条报告中，有 69%的低质量报告，因信息缺失报告而被迫关闭的报告占报告总数的 15%, 重复报告占 38%。

[4] 在 Gnome 社区中，解决 90%的报告需要 6.29 个月，而解决 90%的信息缺失的报告需要 7.17 个月。

目前许多研究工作致力于提高缺陷报告质量，主要分为重复检测和质量反馈两个方面。前者主要采用自然语言分析等方法计算缺陷报告间的相似度，依据该相似度检测出重复报告。这部分工作大都假定被检测的报告具有完整的信息，然而这个假设对于大量的信息缺失报告（往往由缺乏经验的报告者提交）并不成立。后者主要探索影响报告质量的因素，通过对缺陷报告的质量进行建模，预测其质量并指导报告者完善缺陷报告。这部分工作针对一些基本共性的问题给予报告者一定的指导（例如提示报告者添加出错时的截屏），然而，不同的缺陷有各自的特殊性，报告者需要更有针对性的建议来完善报告信息。

为了给予报告者及时、有针对性的指导，避免信息缺失报告及重复报告的提交，本文分析了现有的缺陷处理流程和低质量报告的特点，设计实现了基于相似检测的交互式报告工具。工具采用交互的方式指导报告者逐步完善报告信息并发现可能存在的重复报告。在每一轮交互中，工具以挖掘项目的历史缺陷报告为基础，通过检测、分析新提交报告的相似报告，为报告者生成有针对性的建议，帮助其完善报告信息并发现重复报告，从而减少信息缺失的报告及重复报告的提交，提高缺陷报告的质量。

本文解决了以下难点：1、缺陷报告信息的提取和建模。本文工具使用 Gnome 及 Mozilla 社区近 10 年的来自近 287 个产品[5]的历史缺陷报告，不同的社区、不同的产品在不同的时期所使用的报告格式不尽相同。工具针对每种情况设计相应的策略，从上述报告中提取产品、详细描述、堆栈等信息。2、信息缺失报告的相似检测。现有的重复检测算法假定报告信息完整，而本文分析信息缺失报告的特征，综合报告文本及堆栈信息，通过计算、分析现有报告与被检测报告的不对称相似度[6]，检测相似报告和重复报告。3、有针对性的完善建议的生成。工具生成的完善信息既应针对性，又应易于报告者理解。本文根据最大信息增益原理，从相似报告中总结出关键词，并通过对包含关键词的例句进行聚类，为关键词寻找典型例句。工

[5] 包括 Gnome 社区 2000 年至 2010 年的关于 220 个产品的 419k 条缺陷报告，以及 Mozilla 社区 2000 年至 2010 年的关于 67 个产品的 513k 条缺陷报告。

[6] 报告 A 与报告 B 的不对称相似度度量了报告 A 与报告 B 所共有的信息占报告 A 的信息的百分比。

具以“关键词+例句”的形式给予报告者完善报告的建议。

本文的主要贡献在于：1、提出了一个基于相似检测的交互式报告完善方法。与现有报告方式不同，该工具以交互的方式协助报告者逐步完善报告信息并发现重复报告。在每一轮交互中，工具根据自然语言相似度及堆栈相似度检测项目中的相似报告及重复报告，并根据相似度的分布，判断新提交报告的信息是否缺失。如果报告的信息缺失，工具利用最大信息增益原理分析相似报告并生成有针对性的建议，帮助报告者完善报告信息。2、使用 MongoDB、Flask 及 Bootstrap 实现了基于 B/S 架构的交互式缺陷报告工具 Intereport。在实验评估中，本文从 Gnome 社区中随机抽取了 60 条报告，并使用 Intereport 对其进行相似检测。结果显示，随着报告信息的不断完善，相似检测逐步精确。Intereport 检测出的重复报告列表（最大长度为 7）中，有 50%～60%的列表包含了真正的重复报告。此外，本文还模拟了报告者使用 Intereport 提交报告的过程。

本文内容按照如下方式组织：第二章介绍相关工作；第三章进行问题分析并介绍工具的设计框架；第四章、第五章分别介绍相似检测算法及建议生成算法的设计；第六章对工具的实现进行描述；第七章评估工具的效果；第八章对本文工作进行总结，并对今后工作进行展望。

2 工作背景

本章首先对缺陷追踪系统进行介绍。然后对现有的重复检测和质量反馈工作进行简要介绍。

2.1 缺陷追踪系统

缺陷追踪系统（Issue-tracking System）是用于记录和追踪用户反馈（特别是软件缺陷）的软件系统。该系统提供了一个合作进行缺陷的报告、讨论、跟踪、分类汇总及归档的平台。常见的缺陷追踪系统有 Bugzilla、Jira。值得注意的是，这里的“缺陷报告”并不仅仅指针对软件缺陷的报告，还包括相当数量的“功能需求”（Feature Request）等方面的报告。

缺陷追踪系统以缺陷报告（Issue Report）为核心对缺陷进行追踪。以 Bugzilla 为例，系统对报告数据的记录分为“基本信息”和“历史操作”两部分。“基本信息”记录了报告的属性（见图 2.1 缺陷追踪系统 Bugzilla 中的一条报告的基本信息），主要包括：报告的 ID（Report ID）[7]、报告者（Reporter）、报告创建日期（Creation Timestamp）、所属产品（Product）、概述（Summary）、详细描述（Description）、报告当前的状态（State）和处理意见（Resolution）。如果该报告所反映的缺陷导致软件崩溃（Crash），详细描述中往往会出现崩溃时的堆栈（Stacktrace）。如果该报告与另一条报告反馈了相同的缺陷，该报告中会包“与哪条报告重复”。此外，系统还记录了在该报告下进行的相关评论（Comment），记录内容包括：评论时间、评论者、评论内容。“历史操作”记录了历史上在该报告上所进行的操作（见图 2.2 Bugzilla 中一条报告的修改记录页面）。对每个操作，记录内容包括：操作人（Who）、操作日期（When）、修改了什么数据（What）、修改前数据内容（Old）、修改后的数据内容（New）。

[7] 报告提交时 Bugzilla 为该报告分配的唯一的标识。

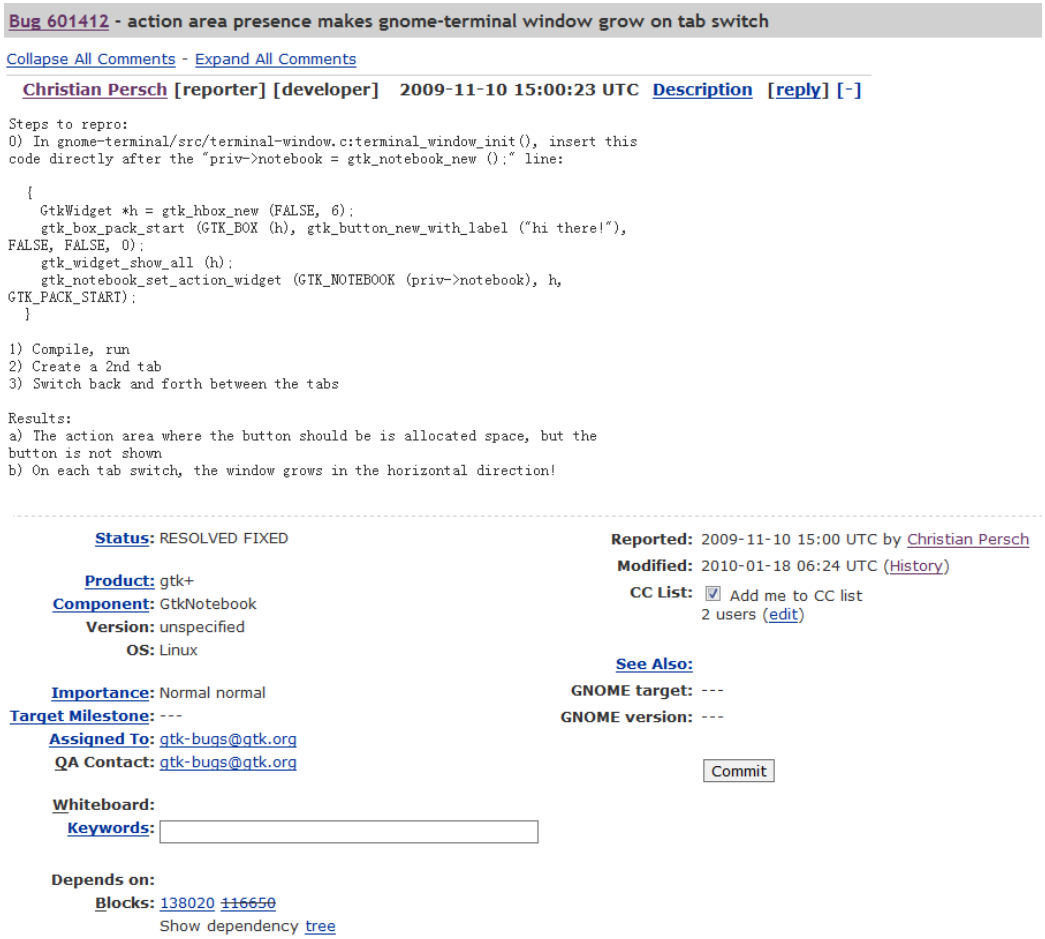

图 2.1 缺陷追踪系统 Bugzilla 中的一条报告的基本信息

Who	When	What	Removed	Added
chpe@gnome.org	2009-11-16 12:55:22 UTC	Blocks		138020
mclasen@redhat.com	2009-11-29 23:11:41 UTC	CC		jhs@gnome.org, mclasen@redhat.com
jhs@gnome.org	2009-12-02 09:20:36 UTC	Status	UNCONFIRMED	NEW
		Ever confirmed	0	1
jhs@gnome.org	2009-12-02 10:53:07 UTC	Attachment #148892 Attachment is obsolete	0	1
mclasen@redhat.com	2010-01-18 06:24:40 UTC	Status	NEW	RESOLVED
		Resolution		FIXED

图 2.2 Bugzilla 中一条报告的修改记录页面

2.2 相关工作

2.2.1 重复检测

重复检测是指通过自动化的手段在缺陷追踪系统的中寻找与被检测报告重复的报告[8]。Hiew 于 2006 年提出的基于自然语言相似度（Natural Language Similarity）检测重复报告的方法[6]。该方法主要采用自然语言处理（Natural Language Processing）

[8] 为了简便，后文称为“被检测报告”和“现有报告”。

技术将缺陷报告的概要和描述建模为 TF-IDF（Term Frequency – Inverse Document Frequency）向量，通过 TF-IDF 向量之间的相似度判定两个报告是否描述了相同缺陷，从而发现与被检测报告重复的现有报告。在此基础上，Wang 等人综合了缺陷报告的堆栈信息相似度，提升了重复检测的效果[4]。下面就上述重复报告检测技术进行介绍。

2.2.1.1 使用自然语言相似度的方法

使用自然语言相似度的方法根据报告的自然语言部分[9]的相似度来判定报告是否描述了相同的缺陷。报告的自然语言部分包括概要和详细描述。由于该方法对这两部分内容的处理方式相似，本文将它们统称为文档（Document）。该方法的流程如图 2.3 所示。

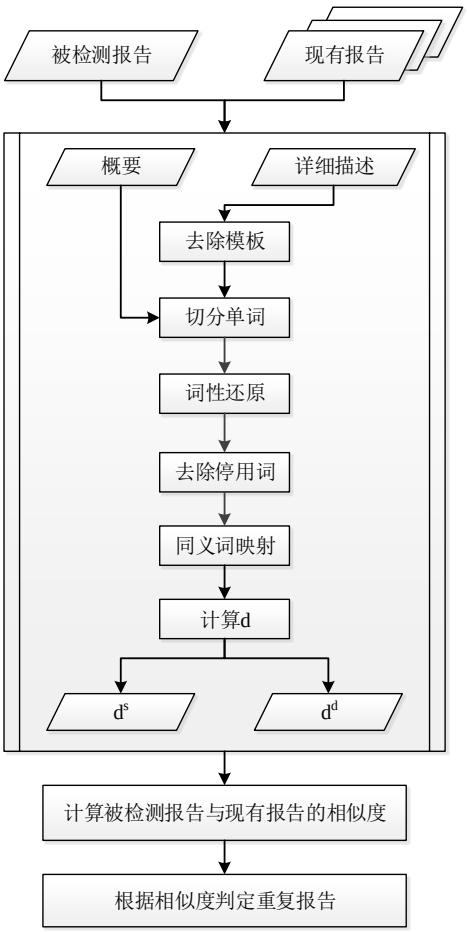

图 2.3 重复报告检测流程

[9] 包括报告的概要（Summary）和详细描述（Description）。后同。

首先，报告的详细描述中有可能出现一些预设的模板，比如“Steps to Reproduce:”，“Actual Results:”。它们会干扰重复检测，应将其过滤。接着，文档被切分（Tokenize）成单词（Term）。考虑到一些单词被普遍使用并且本身并没有实际意义（称为停用词，如 the, is, at 等），它们被从上述单词中过滤。不同词性的单词有可能具有相似的含义，因而需要对过滤后的单词进行词性还原（Stemming）。例如：visualization 和 visualize 在求词根后都成为 visual（使用 NLTK 中的 Snowball Stemmer 算法）。此外，同义词之间表达了相同的含义，有必要使用同义词表进行同义词映射（例如，“quit unexpectly”被映射为“crash”）。

文档 D_i 经过上述流程后，得到单词的集合为$\{w_{i,1}, w_{i,2}, \ldots, w_{i,n}\}$。可使用向量 $\mathbf{d}_i = (t_{i,1}, t_{i,2}, \ldots, t_{i,n})$代表 D_i 所包含的信息。其中，$t_{i,j}$ 为单词 $w_{i,j}$ 在 D_i 中的“词频-逆文档频率”（TF-IDF, Term Frequency – Inverse Document Frequency）：

$$t_{i,j} = TF_{i,j} \cdot IDF_j$$

$t_{i,j}$ 描述了 D_i 在单词 w_i 上的性质。$TF_{i,j}$ 表示 w_i 在 D_i 中出现的频数，IDF_j 称为逆文档频率。假设整个文档集合大小为 N，有 DF_j 个文档包含 w_j,则：

$$IDF_j = log\frac{N}{1 + DF_j}$$

IDF_j 表征了一个单词 w_j 对于文档特征的重要性。它使得稀有单词更能代表文档的内容。例如在有 1000 篇文档的的文档集合中，单词“bug”在 900 个文档中出现，则 IDF_{bug}=log(1000/901)=0.05; 而词“plugin”仅在 10 个文档中出现，IDF_{plugin}=log(1000/11)=2.00 > IDF_{bug}，说明“plugin”比“bug”对文档的特性更为重要。

文档 D_i，D_j 间的相似度使用文档向量 $\mathbf{d}_i$，$\mathbf{d}_j$ 间的余弦相似度度量：

$$cos_{norm} = \frac{\boldsymbol{d}_i \cdot \boldsymbol{d}_j}{\sqrt{\sum_{i=1}^{n} \boldsymbol{d}_i^2} \cdot \sqrt{\sum_{i=1}^{n} \boldsymbol{d}_j^2}}$$

$cos_{norm} \in [0,1]$，其含义是“相似的文档的单词分布相近”。上式对 $\mathbf{d}_i$,$\mathbf{d}_j$ 进行了单位化，避免了文档长度对相似度计算的影响。

报告间相似度的计算综合了概要和详细描述两部分的相似度。对报告 R_i 和 R_j，假设其概要之间的相似度为 $scos_{i,j}$，详细描述之间的相似度为 $dcos_{i,j}$，则 R_i 和 R_j 的相似度 $sim_{i,j}$ 为：

$$Sim_{i,j} = \alpha \cdot scos_{i,j} + (1-\alpha) \cdot dcos_{i,j}$$

其中，$\alpha \in [0,1]$，表示概要的相似度在衡量报告相似度时所占的权重。不同的工作在α的取值上采取了不同的策略。Hiew 认为概要和详细描述同等重要（α=0.5）[6]；Rnneson 等人认为概要的权重应为详细描述的 2 倍(α=0.66)[7]；Ko 等人认为仅使用概要部分即可（α=1.0）[8]。

使用上述报告相似度计算方法，重复报告的检测过程为：首先，筛选出缺陷追踪系统中的“可解决报告”（没有“处理意见”或“处理意见”为 FIXED, DUPLICATED 的报告）。然后，计算“可解决报告”与被检测报告间的相似度，取相似度最大的 k 条（Hiew 取 k=3 或 k=7）报告作为被检测报告的重复报告。

2.2.1.2 使用自然语言及堆栈相似度的方法

单纯根据报告自然语言部分进行相似检测效果欠佳，其原因主要有两点：1、自然语言存在多样性，仅使用自然语言信息检测出处理含义相同但表达方式不同的重复报告。2、报告自然语言部分仅描述了程序的外在行为。程序的内在行为存在于软件崩溃时的堆栈信息中[4]。因此，在使用自然语言相似度的方法的基础上，Wang 等人加入了对堆栈相似度的计算。基于图 2.3 所示的框架，该工作额外地对报告的堆栈做如下处理：

1. 对于一条报告，从堆栈中提取函数签名，仅考虑一个函数签名是否出现过，而忽略出现次数
2. 为报告 R_i 的堆栈计算函数签名向量 $\mathbf{s}_i=(s_{i,1},s_{i,2}, \ldots, s_{i,n})$。其中 $s_{i,j}$ 为 0 或 1，表示函数签名 j 是否出现在报告 i 的堆栈中
3. 使用向量间的 cos 相似度度量两条报告的函数签名向量间的相似度
4. 采用启发式的策略计算报告间的相似度。若现有报告与被检测报告的自然

语言相似度为 SIM_{nlp},堆栈相似度为 SIM_{exe},设阈值 CT_{NL-S}, $CT_{NL-E} \in [0,1]$,则使用表 2.1 中策略对报告进行分类并计算综合相似度 $SIM_{combined}$。最终的报告相似度排名（相似度由高到低）首先根据类别由低到高排序，再在每类中根据 $SIM_{combined}$ 由高到低排序。方法最后取最终排名的前 k（$0<k<7$）条报告作为相似报告。

表 2.1 计算报告相似度的启发式策略

$SIM_{nlp}>CT_{NL-S}$?	$SIM_{exe}>CT_{NL-E}$?	类别	$SIM_{combined}$
是	是	1	$(SIM_{nlp}+SIM_{exe})/2$
是	否	2	SIM_{nlp}
否	是	3	SIM_{exe}
否	否	4	$(SIM_{nlp}+SIM_{exe})/2$

2.2.2 质量反馈

质量反馈方面的现有工作主要通过对历史报告的分析、总结，发现了高质量报告所应包含的信息，并在此基础上，设计了质量预测、反馈机制。

2.2.2.1 质量预测

一部分研究工作使用实证软件的方法，挖掘缺陷追踪系统历史缺陷，探寻了优秀的报告需要包含的信息。Sandusky 等人使用 Card Sort 方法对 Mozilla 及 Eclipse 社区中的随机抽取的 600 条报告进行了分析[5]。分析调查了项目成员在报告者提交报告后与报告者进行的交互，总结了报告者在报告中应当提供的信息。这些信息可归纳为 8 类：报告缺失信息（例如：堆栈、截屏、缺陷所在的软件版本等）、缺陷确认、缺陷报告分发、调试信息、修复方法、报告状态询问、报告处理意见、报告处理流程。其中，报告缺失信息、缺陷确认及调试信息在报告初期需求最为迫切。

Just 等人通过对 156 名来自 APACHE, ECLIPSE 和 MOZILLA 的开发者调查后发现，重演缺陷的步骤（Steps to Reproduce）和堆栈信息是对了解和修复缺陷最为重要的信息[1]。基于上述调查结果，工作设计了一个交互式的报告质量预测及反馈工具 CUZILLA。该工具以 289 条被开发者标记了质量评分的缺陷报告为学习预料，分析了缺陷报告的 7 项特征（如下表所示），通过广义线性回归（Generalized linear

regression）、步进式线性回归（Stepwise linear regression）及支撑向量机（SVM）等方式建立了这 7 项特征与报告质量评分的预测模型。通过该模型，CUZILLA 预测新提交的缺陷报告的质量评分，并通过寻找该报告的最大失分特征，给予报告者补全信息的提示。例如，预测模型中有截屏这一特征与高质量评分呈正相关。如果报告者提交了一条没有截屏的报告，而 CUZILLA 根据模型预测，增加截屏信息后该报告的质量将得到最大幅度的提升，则会提示报告者“是否可以考虑提供一张截屏？”

2.2.2.2 反馈机制

Mozilla 社区的 Bugzilla 加入了相似报告检测的反馈机制[10]。在报告者创建新报告前，系统提示报告者检索相似的报告。另外，一些在线问答平台使用了交互式的提问系统，例如 IT 技术问答网站 Stack Overflow[11]。用户通过该平台创建问题时，系统会同时检索并罗列出相似的问题。

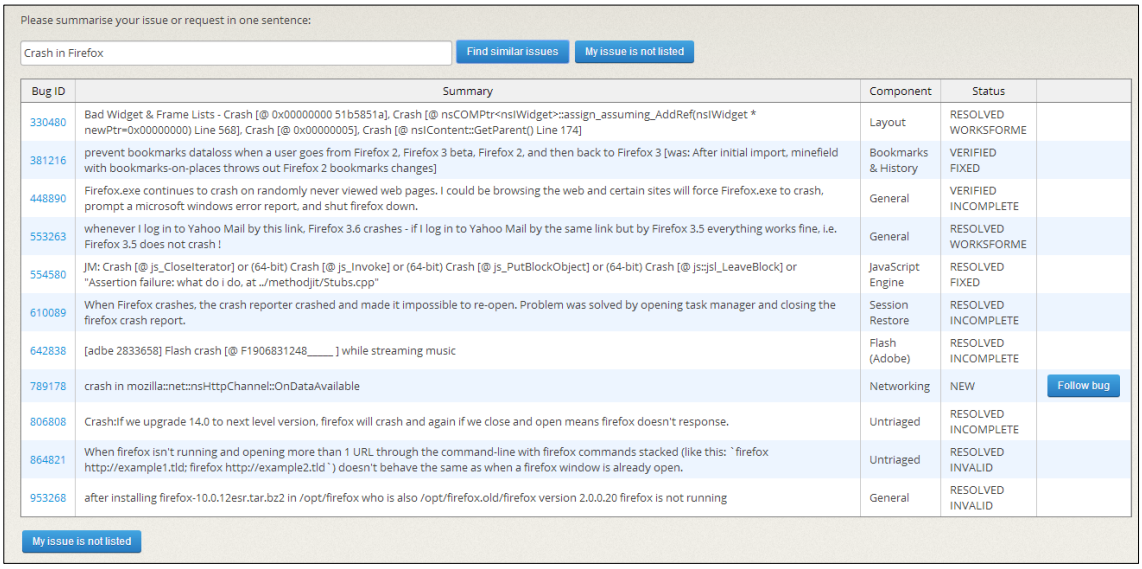

图 2.4 Bugzilla 创建新报告前的相似报告检测页面

[10] http://bugzilla.mozilla.com/
[11] http://stackoverflow.com

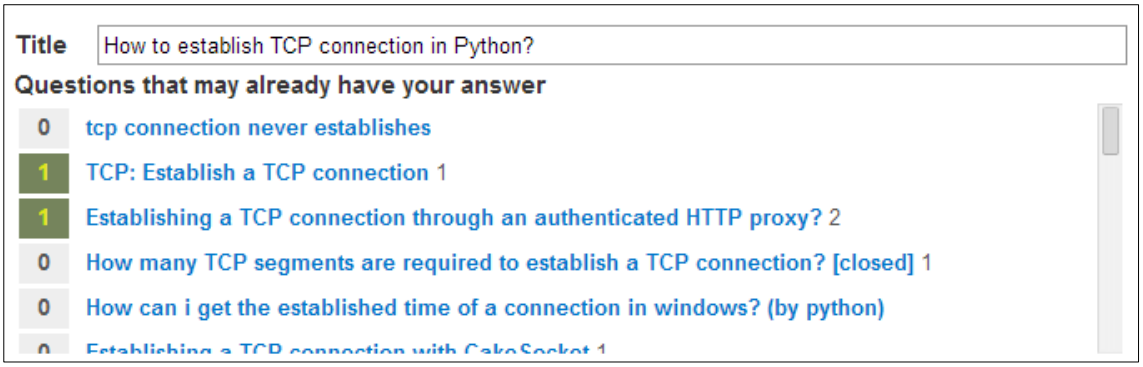

图 2.5 Stack Overflow 创建新问题时的反馈

3 问题分析

本章首先分析开源社区缺陷追踪系统的报告处理流程，对参与流程的角色以及他们在流程中的行为进行分析，并归纳现有流程中存在的问题。在此基础上，本章提出解决方案——使用相似报告判断新提交报告的完整程度，并帮助报告者补充报告信息。

