



同濟大學

TONGJI UNIVERSITY

**硕士学位论文**

(专业学位)

**面向云开发环境下代码推荐的源代码实  
时分析系统**

姓 名：秦乙丹

学 号：1641479

所在院系：软件学院

专业学位名称：

专业领域：

指导教师：刘琴

副指导教师：

二〇一九年五月



同濟大學  
TONGJI UNIVERSITY

A dissertation submitted to  
Tongji University in conformity with the requirements for  
the degree of Master

## **Real-time Analysis System of Source Code for Code Recommendation in Cloud IDE**

Candidate: Qin Yidan

Student Number: 1641479

School/Department:

Discipline:

Major:

Supervisor:

May, 2019

中文题目
姓名
同济大学

5cm 左右

5cm 左右

# 学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年 月 日

## 同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，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，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年      月      日

## 摘要

云计算概念已经兴起多年，在这期间各种新技术层出不穷。IaaS, PaaS, SaaS 等接踵而至，借此机会，许多本地的功能都有了云端的产品。顺应此潮流，本地的集成开发环境（IDE）也出现了对应的云产品，但是使用的人数并不多。云端集成开发环境可以使开发者减少在本地配置环境所消耗的时间，更快捷地进入开发过程；并且可以不受物理硬件位置所限，随时随地进行开发。但是目前云端开发环境还不够智能，这也可能是云端开发环境不受欢迎的原因。为此，整个课题组围绕智能化云端开发环境，以 Eclipse Che 为实例展开研究。本文承担的是将云端开发环境与其他课题组的代码推荐算法服务连接起来，以及检测收集开发过程中生产率有关的数据的任务。本文亦负责提供一个插件管理中心，将其他组可能开发的插件汇总起来。为此本文设计了对开发者 Eclipse Che 上正在编写的源代码进行分析并抽取特征的解决方案、对开发者开发源代码时的生产率进行实时统计的解决方案以及对插件进行管理的解决方案。并将这三套方案以插件的方式实现出来并部署到 Eclipse Che 产品上。

**关键词：**代码特征提取，软件生产率，云端集成开发环境，Eclipse Che

## ABSTRACT

The concept of cloud computing has been on the rise for many years, during which various new technologies emerged. IaaS, PaaS, SaaS, etc. came one after another, and many local features have cloud products. In line with this trend, the local integrated development environment (IDE) has also appeared corresponding cloud products, but the number of people using it is not much. The cloud-IDE enables developers to reduce the time spent on configuring local environment and enter the development process more quickly. It also makes developing anywhere, anytime, without being limited by physical hardware location. However, the current cloud -IDE is not smart enough, which may be the reason why the cloud-IDE is not popular. Therefore, the entire research group focused on the establishment of intelligent cloud-IDE, with Eclipse Che as an example. This paper undertakes the task of connecting the cloud development environment with the code recommendation algorithm services of other research groups and detecting the productivity-related data during the development process. This paper is also responsible for providing a plugin management center that aggregates plugins that may be developed by other groups. To this end, this paper designs a solution for analyzing and extracting the source code being written by a developer on Eclipse Che, a solution for recording the productivity of the developer when developing the source code, and a management solution for the plugins. The three sets of solutions were implemented as plugins and deployed to the Eclipse Che product.

**Key Words:** code feature extraction, software productivity, cloud IDE, Eclipse Che

# 目录

摘要 .....	I
目录 .....	III
第 1 章 引言 .....	6
1.1 研究背景 .....	6
1.1.1 代码推荐背景.....	6
1.1.2 生产率相关背景.....	8
1.2 研究目标及意义 .....	9
1.3 研究对象 .....	9
1.4 本文组织结构 .....	10
第 2 章 相关技术 .....	11
2.1 Eclipse Che 历史及现状 .....	11
2.2 Eclipse Che 的相关技术变更历史.....	11
2.2.1 部署运行方式的变更 .....	11
2.2.2 Eclipse Che 插件开发的技术变更.....	12
2.3 Eclipse Che 开源项目结构简介.....	12
2.2.1 dashboard.....	13
2.2.2 plugins.....	13
2.2.3 assembly.....	13
2.2.4 其他.....	13
2.4 与 Eclipse Che 有关的其它技术.....	14
2.4.1 Docker.....	14
2.4.2 Maven.....	14
2.4.3 GWT.....	14
2.5 本章小结 .....	14
第 3 章 系统分析 .....	15
3.1 总体需求 .....	15
3.2 功能性需求分析 .....	15
3.2.1 角色定义.....	15
3.2.2 源代码实时分析模块的用例设计.....	16
3.2.3 生产率统计模块的用例设计.....	16
3.2.4 智能化插件管理中心模块的用例设计.....	17
3.3 非功能性需求分析 .....	19
3.3.1 效率需求.....	19
3.3.2 稳定性需求.....	20



3.3.3 可扩充性与可重用性.....	20
3.4 设计约束.....	20
3.5 本章小结.....	20
第4章 系统设计.....	21
4.1 系统架构设计.....	21
4.2 源代码实时分析模块设计.....	23
4.2.1 源代码实时分析模块 IDE 子模块设计.....	23
4.2.2 源代码实时分析模块 WsAgent 子模块设计.....	24
4.3 生产率统计模块设计.....	25
4.3.1 生产率统计模块 IDE 子模块设计.....	25
4.3.2 生产率统计模块 WsAgent 子模块设计.....	26
4.4 智能化插件管理中心模块设计.....	26
4.4.1 智能化插件管理中心模块 IDE 子模块设计.....	26
4.4.2 智能化插件管理中心模块 WsAgent 子模块设计.....	27
4.5 本章小结.....	28
第5章 系统实现.....	29
5.1 开发环境配置.....	29
5.1.1 相关软件安装.....	29
5.1.2 项目配置.....	29
5.2.3 运行参数配置.....	34
5.2 IDE 层常用功能的实现.....	34
5.2.1 类的注册与绑定.....	34
5.2.2 输入缓存实现.....	35
5.2.3 界面实现.....	36
5.3 源代码实时分析模块实现.....	37
5.3.1 源代码实时分析 IDE 子模块实现.....	37
5.3.2 源代码实时分析 Workspace Agent 层实现.....	38
5.3.3 推荐结果显示实现.....	39
5.4 生产率统计模块实现.....	39
5.4.1 文件类型定义.....	39
5.4.2 功能实现.....	40
5.4.3 心跳机制.....	41
5.5 智能化插件管理中心模块实现.....	41
5.5.1 智能化插件管理中心模块 IDE 子模块实现.....	41
5.5.2 WsAgent 层实现.....	42
5.6 本章小结.....	43
第6章 测试.....	44
6.1 源代码实时分析模块测试用例.....	44

6.2 生产率统计模块测试用例 .....	44
6.3 智能化插件管理中心.....	44
6.4 本章小结 .....	44
第7章 结论与展望 .....	45
7.1 结论 .....	45
7.2 展望 .....	45
致谢 .....	46
参考文献 .....	47
个人简历、在读期间发表的学术论文与研究成果 .....	49

## 第 1 章 引言

### 1.1 研究背景

#### 1.1.1 代码推荐背景

随着开源代码库的日渐丰富和各商业机构提供的 API 和框架的日渐完善,越来越多的程序开发者趋向于调用 API、使用框架或复用代码来实现他们想要的功能。而每个 API 库通常包括数百个类,每个类又包括数十个方法。开发者通常要花费大量时间来学习使用 API 中的方法,并以特定的序列调用这些方法来完成一些任务<sup>[1]</sup>。这些时间包括查找文档、阅读文档和示例代码等,即使可能只需要少数方法,也可能需要大量的时间从所有文档中查找出有关的部分。一些新手开发者有时还会遇到不知道一些 API 的存在的问题。开发者在使用框架时也会遇到与 API 类似的问题。而开发者在复用代码片段时,更容易遇到文档不完整的问题,从而将大量时间花费在阅读和理解代码上。综上所述,如何让程序开发者快速地使用 API,框架和代码片段来开发出可以实现目标功能的代码成为急需解决的问题。