3.1 缺陷追踪系统的流程机制

大型开源社区的缺陷追踪系统面临大量的缺陷报告，一般来说，社区有规范的缺陷处理流程，例如，Mozilla 和 Gnome 在其网站上进行了定义[16][17]。在此流程中涉及到担负不同责任的不同角色，协作完成缺陷的发现和解决。

3.1.1 流程参与角色[12]

报告者（Reporter）和开发者（Developer）是缺陷处理流程中最为核心的两个角色。大型开源社区的缺陷追踪系统面临大量的缺陷报告，为了使缺陷报告能够被高效地修复，社区引入了审核者（Triager）和管理者（Bug Master）两个角色。下面对这些角色分别进行介绍。

报告者

报告者是向缺陷追踪系统提交缺陷报告的角色。报告者既可能是项目的开发人员、管理人员，也可能是普通的用户。由于不同的报告者对社区的了解程度及相关专业知识技能的熟悉程度不相同，他们提交的报告的质量也参差不齐。开发人员具有专业知识，因而他们在提交报告前往往会首先确认不存在重复报告，然后在报告中提供用于缺陷重现、定位、修复的信息。相反，普通的用户很可能不会先搜索重复报告，并仅在新报告中提供了部分信息。

[12] “人与“角色”不是对等关系——一个“人”（即社区的贡献者）可以有多个“角色”

开发者

缺陷追踪系统中，开发者指修复缺陷的角色。他们拥有对项目代码修改、提交的权限。在 Gnome、Mozilla 这样的大型社区中，开发者的数量极为有限[10]。因此，如何使开发者避免干扰，将其有限的精力投入缺陷修复中，是提高缺陷修复效率的重要问题。

审核者

低质量的报告花费了开发者宝贵的时间，影响了缺陷的修复。为了提高报告质量，使开发者专注于缺陷修复，社区引入了审核者角色。审核者帮助开发者确认报告中的信息正确、完整，保证报告反映的缺陷确实存在并可重演。他们过滤低质量报告（完善、更正报告信息，关闭不可重演或者非软件缺陷的报告），并将高质量报告递交开发者。相对于开发者，审核者所需的专业知识较少，因而有数量可观的贡献者担当该角色。另外，通过审核报告，贡献者逐步提高对社区及专业知识的了解，为其成为开发者甚至核心开发者打下基础。因此，“审核者”被称为“进入社区的第一步”。

管理者

管理者数量很少，他们负责制定缺陷追踪系统的报告制度流程，对系统进行升级、备份等。

3.1.2 报告处理流程

在缺陷追踪系统中，各角色遵从一定的流程提交、处理缺陷报告。流程规定，一条报告的生命周期（Lifecycle）自报告被提交起到报告被关闭止，期间报告经历若干状态（State）。特定的状态表明该报告当前所等待的处理步骤。图 3.1 展示了一条报告的整个生命周期所经历的状态，以及相应的处理步骤。

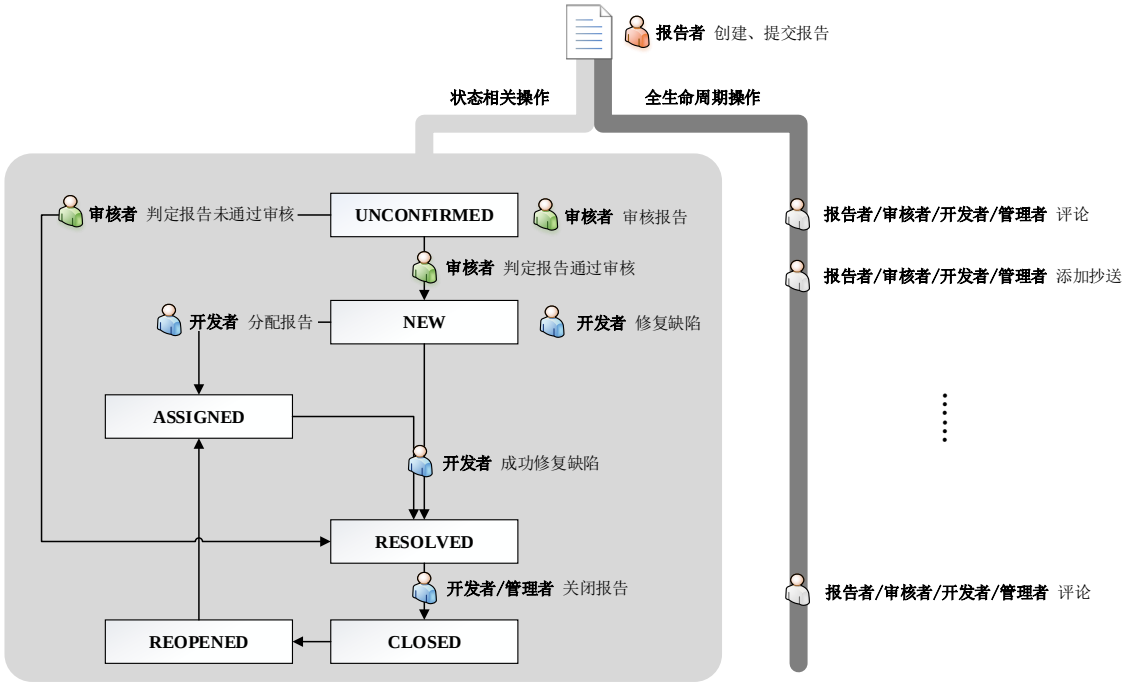

图 3.1 缺陷报告处理流程

创建、提交报告

报告者通过缺陷追踪系统创建、提交报告。报告创建后，系统为其分配一个全局唯一的编号（Report ID）作为标识。报告者在新报告中必须提供的信息包括：“概要”（Summary）、“详细描述”（Description）、“产品”（Product）、“版本”（Version）等。此外，报告者还可以提供额外的信息，包括：“堆栈”（Stacktrace）、“标签”（Tag）、“附件”（Attachment，包括代码片段、截屏等）等信息。

报告者可能由于不同的原因创建新报告。其中最典型的是“报告反馈缺陷”。报告者发现了缺陷，将其上报。经验丰富的报告者会在报告中包含“重现缺陷的步骤”、 “截图”甚至“重现缺陷的测试样例（Testcase）”。有的报告是“报告提供解决方案”。报告者发现并修复了某个缺陷，将其供开发人员审查。这样的报告附加了修复缺陷“补丁”（Patch）。有的报告是“报告反馈需求”，比如希望增加某个功能（Feature）。还有的报告是“报告进行询问”。这往往是初级的用户在使用软件中出现了疑问，通过缺陷追踪系统进行提问。

全生命周期操作

部分操作不受报告状态的约束，可在报告的整个生命周期内进行，包括：评论（Comment）、抄送（CC，Carbon Copy）。

评论提供了类似留言板的机制。参与报告流程的各个角色通过评论进行交流。例如，审核人员向报告者询问信息，开发人员对报告的修复进行讨论等等。评论可附带补丁、测试样例、截图等附件。

抄送提供报告进展的追踪机制。系统会将报告的变动发送给该报告抄送列表中的人员。

审核报告：UNCONFIRMED 状态

绝大多数报告提交后进入“待审核”（UNCONFIRMED）状态[13]。该状态下主要由审核者对缺陷报告进行审核。审核工作主要包括 3 项任务：确保报告信息完整、确保报告的有效性、产品分配。

确保报告信息完整是指审核者应尽可能保证报告提供了重演、修复缺陷所需的全部信息，以避免开发者因信息不足而无法修复报告。该任务要求审核者在必要时向报告者所求信息，甚至指导报告者获取所需信息。由于双方的交互主要通过系统的评论机制进行，获取信息的过程可能十分漫长。有时，报告者长时间没有提供反馈，审核者会将这类报告关闭[14]。

确保报告的有效性包括 2 层含义。首先，报告提供的信息应该是有效的。审核者应纠正其中错误的信息。其次，报告反映的缺陷应当确实存在。例如，“报告进行询问”并没有反馈缺陷或者功能需求，审核者可以直接回复报告者的问题，并将该报告关闭。

产品分配指审核者确保报告的“所属产品”属性填写正确，并在必要时为报告重新分配产品。大型项目中包含众多产品，产品间有较复杂的依赖关系。报告者

[13] Mozilla 中，本产品的开发者提交的报告可跳过 UNCONFIRMED 状态直接进入 NEW 状态

[14] Gnome 中规定，如果报告者超过至少 2 个月没有提供反馈，则应将报告关闭。见 https://wiki.gnome.org/Bugsquad/TriageGuide

很可能填写错误的“产品”。“产品”属性极为关键，因为如果报告分配到了错误的产品中，很可能得不到重视，导致错误分配难以被纠正，缺陷无法修复。需要指出的是，由于审核者大多不是开发人员，对产品内部结构、开发团队的人员组成并不了解，他们往往仅将报告分配到产品，而不会分配到具体的“组件”（Component）或者开发者。到“组件”和人员的分配将由开发者在报告的 NEW 状态下完成。

审核者将通过审核的报告的状态修改为 NEW，表示“该报告通过审核，等待修复”。如果报告不能通过审核，审核者将报告的状态修改为 RESOLVE（表示“已处理”），并在“处理意见”（Resolution）中记录原因。

修复报告：NEW 状态

经过审核者的过滤，被标记为 NEW 状态的报告质量较高。开发者往往通过系统的筛选供功能，提取出 NEW 状态的报告进行修复[15]。有时，开发者会首先将报告分配到具体的“组件”甚至个人，因而有可能将分配后的报告状态修改为 ASSIGNED（表示“已分配”）。但在实际操作中，开发者有可能跳过 ASSIGNED 状态直接修复缺陷。缺陷被修复后，开发者将报告状态修改为 RESOLVED，并将“处理意见”标为“已修复”（FIXED）。

已处理：RESOLVED 状态

已经修复的或未通过审核的报告都将进入 RESOLVED 状态。该状态下的报告具有“处理意见”属性。该属性表明对报告被处理的方式。表 3.1 列举了常见的“处理意见”。大多数情况下，进入该状态即标志着报告生命周期的结束。但是，如果发现需要对该报告做进一步处理，贡献者可将报告的状态修改为上述的任何状态。

[15] 开发者有权限查看并修复任何未修复的报告。

表 3.1 常见的“处理意见”（Resolution）

处理意见	含义
FIXED	报告的缺陷已被修复
DUPLICATE	该报告与其他报告重复[16]
INCOMPLETE	报告的信息不足
WONFIX	不予以修复/不予以实现功能
NOTABUG	报告反馈的不是缺陷或功能需求
OBSOLATE	已不对缺陷所在的版本进行维护

关闭报告与重新开启：CLOSED 状态与 REOPENED 状态

有时候，社区会将处于 RESOLVED 状态的报告归档，即将该报告的状态修改为 CLOSED。如果需要对归档后的报告进行修改（例如发现缺陷仍然存在），则应首先将其状态修改为 REOPENED（重新开启）。

3.1.3 现有流程存在的问题

现有流程存在的问题是，如果报告者提交了信息缺失的报告，审核者（甚至开发者）就需要与报告者经过反复交流以完善信息。交流大多通过评论的方式完成，因此需要双方耗费大量时间等待对方回复，效率不高。并且对于普通用户而言，他们在提交报告以后很可能不会及时登录系统提供反馈，造成报告因长时间得不到反馈而被迫关闭。另外，重复检测工具难以对信息缺失的报告做出较准确的检测，因而当前流程的完善报告信息的方式还可能导致无法及时发现重复报告。

综上所述，目前流程存在的问题是：缺乏及时地给报告者完善报告的提示，因此失去了在第一时间获取信息的机会。如果能在报告提交时能及时给予报告者完善建议，就能及时获取缺陷信息。这不仅有利于减少报告的缺失，还能及时定位缺陷报告，避免重复报告的提交。

3.2 解决方案

本文希望帮助报告者提高报告质量，因此报告者的动机尤为重要。如果报告者反馈的缺陷已经被报告（即项目中存在重复报告），说明该缺陷正在被处理甚至可

[16] 报告中会相应地记录该报告与那些报告重复。

能已经被解决。报告者若能及时发现该重复报告，既能避免撰写、提交重复的报告，又能及早发现解决缺陷的方案。如果项目中不存在重复报告，则报告者希望让自己的报告得到尽快修复。换而言之，报告者至少必须保证，开发者根据报告的描述，能将该报告反映的缺陷与其他缺陷区别开来。由此可见，对于报告者而言，报告信息充足的必要条件[17]是：

通过报告所包含的信息能判定该报告与现有报告重复，或与现有报告区分

换而言之，信息缺失的报告则可能存在多个疑似的重复报告。本文将这些疑似重复报告称为相似报告。例如：信息缺失的报告 R_n 所包含信息的集合为$\{I_a\}$,现有完整报告 R_1，R_2 包含的信息集合分别为$\{I_a, I_b\}, \{I_a, I_c\}$. 根据 R_n 的信息，仅能判断 R_n 可能与 R_1 或 R_2 重复（或为新的报告）。此时，I_b 或 I_c 成为完善 R_n 的关键信息。如果报告者更新了报告 $R`_n$ 的信息集合为$\{I_a, I_b\}$，则可判定，该报告或者与 R_1 重复，或者为新的报告（如果报告者还能进一步提供 I_d，使 R_n 与 R_1 区别）。由此可见，可以以相似报告数量度量报告的信息完整性。

由上述例子还可看出，相似报告（R_1, R_2）不仅可以用于判断 R_n 的信息完整性，还可以为完善报告提供线索（提示 R_n 的报告者，应完善的信息可能为 I_b、 I_c，或与之不同）。大型开源项目有众多的用户，积累了大量的缺陷报告。如果能“以史为鉴”，则可充分利用丰富的历史知识，从相似报告中挖掘出完善现有报告的线索，从而帮助报告者完善报告信息，减少报告信息缺失的几率，并定位可能存在的重复报告。

[17]由于报告在处理阶段可能需要更多的信息，因此该条件为报告信息充足的必要条件

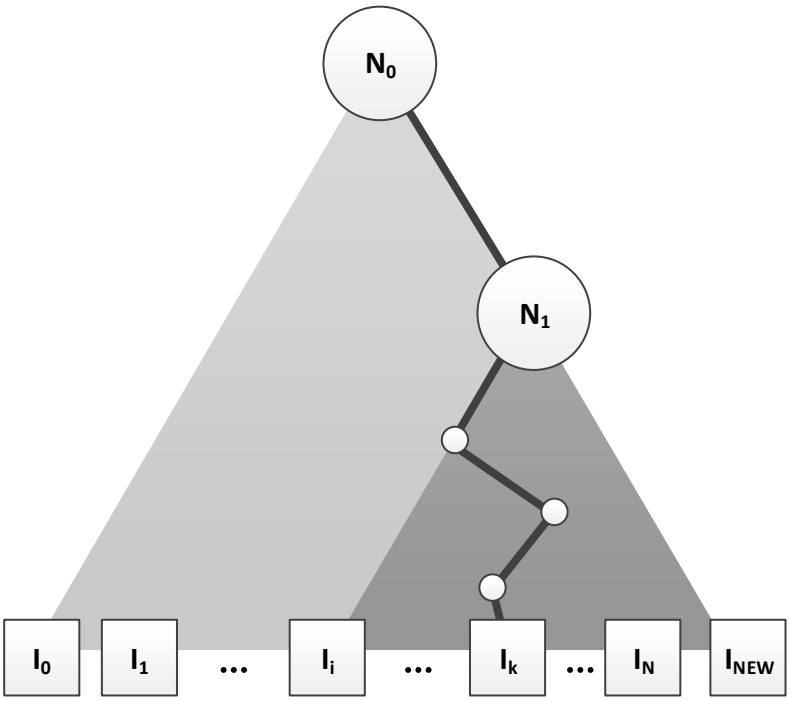

图 3.2 交互式完善报告信息的过程

利用相似报告完善信息、发现重复报告的过程，可类比报告者使用决策数对新提交报告进行分类的过程。如图 3.2 所示，假设缺陷追踪系统中现有 N 条非重复的、信息完整的报告（设这些报告组成的集合为 $I=\{I_0, I_1, \ldots, I_N\}$），则对应的决策树有(N+1)个叶节点——新提交报告可能是重复报告，也可能是新提交报告。然而，该决策树的结构是未知的。交互式报告的过程，就是报告者在工具的提示下，不断补充报告信息，寻找一条从决策树根节点到一个叶节点的路径的过程。假设初始的报告为 R_0，其中未包含任何信息（一个极端情况），则 R_0 的相似报告集合 $S_0=I$，表示 R_0 可能与任何一条报告重复，也可能是新的报告。在决策树中，R_0 对应了整棵树的根节点 N_0。当报告者根据提示，完善信息后得到新版本的报告 R_1。由于 R_1 的信息比 R_0 丰富，因而其相似报告集合 $S_1 \subseteq S_0$。在决策树中，R_1 对应节点 N_1 为 N_0 的子节点。随着交互进行，报告信息不断丰富，报告对应的节点向叶节点移动。对应节点到达叶节点表示了交互的结束——或者已经找到重复报告，或者报告信息已与现有报告区分。

由此可见，利用相似报告，缺陷追踪系统可以通过交互的方式在报告创建、提交阶段给予报告者有针对性的指导，帮助其完善报告信息、发现重复报告，从而避

免了信息缺失和重复报告的提交。该过程包含两个关键的问题：1、如何根据报告 R_i 所包含信息，确定相似报告集合 I_i；2、如何根据 I_i 生成完善报告的建议，使得报告对应节点更快向叶节点移动，即使得 $(|I_i|-|I_{i+1}|)$有最大值。

4 工具设计

根据上一章的分析，本章首先设计了交互式的缺陷报告流程，然后提出工具的框架。最后，对工具框架中的两个重要模块——相似检测模块和建议生成模块进行设计。

4.1 交互式报告流程设计

本文对工具的交互流程设计如图 4.1 交互式报告流程所示。首先，与现有缺陷报告系统相似，报告者创建缺陷报告，填写“产品”“概要”“详细描述”“堆栈”等信息，并提交报告。然后，工具检索项目中的相似报告及重复报告，并根据相似报告的分布给予报告者反馈：如果不存在相似报告，则工具认为新提交报告的信息充足，可以提交至缺陷追踪系统；如果存在相似报告，工具分析这些相似报告，并给予报告者完善报告信息的提示。此外，如果相似报告的数量较少（少于一个预先设定的整数，例如 7），工具还将这些相似报告作为可能的“重复报告”反馈给报告者，供其确认。报告者可根据工具的提示补充相应的信息：如果“重复报告”中的某个报告与新提交报告相同，报告者可确认重复报告；如果工具给予的完善提示与新提交报告有关，报告者根据提示补充完善报告信息；如果工具给予的完善提示与新提交报告无关，报告者可否定该提示的相关性（提供负面反馈）。报告者采取上述行为补充相应后，交互进入新的一轮。此时新提交的报告与上一轮相比信息更加丰富，因而相似检测也更加精确。最终，报告者在工具的协助下发现重复报告（如果存在）或完成新报告的提交（此时不存在相似报告）。

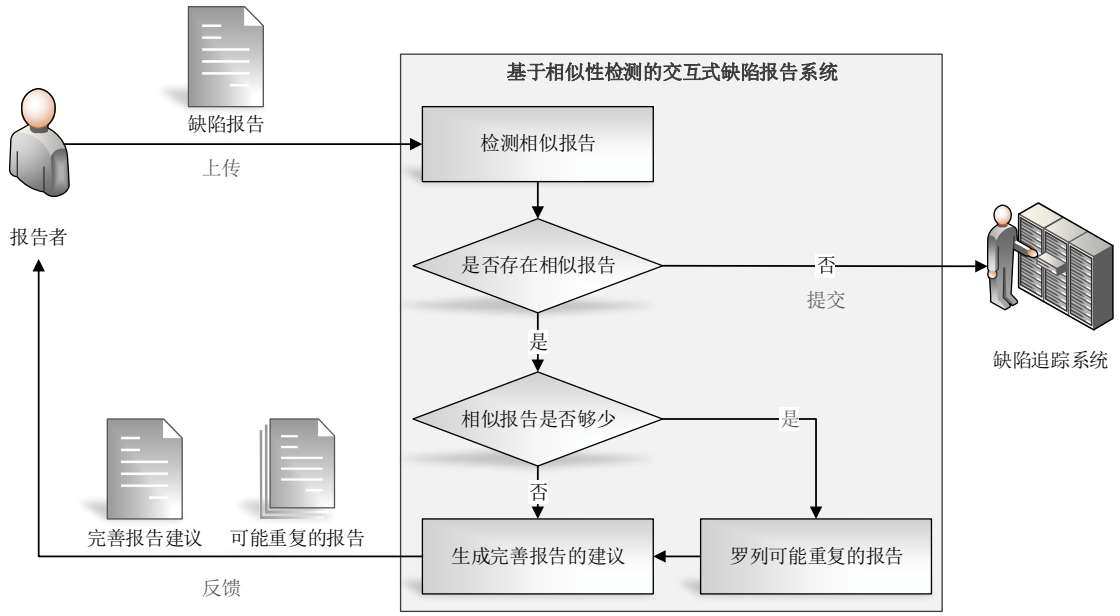

图 4.1 交互式报告流程

4.2 工具框架设计

为了实现上述流程，本文工具的框架设计如图 4.2 所示。对应工具需要解决的两个关键问题，框架的核心算法部分包含两个关键的功能模块：相似检测、建议生成。此外，工具以缺陷追踪系统历史数据为基础，需要具有数据访问、日志记录、参数配置等基础功能作为支撑。最后，会话交互部分通过调用核心算法提供的功能实现与报告者交互完成报告提交。

图 4.2 模块设计图

下面对核心算法中的两个关键模块进行说明。

4.2.1 相似检测模块

该模块负责对报告信息进行提取、建模，计算新提交报告与现有报告的相似度，从而检测出相似报告及重复报告。模块的输入来自两部分：缺陷追踪系统中的现有报告、报告者创建的新报告。对现有报告，模块进行批量处理：提取报告信息，计算用于相似检测的特征，并为其生成索引。对新提交报告，模块计算其与现有报告的相似度，并分析相似度分布，输出该报告的相似报告及重复报告。

4.2.2 建议生成模块

该模块负责分析新提交报告与相似报告，并生成完善报告的建议。模块的输入为新提交报告和相似检测模块输出的相似报告，输出为完善建议——包括一个代表需完善信息的关键词，以及一组用于说明该关键词的例句。

4.3 相似检测模块设计

本节针对信息缺失报告的特点以及相似检测的目标等因素，对 2.2.1 中的重复检测算法进行了改进，设计了相似检测算法。本节首先分析相似检测的需求，然后对相似检测算法进行设计。

4.3.1 需求分析

现有的重复报告检测工具主要基于自然语言相似性。其输入为“可解决的”（还没有“处理意见”的或“处理意见”为 FIXED、DUPLICATED 的）、信息完整的报告，输出为与输入报告重复的报告。与现有工作相比，本模块所涉及的情况有如下不同：

1. 检测目标不完全相同

现有的重复报告检测工具的目标是发现与被检测报告[18]重复的报告。而本文的相似检测目标是发现“相似报告”，即可能与被检测报告重复的报告。当“相似报告”较多时，它们被用于生成完善建议；否则，被作为重复报告反馈给报告者。

[18] 本章中的“被检测报告”即“新提交报告”。后同。

2. 输入输出不完全相同

由于检测目标不同，本模块的输入输出也与重复检测工具不完全相同。现有重复检测工具中，输入的被检测报告应该是“信息完整”（“处理意见”不为INCOMPLETE）的报告，本模块的被检测报告由报告者上传，有可能是“信息缺失”的；重复检测工具中，输入的现有报告为“可解决”的报告，本而模块需要对所有缺陷报告进行反馈，因此输入的现有报告包括所有“信息完整”的报告；重复检测工具的输出是重复报告，相似检测的输出是“相似报告”。另外，本模块的被检测报告中中还可能包含“负面反馈”，例如：“报告与‘browser’无关”。此时相似检测算法应当避免返回包含“browser”的报告。