为了解决上述问题,各种代码推荐系统逐渐出现。代码推荐系统<sup>[2]</sup>根据开发者的输入为开发者推荐特定的 API 序列或代码片段。目前市场上还没有占据主导地位的代码推荐产品,各种代码推荐系统都在理论探索中。大多数的各种代码推荐系统的最终输出基本相同,均为 API 序列,代码片段或类似结构。然而各种系统的输入却大不相同,有的以开发者的自然语言查询语句作为输入,有的以代码中的有效关键词作为输入,还有的以测试用例作为输入,之后各个推荐算法针对输入的内容进行分析。并且大部分的系统并没有针对实时性进行设计,也没有在云端集成开发环境下的成功实现。

按照代码搜索方法的输入类型划分,相关文献可分为以自由文本作为输入的研究、以 API 作为输入的研究、其它形式作为输入的研究。而推荐的内容要么是相关的代码片段,要么单个 API,或多个 API 的调用序列。这里,把推荐代码的研究称为代码搜索,把推荐 API 及其序列的研究称为 API 推荐<sup>[3]</sup>。

在代码推荐中,有的以自由文本作为查询输入。Zhong 等人<sup>[4]</sup>提出了 MAPO 系统,通过提取代码片段的方法名和调用序列等特征,将代码片段聚类。对于聚类产生的每一集群,MAPO 采用了一个频率子序列挖掘工具来挖掘集群中代码

片段的使用模式,将相应模式中的代码调用顺序推荐给开发者。Zhong 等人认为在聚类中应该保留用户类的类名或用户方法的方法名,并将其作为特征来处理,以免在挖掘过程中重要的信息被遗漏掉。并且在处理过程中,MAPO 只考虑了第三方 API 的调用顺序。

类似的,Bjracharya 等人<sup>[5]</sup>综合 API 使用模式的相似性等一系列特征进行代码片段的推荐,他们希望以此来解决源代码文件中单词的稀缺性以此来检索更多的代码片段。在 Bjracharya 等人的研究中,其关注了代码片段中的完全限定名称(Fully Qualified Names),而没有关注代码片段所在的用户类的名称。

McMillan 等人<sup>[6]</sup>在信息检索的基础上,利用 PageRank 和传播模型推荐具有依赖关系的函数。Keivanloo 等人<sup>[7]</sup>利用向量空间模型、频繁项挖掘等技术为自由文本的查询进行代码推荐。

Raghothaman 等人<sup>[8]</sup>提出了一种新的方法 SWIM,该方法首先从用户点击数据中找到与用户输入的自由文本查询相关的 APIs,同时从 GitHub 的代码中抽取出结构化的 API 调用序列,通过匹配找到 APIs 相应的调用序列,进而产生合成的代码片段。以上提到的方法都是基于信息检索的技术。

不同于这些方法,Jiang 等人于 2016 年尝试使用监督学习的方法为自由文本的查询提供相关的代码片段<sup>[9]</sup>。其以 Java 语言为主要研究对象,从文本,主题结构三个方面分析代码特征。结构方面提取了代码片段的长度。主题方面提取了候选代码片段的主题和查询的主题的相似性。文本方面主要通过代码片段的一些属性与查询的文本相似性来描述特征。代码的属性包括:代码的文字内容(代码的),代码片段的完整标题(代码片段所在的包名,类名和方法名),代码片段的名称(仅包含代码片段所在的方法名),与候选代码片段同一 Java 文件下的兄弟方法名称,引用部分中 Android 包下的引入语句(该文章在测试时使用的是 Android 项目)以及引用部分中 Java 标准库下的引用语句。

另外,Li 等人<sup>[10]</sup>提出了基于关系的代码搜索方法,在该方法中,代码片段中的 API 使用模式和用户的查询分别被表示为方法之间的关系调用图和行为关系图。这样,代码搜索问题就被转换为了图搜索问题。

在以信息检索为主要匹配方法的代码搜索研究中,常常因为查询词过短以及查询和匹配的代码片段使用不同的语言而导致词项失配的问题。为了解决这一问题,研究者提出了一系列的基于查询扩展的代码搜索方法。例如,Wang 等人<sup>[11]</sup>利用半自动的相关反馈方法为代码搜索引擎推荐的代码段列表进行评价,而后再根据评价结果进行重排,进而得到了更好的推荐结果。另外,研究者也考虑从 WordNet 中抽取出近义词来扩展查询<sup>[12]</sup>或测试用例<sup>[13]</sup>。为了获取与软件开发和编程相关的近义词,研究者尝试从 StackOverflow 的编程问答对中抽取出相关的近义

词<sup>[14]</sup>或 API<sup>[15]</sup>来扩展原始查询,并取得了好的效果。

在以 API 作为查询的代码搜索研究中,Subramanian 等人<sup>[16]</sup>提出了 Baker 方法,其利用代码片段来丰富 API 文档;Moreno 等人<sup>[17]</sup>提出了 MUSE 方法,该方法利用静态切片和代码克隆技术,从一系列相似的代码片段中总结出一套调用某个 API 的步骤并呈现给开发者。

在以其他类型内容作为查询的代码搜索研究中,Lemos 等人<sup>[18]</sup>提出以测试用例作为输入进行搜索,其目的是把开发者的搜索需求尽可能地在测试用例中描述清楚;Stolee 等人<sup>[19]</sup>以开发者期待的某段代码的输入和输出状态的配对作为输入来推荐代码片段;Inoue 等人<sup>[20]</sup>提出 IchiTracker,其以开发者当前编写的代码片段作为输入,推荐与输入相符的开发者可能需要的其他代码。

除了推荐代码片段,也有较多学者关注 APIs 的推荐。例如,Thung 等人<sup>[21]</sup>提出从版本控制系统的代码历史变更信息中抽取知识,从而推荐相关的 API, Gu 等人<sup>[21]</sup>提出了一种基于深度学习的 API 序列推荐方法——DeepAPI。该方法把 API 推荐问题转换成监督学习问题,把用户输入的自由文本查询转换成相关的 API 序列。类似地,Niu 等人<sup>[23]</sup>提出了使用监督学习的方法,在两阶段的基础上,借助于多种特征进行 API 的推荐。另外,Rahman 等人<sup>[24]</sup>尝试从 StackOverflow 的问答对中抽取出与开发者的自由文本的查询相关的 API。

### 1.1.2 生产率相关背景

随着软件开发进入 DevOps 时代<sup>[25][26][27][28]</sup>,将 DevOps 多源、异构、多周期数据分析用于提高项目生产率是近几年软件工程领域的研究热点之一<sup>[29]</sup>。基于 DevOps 开发模式的应用软件需要同时在保证项目质量的前提下,灵活快速的针对用户提出的新的需求特性进行即时发布和交付,在这一过程中如何能够尽可能缩短发布周期(即提高有效生产率)是至关重要的,往往是考验初创企业能否存活的严峻问题。在现代软件开发中呈现的开源与开发结合、多周期迭代、项目涉众泛化、开发环境直接依托互联网等软件开发环境和形态的变化中,如何动态提取项目的阶段性特征信息、了解生产率状况,并精准预分析和配置下一代周期的资源等,是新型软件项目能够成功的关键问题。