3. 数据规模和效率要求不同

现有重复检测工具的检测范围限于单个工程中，其检测规模不大，对检测效率没有太严格的要求。而 Gnome、Mozilla 等社区的缺陷追踪系统为社区的多个工程提供统一的缺陷报告入口。因此，本模块需要对不同工程的报告进行相似检测，需要对大量的数据[19]进行处理计算。另外，本模块运行于交互式工具中，要求能在短时间内返回结果，否则将极大影响工具的可用性。

4.3.2 相似检测算法设计

综合上述四个特点，本文基于现有的重复检测算法，设计了相似检测算法。算法综合考虑了报告的概要、详细描述、堆栈等信息的相似度，采用了二值化的词频生成报告的 BTF-IDF 向量，以“现有报告符合被检测报告的程度”来度量两者的相似性，并通过现有报告的相似度分布，选取相似报告及重复报告。另外，算法利用重复报告在产品、报告时间上的分布特点，对检索效率进行优化。

本模块工作流程如图 4.3 所示。表 4.1 对流程中步骤的功能、输入输出进行了说明。该流程在图 2.3 的基础上进行了改进。另外，本文经过对 Gnome、Mozilla

[19] 本文工作所使用的 Gnome 社区数据共包含 220 个产品 419k 条报告，其中报告数量最多的产品 Evolution 包含 56k 条报告。Mozilla 社区的数据共包含 67 个产品 513k 条报告，其中报告数量最多的产品 Core 包含 217k 条报告。

等社区报告的分析、测试，对流程中的多个环节进行了改进。下面将根据流程图对各个环节的设计进行介绍。

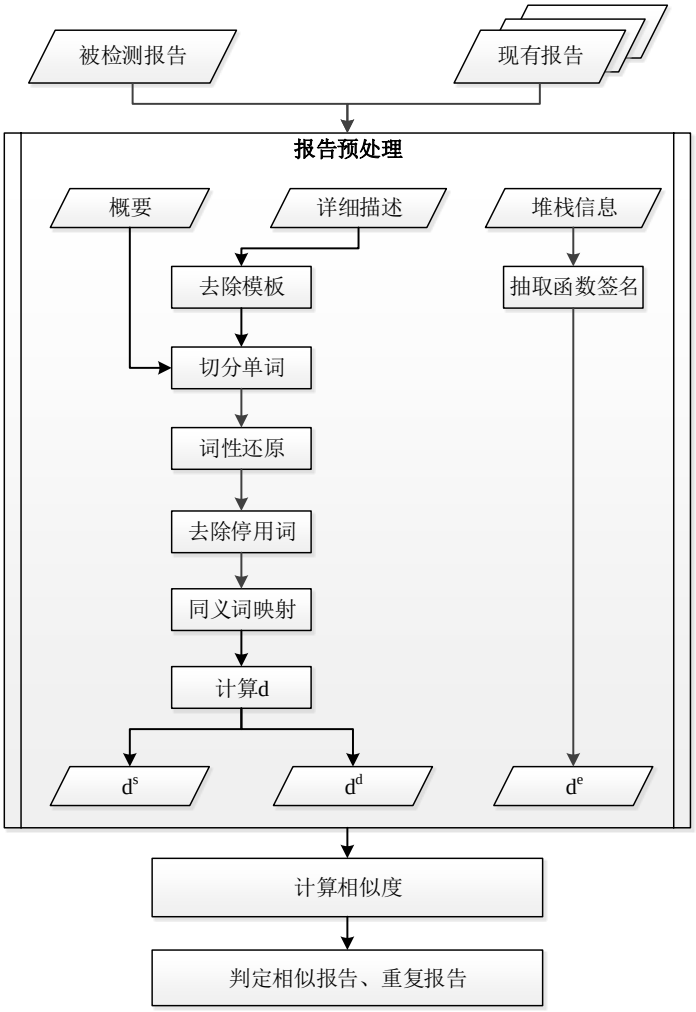

图 4.3 相似检测流程

表 4.1 相似检测模块各步骤介绍

步骤名称	功能	输入	输出
报告预处理	对报告信息进行提取、建模	报告的概要、详细描述、堆栈文本	概要及详细描述的 BTF-IDF 向量、堆栈的函数签名表
计算相似度	计算现有报告与被检测报告的相似度	被检测报告与现有报告的报告提交时间、产品、概要及详细描述的 BTF-IDF 向量、堆栈的函数签名表	现有报告与被检测报告的相似度
判定相似报告及重复报告	根据现有报告的相似度分布判定相似报告和重复报告	现有报告与被检测报告的相似度	相似报告及重复报告

4.3.2.1 报告预处理

本文使用的数据来自 Gnome 及 Mozilla 社区的缺陷追踪系统。在北京大学数据分析小组的工作下，已经将报告的信息以 key-value 方式存储。这一阶段，工具主要使用自然语言处理的方法对概要及详细描述进行信息提取，并对报告的堆栈信息进行信息提取。

去除模板

缺陷报告工具在创建新报告时常常提供一些报告模板，以提示报告者提供相关信息。模板的大量出现会干扰报告的重复检测。因为即使两个原本毫不相干的报告也会因为使用的模板相同而在文字上产生一定的相似度（如图 4.4 所示）。因此，工具使用正则表达式匹配检索并除去文本中的模板。

```
Distribution: Fedora Core release 3 (Heidelberg)
Package: gnome-utils
Severity: normal
Version: GNOME2.8.0 2.8.0
Gnome-Distributor: Red Hat, Inc
Bugzilla-Product: gnome-utils
Bugzilla-Component: gdict
Bugzilla-Version: 2.8.0
BugBuddy-GnomeVersion: 2.0 (2.8.0)
Description:
```

```
Package: gnome-panel
Severity: normal
Version: 2.0.6
Synopsis: gnome-panel crash
Bugzilla-Product: gnome-panel
Bugzilla-Component: Show Desktop Button
BugBuddy-GnomeVersion: 2.0 (2.0.3)
Description:
```

图 4.4 两个无关报告由于使用了相似的模板，而包含大量相同的词

切分单词

为了提取文本的词频信息，工具首先需要对其进行分词处理，即把整段文本分割为若干单词。不同的分词策略在不同的使用环境中可能对相似检测产生较大影响。这导致了一些通用的分词策略在当前环境下不能取得好的效果。例如，一种简洁、通用的分词策略是，使用非字母字符作为分割符。在该策略下，“mouse-down”被分割成“mouse”和“down”两个词条。而在对软件行为的描述中，“mouse-down”作为一个常用的描述方式，将其作为一个词条显然更为准确。否则“mouse-up”会与“mouse-down”产生一定的相似性，与常理不相符。本文经过对报告内容的分析，最终确定的分词策略为：

以字母开头的包含数字、点、连词符、下划线的连续字符串均为词条

下表罗列了一些文本在不同分词策略上的效果：

表 4.2 本文的分词策略与采用非字符的分词策略对比

文本	非字符串分割策略	本文策略
ubuntu13.04	ubuntu	Ubuntu13.04
mouse-down	mouse, down	mouse-down
set_background_color()	set, background, color	set_background_color
github.com	github, com	github.com

词性还原

不同词性的同根单词间可能具有相同的含义。因此，工具需要对单词进行词性还原。词根化算法没有绝对的标准，不同的算法对同一个单词可能产生不同的输出。

下表列举了一些词性还原算法以及输入输出样例。

表 4.3 不同词性还原算法的对比

输入样例	算法名称	输出样例
visualization, visualizing	Lancaster	vis, vis
	Porter	visual, visual
	Snowball (Porter2)	visual, visual
was	Lancaster	was
	Porter	wa
	Snowball (Porter2)	was

Porter 是最为经典的词性还原算法。Snowball（Porter2）在 Porter 算法的基础上进行了改进。相比 Snowball 算法，Lancaster 算法较为激进。这使得 Lancaster 算法有更大的可能将词根本不相同的词映射为相同的词，从而导致本不应该相似的两个文本被判定相似。另外，由于建议生成模块需要将关键词作为建议提示反馈给报告者，Lancaster 算法易导致单词求词根后难于辨认。综合上述原因，模块选择 Snowball 算法。

去除停用词

停用词本身没有具体含义，并且大量出现在报告中，既降低了计算、存储效率，又可能干扰相似检测。工具使用 NLTK（Natural Language Toolkit）定义的停用词表。另外，观察到报告中会出现一些长度极短（少于三个字符）的词条，它们不仅

对相似检测帮助不大，还可能损害算法效率、检测效果。工具将这部分词条过滤。

计算文档向量[20]

与普遍意义的文档相比，缺陷报告在篇幅、措辞上有一定的特殊性。另外，被检测报告有可能信息缺失。因此本模块对文档向量的计算进行了较大调整。

首先，与一般文档相比，缺陷报告具有较短的篇幅，其词条都有较强的指向性，常常一个关键词的出现即确定了其描述的方向。相比之下，词频对相似度的贡献并不大。例如在表 4.4 中，D_{new} 是被检测文档，D_i, i=1,2,…,4 为现有文档。D_1 与 D_{new} 都描述了“browser crash”。然而，由于 D_2 反复强调了“crash”，因而采用 2.2.1.1 中的 TF-IDF 向量时，D_2 评分较高。而实际上，从 D_{new} 的角度出发，D_1 与 D_2 应具有同等的相似度。可见，词频可能对相似检测产生误导。综上所述，对于报告 D_i 一个词 $w_{i,j}$，本文计算其二值词频 $BTF_{i,j}$：

$$BTF_{i,j} = \begin{cases} 1, w_{i,j} > 0 \\ 0, \quad else \end{cases}$$

对应 $w_{i,j}$ 的 BTF-IDF 值为：

$$BTFIDF_{i,j} = BTF_{i,j} \cdot IDF_j$$

其次，由于信息缺失的报告只包含部分信息，其词的分布与对应的完整报告差距较大。如表 4.4 中 D_1 虽然与 D_{new} 都存在“browser”“crash”，但由于 D_1 包含许多额外信息，致使单位化后这两个词的权重下降。作为比较，虽然 D_3 只有“browser”一词与 D_{new} 重叠，但由于“browser”在 D_3 中占据了词条总数的 50%，因此使用 TF-IDF 向量计算的相似度认为 D_3 比 D_1 更接近 D_{new} （0.50>0.47）。 基于上述考虑，本模块不对文档的 BTF-IDF 向量进行单位化。

综上所述，若文档 D_i 的单词集合为$\{w_{i,1}, w_{i,2}, \ldots, w_{i,n}\}$，则其文档向量为：

$$\boldsymbol{d_i} = (BTFIDF_{i,1}, BTFIDF_{i,2}, \ldots, BTFIDF_{i,n})$$

[20] 由于模块对报告的概要和详细描述进行相似的处理，因而这里统称为“文档”。

堆栈的处理

图 4.5 展现了 Gnome 社区中一条报告的部分堆栈。一条完整的堆栈信息首先由若干线程（Thread）组成。每一个线程包含若干帧（Frame）。一般来说，一帧对应了一个函数的调用。其包含的信息有帧序号、内存地址、函数名及函数所在文件。

```
Thread 2 (Thread -1211249776 (LWP 3785)):
#0  0x0012d402 in __kernel_vsyscall ()
No symbol table info available.
#1  0x00ef4c5b in waitpid () from /lib/libpthread.so.0
No symbol table info available.
#2  0x003cb776 in ?? () from /usr/lib/libgnomeui-2.so.0
No symbol table info available.
#3  <signal handler called>
No symbol table info available.
#4  0x0012d402 in __kernel_vsyscall ()
No symbol table info available.
#5  0x00f28fa0 in raise () from /lib/libc.so.6
No symbol table info available.
#6  0x00f2a8b1 in abort () from /lib/libc.so.6
No symbol table info available.
#7  0x04169f75 in ?? () from /lib/libdbus-1.so.3
No symbol table info available.
```

图 4.5 一条报告中包含的堆栈片段

模块对堆栈信息进行了过滤：帧序号是从 0 开始的顺序编排的整数，该信息对于相似检测意义不大；应用程序可能被动态加载到内存中，即使堆栈反映的是相同缺陷，其内存地址有可能不同，因此内存地址也不宜作为相似检测的依据；相比函数所在文件，函数名更具有区分度，也更接近缺陷本质。并且考虑到堆栈中出现连续的多个重名函数的可能性不大，故工具仅保留帧的函数名。另外，考虑到重载函数（名称相同而参数不同）间有较强的关系，工具将多个重载的函数当作相同函数处理。

4.3.2.2 计算相似度

模块使用 BTF-IDF 向量计算报告的文档相似度，使用堆栈信息计算堆栈相似度。模块使用条件加权方法综合文档和堆栈相似度，计算报告间的综合相似度。另

外，模块利用重复报告在产品、报告时间上的分布特点对相似检测范围进行限制，提高了检测效率。

文档相似度计算

模块计算相似度的目的是检测相似报告，即检测出符合被检测报告的现有报告。由此可见，相似度是衡量现有报告符合被检测报告的程度，是非对称的。定义文档向量 $\mathbf{d_i}$, $\mathbf{d_j}$ 间的相似度 $RawSim_{i,j}$ 为：

$$RawSim_{i,j} = \frac{\boldsymbol{d_i} \cdot \boldsymbol{d_j}}{\sum_{i=0}^{n} \boldsymbol{d_i^2}}$$

其中 $\mathbf{d_i}$ 被检测报告中的文档，$\mathbf{d_j}$ 为现有报告中的文档，$RawSim_{i,j} \in [0,1]$。例如，表 4.4 中 D_{new} 可能与 D_1 或 D_4 重复，有 $RawSim_{new,1}=RawSim_{new,4}=1.0$.

另外，报告者在交互过程中可能对详细描述部分提出“负反馈”，即指定某些词与报告完全无关，不会出现。因此模块对出现这部分词的详细描述的相似度进行“惩罚”。设“负反馈”词条集合为 W_{neg}，则 D_i 与 D_j 的相似度为：

$$Sim_{i,j} = \max(0, RawSim_{i,j} - \beta \cdot \sum_{w \in T_{neg} \cap w_j} BTFIDF_{j,w})$$

其中β为惩罚系数。该值越大，“负反馈”的效果越明显。

表 4.4 不同策略下的报告相似度

报告	文本	词频	单位化 TFIDF	TF-IDF 相似度	BTF-IDF	BTF-IDF 相似度
R_{new}	Browser crash	browser:1 crash:1	browser:0.71, crash:0.71	1.0	brower:1 crash:1	1.0
R_1	Browser crashed when I opened new tab and click close button.	browser:1 crash:1 when:1 open:1 new:1 tab:1 click:1 close:1 button:1	browser:0.33 crash:0.33 when:0.33 open:0.33 new:0.33 tab:0.33 click:0.33 close:0.33 button:0.33	0.47	browser:1, crash:1 when:1 open:1 new:1 tab:1 click:1 close:1 button:1	1.0
R_2	Crash, crash, crash!	crash: 3	crash: 1	0.71	crash:1	0.5
R_3	Browser slow.	browser:1 slow:1	browser: 0.71 slow:0.71	0.50	browser:1 slow:1	0.5
R_4	Browser crash when I’m updating.	browser:1 crash:1 when：1 update:1	browser:0.5 crash:0.5 when:0.5 update:0.5	0.71	browser:1 crash:1 when:1 update:1	1.0

堆栈相似度计算

与文档相似度类似，本文的堆栈相似度度量了现有报告的堆栈符合被检测报告的程度。依据 Gnome 社区提供堆栈比较方法[15]，本文计算相似度的方法如下:首先，仅比较堆栈顶部的线程，因为该线程导致了程序的崩溃。然后，对于顶部的线程，从栈顶向栈底寻找第一个出现“<signal handler called>”符号的位置。该位置以上的函数调用是引起程序碰亏的直接原因。接着，比较两个堆栈在上述位置以上的函数签名。设被检测报告堆栈的这部分函数签名集合为 ST_{new}，现有报告堆栈的这部分签名集合为 ST_R，则堆栈相似度 $STSim_{new,R}$ 描述了 ST_{new} 与 ST_R 的共有部分占 ST_{new} 的比例：

$$STSim_{new,R} = \begin{cases} \dfrac{ST_{new} \cap ST_R}{ST_{new}}, 若 ST_{new} \neq 0 \\ 1, 否则 \end{cases}$$

综合相似度计算

被检测报告与现有报告的综合相似度度量了现有报告信息符合被检测报告信息的程度。为了根据相似度进行相似报告及重复报告分析，本文的综合相似度取值在[0,1]内。Wang 等人（见 2.2.1.2）综合文本相似度及堆栈相似度计算报告的综合相似度。然而，该策略仅能得到相似程度的排序，而本文需要得到相似度值。因此，模块在上述方法的基础上进行如下设计：

设被检测报告与现有报告间的文档相似度为 $DSim_{new,R}$，堆栈相似度为 $STSim_{new,R}$。相似度较大的因素应占主导，因此有:

$$Sim_{new,R} = \begin{cases} \alpha \cdot DSim_{new,R} + (1-\alpha) \cdot STSim_{new,R}, \text{如果} DSim_{new,R} > STSim_{new,R} \\ \alpha \cdot STSim_{new,R} + (1-\alpha) \cdot DSim_{new,R}, \text{否则} \end{cases}$$

其中$\alpha \in [0.5,1]$,表示主导因素对综合相似度的权重。

为了找到所有与被检测报告相似的现有报告，理论上需要计算被检索报告 R_{new} 与每一条现有报告 R_i 的相似度 $Sim_{new,i}$。然而，大型社区的报告数量多，穷尽计算所有报告的相似度将效率低下。本文通过分析重复报告的分布，发现重复报告在报告时间及产品上的分布较为集中。如图 4.6 和图 4.7 所示，在 Gnome 社区的数据中，如果把相互重复的报告归为一组，则有近 99.9%的组的报告在两年内创建，有大于 97.5%的组的报告来自于同一个产品。Mozilla 社区的数据也呈现类似的分布。出于效率与检测效果的权衡，工具仅计算与被检测报告属于同一个产品，且在两年内创建的报告的相似度。

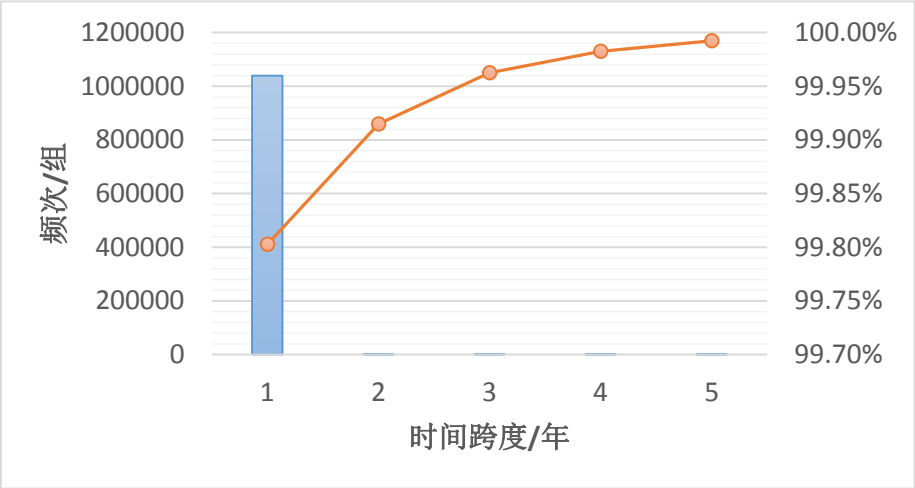

图 4.6 重复报告组的报告时间跨度分布直方图

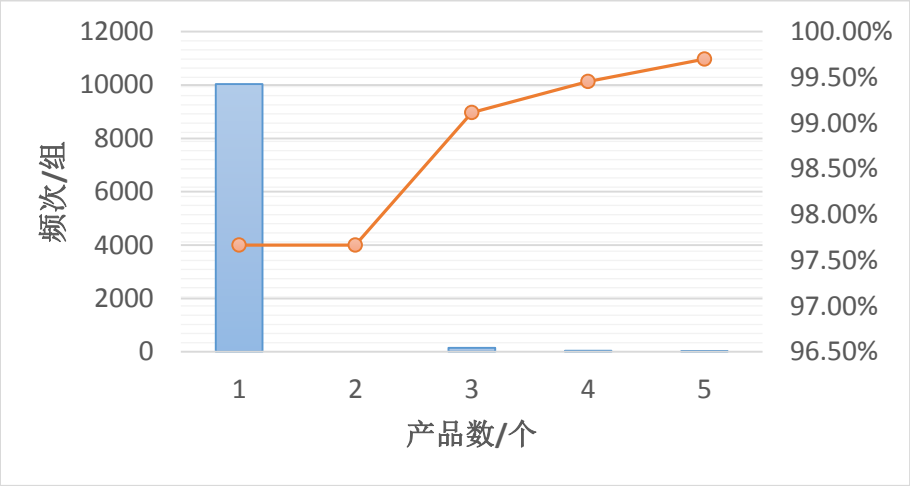

图 4.7 重复报告组的产品数的分布直方图

4.3.2.3 判定相似报告、重复报告

本模块通过分析被检测报告与所有现有报告间的相似度，得到被检测报告的相似报告集合和重复报告集合。相似报告指的是有可能与被检测报告重复的报告。如果现有报告 R 与被检测报告 R_{new} 综合相似度较低，说明 R 不太符合 R_{new} 的信息，因而其属于相似报告的可能性较小；反之，如果 R 与 R_{new} 综合相似度较高，则其成为相似报告的可能性较大。因此，相似报告集合 $SimSet_{new}$ 为：

$$SimSet_{new} = \{R \in G | Sim_{new,R} > \sigma\}$$

其中$\sigma \in [0,1)$为判定相似报告的阈值。该值越大，相似报告数量较少，工具能更快地排除非相似的报告。但此时相似检测容易受噪声（如被检测报告中提供的重

复报告中不存在的额外细节，或被检测报告中的新的措辞）影响，导致相似报告集合中遗漏一部分重复报告。相反，σ越小，相似检测对噪声容错能力越强，但工具对非相似报告的排除速度较慢。报告者需要经过更多轮的交互才能完成报告。本文将在实验部分对σ的不同值产生的效果进行比较。

随着报告信息逐步完善，大部分现有报告与被检测报告相似度越来越低，只有重复报告的相似度基本不变。当报告信息较为完善时，仅有少数相似报告存在。这是可以将其作为重复报告，让报告者审阅。因此，重复报告集合 DupSet_{new} 为：

$$\mathrm{DupSet}_{new} = \begin{cases} SimSet_{new}, \text{如果} |SimSet_{new}| < \theta \\ 0, \text{否则} \end{cases}$$

其中θ为正整数。该值越大，算法越早给出重复报告供报告者审查，但此时报告者需要审查的报告数量也较多；反之该值越小，算法越晚提出重复报告，但报告者需要审查的报告数量也较少。综合考虑报告者的负担以及交互次数，本文建议该值一般不超过 7。

4.3.3 类设计

根据上述算法分析，设计相似检测模块的类如图 4.8 所示。

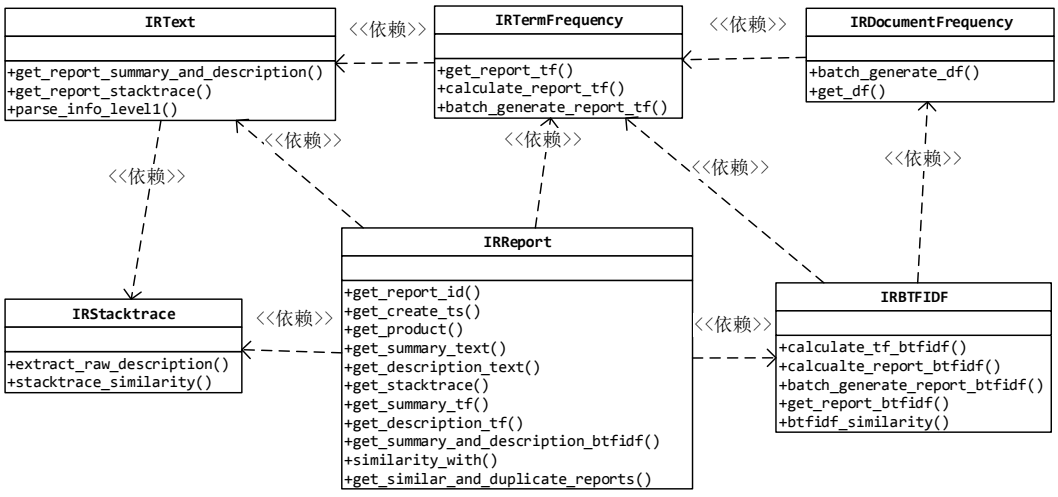

图 4.8 相似检测模块概要设计类图

IRText 类负责从缺陷追踪系统中提取所需信息。IRTermFrequency 类计算词频。

IRDocumentFrequency 计算文档频率。IRBTFIDF 类计算 BTF-IDF 向量。IRStacktrace 负责分析、比较堆栈信息。IRReport 类对应了缺陷报告。通过该类不仅可以访问一条报告的信息，还可计算该报告与其他报告的相似度、计算该报告的相似报告及重复报告。

4.4 建议生成模块设计

本模块通过分析新提交报告与相似报告，生成完善报告的建议。本章首先对建议生成的需求进行分析，然后设计建议生成算法。

4.4.1 需求分析

建议生成模块根据相似检测模块发现的相似报告，生成完善报告信息的建议。由于缺陷追踪系统对报告的概要长度有 255 字的限制，因此不需要对概要部分进行信息完善。报告的堆栈由崩溃程序生成，非有即无，不需要提供完善建议。而详细描述部分可包括大量详实的信息，因此，模块仅对如何完善详细描述提出建议。

为了有效地协助报告者，模块生成的建议应该满足如下条件：首先，建议报告者完善的信息应该未在上传的报告[21]中出现，否则该建议毫无意义。其次，建议完善的信息应该尽可能是关键信息。当报告者提供该信息后，相似报告的数量应较大幅度地可能减少。最后，生成的建议应该易于理解。

4.4.2 建议生成算法设计

模块使用“词”作为反馈建议信息的方式，有以下原因：首先，与 0 中使用 BTF-IDF 原因相似，关键的单词在缺陷报告中具有足够的意义和指向性。报告者根据关键词不难了解需要完善的信息；其次，词是英语表意的最小单位，报告者容易对词做出正面或负面反馈。报告者肯定一个词，如“crash”，是指该词与报告有关，否定一个词指该词与报告无关（该报告与“crash”无关）。而否定一个句子，如“Brower crashed.”，工具很难知道究竟是在否定“brower”（崩溃了，但不是浏览器崩溃），还是否定“crash”（没有崩溃发生）。最后，相似检测模块使用词对

[21] 指报告者上传至报告工具的报告，即相似检测中的被检测报告。

报告信息建模，因而建议生成模块可以很方便地使用词作为反馈建议的方式。为了对关键词进行说明，模块还从相似报告中提取关键词的“例句”供报告者参考。

为了保证建议信息未在上传的报告中出现，模块首先寻找在相似报告中出现，但未在上传报告中出现的词，作为候选关键词。接着，为使建议信息尽可能代表关键信息，模块使用最大信息增益原理，从候选关键词中选取关键词。最后，为了使报告者易于理解关键词所描述的信息，模块从相似报告中提取包含关键词的例句，并使用 k-means 聚类算法挑选出有代表性的例句，帮助报告者理解关键词。模块的流程如图 4.9 所示。

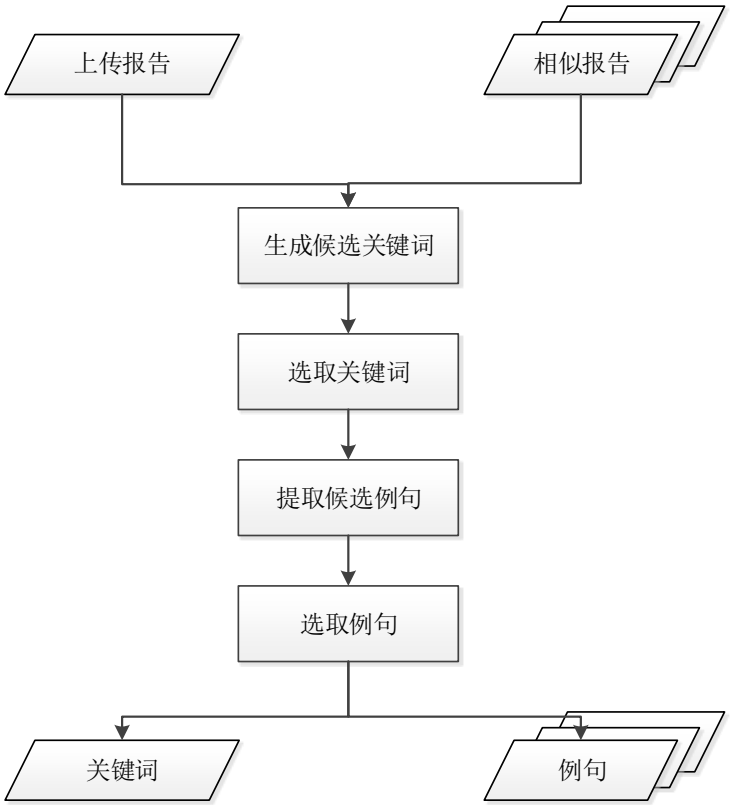

图 4.9 生成完善建议的流程

4.4.2.1 生成候选关键词

为了保证建议信息未在上传报告中出现，模块首先求出候选关键词集合 W_c。该集合中的词在相似报告中出现，未在上传报告中出现。如果被检测报告为 R_{new}，相似报告集合为$\mathbb{R}$. 对于报告 R，其概要和详细描述的词的集合分别为SW_R及DW_R，则:

$$W_c = \bigcup_{R \in \mathbb{R}} DW_R - SW_{R_{new}} - DW_{R_{new}}$$

4.4.2.2 信息增益

信息增益（Information Gain）指的是某特征的信息能给分类系统带来的信息量。对一个特征来说，其信息增益就是系统存在该特征的信息该信息与不存在该特征的信息之间的信息量差值。若 H(X)表示系统没有特征 T 的信息时的熵，H(X|T)表示系统有特征 T 的信息时的熵，则特征 T 的信息增益为：

$$\mathrm{IG(T)} = \mathrm{H(X)} - H(X|T)$$

信息增益可以作为分类系统中的特征选择依据。例如，ID3 决策树算法使用当前具有最大信息增益的特征作为分类特征。例如在一个分类系统中，一条数据由 F_1,F_2 两个特征构成，F_1 的取值集合为{f_{11},f_{12},..,f_{1p}},F_2 的取值集合为{f_{21},f_{22},⋯f_{2q}}。数据可归入 c_1,c_2,⋯,c_n 中的一类中。若在有 N 条数据的训练集中，属于每一类的数据数量分别为 k_{c1},k_{c2},⋯,k_{cn}，令$\mathrm{N} = \sum_{i=1}^{n} k_{ci}$，则系统的熵为：

$$\mathrm{H(C)} = -\sum_{i=1}^{n} \frac{k_{ci}}{N} \log \frac{k_{ci}}{N}$$

若这 N 条数据在 F_1 上的取 f_{1i} 值的数据数量为 $k_{f1i,}$则：

$$\mathrm{H(C}|F_1) = \sum_{i=1}^{p} \frac{k_{f1i}}{N} \cdot H(C|F_1 = f_{1i})$$

则：

$$\mathrm{IG}(F_1) = H(C) - H(C|F_1)$$

同理：

$$\mathrm{IG}(F_2) = H(C) - H(C|F_2)$$

若 IG(F_1)较大，说明特征 F_1 提供了更多的信息量，则优先选择 F_1 作为分类特征。反之，则应选择 F_2.

4.4.2.3 选择关键词

对于关键词而言，选取的关键词应该对增加报告信息有较大贡献。换而言之，如果将交互式报告过程看作分类过程，单词是数据的特征，则应在候选关键词集合$W_c=\{w_1,w_2,..,w_m\}$选择对分类具有最大信息增益的关键词。如果把彼此重复的报告归为一组[22]，相似报告中包含 k 组重复报告，则被上传报告等可能属于(k+1)类的一类——可能与 k 组报告中的一组重复，或者是一个新的报告。如果重复报告组 D_i 包含$|D_i|$条报告，且$N=\sum_{i=1}^{k}|D_i|$，则上传报告 R_{new} 的信息熵为：

$$H(D)=-\sum_{i=1}^{k}\frac{|D_i|}{N}\log\frac{|D_i|}{N}$$

每一个候选词条对应了信息增益问题中的一个特征。报告者在该特征上的选择有两种：该关键词与报告有关，或，该关键词与报告无关。设函数：

$$e(D,w)=\begin{cases}1, w\text{与报告组 D 中报告有关}\\ 0, \text{否则}\end{cases}$$

令 S 表示与 w_i 有关的报告数量，有$S=\sum_{i=1}^{k}[e(D_i,w_i)\cdot|D_i|]$. 则：

$$H(D|w_i)=\frac{S}{N}H\left(D\middle|\text{报告与}w_i\text{有关}\right)+\left(1-\frac{S}{N}\right)H(D|\text{报告与}w_i\text{无关})$$

其中：

$$H\left(D\middle|\text{报告与}w_i\text{有关}\right)=-\sum_{i=1}^{k}\frac{e(D_i,w_i)\cdot|D_i|}{S}\log\frac{e(D_i,w_i)\cdot|D_i|}{S}$$

$$H\left(D\middle|\text{报告与}w_i\text{无关}\right)=-\sum_{i=1}^{k}\frac{(1-e(D_i,w_i))\cdot|D_i|}{N-S}\log\frac{(1-e(D_i,w_i))\cdot|D_i|}{S}$$

因此：

$$IG(w_i)=H(D)-H(D|w_i)$$

[22] 下文称这样的组为重复报告组

根据最大信息增益，应选取 w_b 满足：

$$w_b = Maxarg_{w_i \in W_c} IG(w_i)$$

4.4.2.4 选取候选例句

为了使报告者易于理解关键词所描述的信息，模块为关键词提供例句进行说明。为了保证例句与关键词的相关性，这些例句将从相似报告中选取。因此，模块首先需要从相似报告中抽取包含关键词的句子，作为候选例句。为了报告文本中一行的长度（例如小于 80 字符），句子中常常会插入换行符。因此，模块仅使用‘.’ ‘?’ ‘!’及空行作为句子间的间隔标识。

4.4.2.5 K-means 聚类算法

聚类算法用于对样本进行归类。算法的输入为若干样本 $t_1,t_2,\ldots,t_n$。t_i 表示第 i 个样本的特征组成的向量。算法根据样本特征，将样本分为若干类，并使得同属一类的样本具有较大的相似度，属于不同类的样本有较大的差异。

K-means 算法是聚类算法的一种。它使用迭代的方法，通过考察样本间的距离，将样本分为 k 类（k 为调用者预定的值）。同属一类的样本间的距离较小，而属于不同类的样本间距离较大。这里的“距离”可以是欧式距离、夹角余弦等计算两个特征向量相似度的函数。每一类中，类的质心（Centroid）为该类中的样本的均值。K-means 算法的伪代码如代码 4.1 所示。Cendroid 记录了每一类的质心，其初始值随机从样本中选取；cluster[i]为样本 i 所属的类的序号；isConveraged 说明算法是否收敛。若算法未收敛，则不断进行迭代。第 8 行计算与样本 i 最近的类的质心，并将样本放入该类中；由于更新了样本所属的类，9~17 行根据新的分类情况重新计算每个类的质心； JudgeConverage 通过比较迭代前后质心差异判断计算是否已经收敛。最后，聚类返回结果是每个样本所属的类的序号。

```
01 K-MEANS(sample, k)
02   centroid ← RandomSelect(sample, k)
03   cluster ← [0,…,0]
04   isConveraged ← False
05   while isConveraged = False
06     do // 计算每个样本所属的类
```

```
07       for i←0 to length[sample]
08         do cluster[i]←NeareastCentroidIndex(centroid,
   sample[i])
09        // 更新每个类的质心
10        newCentroid←[0,…,0]
11        newCentroidCount←[0,…0]
12        for i←0 to length[sample]
13          do index←cluster[i]
14              newCentroid[index]←newCentroid[index]+sample[i]
15              newCentroidCount[index]←newCentroidCount[index]+1
16        for i←0 to length[newCentroid]
17          do newCentroid[i]←newCentroid[i]/newCentroidCount[i]
18        // 判断是否收敛
19        isConveraged←JudgeConverage(centroid, newCentroid)
20        centroid←newCentroid
21   return cluster
```

代码 4.1 K-means 聚类算法伪代码

4.4.2.6 选取例句

候选例句的数量可能很大，如果将它们全部呈现给报告者，很可能增加报告者的负担。由于例句在相似报告中选取，候选例句中往往出现一些意思相同或者相近的例句。对于这些意思相近的例句，显然没有必要将它们都反馈给报告者。因此，当候选例句数量较多时（大于一个预定的整数 k。k 值可由报告者设定。），模块使用 K-means 聚类算法将候选句子按语义归为 k 类，然后从每一类中挑选离该类质心最近的句子作为例句反馈给报告者。

在本聚类问题中，候选例句对应 4.4.2.5 中的样本。首先模块需要计算每个候选例句的特征向量。模块使用$$中对文档的预处理方式提取句子的词频，并计算 TF-IDF 向量作为句子的特征向量。然后，模块使用夹角余弦值作为距离函数，对候选例句的特征向量进行 k 聚类。最后，模块选取与该类质心距离最近的句子，作为最终例句反馈给报告者。

4.4.3 类设计

根据上述算法设计，对建议生成模块进行类设计。类图如图 4.10 所示。

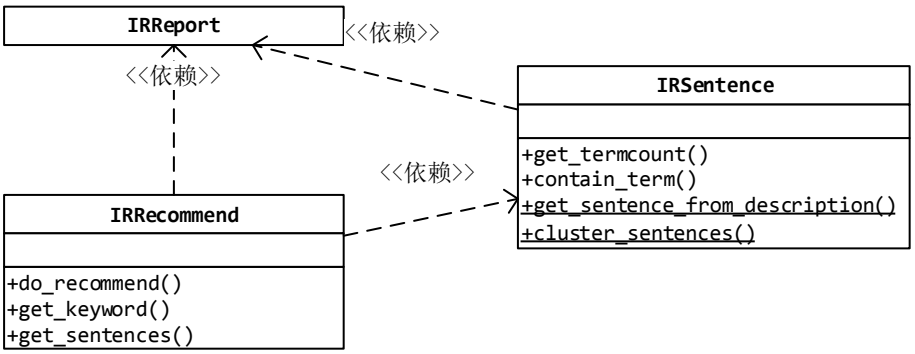

图 4.10 建议生成模块概要设计类图

当IRReport对象计算出相似报告以后，IRRecommend对象调用do_recommend()函数计算关键词和例句。该函数首先调用 get_keyword()计算关键词，然后在get_sentences()函数中使用 IRSentence 类完成候选例句抽取、聚类，最终得到例句。

5 工具实现

根据上述设计，本文实现了基于相似检测的交互式的缺陷报告工具 Intereport。为了与现有缺陷追踪系统的报告方式保持一致，Intereport 基于 B/S 架构实现。其中，服务器端使用了 MongoDB 数据库，采用 Python 语言实现。浏览器端使用 Bootstrap 框架及 JQuery 实现。

本章首先对 Intereport 的相关语言、工具简要介绍，然后给出模块设计图，最后详细介绍每个模块的设计及实现。

5.1 Intereport 涉及的相关语言及工具

5.1.1 MongoDB

MongoDB 是一种基于文件的非关系型开源数据库。Intereport 使用 MongoDB 进行报告信息及相关数据的存储，有如下原因：

1、 MongoDB 是非关系型数据库。关系型数据库使用表结构存储数据，要求数据有确定的结构。关系型数据库可有效避免数据冗余，可通过事务处理保证数据的一致性，并支持诸如连接等复杂操作。非关系型数据库不要求数据有确定的结构，存储更为灵活。但非关系型数据库存储冗余较大，不能保证严格的一致性，且对复杂操作的支持有限。对于 Intereport 而言，首先 Intereport 所要存储的数据结构差异很大。例如报告文本的词频向量。不同的报告文本包含不同的词条，词频向量的维度差别很大。如果使用关系型数据库，需要建立多个表。并且需要对这些表进行连接操作才能得到一条报告文本的词频向量。相比之下，非关系型数据库只需要存入一个“词条•词频”字典即可。其次，Intereport 对数据一致性及避免冗余的要求不高。Intereport 对数据库的修改操作大多是批处理写入操作，不涉及“读•写”一致性问题。综上所述，非关系型数据库更适合 Intereport.

2、 MongoDB 对支持对象的存储操作。这意味着可以将内存中的整个对象映射到数据库中，避免了关系型数据库存取时从对象结构到表结构的变化，从而极大地简化了数据库操作。

3、 MongoDB 对分布式存储及计算支持较好。Intereport 需要批量预处理现有报告，计算新提交报告与现有报告的相似度。这些计算过程在现有报告之间不存在计算及数据相关，因而是可高度并行化的。MongoDB 支持分布式存储，并支持使用 Map/Reduce 进行计算，对进一步提升 Intereport 计算效率有较大帮助。

4、 MongoDB 是开源软件，有来自其开源社区的强大支持

基于上述原因，Intereport 使用 MongoDB 2.4.8 进行数据存储。下面简要介绍 MongoDB 的一些主要元素。

如下图所示，MongoDB 中的一个数据库（Database）中可包含多个集合（Collection），每个集合中包含若干文档（Document）。 集合对应关系型数据库中的表（Table）。与数据库中的表不同的是，同一个表中的所有行必须具有相同的列，而同一个集合中的文档结构可以不相同。一个文档是一个“键 值”对集合。其中“值”可以是一个文档。MongoDB 中可以对集合根据某些“键”生成索引。索引会消耗一定的存储空间，但可带来一定效率的提升。

5.1.2 Python

Python 是一种开源的、面向对象的、解释型的高级计算机程序设计语言。Python 灵活性较好。它支持动态类型，其的代码具有良好的跨平台能力。Python 易用性较强。它提供自动内存管理，运行前无需经过漫长繁琐的连接、编译过程，让程序编写者无需关心繁琐的底层细节，将主要精力投入程序的核心逻辑。此外，Python 拥有功能强大的标准库，以及来自开源社区大量的工具库，还可以方便地连接采用其他语言实现的模块。上述特性使得 Python 尤其适合用于快速搭建原型程序及验证设计思路。

Intereport 的需要使用数据库连接、自然语言处理工具库、聚类算法工具库等众多工具库支持，并且存在众多算法细节需要调优。Intereport 的服务器端使用 Python 2.7 实现。

5.1.3 NLTK

NLTK 是开源自然语言工具包（Natural Language Toolkit）的缩写，是 Python

中自然语言处理的工具包。NLTK 不仅包含超过 50 套语料库及词汇资源，还提供了一些列自然语言处理所需的算法，例如词条切分、词性还原等。Intereport 使用了 NLTK 3.0 进行词条切分、去停用词以及词性还原。

5.1.4 C Clustering Library and PyCluster

C Clustering Library 是一条包含了常用的聚类算法的 C 语言实现的开源工具包。PyCluster 利用 Python 可与 C 语言程序库集成的特性，为 C Clustering Library 实现了一套 Python 访问接口。Python 程序可通过 PyCluster 完成聚类操作。PyCluster 提供的聚类操作有：

- 层级式聚类
- 自组织图
- K-means 聚类
- 主成分分析

Intereport 使用了 PyCluster 1.52 的 K-means 聚类方法。

5.1.5 pyMongo

pyMongo 是 MongoDB 的 Python 接口开发包，提供了在 Python 中操作 MongoDB 的一系列方法。

5.2 本文工作使用的数据

本文工作使用的数据是北京大学软件工程实验室数据分析小组与 Avaya 实验室合作收集的数据。Mockus[9]在 2009 年对该数据的数据类型及收集方式进行了详细的介绍。其中的缺陷报告数据来自著名开源社区 Gnome、Mozilla 的缺陷追踪系统。这两个社区均使用 Bugzilla。Gnome 社区的数据包括 2000 年至 2010 年间提交的 419k 条报告；Mozilla 社区的数据包括 2000 年至 2010 年间提交的 513k 条报告。

5.3 总体模块图

Intereport 分为逻辑服务器部分和页面交互部分。逻辑服务器部分分为基础模块、核心算法模块和会话模块。基础设施提供了日志、配置、数据库访问等功能；核心算法模块实现了 4 和 4.4 中描述的核心算法；会话模块通过 TCP/IP 协议响应

外部请求。页面交互部分负责与报告者交互：生成浏览器页面，将报告者的操作转换为 TCP/IPC 消息转发给逻辑服务器，并根据逻辑服务器返回结果更新页面。下面首先介绍逻辑服务器端的实现，再介绍页面交互的实现。

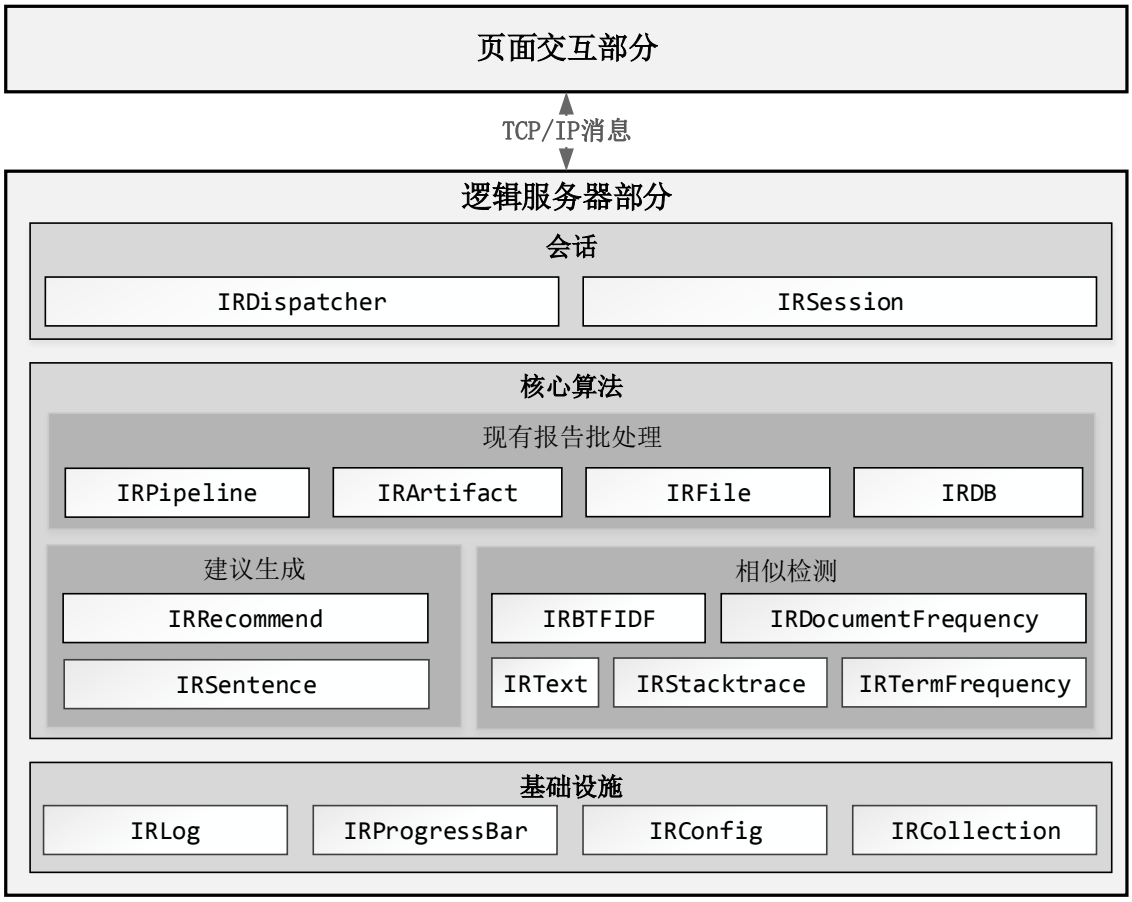

图 5.1 Intereport 模块与类的分布

5.4 逻辑服务器端

根据 4.2 的设计，逻辑服务器端可分为基础设施模块、核心算法模块、会话模块。

5.4.1 基础设施模块

基础设施提供了日志、配置、数据库访问等功能。其中 IRLog 和 IRProgressBar 负责日志记录；IRConfig 负责配置读取；IRMongodbHelper 及 IRCollection 负责 MongoDB 的读写操作。

5.4.1.1 日志记录：IRLog, IRProgressBar

需求分析

工具的调试、测试、监控离不开运行时信息的输出。IRLog 类负责统一管理系统的信息输出，IRProgressBar 使用 IRLog 实现了进度显示功能。工具需要一个统一的信息输出对象。该对象可在被创建时制定信息的输出方式，包括是否输出到标准输出，是否将输出写入文件。对象一旦被创建，则可全局访问。对于输出的消息，需要能够为其指定“消息级别”——分析信息量极大的日志时，需要通过“消息级别”对消息进行筛选。此外，预处理缺陷追踪系统现有报告以及检测相似报告的过程常常涉及对 Collection、字典的遍历。为了能有效发现、定位工具可能存在的问题，遍历过程中需要输出遍历进度信息。综上所属，对日志记录有如下需求：

需求1 日志工具一旦创建，则可全局访问。它提供统一的输出接口，并可定制日志输出方式。

需求2 日志工具可为输出的消息指定“消息级别”。

需求3 日志工具为遍历过程提供“进度条”，支持 Collection、字典的遍历。

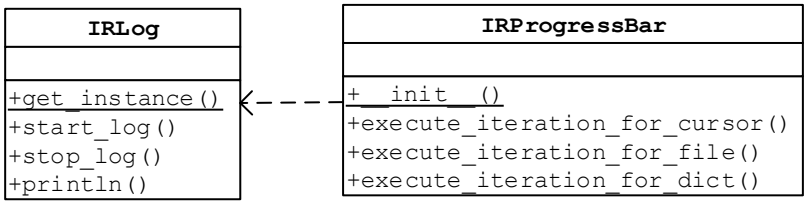

图 5.2 日志工具类图

功能设计

对于需求 1 ，上述对输出信息的需求符合“单件”（Singleton）设计模式，因此 IRLog 被实现为“单件”。IRLog 对象在第一次 IRLog.get_instance 函数被调用时创建，并通过 start_log 函数指定消息是否打印到标准输出，以及是否输出到文件。此后，可通过 IRLog.get_instance 全局访问该对象。println 方法将消息输出到预先指定的位置。其中 msg_level 指定了信息的“级别”。最后，stop_log()关闭输出通道。

对于需求 3 ，IRProgressBar 分别对 Collection、字典和文件实现了显示进度的遍历器。以 Collection 为例，遍历器首先获得 Collection 中 Document 的数量，然后遍历每个 Document：首先使用 IRLog 输出进度，然后以当前 Document 作为参数调用函数 func。代码 5.1 和代码 5.2 分别展示了使用遍历器前后的代码。

```
01 size = collection.count()
02 for item in collection:
03     progress = 'Progress ' + str(i*100/size) + '%'
04     print progress
05     log.write('progress\n')
06     # 循环中的逻辑
```

代码 5.1 未使用 IRProgressBar 进行遍历进度输出

```
01 def func(item):
02     # 循环中的逻辑
03     IRProgressBar.execute_iteration_for_cursor(collection,
04 func)
```

代码 5.2 使用 IRProgressBar 进行遍历进度输出

5.4.1.2 参数配置：IRConfig.