针对这个问题,近几年来将基于 DevOps 开发模型的全面、系统的软件项目数据收集和分析用于生产进度监测和资源配置的理念和方法引起了学术界和产业界的日益重视<sup>[25][26][27][28]</sup>。然而现有的软件项目生产率的度量和建模方法多围绕着产品特征数据和基于传统生产模型的过程特征数据构建,极少覆盖相关的项目资源特征数据,如专业用户(如开发、测试人员)和普通用户交互行为数据等,更难以满足基于 DevOps 软件项目的多源、异构、多周期数据的汇聚分析以及其

资源配监测的客观需求。为了满足 DevOps 下软件检测的

在 DevOps 模式下软件的，软件的产出和成本的定义的到了扩展，软件的产出可以探索式扩展为产出物包含了自研源代码、复用（开源）源代码、各类自动化脚本、报告缺陷、修复缺陷等。所消耗成本除了用于开发可执行源代码所消耗的人月之外，还包括花费在开发环境搭建和部署、自动化测试代码的编写、编译与部署脚本的编写、数据爬取脚本的编写、其他脚本的编写、自动化测试的执行和缺陷报告的分析等方面的人月。

## 1.2 研究目标及意义

目标一是提出了一种实时的源代码分析的解决方案，从云开发环境的编辑器端实时获取代码，在没有语法树支持、源代码包含错误的情况下，尽可能多地抽源代码的特征，包括上述提到的自由文本，API 调用序列等输入到外部的代码推荐服务器中。

目标二是实时分析开发者输入的代码，将其与生产率所定义的产出相对应，统计开发者在不同产出的项目上所花费时间。

目标三是提出一个管理项目组中各个插件的解决方案，并将上述三种方案以 Eclipse Che 的方式实现出来。

本文并不指定具体的额推荐算法，本文所属的上级课的其他小组将负责代码推荐算法的具体实现，他们将以本课题的产出为模板，进一步开发自己的展示平台，未来多个插件在调试后将整合到统一平台上，为了方便管理而设置管理中心，这就是目标三的由来。

本文是一个开创性研究，理论上，生产率的定义还在探究与调整中，本文提出的统计生产率的解决方案提供的数据将有助于生产率的进一步探索。虽然有很多对于代码推荐的研究，但是很少有在云端集成开发环境上实现的，因此本文所的工作在将云开发环境变得智能化方面具有探索性意义。于此同时，本文所开发的插件还会为其他课题组提供技术示范，为其摸清开发过程中可能遇到的障碍，属于技术上的探索。

## 1.3 研究对象

本文研究对象为在云端集成开发环境中，抽取开发者正在编写的代码特征、在开发者编写代码过程中统计生产率，以及对个部分最终展示插件进行管理的解决方案。研究对象的实现为云端集成开发环境 Eclipse Che 上的插件。

## 1.4 本文组织结构

本文第一章中介绍代码推荐的相关研究背景和生产率的相关概念。第二章介绍了 Eclipse Che 及其相关技术变迁。第三到第六章详细介绍了在将技术与实践结合的理念下，本系统从需求制定，设计，实现，测试的全过程。第七章对整个实现进行了总结，并提出对未来的看法。

## 第 2 章 相关技术

### 2.1 Eclipse Che 历史及现状

Eclipse Che 是一个基于 Docker 的云端集成开发环境。第一版发布于 2014 年，其名来源与它的主要开发工作进行的所在地——乌克兰的切尔卡瑟（Cherkasy）。Eclipse Che 启动后可以通过浏览器进行访问，在服务其上搭建多个 Workspace。可以为每个 Workspace 配置不同的语言支持，框架支持与开发工具支持。在每个 Workspace 中可以创作多个项目（Project），对于 Workspace 可以进行代码编写，版本控制，编译部署等操作。

经过多年的发展，Eclipse Che 已经发展出了独立的开发产品 Codenvy，并且支持多种开发语言、框架和开发部署、依赖管理以及版本控制工具。其中开发语言包括 C、C++、C#、F#、GO、Java、JavaScript、PHP、Python、Ruby、SQL 以及 TypeScript 等；框架包括 AngularJS、Docker、.Net 2.0、Kubernetes、OpenShift 以及 Yeoman 等；开发部署、依赖管理以及版本控制工具包括：Ant、Bower、Grunt、Gulp、Maven、Npm、Git、Orion、SSH 以及 Subversion 等。<sup>[30]</sup>

Eclipse Che 在 github 上发布了开源项目，最早的标签是 2016 年 4.X.X 版本发布标签，之后一直以周为周期快速迭代，目前已经迭代到 7.X.X 版本的 beta 版，同时 6.X.X 版本也在更新维护，4.X.X，5.X.X 版本基本不再更新维护，支持文档已删除。在进行本研究时，兼顾产品更新支持与稳定性，选择 6.16.0 版本作为研究对象，定义 6.16.0 为当前版本。同时在下文中，我通过回忆预研究阶段的内容与查找 github 上的历史记录等方式，列出了一些过去的功能，目的是在以后的研究中，若在网上搜到旧版本的相关资料时，可以快速绕开，避免不必要的麻烦。

### 2.2 Eclipse Che 的相关技术变更历史

#### 2.2.1 部署运行方式的变更

Eclipse Che 的部署方式也随版本更新而更新。本人在其 4.X.X 版本时进行相关预研究，版本四支持以 zip 压缩包的形式进行发布，需要安装 Docker，解压



之后通过脚本将相关组件挂载在到 Docker 上运行，从而启动产品实例。在版本四之后，随着 Eclipse Che 的 Docker images 的发布，zip 压缩包的发布方法不再受到官方的支持，从而变为用户个性化定制的开发方法。即用户对产品进行部分修改之后将修改过的组件挂载到 Docker 上进行运行或调试。

而在运行 5.X.X 版本以及之后的原生的 Eclipse Che 产品实例时，需要通过命令行启动 Docker 程序，并从 Docker 镜像仓库中查找并拉取主要镜像，主要镜像在运行过程中会继续拉取其他需要的镜像，之后启动 Eclipse Che 实例。其中 Eclipse Che 的主要镜像既可以使用默认的最新版本，也可以指定某一版本。因此，用户在发布的时候可以考虑发布带有用户定制版本号的 Docker 镜像。此外，还可以将编译打包好的插件部署到 Maven 远程仓库中，实现单个插件的发布，或者将组件（assembly）打包好之后进行发布。6.X.X 目前也在继续支持这种运行方式和发布方式。

除此之外，从 5.X.X 版本开始，Eclipse Che 也开始支持多人模式。并且还加入了不经网页端，而通过 Restful API 对 Workspace 进行直接操作的功能。目前，Che 的 Workspace 和用户相关配置通过 Java 持久化 API（Java Persistence API，JPA）存在数据库中，项目代码中有关于 MySQL 和 PostgreSQL 的支持和相关测试代码，但并没有在实际产品中使用。

## 2.2.2 Eclipse Che 插件开发的技术变更

在 4.X.X 版本时期，项目本身并没有提供特殊的插件开发方法，开发者可以通过直接在源代码项目中 plugins 模块下添加代码以及在适当的位置注册插件和扩展从而达到改进 Che 功能的目的。

在 5.X.X 版本时期，该项目通过名为 che-dev 的 Docker 镜像提供了 archetype 服务。即可以通过运行该镜像生成一些简单的插件模板，通过修改之后即可进行编译，加入现有的组件中生成新的产品实例。但是这种开发方法在 6.X.X 版本中不再继续支持。<sup>[31]</sup>

在 6.X.X 版本中，Eclipse Che 本身提供了带有 Eclipse Che 开发环境的 Workspace 模板，并提供了一些 Eclipse Che 插件的项目模板，就像上一版本中的 archetype 一样，不同的是这次可以在 Eclipse Che 产品中进行，即 Eclipse Che 可以编译自身，并部署调试运行。

## 2.3 Eclipse Che 开源项目结构简介

Eclipse 使用 Maven 进行项目依赖与编译管理，Che 项目下也分为多个

Maven 模块，下文将介绍几个主要的 Maven 模块。

### 2.2.1 dashboard

负责 Eclipse Che 实例启动时的起始控制界面，显示关于添加，删除，配置 Workspace 的相关选项。由于该模块需要额外的环境配置，一般建议跳过该模块的编译，直接从 Maven 模块中引用已经编译好的代码库。

### 2.2.2 plugins

该模块包含了 Eclipse Che 的主要插件，一个插件中包括前端扩展模块（ide），后端扩展模块（server），前后端共享扩展模块（share）中的一个或多个。各个插件在上层 pom.xml 中的<dependencyManagement>标签下注册，并在对应的 assembly 中注册，等待被编译到产品中。

### 2.2.3 assembly

assembly 是 Eclipse 项目下负责组织编译的模块，通过 assembly 模块下的 assembly-main 模块可以生成完整的 Eclipse Che 产品文件，而具体各个部分的编译会依赖其他 assembly 来进行：如包含前端代码和 IDE 扩展，负责界面显示的 assembly-ide;包含各种以 jar 包形式存在的，对 Workspace 进行操作的扩展的 assembly-wsagent-war; 用于将各个插件打包进 Tomcat 并添加配置文件的 assembly-wsagent-server ; 以及包含核心代码和 Workspace API 的 assembly-wsmaster-war。通过上述模块，最后在 assembly-main 的 target 下面生成了可以被挂载到 Docker 镜像下进行调试和运行的产品实例。在官方文档中，经常以 Workspace Server, Workspace Anget 和 IDE 三个组件来描述 Eclipse Che, 本文在下文中也将使用这种描述方法。

### 2.2.4 其他

ide 模块主要包含前端 ide 的核心代码，包括 ide 界面各部分（如菜单，编辑器，控制台等）的基本定义等。Wsagent 模块提供各种通过各个 workspace 独立的端口可以进行访问的 API。Wsmaster 模块提供通过项目启动的主要端口可以访问的 API。core 模块提供包括数据库访问，API 部署，基础的事件定义和 Exception 定义等在内的基础代码。agent 模块包括各个部署在 Workspace Server 中的扩展，主要是各个语言的 installer，用于在 Workspace 创建时向 Workspace Agent 中添加运行时环境。主要用于添加各种语言的扩展，因此不太经常被使用。

## 2.4 与 Eclipse Che 有关的其它技术

### 2.4.1 Docker

Docker 是一款开源应用容器引擎，为开发人员提供用于构建，发布和运行的分布式应用平台<sup>[32]</sup>。Docker 主要包括 Docker 镜像，Docker 容器和 Docker 仓库。Docker 镜像相当与一个只读的程序模板，具有完整的运行环境和程序包。Docker 容器相当与一个镜像的实例或者镜像上的可读写层，依赖于镜像，可以存储数据，可以随时启动，停止和移除。Docker 仓库用于存储镜像，分为共有和私有。<sup>[33]</sup>Eclipse Che 的默认 Workspace 功能模块会以 Docker 镜像的形式存储在 Docker 仓库中，而 Workspace 实例在运行时会以 Docker 容器的形式运行，停止后会被写入挂载的数据文件夹中。

### 2.4.2 Maven

Maven 是 Apache 下一款基于项目对象模型 Project Object Model 的优秀的项目工程管理工作。能为开发者处理繁琐的代码清理，编译，打包和部署等工作。<sup>[34]</sup>Maven 通过插件来进行对项目测试操作，系统默认的插件包括 clean, compiler, deploy, install, verifier 等，用户还可以添加第三方插件在特定的阶段添加个性化定制的操作。<sup>[35]</sup>

### 2.4.3 GWT

GWT (Google Web Toolkit)，是一款由谷歌开发的前端开发框架。它允许开发者以 xml 的形式设计页面布局，以 Java 语言开发前端逻辑代码，并且不必为了开发网页端代码而去专门学习 XMLHttpRequest 或 JavaScript 相关知识。项目最终会被以 Javascript 的形式部署到网页端。<sup>[36]</sup>

## 2.5 本章小结

本章前几小结介绍了 Eclipse Che 的发展历史，技术变更和现行主要模块。主要目的是为了提示读者在搜集资料时不会被过时的信息所迷惑。同时为在后续版本若出现与现在版本不一样的行为时，及时识别出是版本问题并快速剔除过时内容提供帮助。之后用一节介绍了 Eclipse 所用的其他技术，作为对上文的补充。

## 第 3 章 系统分析

### 3.1 总体需求

本文总的需求是在云端集成开发环境 Eclipse Che 中分析开发者正在输入的源代码，为智能化开发环境的下一阶段分析提供输入。同时为了同级课题组所产生的功能模块提供统一的管理模块。

目前，本文所要对应的下一阶段的功能有两个，一个是代码推荐，一个是生产率的研究。因此本文主要的功能模块分为为代码推荐提供输入数据的源代码实时分析模块和为生产率研究提供输入的生产率统计模块。这两个模块属于并列关系。由于同课题下其他课题小组也会在 Eclipse Che 上集成他们的成果，会有和上述两模块并列的其他模块集成进来，需要一个管理模块来控制各模块的是否启用，以免造成各个模块争相抢占消息总线或结果显示等资源，因此提出智能化插件管理中心模块。智能化插件管理中心模块与其他所有模块为管理与被管理的关系。

### 3.2 功能性需求分析

#### 3.2.1 角色定义

由上文可知，本系统包含三个模块：源代码实时分析模块，生产率统计模块和智能化插件管理模块。源代码实时分析模块，为项目的开发者提供代码推荐过程中的文件分析和结果展示服务，需要外部的代码推荐算法服务器为其提供推荐服务。生产率统计模块为项目管理者提供 DevOps 的生产率计算服务，在显示结果时需要管理权限。智能化插件管理中心模块需要有权限的管理者来对包括源代码实时分析插件的所有智能化插件进行配置。

上述描述中，出现外部代码推荐系统，普通开发者，有查看软件生产率权限的项目管理者，可以配置智能化插件的项目管理者四个角色。Eclipse Che 本身没有很强的用户权限管理系统，其权限是以 Workspace 为单位，划分为两种角色：

(1) Workspace 的创建者，可以编辑 Workspace 相关配置 (2) Workspace 使用者，即普通开发者，没有配置权限。因此，本系统继承 Eclipse Che 的权限管理模式，以 Workspace 创建者作为有查看软件生产率权限的项目管理者和可以配置

智能化插件的项目管理者。综上，需求分析中将出现如下三个角色：

- 开发者：使用创建好的 Workspace 进行代码开发。
- Workspace 创建者：创建 Workspace，通过智能化插件管理系统配置智能化插件，查看软件生产率相关历史数据。
- 外部代码推荐系统：为本系统提供具体的代码推荐算法服务。