需求分析

由于工具需要使用于多个社区，数据时间跨度大、格式繁杂。为了适应不同的数据，工具需要使用参数进行配置。例如数据库位置、报告模板的格式、堆栈的标识、相似检测算法的相似计算的权值等等。这些参数需要使用一个统一的方式进行配置，并可在代码各处进行访问。

> **需求4** 参数配置工具能读取、设置配置，并提供全局可访问的函数，供各模块获取配置参数。

功能设计

IRConfig 使用配置文件保存所涉及的参数，因为：1. 配置文件易读、易修改；2. 工具可通过读取不同的配置文件来适应不同的情境。这为极大地方便了本文进行不同算法的效果比较实验。所示，配置文件包括若干行，每行为一个配置项，以“键:=值”方式表示。以‘#’开头的行表示注释。

```
01 #basic information
02 community:=gnome
03 #db connection
04 db_host:=localhost
05 db_port:=27017
06 #collection key
07 bug_id_name:=bug_id
08 bug_summary_name:=summ
09 bug_description_name:=desc
```

代码 5.3 Intereport 的参数配置文件

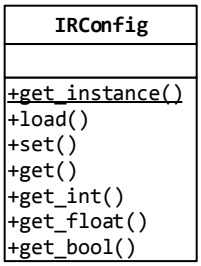

图 5.3 参数配置类 IRConfig 的类图

工具使用 IRConfig 类进行配置文件的读取和配置的访问。需求 4 符合对单件的要求，因而 IRConfig 被设计为单件。load 函数读入配置文件，将配置以字典[23]形式保存。get 函数返回相应名称的配置项，如果该名称的配置项不存在，则返回第二个参数。为了增强易用性，设计 get_int, get_float, get_bool 将 get 函数的返回值转换为指定类型。

5.4.1.3 数据库访问：IRCollection

需求分析

工具使用 MongoDB 数据库，需要根据配置中对数据库名、Collection 名的配置，找到所需数据库。另外，批处理现有报告涉及多个 Collection 的读和写，并且这些 Collection 之间存在依赖：要生成“单词文档频率 Collection”，需要读取“词频 Collection”。如果被依赖 Collection（“词频 Collection”）未能成功生成，则读取该 Collection 的代码（生成“单词文档频率 Collection”的代码）不应被执行。然而，由于 MongoDB 不提供事务操作，工具需要额外记录最后一次对 Collection 的

[23] 指 python 中的 dict 类型

修改是否成功。综上所述，有需求：

需求5 根据配置文件中的数据库地址、数据库名、Collection 名，得到相应的 Collection.

需求6 记录所有对 Collection 的批处理修改是否完成，并提供查询。

功能设计

为了满足需求 6 ，工具将对每个 Collection 的修改操作写入一个特殊的 Collection——Meta Collection 中。该 Collection 中文档的结构如表 5.1 所示。

表 5.1 Meta Collection 中文档的格式

键	说明	例
collection_name	记录的 Collection 的名字	btfidf_collection
last_modified	最后一次修改该 Collection 的时间戳（1970 年 1 月 1 日 0：00 至今的秒数）	1388534400
is_success	最后一次修改是否成功	True

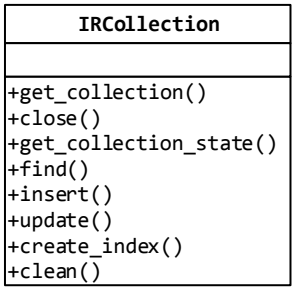

图 5.4 数据库访问类 IRCollection 的类图

为了能捕获所有在工具代码中对 Collection 进行的修改，IRCollection 类对 pymongo 中 Collection 类进行封装：通过 IRCollection.get_collection 获得相应 Collection 的 IRCollection 对象（需求 5 ）。参数 mode 指定了 Collection 的打开方式。如果以修改方式（“写入”或者“追加”）打开 Collection，则 IRCollection 会在 Meta Collection 中将该 Collection 的“is_success”设为 False. 只有对该 IRCollection 对象调用 close 函数，才会将“is_success”设为 True. 这样，如果批量写入操作未能执行完成，则可通过 get_collection_state 函数查询到该 Collection 的“写入是否

成功”为 False.

5.4.2 核心算法模块

根据总体设计框架 4.2 中的核心算法模块设计，本模块需要提供两大功能：相似检测、建议生成。在实现阶段，出于对效率的考虑，模块需要预先处理、计算缺陷追踪系统中现有报告的信息。因此，本模块涉及两条执行路线：对现有报告的批处理、对新提交报告的相似检测及建议生成。模块首先从需求出发进行功能分析，并发现类。然后，根据上述两条执行路线，设计实现各个类的具体职责功能。

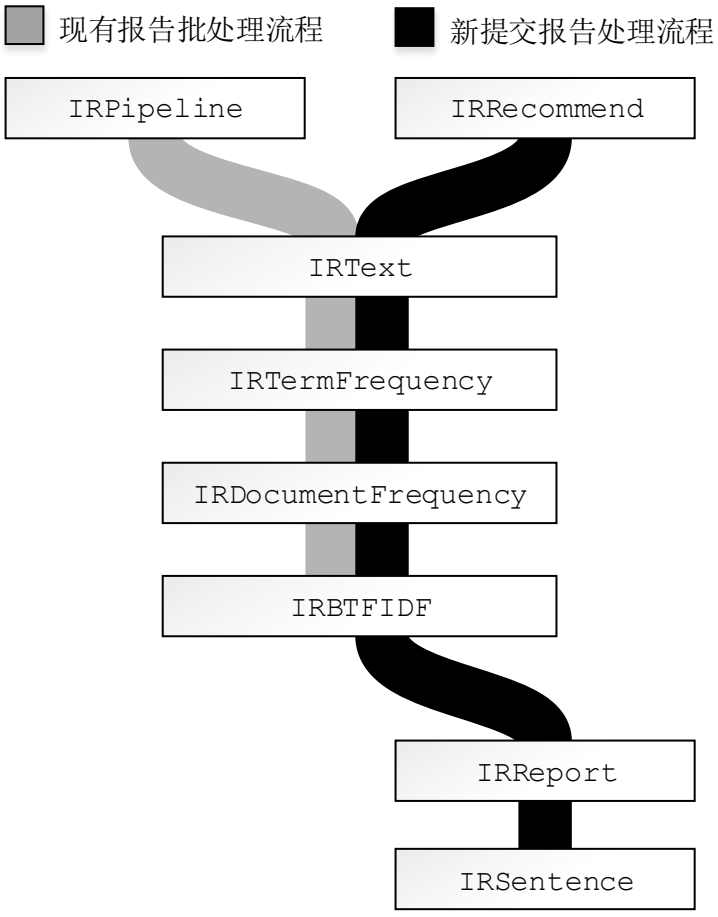

图 5.5 核心算法模块的类与流程关系图

图 5.5 描述了本模块包含的类以及它们在上述两个流程中的位置。IRText，ITTermFrequency，IRDocumentFrequency 和 IRBTFIDF 实现图 4.3 中对报告的预处理，分别负责处理文本堆栈、词频、文档频率、BTF-IDF 信息。它们既需要实现对现有报告的批处理，又要实现对单个新提交报告的处理。IRReport 是对缺陷报告的

抽象，提供统一的访问报告的各种信息的接口，并实现了报告相似度的计算。IRSentence 是对文本句子的抽象。与两个流程对应，IRPipeline 类提供批量处理现有报告的程序入口。IRRecommend 类提供了新提交报告的相似检测及建议生成的程序入口。

5.4.2.1 **文本堆栈信息：IRText**

需求分析

IRText 负责从缺陷追踪系统数据中抽取工具所需数据并存入数据库。需要提取的内容有：报告 ID、报告创建时间、所属产品、概要、详细描述、堆栈、重复信息。Gnome 的缺陷追踪系统数据为 XML 格式，而 Mozilla 的为 MySQL 格式，需要分别针对两种格式的数据设计提取器。另外，“处理意见”为 INCOMPLETE 的报告将被剔除。最后，需要提供数据的读取，包括：批量读取（批处理时使用）和读取单条报告信息（生成建议时使用）。

Intereport 在生成推荐信息环节的关键词计算时，需要知道缺陷追踪系统中现有报告之间的重复情况。因而，希望根据一个报告的 ID，得到与该报告重复的报告的 ID。综上所述，有需求：

需求7 从 XML 格式的缺陷追踪系统数据中提取报告信息，并存入数据库

需求8 从 MySQL 格式的缺陷追踪系统数据中提取报告信息，并存入数据库

需求9 从数据库中批量读取报告信息

需求10 从数据库中获取指定报告的信息

需求11 提取、存储报告的重复信息

需求12 返回与指定报告重复的报告

功能设计

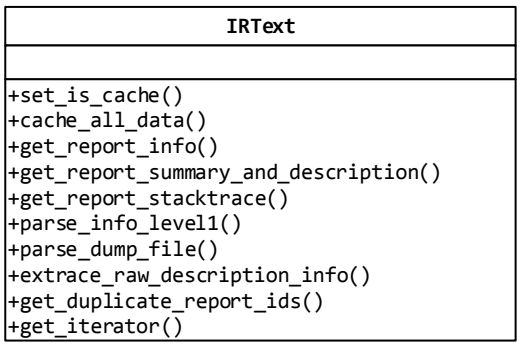

图 5.6 IRText 类图

parse_info_level1 函数和 parse_dump_file 函数分别实现了需求 7 和需求 8 。由于 Gnome 和 Mozilla 两个社区中报告的堆栈信息以文本的形式追加在详细描述后，因此需要 IRStacktrace.extrace_raw_description_info 函数通过 IRStacktrace 类（见 0）提供的功能，将详细描述分割为字符串表示的文本部分和 0 所述的数组表示的堆栈信息。

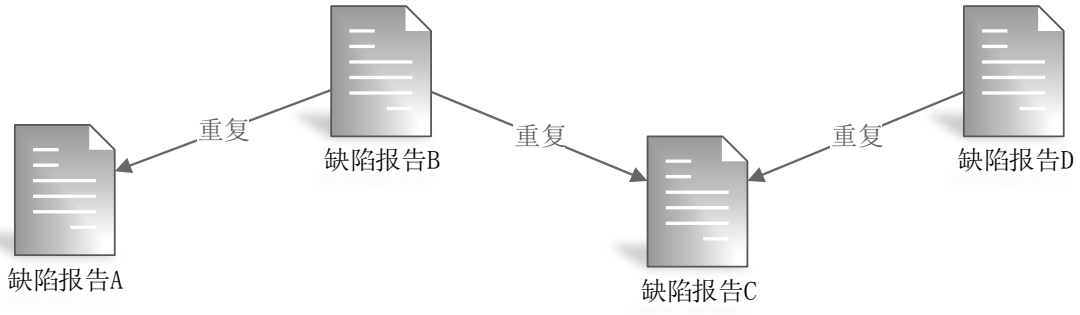

图 5.7 缺陷追踪系统对重复报告的记录方式

为了能查询指定的报告与那些报告重复，最简单的解决办法是，对于任何一条报告，记录与其重复的报告。然而这样的记录方式空间消耗太大，因此，这里引入“重复报告集合”概念。重复报告集合中包含一组相互重复的报告。每个重复报告集合用唯一的 ID 标识。这样，通过建立报告 ID 和重复报告集合 ID 的双向映射，即可求出指定报告的重复报告。建立这个双向映射面临一个问题：如图 5.7 所示，缺陷报告系统数据可能仅单向记录报告的重复情况（即图中的边）。对于本工作而言，报告间的重复关系没有方向性，因而上图中的报告相互重复，它们属于同一个

重复报告集合。为了建立重复报告集合信息（需求 11 ），模块采用归并的方式生成重复报告集合。代码 5.4 为该过程的伪代码。

```
01 GENERATE_DUPLICATE_MAP:
02     report2set ← new dict()
03     for reportID in reportIDs
04         do new_set ← set([reportID])
05             report2set[reportID] ← new_set
06     for duplicate in duplicates
07         do reportID1 ← duplicates.from
08             reportID2 ← duplicates.to
09             new_set ←
10             report2set[reportID1]+report2set[reportID2]
11             report2set[reportID1] ← new_set
12             report2set[reportID2] ← new_set
13     for reportID, set in report2set
14         do insert_into_database(reportID, set.id)
```

代码 5.4 建立重复报告组的伪代码

report2set 是保存了从报告 ID 到重复组 ID 的映射。3~5 行，函数将每个报告创建一个重复报告集合，集合创建时带有唯一的编号（ID）；6~12 行，函数根据一组重复关系，合并对应的两个重复报告集合；13~14 行，报告重复集合信息在数据库中保存为（报告 ID，重复集合 ID）对。

相应地，通过访问数据库中的重复报告组信息，get_duplicate_report_ids 函数提供了获取指定报告的重复报告的功能。伪码如代码 5.5 所示。

```
01  GET_DUPLICATE_REPORT_IDS(reportID):
02      setID ← find set_id in database where report_id=reportID
03      if setID = NIL
04          return [reportID]
05      return find bug_id in database where set_id = setId
```

代码 5.5 获取指定报告的重复报告的伪代码

第 2 行尝试在数据库中查找 reportID 的重复报告集合 ID。如果该 ID 不存在，说明该报告不存在重复报告，因此其重复报告集合仅包括它自己（3~4 行）；否则，根据重复报告集合 ID 找到所有该集合中的报告（第 5 行）。

对应需求 10，get_report_info, get_report_summary_and_description, get_report_stacktrace 函数分别负责返回数据库中指定报告（通过报告 ID）的基本信息（创建时间、所属产品）、文本信息（概要和描述的文本部分）、堆栈。get_iterator 返回一个遍历所有报告的信息的迭代器。（需求 12 ）

存储设计

表 5.2 text_collection 结构

键	说明	例
report_id	报告 ID	100100
summary	概要	“Browser crashed”
description	详细描述	“Browser crashed when I …”
stacktrace	堆栈	[[“on_mouse_down”, …], …]

表 5.3 basic_collection 结构

键	说明	例
report_id	报告 ID	100100
create_ts	报告创建时间，1970 年 1 月 1 日起的累计秒数	212156541
product	产品名称	“evolution”

表 5.4 duplicate_info 结构

键	说明	例
report_id	报告 ID	100100
set_id	重复报告组 ID	56

由于报告的概要、详细描述、堆栈需要用于批处理（批量读取），而报告的创建时间、所属产品则仅用于相似检索时的非批量查询，报告的重复报告仅在生成建议时使用，因此，IRText 的数据存取分为 3 个 Collection：text_collection 保存报告的概要、详细描述、堆栈，basic_collection 保存报告的创建时间、所属产品，duplicate_report 保存报告与其对应的重复报告集合的关系。

5.4.2.2 词频信息：IRTermFrequency

需求分析

IRTermFrequency 类负责根据缺陷报告的文本生成词频。在现有报告批处理流程中，text_collection 中的文本信息被批量读入，对应的词频被批量计算并存入数据库。相应地，该类提供指定现有报告词频的读取，以及现有报告词频的批量读取。在提交报告处理流程中，IRTermFrequency 读入新报告的文本，输出对应的词频。

需求13 批量处理报告文本信息，生成对应词频，存入数据库

需求14 处理单个报告的文本信息，生成对应词频

需求15 从数据库中读取指定报告的词频

需求16 批量读取数据库中的词频信息

功能设计

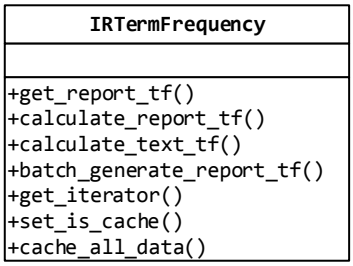

图 5.8 IRTermFrequency 类图

IRTermFrequency 的核心功能函数 calculate_text_tf 函数实现了词频向量的计算。其输入为一段文本（概要或者详细描述），输出本文的词频向量。calculate_text_tf 的算法流程依照图 4.3 所示。函数使用 Python 的 dict 类将一段文本的词频向量保存为“键 • 值”对。dict 类具有较高的检索性能（O(1)），且该结构可直接存储到 MongoDB 中。

calculate_report_tf 函数使用 calculate_text_tf 函数分别计算报告的概要和详细描述的词频向量。batch_generate_report_tf 批量计算现有报告的词频向量。它遍历 text_collection 中的每条报告，通过调用 calcuate_report_tf 函数计算其词频向量，并

存入数据库。与 IRText 的设计类似，get_report_tf 函数返回一条报告的词频向量（概要和详细描述的），get_iterator 函数返回词频数据库的遍历迭代器。

存储设计

tf_collection 存储了现有报告的词频向量。其中 summary 和 description 为报告的概要及详细描述的词频向量。

表 5.5 tf_collection 结构

键	说明	例
report_id	报告 ID	100100
summary	概要的词频向量	{“browser”:1, “crash”:1}
description	详细描述的词频向量	{“browser”:1, “crash”:1,“when”:1}

5.4.2.3 文档频率信息：IRDocumentFrequency

需求分析

IRDocumentFrequency 类在现有报告批处理流程中计算报告的文档频率并存入数据库。相应地，提供对指定单词文档频率的检索。

需求17 批量计算现有报告的单词的文档频率，并存入数据库

需求18 从数据库中读取指定单词的文档频率

功能设计

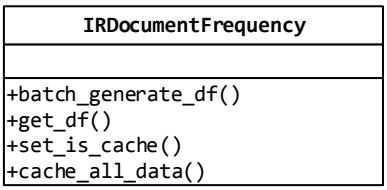

图 5.9 IRDocumentFrequency 的类图

batch_genrate_df 函数读入 tf_collection 的词频信息，统计所有出现过的单词的文档频率。由于报告分为概要和详细描述两部分，并且相同的单词在不同的部分的重要性（文档频率的意义，见 2.2.1.1）有可能不同，因而函数分别计算单词在这两部分各自的文档频率。相应地 get_df 函数读取数据库并返回给定单词 term 的文

档频率。

存储设计

df_collection 分别存储了概要及详细描述两部分词频，其中 term 为单词，dfs 为一个有两个元素的数组，分别表示 summary 和 description 的文档频率。

表 5.6 df_collection 结构

键	说明	例
term	单词	“browser”
dfs	文档频率	340

5.4.2.4 BTF-IDF 信息：IRBTFIDF

需求分析

IRBTFIDF 类负责计算报告的 BTF-IDF 向量。在现有报告批处理流程中，现有报告的 BTF-IDF 向量被批量计算，并存入数据库。对应地，该类提供查询现有报告的 BTF-IDF 向量功能。在新提交报告处理流程的中，IRBTFIDF 读入新报告的词频信息，输出对应 BTF-IDF 向量。最后，由于报告相似度基于报告文本的 BTF-IDF 向量的相似度，该类需要提供两个 BTF-IDF 向量相似度的计算。

需求19 根据报告的词频向量，计算该报告的 BTF-IDF 向量

需求20 批量计算现有报告的 BTF-IDF 向量，并存入数据库

需求21 从数据库中读取指定现有报告的 BTF-IDF 向量

需求22 计算两个 BTF-IDF 向量间的相似度。

功能实现

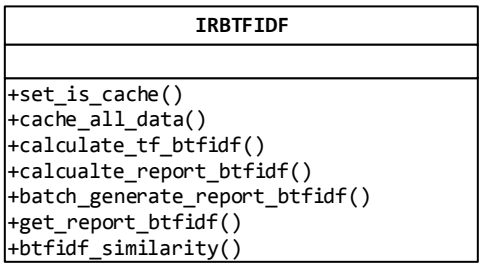

图 5.10 IRBTFIDF 类的类图

calculate_tf_btfidf 函数是计算 BTF-IDF 的核心。该函数以词频向量为参数，返回该词频向量对应的 BTF-IDF 向量。根据 0 中文本相似度计算公式,该函数伪代码如代码 5.6 BTF-IDF 向量计算伪代码所示。

```
01 CALCULATE_TF_BTFIDF(tf, N)
02     btfidf ← new dict()
03     for term, freq in tf
04     do df ← IRDocumentFrequency.get_df(term)
05        idf = log((N+1)/(df+1))
06        btfidf[term] = idf
07     return btfidf
```

代码 5.6 BTF-IDF 向量计算伪代码

calculate_report_btfidf 函数通过调用 calculate_tf_btfidf 函数分别计算报告的概要及详细描述的 BTF-IDF 向量。batch_generate_report_btfidf 函数通过遍历 tf_collection，计算每条现有报告的 BTF-IDF 向量，并存入数据库中。相应地，get_report_btfidf 函数从数据库中检索给定报告的 BTF-IDF 向量。

btfidf_similarity 函数计算两 BTF-IDF 向量的相似度，结果用[0.0,1.0]间的浮点数表示。根据 0 对文档相似度计算的设计，该相似度是非对称的，因此两个被计算的向量的参数顺序极为重要。第一个参数 primary 对应被检测的报告的向量，第二个参数 secondary 对应现有报告的向量。第三个参数为用户负反馈标记的“完全无关”的词条。计算过程如下：

```
01   BTFIDF_SIMILARITY(primary, secondary, penalty)
02      plength ← get_squared_length(primary)
03      if plength < epsilon
04         return 1
05      similarity = 0
06      for term, tf in primary
07         do if term in secondary
08               similarity += tf * secondary[term]
09      neg = 0
10      for term in penalty
11         do if term in penalty
12               neg += secondary[term]
13      return max(0, (similarity - alpha*neg) / plength))
```

代码 5.7 BTF-IDF 向量相似度计算伪代码

第 2 行首先计算被检测报告向量 primary 的长度。如果长度为 0（epsilon 是一个很小的正浮点数），说明报告中没有提供信息，根据工具对相似度的定义——可能的重复报告，该值应该为 1.0，即当前报告可能与任何一条报告相似。5~8 行计算向量间的非单位化的相似度；9~12 行检测现有报告向量中是否包含负反馈中的词。如果有，则对相似度进行“惩罚”。第 13 行中，alpha 是惩罚的系数。对惩罚后的值进行单位化。由于惩罚后的相似分值有可能为负，需要将负数的分值截取为 0。