### 3.2.2 源代码实时分析模块的用例设计

源代码实时分析所需要完成的基本任务是在开发者编写代码时为用户提供适当的代码推荐。由于本源代码实时分析插件不涉及代码推荐算法本身，因此需要引入承载的具体的代码推荐算法的第三方代码推荐系统。源代码实时分析插件应完成的功能为能够根据用户的输入判触发代码推荐动作；能接信息当前代码 API 调用顺序、自由文本等信息，以及包括项目依赖库、项目结构、当前代码所处位置等在内的代码上下文信息，传递给代码推荐算法；能够进行结果展示；能将用户选择的结果插入到编辑器指定位置。用例图如图 3.1：

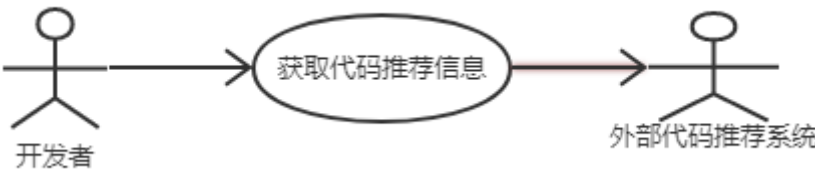


图 3.1 获取代码推荐信息用例图

表 3.2 开发者获取代码推荐信息用例

用例名称	开发者获取代码推荐信息
角色	开发者，外部代码推荐系统
前置条件	Workspace 创建者已经启用了源代码实时分析插件，开发者进入了这个 Workspace。
基本事件流	<ol style="list-style-type: none"><li>1. 开发者编写代码。</li><li>2. 源代码实时分析插件监听到触发条件，分析文件，并向外部推荐系统发相关信息以请求推荐。</li><li>3. 外部代码推荐系统返回推荐结果。</li><li>4. 源代码实时分析插件展示结果供开发者选择。</li><li>5. 开发者选取想要的推荐结果。</li><li>6. 智能化插件将推荐结果添加到代码中。</li></ol>
异常流	步骤 3 中若外部推荐系统无法返回推荐结果，则结束用例。
备选流	步骤 5 中若开发者不选择推荐结果，则结束用例。
后置条件	开发者继续编辑代码

### 3.2.3 生产率统计模块的用例设计

在本文之前的研究中，我们提到了代码生产率这一概念。传统的生产率指产出与成本的比值。之前的研究因为条件的限制，没有获得详细的代码生产率数据，只是通过对 Github Commit 的抓取，只是发现了生产率定义中的部分数值随着代码开发的生命周期进行周期性变化。因此本文中设置代码生产率统计插件，来收集生产率信息，为后续研究提供更详细的数据。在本文中，对于有些不太容易获取的属性，如报告缺陷、被修复的缺陷等，本文暂时忽略，而专注于研究以代码行数为度量的产出以及相关的成本。

该插件的工作主要是在用户开发时，由源代码实时分析系统对修改的源代码类型进行分析，对增量进行统计。在查询时，仅需要做简单的计算即可，因此功能比较简单，即显示生产率相关信息即可，相关信息包括生产率，以及各个类型的代码每天增量的历史记录等。当 Workspace 打开相关查看界面时展现给 Workspace 创建者，具体用例如图 3.3，表 3.4：

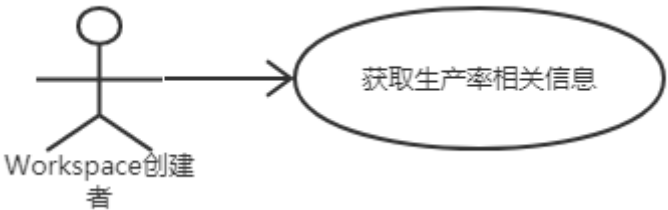


图 3.3 Workspace 创建者查询生产率相关信息用例图

表 3.4 Workspace 创建者查询生产率相关信息

用例名称	Workspace 创建者查询生产率相关信息
角色	Workspace 创建者
前置条件	Workspace 创建者进入了其创建的 Workspace。
基本事件流	1. Workspace 点击了生产率显示按钮。 2. 源代码实时分析系统分析代码生产率并展示。
异常流	当前项目没有创建代码，显示当前开发进度不足。
备选流	无
后置条件	无

3.2.4 智能化插件管理中心模块的用例设计

智能化插件管理中心，需要做到提供一个统一的面板，允许项目管理者，也就是 Workspace 创建者对所有智能化插件的开启或关闭进行逐一配置。一个 Workspace 在刚刚创建时，所有智能化开发相关的插件应该是关闭的。在保存配置时，页面上的结果显示窗口应该做出响应，被关闭插件对应的窗口应被隐藏，被开启的插件对应的窗口应该被显示。有些插件允许具体的配置智能化插件，在

上述面板中，提供按钮打开新的配置面板窗口供开发者对插件的详细信息进行配置，插件的具体配置的存储由插件自行决定。同时智能化插件管理系统的配置结果应该是持久的，在 Workspace 关闭并重新启动之后，应恢复 Workspace 之前的配置。若 Workspace 创建者和开发者同时在不同的浏览器中使用 Workspace，Workspace 创建者更新智能化插件的配置，则不要求该配置对开发者立即生效，因为开发者可能正在使用对应的插件，对于开发者来说，自己的窗口突然消失不是一个很好的用户体验。

综上，智能化插件管理系统应当事先如下功能：提供智能对所有化开发插件配置的开闭进行配置的统一管理界面。对于可以配置的智能化插件，提供配置界面的入口。可以持久化存储当智能化插件的开闭状态。部分用例图如下：

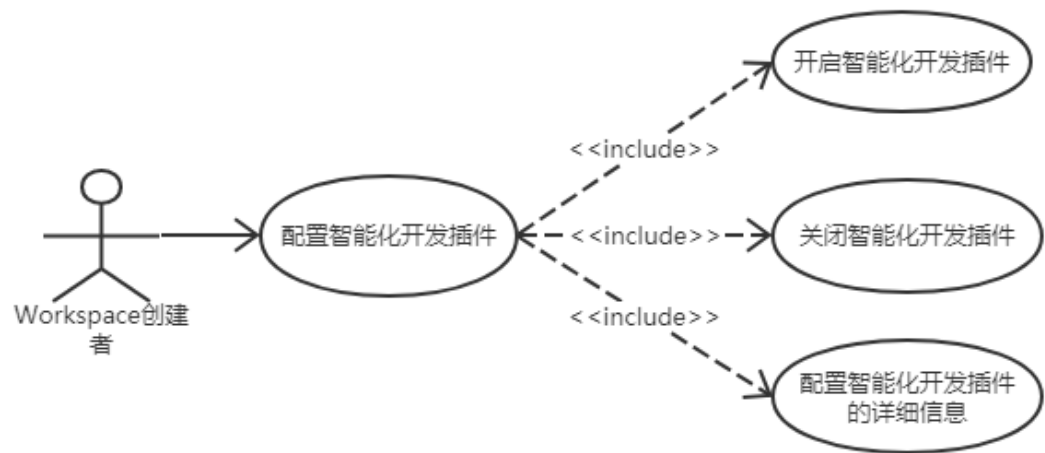


图 3.5 配置智能化插件用例

表 3.6 开启智能化插件用例

用例名称	开启智能化插件
角色	Workspace 创建者
前置条件	Workspace 创建者进入了其创建的 Workspace，一个智能化插件未被开启。
基本事件流	1. Workspace 创建者点击智能化插件管理中心按钮。 2. 智能化插件管理中心显示配置界面。 3. Workspace 创建者开启一个智能化插件并保存 4. 智能化插件管理中心启动对应智能化插件并展示智能化插件的展示界面相关信息。 5. Workspace 创建者关闭窗口。
异常流	无
备选流	备选流 1，步骤 3 中 Workspace 创建者不做任何修改并退出，或修改后点击取消。则直接结束用例。 备选流 2，步骤 3 中 Workspace 创建者做修改后不保存并退出。提示 Workspace 创建者是否保存。
后置条件	无

表 3.7 关闭智能化插件用例

用例名称	关闭智能化插件
角色	Workspace 创建者
前置条件	Workspace 创建者进入了其创建的 Workspace， 一个智能化插件已开启。
基本事件流	<ol style="list-style-type: none"> <li>1. Workspace 创建者点击智能化插件管理中心按钮。</li> <li>2. 智能化插件管理中心显示配置界面。</li> <li>3. Workspace 创建者关闭一个智能化插件并保存</li> <li>4. 智能化插件管理中心关闭对应智能化插件并隐藏智能化插件的结果展示界面。</li> <li>5. Workspace 创建者关闭窗口。</li> </ol>
异常流	无
备选流	<p>备选流 1，步骤 3 中 Workspace 创建者不做任何修改并退出，或修改后点击取消。则直接结束用例。</p> <p>备选流 2，步骤 3 中 Workspace 创建者做修改后不保存并退出。提示 Workspace 创建者是否保存。</p>
后置条件	无

表 3.8 配置智能化插件详细信息用例

用例名称	配置智能化插件详细信息
角色	Workspace 创建者
前置条件	Workspace 创建者进入了其创建的 Workspace， 一个智能化插件可以被配置。
基本事件流	<ol style="list-style-type: none"> <li>1. Workspace 创建者点击智能化插件管理中心按钮。</li> <li>2. 智能化插件管理中心显示配置界面，对于可以编辑详细信息的智能化插件提供编辑按钮。</li> <li>3. Workspace 创建者点击智能化插件的编辑按钮。</li> <li>4. 智能化插件管理中心从智能化插件处获取相关界面并展示。</li> <li>5. Workspace 创建者编辑详细配置并保存，点击详细配置界面的关闭按钮。</li> <li>6. 智能化插件保存相关配置，并关闭详细配置界面。</li> </ol>
异常流	无
备选流	<p>备选流 1，步骤 5 中 Workspace 创建者不做任何修改并退出，具体操作由具体的智能化插件决定。</p> <p>保存并退出。提示 Workspace 创建者是否保存。</p>
后置条件	Workspace 创建者回到智能化插件管理中心界面，并继续配置。

## 3.3 非功能性需求分析

### 3.3.1 效率需求

由于开发者在持续输入，若解析时间或显示时间过长，开发者的输入已经发



生变化,则会给开发者带来不友好的体验,因此,要在用户输入发生巨大变化前,展示结果。当代码已发生变化导致现有的推荐结果并不适合当前代码时,放弃推荐。

### 3.3.2 稳定性需求

由于互联网的链接存在不稳定性,本文需要考虑在网络链接不良时的状态。本文要保证在 Eclipse Che 实例可以使用时,尽可能的实现功能。具体来说在某个已经开启的智能化插件无法访问第三方推荐服务器时,整个 Eclipse Che 系统不可崩溃,开发者开发代码这一行为不应受到影响。可以在原来的结果展示框中提示相关报错,但报错不能过于频繁,不应反复引起用户注意。当用户代码语句输入不完整、输入错误或输入乱码时,可以不触发推荐,但不能导致系统崩溃或大量报错。

### 3.3.3 可扩充性与可重用性

由于部分依赖的项目仍在进行中,以及随着研究的深入,代码推荐所需要的特征可能会发生改变,因此本文需要为这些变化预留扩展空间。支持以后添加与不同的代码推荐算法或代码服务器的适配。为了给项目组中其他成员提供帮助,需要尽可能多的提供对资源的访问方法。

## 3.4 设计约束

根据上级项目的要求,本文所涉及的智能化插件管理系统和样例必须在 Eclipse Che 中实现。

为了应对多个插件同时开启的情况,不允许智能化插件在触发之后自动将推荐结果插入编辑器或以其他形式对代码编辑器中的代码自动修改。所有过程均要求手动触发。此举是为了防止多个插件同时向一处插入代码造成混乱。

## 3.5 本章小结

本章系统分析入手,通过分析系统上下文明确了系统职责,并通过系统职责确认了参与的角色。将系统分为源代码实时分析模块、生产率统计模块和智能化插件管理中心。对这三个插件进行用例设计并提出了功能性需求和非功能性需求。提出了其他设计约束条件。从而完成了对整个系统的分析。

## 第 4 章 系统设计

### 4.1 系统架构设计

根据上文，系统主要需要三个模块：源代码实时分析模块，生产率统计模块和智能化插件管理中心模块。下面要确定这三个功能模块在 Eclipse Che 架构中的位置。

Eclipse Che 是一个基于 Docker 的云端集成开发环境，包含 Workspace Server、Workspace Agent 和 IDE 三层组件，这三层组件也是 Eclipse 可以进行扩展的三个方向，下面介绍这三层组件的功能与关系。

当 Eclipse Che 启动时，启动 docker 容器 che-server 作为 Workspace Server。默认情况下暴露 docker 容器的 8080 端口，在没有其他配置或端口占用的情况下，该端口会映射到服务器的 8080 端口，改端口提供基础的创建、开启、停止、编辑、删除 Workspace 等服务，用户既可以通过 Web 浏览器访问，也可以通过 Restful API 进行访问。通常所说的访问 Workspace Server 即为访问该端口或访问 che-server 容器。访问 Workspace Server 可以访问得到 Workspace 的元数据，但不得到 Workspace 的内容数据。

通过以浏览器访问 Workspace Server，可以得到一个 dashboard 界面，用来对 Workspace 的元数据进行操作。在启动某个 Workspace 时，Workspace 会创建一个临时的 Docker 镜像，以 Workspace 的哈希值加上 dev-machine 命名，在这里称之为 dev-machine 镜像。生成镜像之后会启动对应的 dev-machine 容器。dev-machine 暴露包括 4401，8080 在内的一系列端口，这些端口会被映射到服务器（或者说宿主机）的高位端口上，通过访问这些端口可以访问 Workspace 本身的 API，或 language server 等其他服务。通过这些端口或访问 dev-machine 的容器被称之为访问 Workspace Agent。通过网页的 Dashboard 启动 Workspace 后，dashboard 会访问 Workspace Server 来获得 Workspace Agent 的端口，再将 Workspace 的端口通知 IDE 组件，IDE 组件初始化，并访问 Workspace Agent 获取相关数据。

Workspace 所包含的文件存储在 Workspace Server 的 /data/instance/data/workspaces 目录下，而 Workspace Server 启动时会将宿主机中用户指定目录挂载到的 /data 目录，因此 Workspace 的一系列源代码或项目文件实际是存储在宿主机上的。Workspace Agent 启动时，Workspace Server 下的 /data/instance/data/workspaces 中具体对应的目录会被挂载到 Workspace Agent 的

/project 目录下，因此 Workspace Agent 可以准确地访问到自身的项目文件目录。若 Workspace Server 要访问 Workspace Agent 的项目文件，要知道其 id 在进行访问。目前 Workspace Agent 的核心代码中有支持使用相对于 Workspace 根目录（即 docker 容器的/projects 目录）的相对路径对文件进行读写的 Java 代码级 API，相反 Workspace Server 则没有。基于上述情况，我们说 Workspace Agent 可以访问 Workspace 的文件目录，而 Workspace Server 不能。相比 Workspace Agent 而言，Workspace Server 基本上没有直接对文件的读写，其数据通过 Java Persistence API（JPA）来进行数据库访问，但只供 Workspace Server 内部使用。