存储设计

报告的 BTF-IDF 向量的存储与报告词频向量的存储方式类似。

表 5.7 btfidf_collection 结构

键	说明	例
report_id	报告 ID	100100
summary	概要的 BTF-IDF 向量	{“browser”:2.24, “crash”:0.23}
description	详细描述的 BTF-IDF 向量	{“browser”:1.65, “crash”:0.12,“when”:0.04}

5.4.2.5 堆栈分析：IRStacktrace

需求分析

IRStacktrace 类提供了关于堆栈信息的提取、比较功能。不同的社区的堆栈信息存在格式及内容上的差异，因此，IRStacktrace 需要从这些不同格式的堆栈信息中提取出计算堆栈相似度所需的信息，并以统一的方式存储。最后，IRStacktrace 需要计算堆栈间的相似度。

需求23 从现有报告的详细描述中提取堆栈信息，并存入数据库

需求24 获取指定报告的堆栈

需求25 计算堆栈间的相似度

功能实现

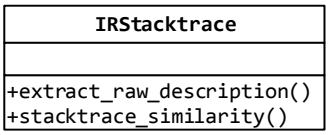

图 5.11 IRStacktrace 类图

extract_raw_description 函数将包含的堆栈信息的详细描述文本分割为字符串表示的文本部分和数组表示的堆栈信息。函数通过正则表达式识别文本中的堆栈。在现有报告批处理流程中，该函数被 IRText.parse_*函数调用，并将提取的堆栈信息存入 stacktrace_collection。在新提交报告处理流程中，该函数用于提取新报告的堆栈信息。stacktrace_similarity 函数计算两个堆栈信息的相似度。

5.4.2.6 批处理流程入口：IRPipeline

需求分析

IRPipeline 类提供了现有报告批处理流程的入口。该类负责调用动流程中相关类，完成对现有报告的批处理。由于现有报告数量多、批处理计算量较复杂，完整执行批处理流程需要较长的时间。工具调试过程中可能需要根据实验效果调节参数并更新现有报告数据。这时很可能不需要重新进行完整的批处理流程，而只需要

重新计算一部分数据。IRPipeline 通过分析流程各步骤依赖关系，只计算受参数更新影响的部分数据，避免了不必要计算。

例如在图 5.5 核心算法模块的类与流程关系图中，IRTermFrequency 计算依赖 IRText 的计算结果，IRText 的计算依赖于现有报告数据。如果采用简单的顺序更新方式，对 IRTermFrequency 中的计算的调整，将导致整个流程的重新执行。而实际上，仅仅需要重新执行 IRTermFrequency 步骤及其后续步骤即可。

需求26 建立批处理数据依赖关系，并自动根据依赖关系更新相应数据

功能实现

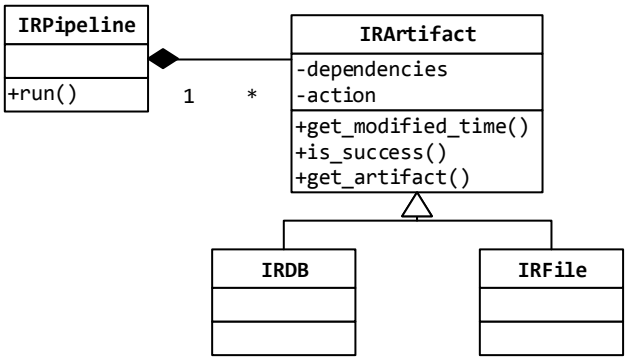

图 5.12 批处理流程涉及的类的类图

Linux 下的工具 make 可根据文件的依赖以及更新时间，判断受否执行相应的生成脚本。make 常被用于 c/c++等工程的编译、链接。然而，对于本工具，make 工具不能很好的满足需求，原因有 3 点：首先，Intereport 大部分数据存储在数据库中，使用 make 不易获取数据库中 Collection 的情况；其次，批处理脚本有可能在中途被中断，因而对应的 Collection 有可能仅被部分写入（即 make 认为 Collection 已经被成功更新，但实际上该更新是失败的）；最后，make 通过 shell 命令调用相应的处理，这迫使每个批处理步骤都需要提供可执行程序的入口，并且无法让 IRLog、IRConfig 等对象生存与流程的整个过程。

综上所述，Intereport 没有使用 make 工具，而进行了如 xx 图所示设计。IRArtifact 类是文件或 Collection 的抽象。其中 dependencies 为该对象所依赖的 IRArtifact 对象。action 指向生成该文件或 Collection 的函数。get_modified_time 和 is_success 函

数为虚函数，分别返回文件或对象的修改时间及修改是否成功。其中的 get_artifact 函数实现了根据依赖生成该对象代表数据的逻辑，如代码 5.8 所示：

```
01 GET_ARTIFACT()
02     need_update ← False
03     for artifact in dependencies
04     do artifact.GET_ARTIFACT()
05         if artifact.get_modified_time() > get_modified_time()
06             need_update = True
07     if is_success() = False
08         need_update = True
09     if need_update
10         action()
```

代码 5.8 根据依赖关系判断是否更新文件或 Collection 的伪代码

2~8 行首先递归地获取所依赖的 IRArtifact 对象，保证被依赖对象已经更新。然后，如果存在被依赖对象的修改时间晚于本对象，或者本对象最后一次更新失败，则需要重新计算本对象对应的文件或 Collection（9~10 行）。

IRFile 和 IRDB 为 IRArtifact 的子类，分别代表文件和和 Collection。它们分别实现 get_modified_time 和 is_success 函数。IRFile 的 get_modified_time 通过 Python 提供的查询文件最后修改时间的函数得到。由于 Intereport 不涉及写入文件，因此 IRFile 的 is_success 返回 True。IRDB 的 get_modified_time 及 is_success 则通过访问 5.4.1.3 中 IRCollection 实现的 collection_meta 得到。

代码 5.9 展示了定义及执行现有报告批处理流程的 Python 脚本。1~5 行定义了文件或 Collection 间的依赖；第 6 行通过调用 btfidf_collection 的更新带动其依赖项的更新。

```
01 info = IRFile('info_level1.dat', dep=[], action=None)
02 text = IRDB('text_collection', dep=[info], action=
   IRText.parse_info_level1)
03 tf = IRDB('tf_collection', dep=[text],
   action=IRTermFrequency.batch_generate_report_tf)
04 df = IRDB('df_collection', dep=[tf],
   action=IRDocumentFrequency.batch_generate_df)
05 btfidf = IRDB('btfidf_collection', dep=[tf, df],
```

```
    action=IRBTFIDF.batch_generate_report_btfidf)
06  btfidf.get_artifact()
```

代码 5.9 使用 IRPipeline 定义的现有报告批处理流程

5.4.2.7 缺陷报告：IRReport

需求分析

IRReport 类是对缺陷报告的抽象。它对上层的 IRRecommender、IRInteractor 屏蔽数据库操作、计算等方面的细节，提供统一的功能接口。这里的“统一”包含 n 层含义：首先，Intereport 所涉及的报告分为两类。一类是缺陷追踪系统中的现有报告。这类报告的信息经过批处理，已经存在数据库中。另一类是报告者提交的新报告，这类报告的信息需要在与报告者交互式通过新报告处理流程得到。IRReport 必须为这两类报告提供统一的信息访问方式。其次，可通过 IRReport 获得有关报告的信息。使用 IRReport 不需要直接对下层的类及数据库进行操作。最后，由于缺陷报告的信息之间存在依赖关系，IRReport 应自动完成被依赖信息的计算、缓冲。例如，对于新提交报告，初始信息只有报告时间、产品、文本（概要、详细描述）、堆栈。假如需要得到该报告的 BTF-IDF 向量，而该信息依赖于词频向量，则 IRReport 应自动计算词频向量，而后计算并返回 BTF-IDF 向量。此外，IRReport 实现了报告间相似度的计算，以及相似报告与重复报告的检索。

需求27 IRReport 对象对现有报告和新提交报告提供统一的访问接口

需求28 获得报告的信息

需求29 计算报告间的相似度

需求30 检测指定报告的相似报告及重复报告

功能实现

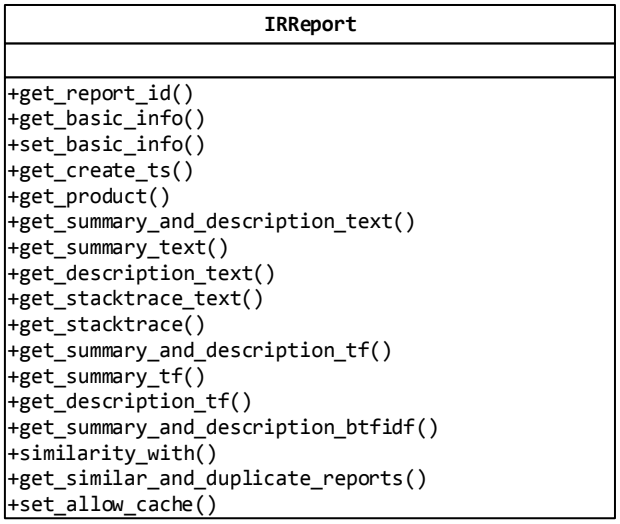

图 5.13 IRReport 类图

对应现有报告和新提交报告，IRReport 对象有有两种创建方式：以现有报告的 ID 作为参数；以新提交报告的报告时间、产品、概要、详细描述及堆栈作为参数。任何一种方式创建的 IRReport 对象都可通过 get_*得到报告相关信息。在 get_*函数内部，如果该 IRReport 对象代表现有报告，由于相关信息已在数据库中，则通过 IRText，IRTermFrequency，IRBTFIDF 等类获得所需信息；如果该 IRReport 对象代表新提交报告，则所需信息可能需要经过递归地计算得到。

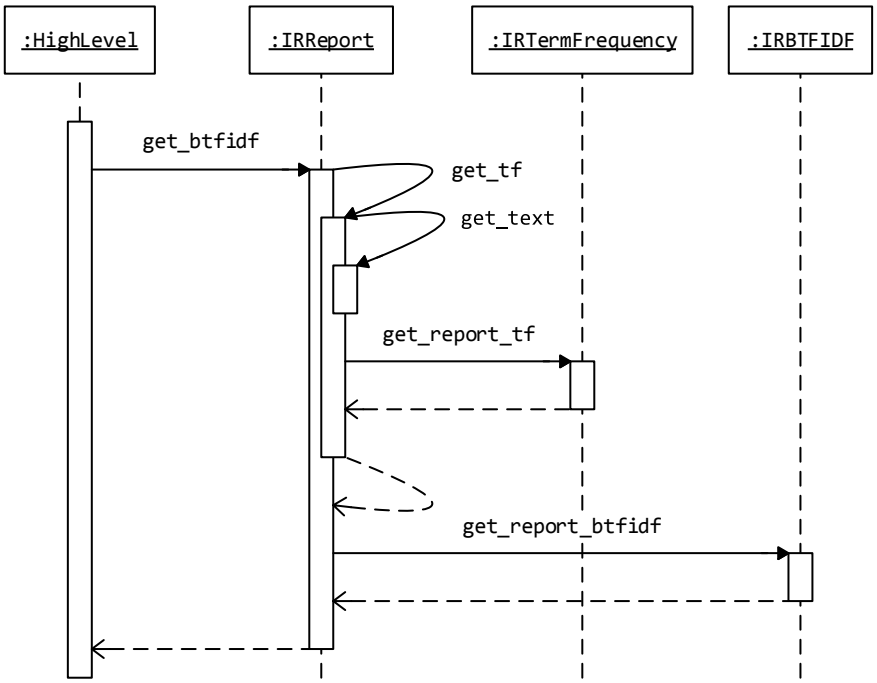

图 5.14 IRReport 获取 BTF-IDF 的顺序图

图 5.14 展示了 get_btfidf 被调用的顺序图。BTF-IDF 向量的计算依赖于词频向量。由于词频向量未生成，IRReport 对象首先使用 get_text 获取文本，然后调用 IRTermFrequency 提供的功能计算词频向量。最后使用 IRBTFIDF 提供的功能计算并返回 BTF-IDF 向量。

similarity_with 函数根据 33 中的方法计算报告间的相似度。由于相似度计算是非对称的，因而该函数约定，当前 IRReport 对象为新提交报告，参数传入的 IRReport 对象为现有报告。基于 similarity_with 函数，get_similar_and_duplicate_reports 函数返回新提交报告的相似报告及重复报告。

5.4.2.8 建议生成：IRRecommender

需求分析

IRRecommender 类依照 4.4.2 设计的算法实现了完善报告信息的建议的生成。该功能输入为一个代表新提交报告的 IRReport 对象，输出为建议信息：关键词、例句、重复报告。

需求31　根据新提交报告生成完善报告信息的建议

实现描述

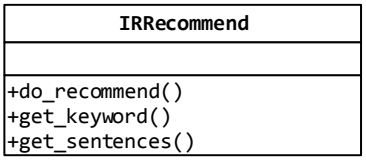

图 5.15 IRRecommend 类图

do_recommend 函数首先通过调用 IRReport 对象的 get_similar_and_duplicate_reports 函数获得相似报告及重复报告。然后依照图 4.9 所设计流程，调用 get_keyword 函数计算关键词，调用 get_sentences 函数计算例句，最后返回关键词、例句和重复报告。

5.4.3 会话模块

报告者使用 Interport 进行报告完善、提交需要与工具进行一系列的交互。工具需要为一次交互维持一个对应的“会话”：报告者创建新报告时，会话被创建；随后，会话负责与该报告者进行交互，完成相似报告检测和建议生成；最后，报告者提交报告，会话结束。因为有可能同时有多个报告者提交报告，因此服务器中有可能同时存在多个“会话”。另外，Intereport 检测相似报告、生成完善建议的过程需要访问大量数据，出于性能考虑，工具进程常驻服务器内存以进行缓存。

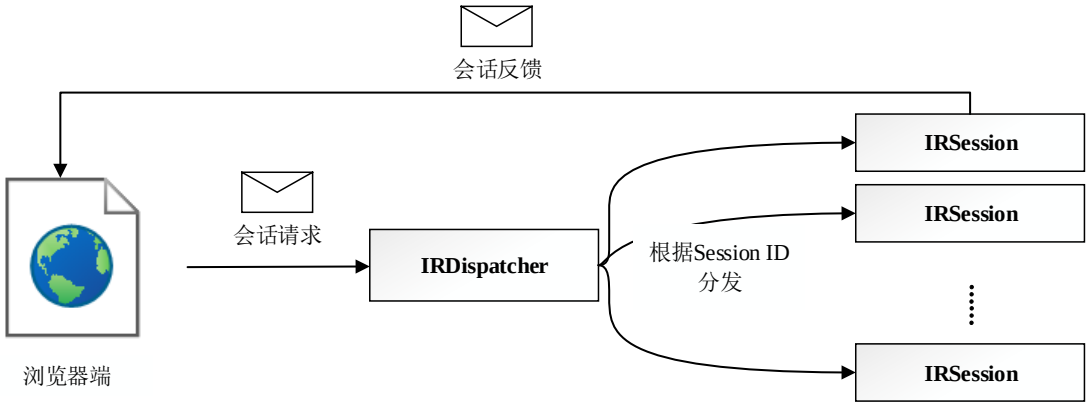

图 5.16 Intereport 的会话交互机制

图 5.16 展示了工具的会话交互机制。IRDispatcher 是 Intereport 的逻辑服务器

端程序入口，通过 TCP/IP 消息与外界交互。当逻辑服务器端程序启动后，IRDispatcher 将监听指定的端口，等待交互请求。当报告者的交互请求到来后，IRDispatcher 首先查看该请求是否属于新创建的报告。如果是，则创建新的 IRSession 对象。如果不是，则将请求转发给对应的 IRSession 对象。IRSeesion 对象接到 IRDispatcher 转发的请求后，更新会话信息，然后进行相似检测、建议生成，并将结果通过 TCP/IP 消息返回给报告者。为了使多个会话能同时进行，IRDispatcher 和 IRSession 对象均运行在独立的线程中。其中 IRDispatcher 为主线程，IRSession 为子线程。

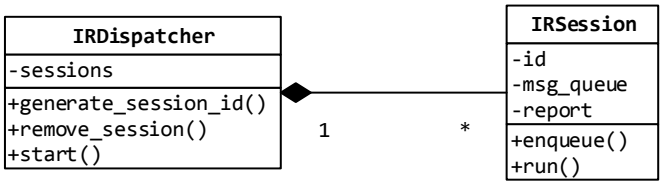

图 5.17 会话交互模块类图

5.4.3.1 TCP/IP 消息包

Intereport 通过 TCP/IP 消息包与外界通讯。其中消息使用“键•值”对存取，实现中采用了 Python 的字典(dict)结构。dict 提供了序列化与反序列化功能，使得程序可以方便地将 dict 对象与可通过 TCP/IP 传输的字符串相互转换。Intereport 接受的消息包的信息表 5.8 所示。

表 5.8 Intereport 接受的信息包

类别	键	值	说明
服务器控制	server_cache	-	开启数据 cache
	server_shutdown	-	关闭服务器
	server_help	-	返回服务器命令说明
会话	session_id	整数	负数表示新会话，正整数表示已存在会话
	set_report_info	IRReport.to_string()的返回值	设置报告内容
	append_penalty	一个关键词	添加负反馈词
	skip_keyword	一个关键词	跳过该关键词
	deny_duplicate	报告 ID	否定指定报告为重复报告
	do_recommend	-	对当前报告进行相似检测并生成建议
	confirm_duplicate	报告 ID	确认指定报告为重复报告
	submit	-	将当前报告作为新报告提交
	cancel	-	取消提交报告

5.4.3.2 **IRDispatcher**

需求分析

IRDispatcher 提供了 Intereport 逻辑服务器端会话交互的入口，负责与前端通讯、数据缓存、信息转发、会话管理。此外，该类还提供服务器的控制管理功能，使得 Intereport 服务器管理者可以采用 TCP/IP 消息的方式操作服务器。

需求32 监听指定端口，处理或分发交互消息

需求33 为新会话创建、管理 IRSession 线程

需求34 处理服务器操作请求：缓存数据、会话查询、关闭服务器

功能设计

如图 5.17 所示，IRDispatcher 为每个会话分配一个全局唯一的正整数（IRSession.id, 后文称为 SessionID），并记录了 SessionID 到对应“会话”对象的映射。每个关于会话的 TCP/IP 消息包中都包含 SessionID 信息。如果 SessionID 为负数，则为新会话。IRDispatcher 创建新的 IRSession。否则，根据 SessionID 在

sessions 中找到相应的 IRSession 对象，将消息转发给该对象。

如果 TCP/IP 消息包中不包含 SessionID，则认为是服务器操作请求。该请求由 IRDispatcher 完成。IRDispatcher 的流程图图如图 5.18 所示：

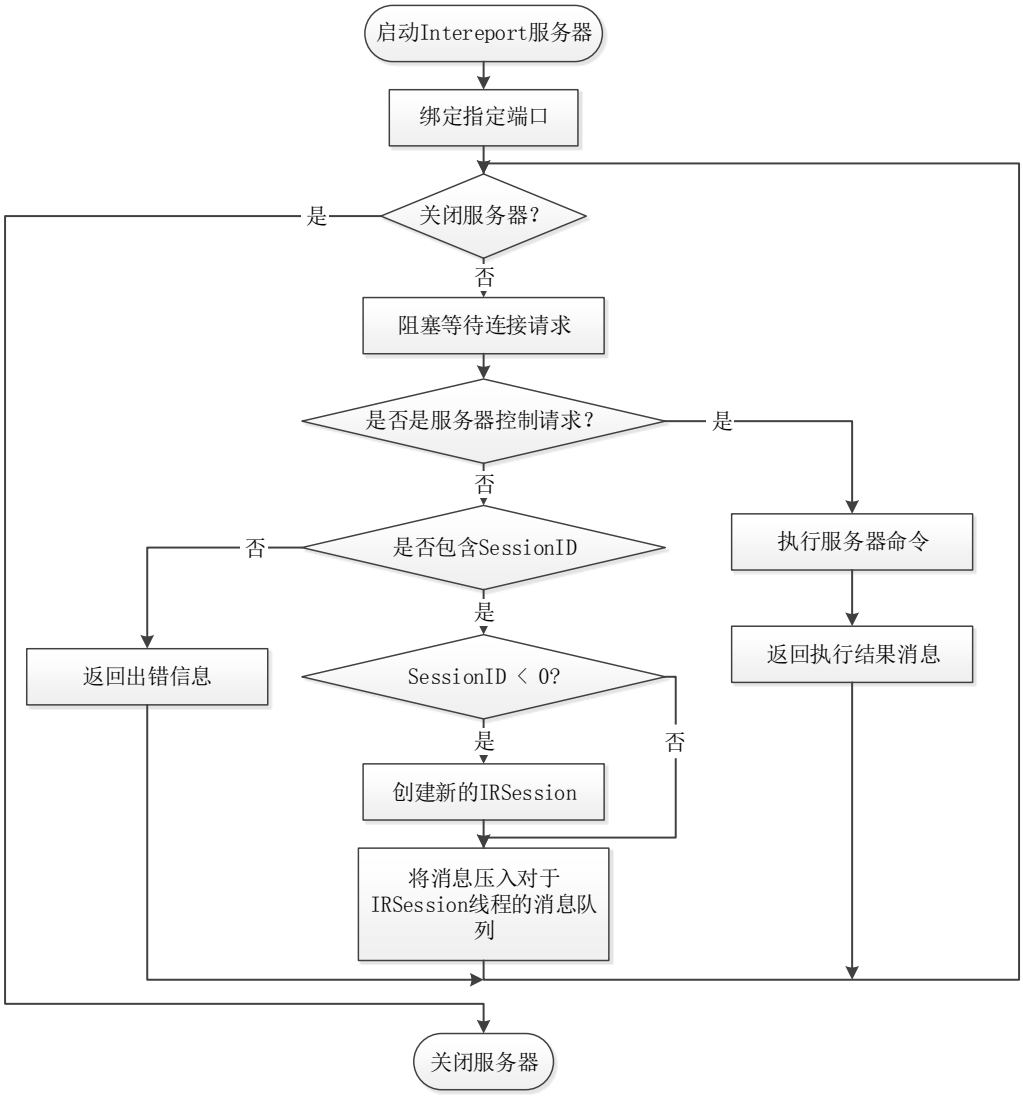

图 5.18 IRDispatcher 流程图

5.4.3.3 **IRSession**

需求分析

IRSession 负责一次对话交互的计算、反馈。同一时间会友多个报告者与服务器交互，因而会话之间应该是并行的。会话与 IRDispatcher 间需要同步机制，以使消息包能分发到 IRSession 中。最后，为了避免链接丢失对服务器的影响，在会话

等待消息超过一定时间后，应主动关闭。

需求35 IRSession 间相互独立，可并行执行

需求36 IRDispatcher 需要将消息发送 IRSession

需求37 完成计算：更新报告内容、添加负反馈词条、否定重复报告、跳过建议关键词、重复检测并生成建议、确认重复报告、提交报告、取消报告

需求38 会话等待超过指定时间则自动关闭

功能设计

由于，IRSession 被设计为线程类，其逻辑在单独的线程中完成。IRDispatcher 与 IRSession 之间的通讯符合“生产者-消费者”（IRDispatcher 通过向 IRSession 分派消息而“生产”消息，IRSession 通过对得到的消息进行计算、反馈而“消费”消息）模式，因而使用 Python 中提供的 Queue 类。Queue 提供了一个线程间的同步队列：put 函数将数据放入队列中（非阻塞写）；get 函数从队列中读取数据。如果队列为空，则线程将一直等待（阻塞读），直到队列中有了数据，或者等待超过指定的时间。对于需求 37 ，IRSession 通过读取消息包中信息并调用 IRReport 及 IRRecommender 中的相关函数完成。图 5.19 为 IRSession 的流程图。

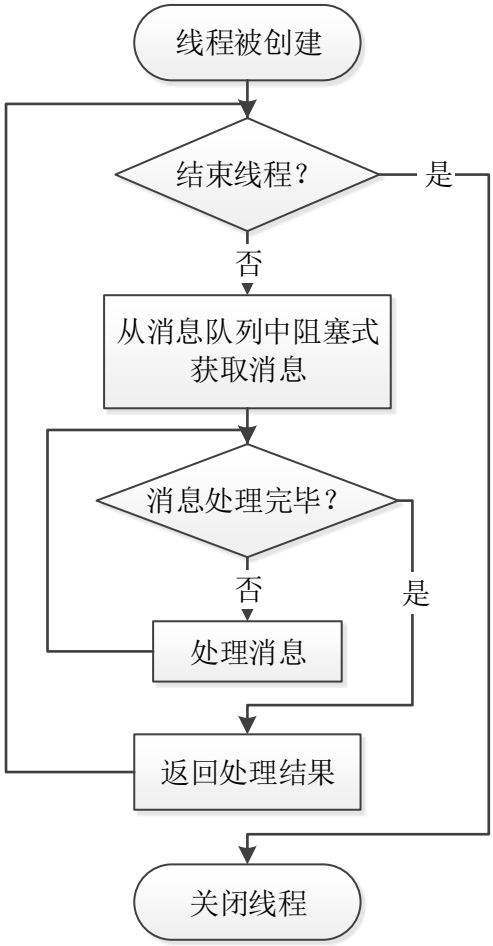

图 5.19 IRSession 流程图

5.5 交互部分

报告者通过网页访问 Intereport，完成报告创建、完善、提交。交互部分负责创建交互页面，将用户的操作转发给逻辑服务器，并根据逻辑服务器的返回结果更新页面。交互部分使用 Apache+Flask 结构实现应用服务器端，使用 JQuery 和 BootStrap 实现前端页面。

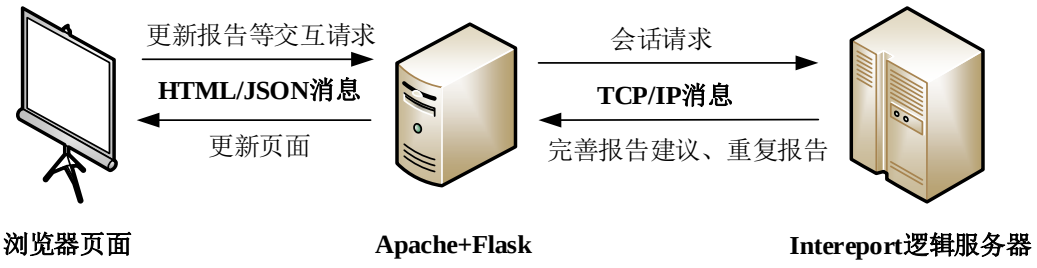

图 5.20 Intereport 交互部分结构图

交互部分共设计 3 个页面：报告创建页面、报告完善页面、提交结果页面。页面之间的跳转关系如下图所示。

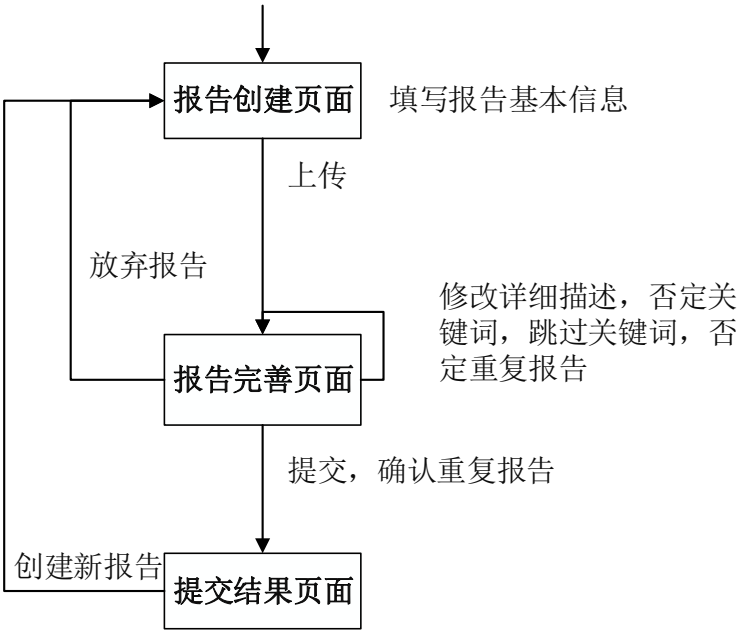

图 5.21 Intereport 页面跳转关系

报告者在报告创建页面中填写报告的基本信息，包括报告所属的产品、概要、初始版本的详细描述和堆栈（可以为空）。

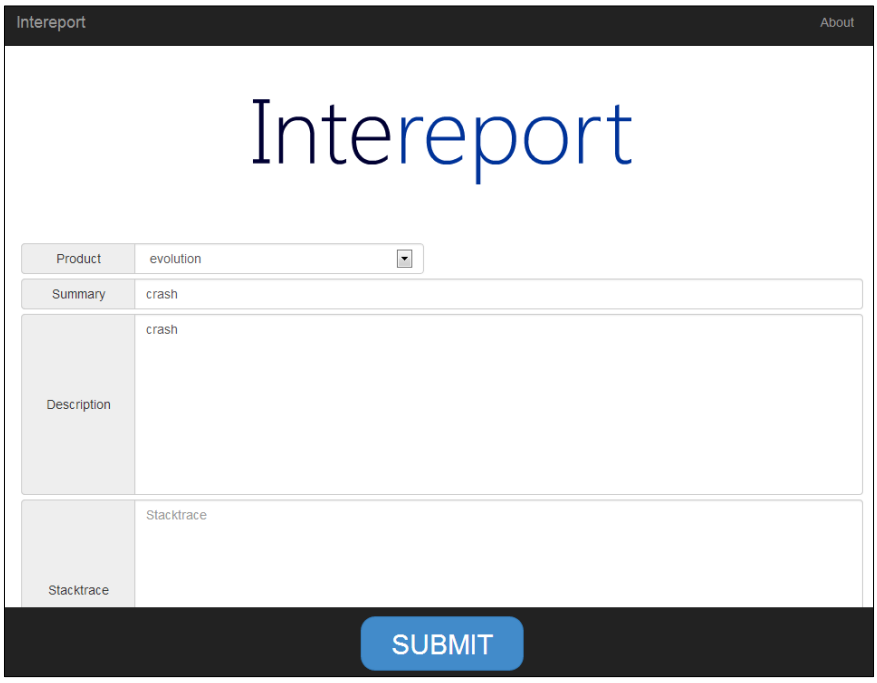

图 5.22 Intereport 报告创建页面

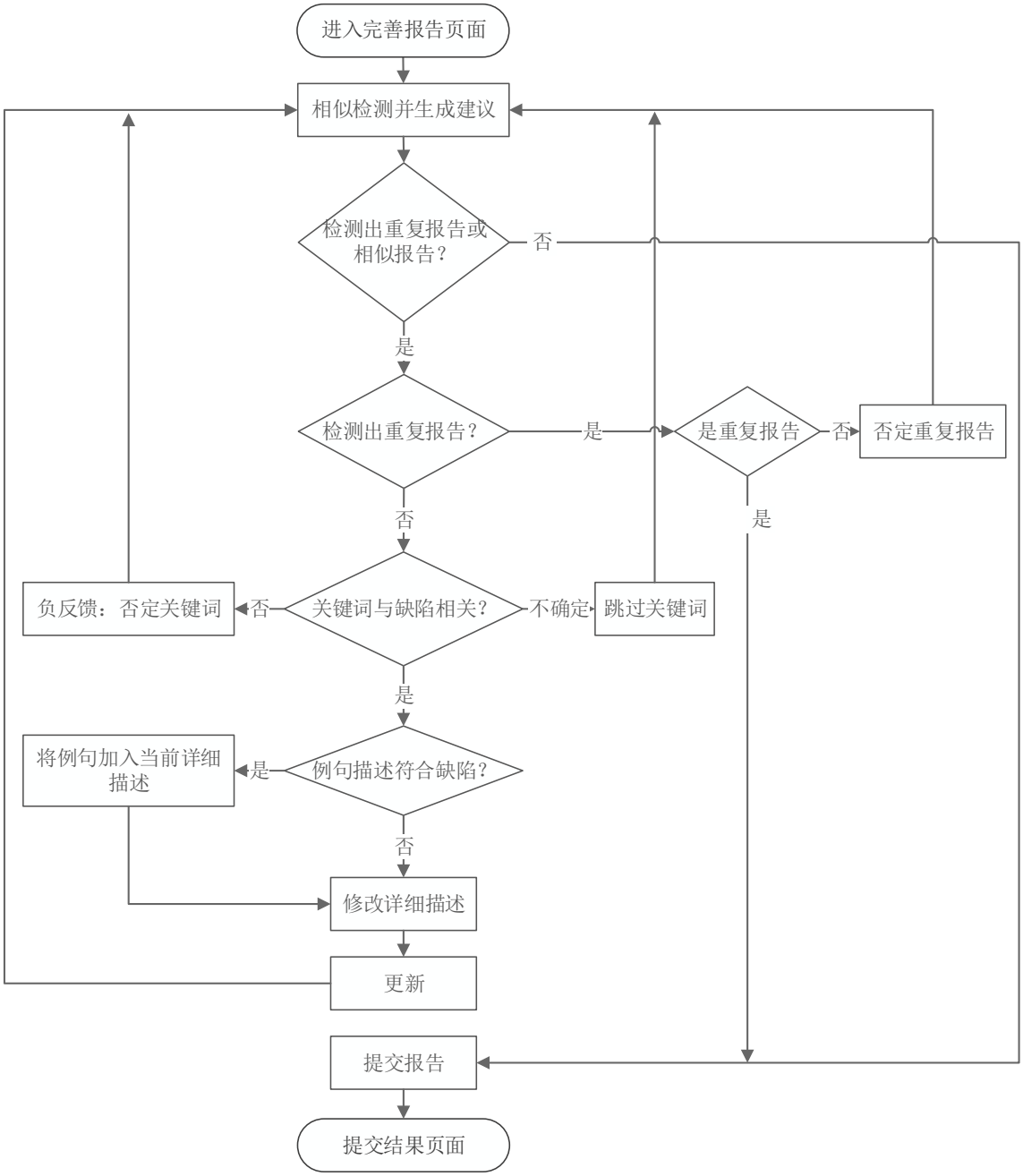

图 5.23 报告完善页面交互流程图

报告创建以后进入报告完善页面。该页面采用多轮交互的方式引导用户逐步完善报告信息，帮助用户发现重复报告。该页面的交互流程如

图 5.23 所示。在每一轮交互中，如果当前报告仍需要补充信息，页面给出代表需要完善的信息的关键词以及例句。如果报告者发现例句与自己的情况完全相符，可通过点击“Me too”按钮将例句加入新报告的的详细描述中（下方），或直接编辑新报告的详细描述；如果报告者认为该关键词完全无关，可点击“Never”

按钮将该关键词加入负反馈列表中；如果报告者不确定该关键词的相关性，可点击“Skip”按钮跳过该词。另外，工具会将检测到的重复报告列出。报告者可以点击“Me too”确认重复，或点击“Not mine”否定重复。如果新报告的信息已经很充足，页面将提示报告者进行报告提交。更新详细描述、提交负反馈词、跳过关键词或否定重复报告都将使 Intereport 进行重新计算，从而进入下一轮交互。确认重复、提交报告将结束交互过程，即完成报告的完善并提交。

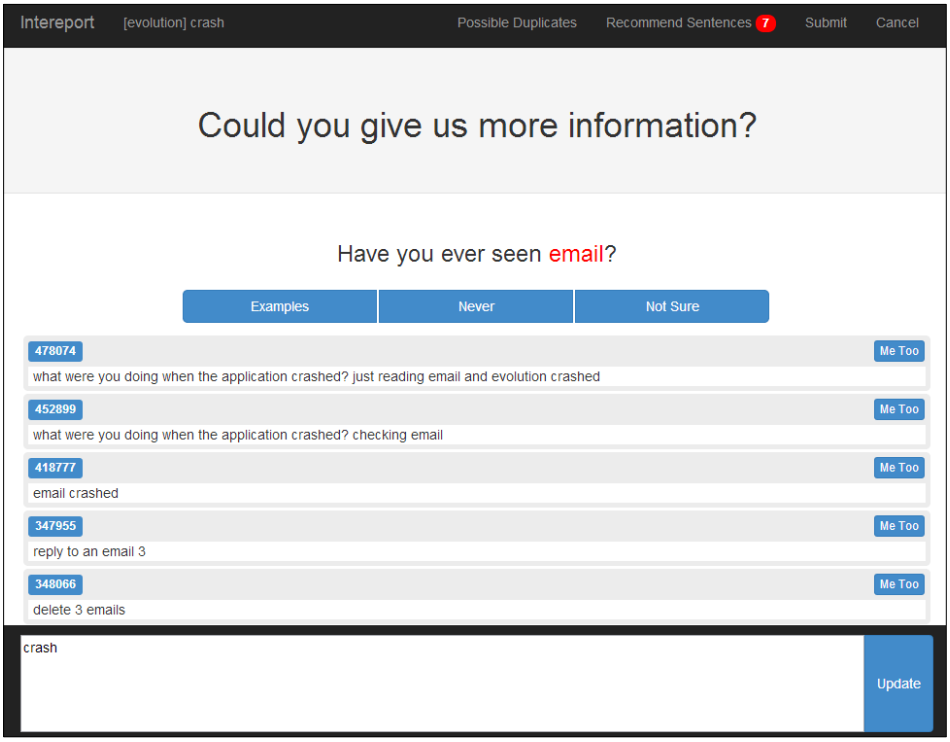

图 5.24 Intereport 报告完善页面（未检测出重复报告时）

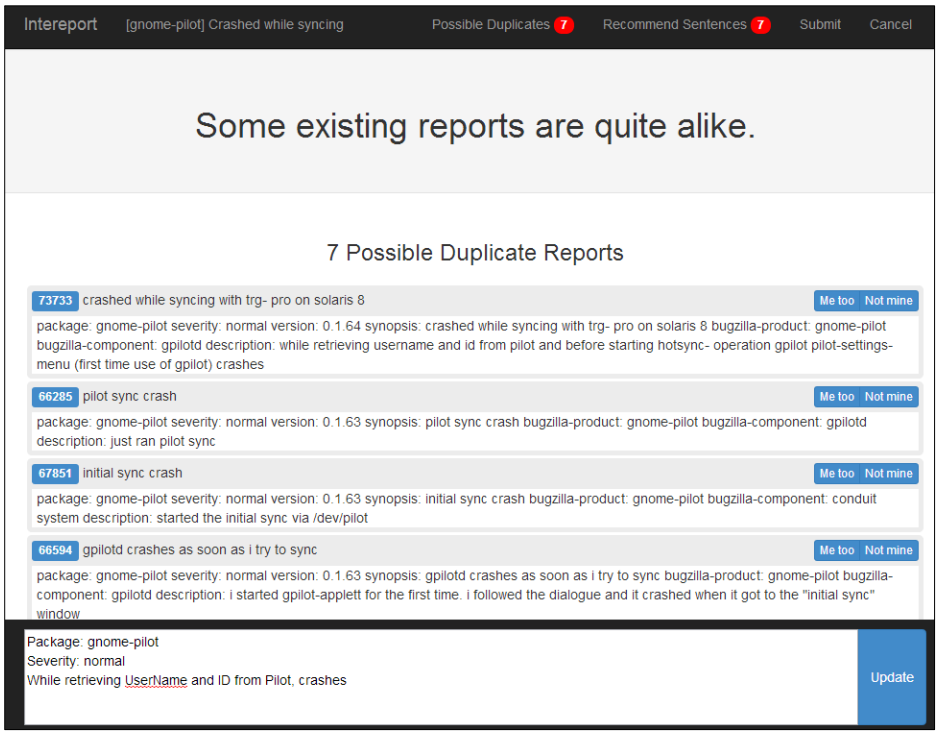

图 5.25 Intereport 报告完善页面（检测出重复报告时）

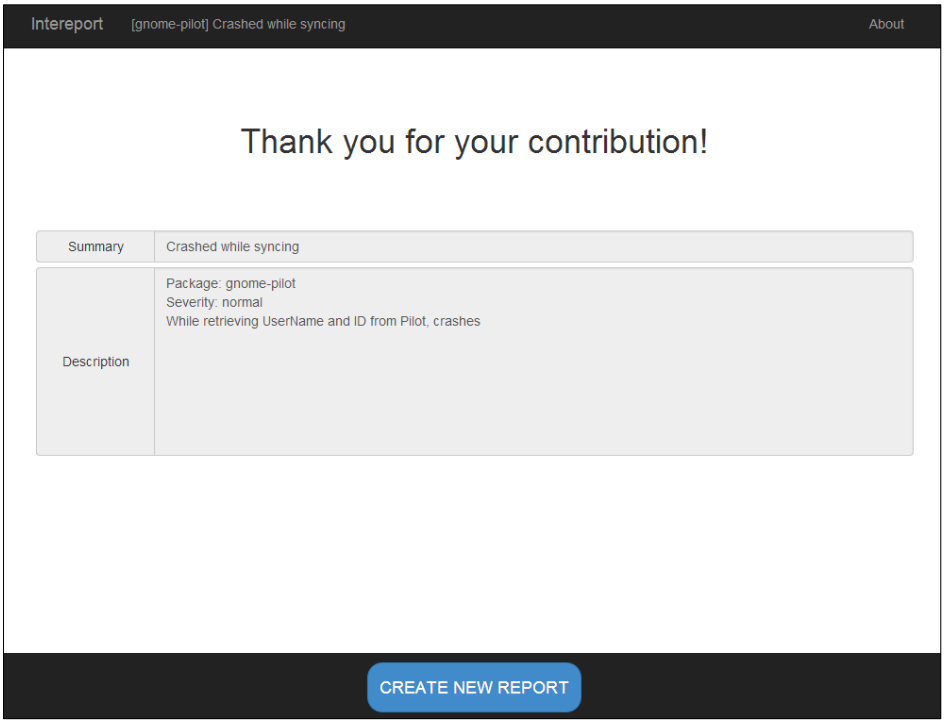

图 5.26 Intereport 提交结果页面

6 实验评估

Intereport 的核心功能由相似检测和建议生成组成。因此，本文针对这两部分分别设计了实验。在关于相似报告检测的实验中，工具随机检测测试数据中的相似报告和重复报告。实验使用检测结果的准确率和召回率来评价检测效果。对于建议生成，本文通过模拟报告者通过与 Intereport 交互完善、提交报告过程，说明工具的有效性。下面分别介绍两部分实验。

6.1 测试数据

本文实验使用了 Gnome 社区自 2011 年至 2010 年的 419k 条报告。其中有重复报告 191k 条。

6.2 相似检测效果评估

6.2.1 实验设计

相似检测输出为相似报告和重复报告，相应地，本实验测试的目的有三个：1、对于输入的被检测报告，工具检索出的相似报告是否包含包含该报告的重复报告。因为只有重复报告被包含在相似报告中，建议生成模块生成的建议才有可能引导报告者提供有效的信息。2、对于被检测报告，工具检索出的相似报告是否准确。3、对比两种堆栈相似度算法（2.2.1.2 中的比较堆栈中所有函数签名的算法，记作 AllSig 及 0 中仅比较关键的函数签名的算法，记作 KeySig）的效果。

为了回答上述问题，本文定义检测结果的准确率与召回率。对于输入的被检测报告 r，不妨设工具检索出的相似报告集合为 S_r，实际上的重复报告集合为 D_r[24]。则准确率和召回率为：

$$\text{SPrecision} = \frac{|S_r \cap D_r|}{|S_r|}$$

[24] 由于已经将被检测报告 r 从“项目报告”中排除，因此 $r \notin S_r$ 且 $r \notin D_r$。

$$\text{SRecall} = \frac{|S_r \cap D_r|}{|D_r|}$$

对于检测出的重复报告集合 M_r，由于报告者可从工具反馈的重复报告列表中确认重复报告，因此可以认为只要有一条真正的重复报告出现在重复报告列表中，即可达到发现重复报告的目的。因此，评估重复报告列表是否包含真正的重复报告，即 MC：

$$\text{MC} = \begin{cases} 1, \text{如果} M_r \cap D_r \neq \phi \\ 0, \text{否则} \end{cases}$$

另外，由于工具使用相似度阈值选取相似报告，因此实验观察每次检测返回的相似报告数量 S#。工具仅在相似报告较少时检测出重复报告，因此实验观察工具是否检测出了重复报告 FD：

$$\text{FD} = \begin{cases} 1, M_r \neq \phi \\ 0, \text{否则} \end{cases}$$

实验从 191k 条报告中随机抽取 60 条报告，然后从 419k 条中除去抽取出的 60 条报告，将余下的报告作为“项目现有报告”。为了模拟信息不足的报告，实验对这 60 条报告中的词分别以 0.2、0.4、0.6、0.8、1.0 的概率随机保留，得到共 300 条报告。对这些报告，分别使用不同的堆栈相似度算法、不同的相似报告阈值（0.65，0.70，0.75）对该组中的每一条报告做相似检测。以<堆栈相似度算法（SA）、单词保留率（Rate），相似报告阈值（Sim-Thre）>作为组别，则实验共包括 2×5×3 组报告，每组有 60 条报告。对每一组中的报告，工具分别进行相似检测，并统计该组的平均相似准确率（AvgSP）、平均相似召回率（AvgSR）、平均相似报告数（AvgS#）、平均重复报告列表包含真正重复报告的概率（AvgMC）以及平均检测出重复报告的概率（AvgFD）。另外，本实验设定 Intereport 返回的重复报告列表的最大长度为 7。

6.2.2 实验结果

表 6.1 相似检测的实验结果

SA	Sim-Thre	Rate	AvgSP	AvgSR	AvgS#	AvgMC	AvgFD
KeySig	0.65	0.2	0.33	0.48	321.35	0.51	0.76
		0.4	0.30	0.44	392.20	0.50	0.74
		0.6	0.38	0.44	287.22	0.58	0.80
		0.8	0.38	0.44	224.50	0.60	0.80
		1.0	0.37	0.41	210.48	0.55	0.81
	0.70	0.2	0.33	0.44	266.61	0.45	0.78
		0.4	0.29	0.32	322.93	0.42	0.80
		0.6	0.37	0.32	197.15	0.49	0.83
		0.8	0.31	0.32	180.02	0.44	0.83
		1.0	0.32	0.28	177.94	0.41	0.85
	0.75	0.2	0.28	0.28	158.54	0.40	0.83
		0.4	0.30	0.22	178.89	0.45	0.91
		0.6	0.35	0.24	111.11	0.47	0.91
		0.8	0.35	0.24	154.69	0.46	0.89
		1.0	0.30	0.17	80.22	0.39	0.91
AllSig	0.65	0.2	0.28	0.47	437.24	0.54	0.65
		0.4	0.29	0.33	393.43	0.40	0.80
		0.6	0.35	0.30	154.37	0.46	0.85
		0.8	0.33	0.29	157.61	0.42	0.83
		1.0	0.27	0.28	149.04	0.37	0.85
	0.70	0.2	0.30	0.38	328.70	0.56	0.76
		0.4	0.26	0.23	328.54	0.35	0.89
		0.6	0.28	0.23	106.57	0.33	0.91
		0.8	0.28	0.24	136.35	0.35	0.89
		1.0	0.27	0.23	128.02	0.31	0.89
	0.75	0.2	0.30	0.33	256.09	0.49	0.80
		0.4	0.28	0.19	180.52	0.35	0.91
		0.6	0.25	0.18	101.04	0.26	0.93
		0.8	0.30	0.19	87.61	0.33	0.89
		1.0	0.28	0.15	49.91	0.31	0.91

通过表 6.1 可以看出，在堆栈相似度算法上，仅考虑关键函数签名的算法（KeySig）比考虑所有函数签名的算法（AllSig）效果更好。使用相同的堆栈相似

度算法时，不同的相似度阈值（Sim-Thre）对结果的影响如下：随着相似度阈值的升高，平均相似准确率（AvgSP）基本不变，而平均相似召回率（AvgSR）降低，平均相似报告数量下降。这说明较高的阈值可能导致相似检测错过一部分真正重复的报告，但可以更快地减小相似报告数量。对于检测出的重复报告，阈值升高降低了平均重复报告列表包含真正重复报告的概率（AvgMC），但平均重复报告检出率（AvgFD）增大。使用相同堆栈相似度算法及相似度阈值时，单词保留率（Rate）对各统计量有不同程度的影响：随着单词保留率的提高，平均相似报告数量（AvgS#）减少，平均相似召回率（AvgSR）和平均重复报告列表包含真正重复报告的概率（AvgMC）基本不变。这表明，随着报告信息的逐步完善，相似报告迅速减少，大部分非相似报告的相似度逐步下降，而重复报告的相似度基本保持不变。因此，当报告信息充分时，可通过相似度将重复报告区别开。