三层的交互主要通过 Restful API 进行，主要为 IDE 访问 Workspace Agent 和 Workspace Server 的 Restful API；Workspace Agent 访问 Workspace Server 的 Restful API。

考虑到源代码分析系统不可避免地会多次读写项目中的源代码，也必将多次访问文件系统；各种配置均与 Workspace 高度相关，存储在 Workspace 的文件中方便和 Workspace 一起通过 Eclipse Che 提供的导入导出功能进行迁移；最终的代码推荐服务器在系统外部，因此留给 Workspace 的用户配置的不会特别多，使用文件系统进行数据存储在效率上也可以接受。综合以上原因后端选择部署在 Workspace Agent 中而非 Workspace Server 中。为了进行界面显示，前端部署在 Eclipse Che 的 IDE 层中。

综合上述插件部署方案和第三章的系统分析结果，本文设计三个模块，分别为：智能化插件管理中心模块，源代码实时分析系统模块和生产率统计模块。三个模块包含 IDE 层子模块和 Workspace Agent 层子模块，其中所有 Workspace Agent 子模块均暴露 Restful API 给 IDE 子模块。系统架构图如图 4.1 所示。图中红框内部，不包括 GWT 和 Eclipse Che IDE 的代码接口，是由本系统实现的部分。在下文中，因为图的大小或标题长度的问题，将时常使用 WsAgent 来代指 Workspace Agent。

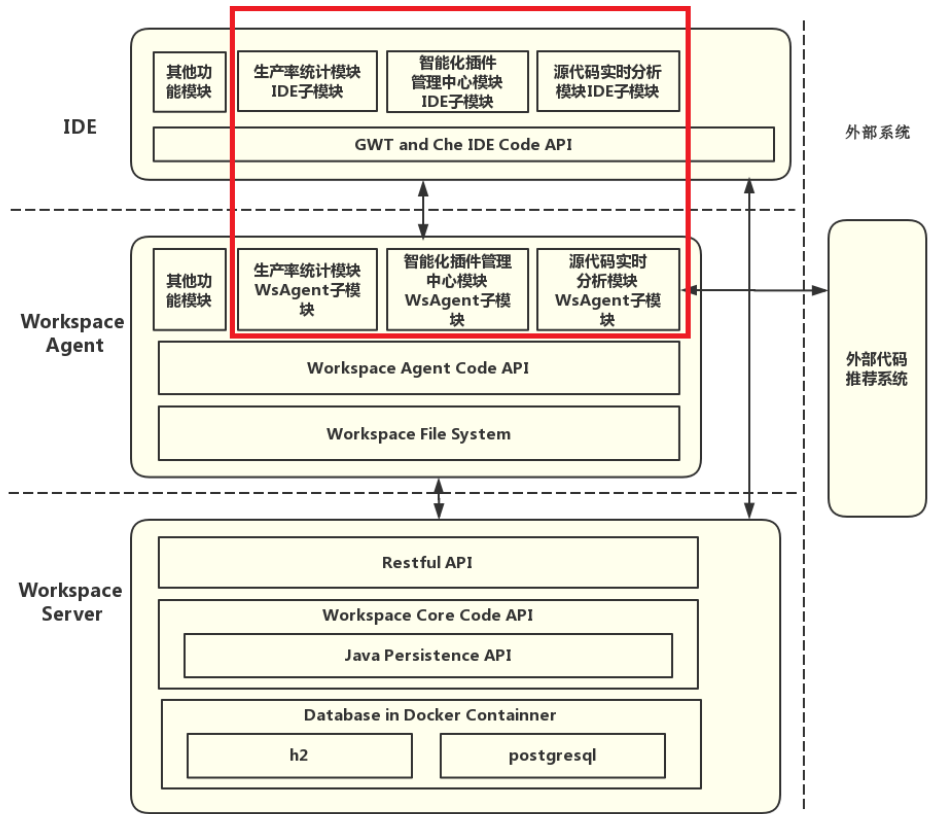


图 4.1 系统架构图

## 4.2 源代码实时分析模块设计

### 4.2.1 源代码实时分析模块 IDE 子模块设计

IDE 模块是直接用户输入进行交互的模块，因此对用户输入的直接缓存和动作判定将设置于此。由于源代码分为多种，有可执行代码、测试代码和项目配置依赖部署管理代码，因此我们对于不同的源代码应采用不同的分析策略。对于可执行代码，我们尝试获取其 API 调用序列以及变量名中有意义的文本，对于项目配置等代码，我们使用其更新代码上下文。

因此，在 IDE 子模块中，设置输入缓存模块，用于缓存开发者的输入；设置分析器调度模块，在活动文件改变时将文件与对应的分析器匹配，在文件打开的或保存的时候请求 Workspace Agent 分析其 API 调用序列；设置分析器工厂，根据当前文件返回分析器实例，每个分析器用于解析具体的某一文件，根据输入决定是否触发代码分析与推荐，请求推荐时输入自上次保存以来用户的修改。针对返回的结果，设置结果解析器，将结果解析为可以被显示的 GWT 元素。同设置

ServiceClient 模块，用于调用 Workspace Agent 的 Restful API；设置结果显示页面用于显示结果；设置远程代码推荐算法选择器，在启动时从 Workspace Agent 获取目前支持的推荐算法服务器，将其显示在结果页面的上方，供开发者选择。协作图如图 4.2，图 4.3：