通过上述实验结果可以看出，随着报告信息的逐步完善，Intereporter 能够定位重复报告（采用 KeySig 堆栈相似度算法、相似报告阈值为 0.65 时，50%～60%的重复报告列表中包含了真正的重复报告），较好地帮助报告者发现可能重复的报告。

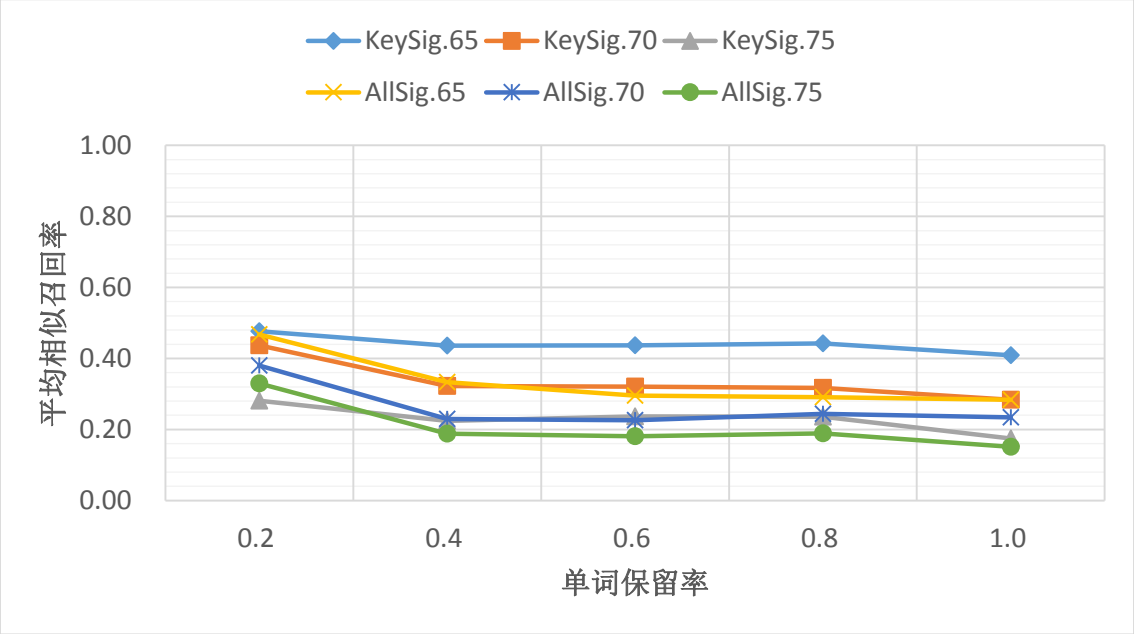

图 6.1 平均相似召回率随单词保留率的变化情况

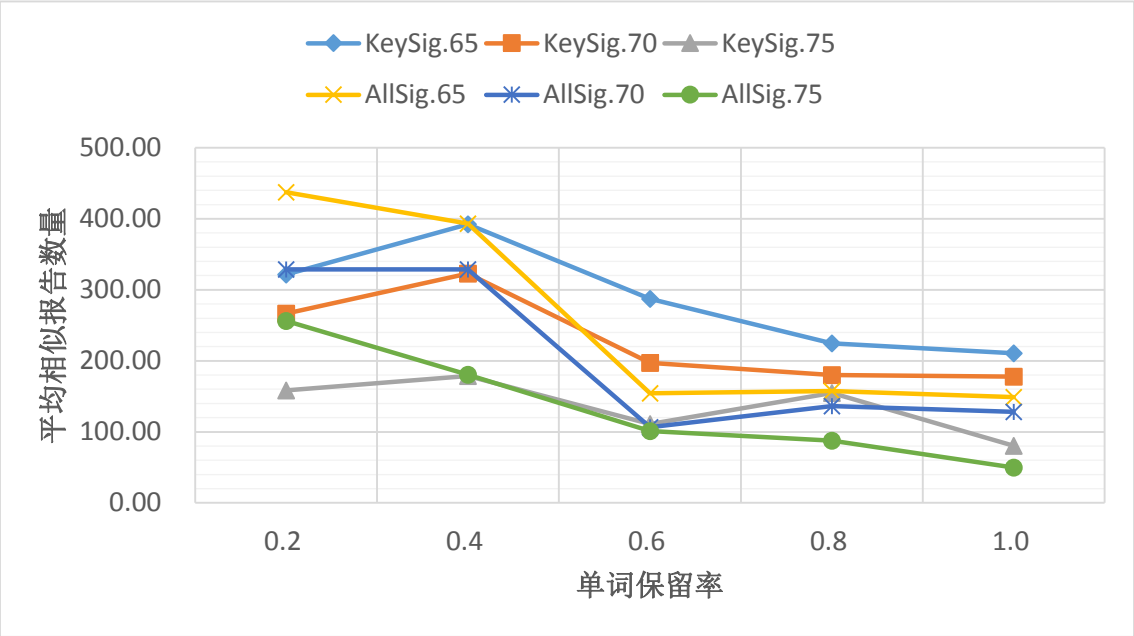

图 6.2 平均相似报告数量随单词保留率的变化情况

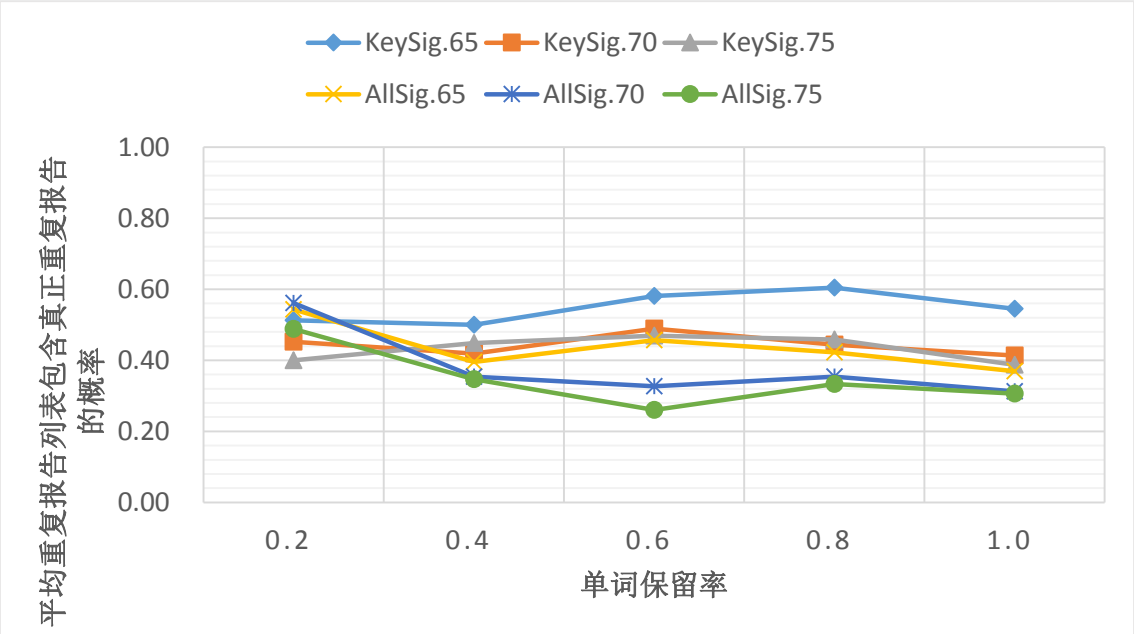

图 6.3 平均重复报告列表包含真正重复报告的概率随单词保留率的变化情况

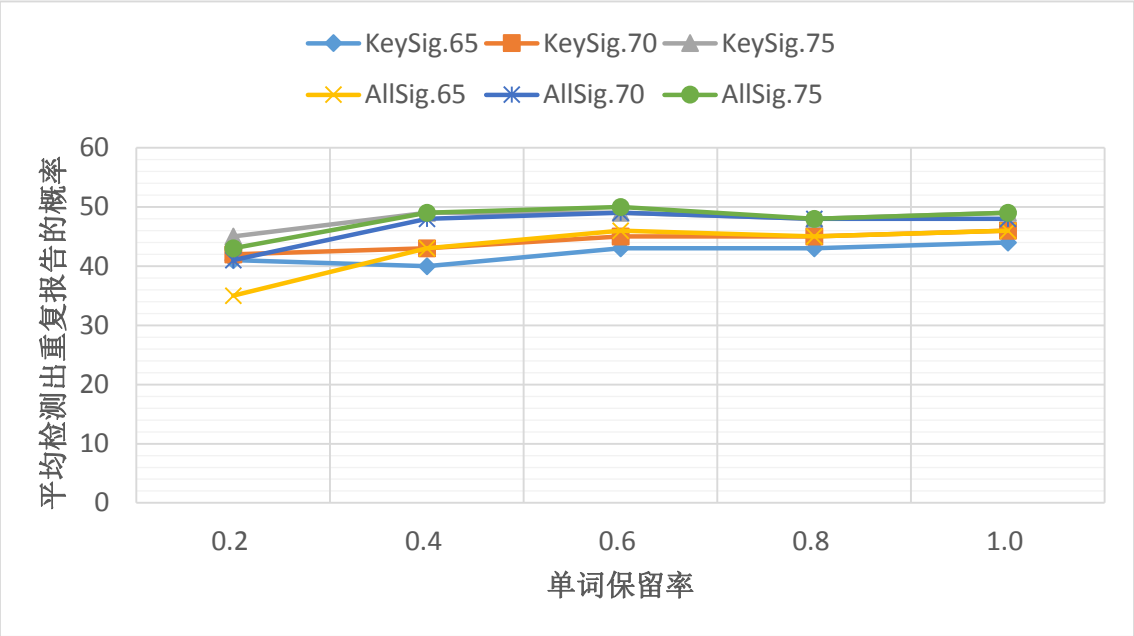

图 6.4 平均检测出的重复报告数量随单词保留率变化情况

6.3 建议生成效果评估

6.3.1 实验设计

本实验模拟报告者通过 Intereport 浏览器端创建、完善、提交报告的过程。实验分别从 419k 条报告中随机抽取报告作为新提交报告的原型[25]，并将余下的报告作为项目现有报告。对抽取出的报告原型，评估人员阅读其详细描述，使用 Intereport 提交该报告：根据 Intereport 给予的提示，评估人员逐步补充信息。实验评估 Intereport 能否引导评估人员补充报告描述，最终发现重复报告。

6.3.2 实验结果

6.3.2.1 报告原型 ID：310338

该报告原型所属的产品为 Evolution。Evolution 是 Gnome 中的一款客户端邮件应用程序。该报告创建于 2005 年 7 月 14 日，反映了 Evolution 的一个按键不工作的缺陷。该报告与 ID 为 274414 的报告重复。报告 310338 的原文如表 6.2 所示。

[25] 后文称之为“报告原型”。

表 6.2 报告 310338 的信息

ID	310338
产品	Evolution
概要	Click on 'Edit' on 'Alarm notification nothing happens
详细描述	Received Alar notification for an appointment in 'Personal Calendar'. Clicked on 'Edit' nothing happens. Expected That appointment has to open

评估人员与 Intereport 的交互如表 6.3 所示。

表 6.3 以报告 310338 为原型的交互式报告过程

序号	行为（R/I[26]）	详细信息
0	R 报告初始信息	概要：Click on 'Edit' on 'Alarm notification nothing happens 详细描述： Received Alar notification for an appointment in 'Personal Calendar'. Clicked on 'Edit' nothing happens. Expected That appointment has to open
1	I 完善建议	remind
	R 补充信息	Alarm notification (meeting reminder)
2	I 完善建议	set
	R 否定	否定 set
3	I 完善建议	often
	R 补充信息	How often dose this happen? Every time
4	I 完善建议	gnome
	R 补充信息	Version: Gnome 2.6
5	I 完善建议	tettnang
	R 补充信息	Distribution: Fedora Core release 2 (Tettnang)
6	I 发现重复报告（ID）	266116、264704、265283、274414、268335

在 Intereport 发现的重复报告中，274414 为真正的重复报告。

[26] R 表示报告者行为，I 表示 Intereport 行为

6.3.2.2 报告原型 ID：487605

该报告所属产品为Evolution。该报告创建于2007年7月17日，反馈了Evolution在撰写新邮件时执行“撤销”命令导致崩溃的缺陷。另外，该报告还包含了堆栈信息。该报告共有 4 条报告与其重复，这些报告的 ID 为：504790、508462、369979、371011。

表 6.4 缺陷报告 487605 的信息

ID	487605
产品	Evolution
概要	Crash in Evolution: I was composing an e-mail
详细描述	What were you doing when the application crashed? I was composing an e-mail, add just pasted a block of text from another application and indented the block (by highlighting it and then using the left-shift button on the menubar). This worked as expected. However after looking at the result, I decided I wanted it unindented so pressed the undo button. evolution then crashed. Distribution: Slackware Slackware 11.0.0 Gnome Release: 2.20.1 2007-10-15 (GARNOME) BugBuddy Version: 2.20.1 System: Linux 2.6.23.1 #71 SMP Sat Oct 13 20:31:02 EDT 2007 i686 X Vendor: The X.Org Foundation X Vendor Release: 60900000 Selinux: No Accessibility: Disabled GTK+ Theme: Clearlooks Icon Theme: gnome

在提交的新报告中，实验观察在不提供堆栈信息时工具是否能找到重复报告。实验过程如下表所示。

表 6.5 以缺陷报告 487605 为原型的交互式报告过程

序号	行为（R/I[27]）	详细信息
0	R 报告初始信息	概要：Crash in Evolution 详细描述：I pressed the undo button. Evolution then crashed.
1	I 完善建议	mail
	R 补充信息	I was composing an e-mail.
2	I 完善建议	send
	R 否定	否定 send
3	I 完善建议	attach
	R 否定	否定 attach
4	I 完善建议	read
	R 否定	否定 read
5	I 完善建议	prefer
	R 否定	否定 prefer
6	I 完善建议	time
	R 跳过	跳过 time，因为原型报告中未提供该信息
7	I 完善建议	paste
	R 补充信息	add just pasted a block of text from another application
8	I 发现重复报告（ID）	441031、638559、369979、456999

在工具发现的重复报告中，369979 为真正的重复报告。

6.3.2.3 报告原型 ID：460835

该报告原型所属的产品为 Nautilus。Nautilus 是 Gnome 中的文件管理器。该报告创建于 2007 年 7 月 27 日，反馈了 Nautilus 在企图通过点击打开文件夹时崩溃的缺陷。报告还包含了堆栈信息。该报告共有 41 条报告与其重复。

[27] R 表示报告者行为，I 表示 Intereport 行为

表 6.6 缺陷报告 460835 的信息

ID	460835
产品	Nautilus
概要	Crash in Open Folder: In the file browser when…
详细描述	Version: 2.18.3 What were you doing when the application crashed? In the file browser when i clicked on some folder to open it, the file browser is crashed. Distribution: Fedora release 7 (Moonshine) Gnome Release: 2.18.3 2007-07-02 (Red Hat, Inc) BugBuddy Version: 2.18.0 System: Linux 2.6.22.1-27.fc7 #1 SMP Tue Jul 17 17:13:26 EDT 2007 i686 X Vendor: The X.Org Foundation X Vendor Release: 10300000 Selinux: Permissive Accessibility: Disabled GTK+ Theme: Clearlooks Icon Theme: Fedora

该报告包含的文字信息极为有限，实验将其堆栈信息提供 Intereport。

表 6.7 以缺陷报告 460835 为原型的交互式报告过程

序号	行为（R/I[28]）	详细信息
0	R 报告初始信息	概要：Crash in Open Folder 详细描述：The file browser is crashed.
1	I 完善建议	close
	R 否定	否定 close
2	I 完善建议	system
	R 补充建议	System: Linux 2.6.22.1-27.fc7
3	I 发现重复报告（ID）	487185、518390、483812、468684

在工具发现的重复报告中，487185、518390、468684 为真正的重复报告。

[28] R 表示报告者行为，I 表示 Intereport 行为

7 结束语

本章在 7.1 小节中总结本文工作，在 7.2 小节中对未来工作进行展望并结束全文。

7.1 工作总结

及时、高效的缺陷收集和处理机制对开发高质量的软件具有重要意义。由于当前的开源项目中有大量缺乏经验的报告者，项目的缺陷追踪系统中存在相当数量的低质量报告。其中，信息缺失和重复报告是报告质量低下的主要原因。这些报告不仅消耗了项目成员大量的时间精力，影响了缺陷的修复，还可能危及软件质量。为了提高缺陷报告质量，本文分析了现有的缺陷处理流程和低质量报告的特点，设计实现了基于相似检测的交互式报告工具。工具通过挖掘项目的历史报告信息，以交互的方式，有针对性地指导报告者完成报告的创建和提交，从而帮助报告者避免信息缺失报告及重复报告的提交，提高报告质量。

本文的主要贡献有：1、提出了一个基于相似检测的交互式报告完善方法。与现有报告方式不同，该方法以交互的方式协助报告者逐步完善报告信息并发现重复报告（如果存在重复报告）。在每一轮交互中，方法使用自然语言相似度及堆栈相似度检测项目中的相似报告及重复报告，根据相似度的分布判断报告的信息是否缺失，并利用最大信息增益原理分析相似报告，进而生成有针对性建议，知道报告者完善报告。2、基于上述设计，使用 MongoDB、Flask 及 Bootstrap 实现了基于 B/S 架构的交互式缺陷报告工具 Intereport。在实验评估中，本文从 Gnome 社区中随机抽取了 60 条报告，使用 Intereport 对其进行相似检测。结果显示，随着报告信息的不断完善，相似检测逐步精确。Intereport 检测出的重复报告列表（最大长度为 7）中，有 50%～60%的列表包含了真正的重复报告。此外,本文还提供 3 个案例，展示了报告者如何使用 Intereport 逐步完善报告的过程。

7.2 工作展望

本文还存在一些有待完善的部分，可在下一步的工作中进行改进。

1、本文所使用的停用词库为 NTLK 提供的停用词集合。然而在缺陷报告这种特定的语境下，可以考虑针对特定的产品完善停用词集合，从而提高相似检测准效果，也能更好地避免没有太大意义的单词成为关键词。

2、可以根据社区提供的“报告提交指南”，通过分析新报告中的信息给予报告者额外的提示。例如，Mozilla 的“报告提交指南”中提示，当缺陷导致了程序崩溃时，应该如何进行堆栈信息获取，报告中至少应当什么样的信息。如果工具能够通过检测报告是否反映了程序崩溃来给予这些基本的建议，那么缺乏经验的报告者可以更好地完善信息。

3、将工具与缺陷追踪系统结合。本文目前实现的 Intereport 仅从缺陷追踪系统定期获取数据并将最终报告提交至缺陷追踪系统。为了让报告者及社区更方便地完成报告提交、检索、管理工作，本文的交互式报告工具应整合到缺陷追踪系统中。

参考文献

[1] Mockus, Audris, Roy T. Fielding, and James D. Herbsleb. "Two case studies of open source software development: Apache and Mozilla." ACM Transactions on Software Engineering and Methodology (TOSEM) 11.3 (2002): 309-346.

[2] Zhou, Minghui, and Audris Mockus. "What make long term contributors: Willingness and opportunity in OSS community." Proceedings of the 2012 International Conference on Software Engineering. IEEE Press, 2012.

[3] Just, Sascha, Rahul Premraj, and Thomas Zimmermann. "Towards the next generation of bug tracking systems." Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on. IEEE, 2008.

[4] Wang, Xiaoyin, et al. "An approach to detecting duplicate bug reports using natural language and execution information." Proceedings of the 30th international conference on Software engineering. ACM, 2008.

[5] Sandusky, Robert J., and Les Gasser. "Negotiation and the coordination of information and activity in distributed software problem management." Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work. ACM, 2005.

[6] Hiew, Lyndon. "Assisted detection of duplicate bug reports." Diss. The University Of British Columbia, 2006.

[7] Runeson, Per, Magnus Alexandersson, and Oskar Nyholm. "Detection of duplicate defect reports using natural language processing." Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, 2007.

[8] Ko, Andrew J., Brad A. Myers, and Duen Horng Chau. "A linguistic analysis of how people describe software problems." Visual Languages and Human-Centric

Computing, 2006. VL/HCC 2006. IEEE Symposium on. IEEE, 2006.

[9] Audris Mockus, Amassing and indexing a large sample of version control systems: Towards the census of public source code history, msr, pp.11-20, 2009 6th IEEE International Working Conference on Mining Software Repositories, 2009

[10] Xie, Jialiang, Minghui Zhou, and Audris Mockus. "Impact of Triage: a Study of Mozilla and Gnome." Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on. IEEE, 2013.

[11] Jalbert, Nicholas, and Westley Weimer. "Automated duplicate detection for bug tracking systems." Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, 2008.

[12] Sureka, Ashish, and Pankaj Jalote. "Detecting duplicate bug report using character n-gram-based features." Software Engineering Conference (APSEC), 2010 17th Asia Pacific. IEEE, 2010.

[13] Sun, Chengnian, et al. "A discriminative model approach for accurate duplicate bug report retrieval." Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010.

[14] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. Vol. 1. Cambridge: Cambridge university press, 2008.

[15] Andre Klapper, "Finding Duplicates." https://wiki.gnome.org/Bugsquad/TriageGuide/FindingDuplicates

[16] Mozilla's "Bugzilla User's Guide" http://www.bugzilla.org/docs/4.2/en/html/using.html

[17] Gnome's "Bugzilla User's Guide" http://www.bugzilla.org/docs/3.4/en/html/lifecycle.html

[18] MongoDB, https://www.mongodb.org/

[19] NoSQL Databases Explained, http://www.mongodb.com/nosql-explained

[20] Python, https://www.python.org/

[21] NLTK, http://www.nltk.org/

[22] de Hoon, Michiel, Seiya Imoto, and Satoru Miyano. "The C clustering library." Institute of Medical Science, Human Genome Center, University of Tokyo (2003).

致 谢

时光如水，生命如歌，七年的燕园生活转瞬即逝。毕业在即，倍感不舍。这段岁月里，入学时的新奇，有刷夜时的困倦，有对不确定的恐惧，有柳暗花明时的欣喜。冬去春来，又是一个春天。北边的小路犹在，燕南的玉兰还香。在毕业论文即将完成之际，我真诚地感谢七年中关怀我、教育我、鼓励我、支持我的老师、同学、家人和朋友。

首先要感谢杨芙清院士、梅宏院士和谢冰老师所领导的北京大学软件工程研究所为我们提供丰富的教育资源和教育环境。这里不仅有良好的环境、齐全的设施，还有浓厚的学术讨论氛围，使我不仅能将自己的科研想法付诸实践，还能在与老师、同学的探讨中感受科研的乐趣。

感谢我的导师周明辉副教授，感谢她三年来对我在学习上、工作上、生活上的悉心指导。周老师十分注重与学生的交流。即便在百忙之中，她仍然经常与我讨论，无论是分析思路、解决方案还是论文撰写，她都给予我及时细致的指导和帮助。在与周老师交流的过程中，我深深地被她对待学术严谨求实的态度所感染。周老师特别注重对学生独立思考能力的培养。我不仅学会了如何解决问题，更重要的是，还学会了如何提出问题、分析问题、表达问题。这帮助我在面对新问题时保持清醒的头脑，让我一生受用。

感谢 Audris，感谢他给予我鼓励和支持。他给予的建议一针见血，让我少走了许多弯路。他对学术一丝不苟的态度让我受益匪浅。

感谢数据分析小组的同学们。感谢马秀娟师姐对我的指导和帮助。感谢郑淇沐，他为本工作做出了贡献，没有他的努力和帮助我无法完成我的工作。感谢已经毕业的尤朝师兄、王梓又师兄、邓飞师兄、大师兄林洪武、陈峰宏师兄和杨果师兄，三年来你们不仅在实验室工作上帮助、支持我，还时常在生活上关心我。感谢朱家鑫、廖钧、Bilal、郭颖、张飞雪、李家耀、林泽燕、涂菲菲对我的工作提出的宝贵意见，和你们在一起工作我感到十分开心。另外，还特别感谢叶治毅同学。他在我准备答

辩幻灯片期间，给我提供了专业级的指导。

最后，感谢我的父亲谢恩、母亲李辉，感谢我的女朋友陈璐瑶，他们对我的帮助和理解使我能更好地投入到学习和科研中。他们的支持和鼓励永远是我前进的最大动力。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明： 所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名： 日期： 年 月 日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名： 导师签名：

日期： 年 月 日