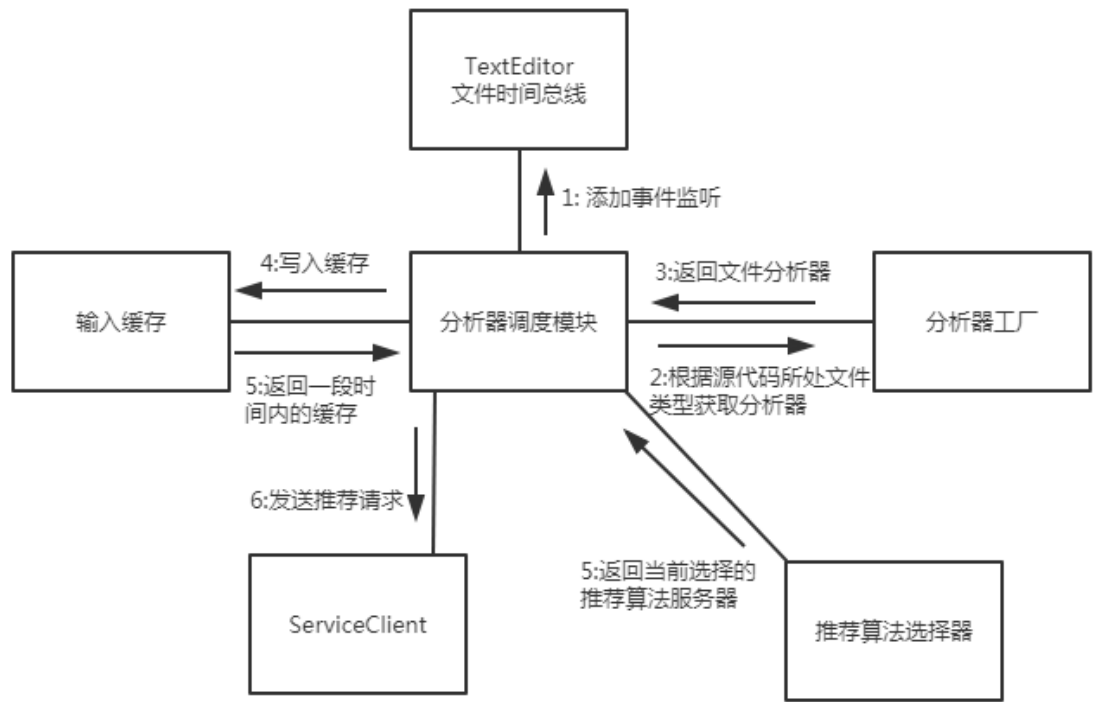


图 4.2 发送推荐请求协作图

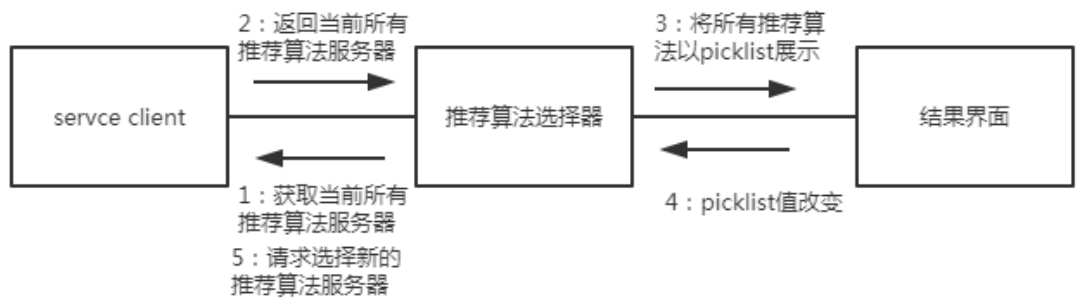


图 4.3 选择新的算法服务器协作图

4.2.2 源代码实时分析模块 WsAgent 子模块设计

由于 IDE 子模块在 GWT 框架下，在开发时可以依赖的库受限，因此语法树解析放在 Workspace Agent 子模块下进行。设计一个异步的语法树解析模块和一个 API 调用序列缓存模块，当收到语法树解析请求时，开启一个线程来解析一段代码的语法树，解析完毕后将转换为 API 调用序列放入缓存。设置算法服务适

配器工厂，用以返回将算法服务器

设置推荐请求控制器，若从 Service Server 处收到推荐请求，尝试从缓存获取已经解析好的 API 序列，若无法获取，则尝试从源码中提取有效文本，同时尝试解析随请求发过来的用户最近输入的文本，生成 API 序列和有效文本集。接着推荐请求控制器从推荐算法服务适配器工厂获取推荐算法服务适配器，用适配器向对应的推荐算法服务器发出请求，获取返回值并通过 Service Server 返回给 IDE 层。

### 4.3 生产率统计模块设计

在软件工程领域，我们可以将生产率的概念其具体化，产出可以使用代码行数来进行度量，而成本可以使用所投入的开发者乘以开发时间。通常情况下，投入的成本以“人月”为单位，而现在本文可以通过编辑器打开的时间来计算一个开发者在当前类型的代码上所消耗的时间。这时，以“人月”为单位会显得过于粗糙，以“人时”或“人分”为单位更加合适。

本文把不同代码的工作量划归为不同的工作类型，具体划分如下：

1. 可执行代码，主要为在 src\main 下的各种语言的可执行代码文件，以后缀名作为识别标志，包括但不限于 .java, .js, .jsp, .html, .go, .py, .c, .cpp, .cs 等。
2. 自动化测试代码，主要包括 src\test 下的 .java 文件。
3. 依赖管理脚本，pom 的 <dependency>, <dependencyManagement> 标签等。
4. 编译部署管理脚本，比如 pom.xml 的 <build> 标签下的内容，makefile 文件等。

#### 4.3.1 生产率统计模块 IDE 子模块设计

设置输入缓存模块，以每种代码对应的每种工作类型进行缓存，记录代码对应工作类型的初始的有效行数，和当前有效行数。如 pom 文件有两种对应的工作类型，依赖管理和编译部署管理，则会产生 2 分缓存，依赖管理缓存的行数指 <dependency> 标签下的代码行数，编译部署脚本缓存的行数为 <build> 标签下的代码行数；不同的 java 文件均属于可执行代码，各自均维持各自的缓存。每个缓存还会缓存存在这个页面上停留的时间，当编辑器切到某个文档的时候，该文档对应的缓存开始累积时间，编辑器离开之后停止计时，若有多个缓存单元则平分缓存时间。一个缓存单元刷新缓存指所用缓存单元将当前行数-初始行数的值和时间返回出来，之后时间清零，起始行数设为当前行数。

设置缓存刷新管理器，监听文件关闭，保存等事件。当有上述事件时，刷新对应缓存单元并将代码行数增量和消耗时间传递给 **Workspace Agent** 进行统计。在缓存刷新管理器中设置心跳发生器，每过一定时间触发一次全体缓存刷新，分类统计代码增量和经过时间，将代码增量和经过时间发送给 **Workspace Agent**。

在代码编辑器打开时，将该编辑器的源代码映射到对应的产出类型中，当编辑器处于活动状态时开始计算时间，并统计这一段时间对应该类型工作的代码行数增量。当编辑器不活动时暂停计时。在编辑器关闭时或自动保存时，将这期间的累计花费时间和产生的代码行数发送回 **Workspace Agent** 进行统计。考虑到整个网页被强行关闭的情况，可能使用心跳机制来维持通讯，并同时借用心跳消息，定时将代码增量发送到 **Workspace Agent** 端。若多次心跳均为 0 代码增量，则进入休眠阶段，不再发送时间，有缓存管理器自行统计；若在稍短的时间内恢复了代码增量，则补发此段时间；若在稍长时间后仍为 0 代码增量，则此段时间不再累加。前者表示开发者临时短时间离开，可能查找资料等；后者模拟开发者下班没有关机，防止时间虚增。

### 4.3.2 生产率统计模块 **WsAgent** 子模块设计

设置接收数据的 **Service Server** 模块和生产率统计模块。生产率统计模块按上述工作分类累加，并写入以日期命名的文件，若有新数据到来则在文件上累加并修改文件内容。

## 4.4 智能化插件管理中心模块设计

智能化插件中心主要负责管理插件的状态。当智能化插件开启或关闭的时需要做两件事，一是将智能化插件的显示端注册到 **IDE** 界面的对应位置，二是通知智能化插件进行相应操作。因此需要为智能化插件提供接口作为模板和用于注册的接口。而各个功能模块的 **Workspace Agent** 资源由 **IDE** 子模块收的释放资源的消息后通过 **Restful API** 通知后端自行释放。

### 4.4.1 智能化插件管理中心模块 **IDE** 子模块设计

在 **IDE** 端，需要设计一个 **IDE** 子模块状态缓存区，注册控制模块，配置项界面以及用于访问 **Workspace Agent** 的 **Service Client**。注册控制模块对外提供注册接口，需要 **Eclipse Che** 提供的由于现实或隐藏页面某部分的接口。配置项界面根将用户的选择传递给注册控制模块，注册控制模块通过 **Service Client** 调用

Restful API 来保存用户的选择。其他功能模块的 IDE 子模块来注册的时候，在没有发现历史记录的情况下，默认模块关闭。当历史记录到来时更新对应历史记录模块的状态；若在已有历史记录的情况下，则直接根据历史记录显示或隐藏对应的界面。整个 IDE 子模块与其他 IDE 子模块以及本身对应的 WsAgent 子模块交互的时序图如图 4.5。

4.4.2 智能化插件管理中心模块 WsAgent 子模块设计

Workspace Agent 子模块比较简单，只需要有暴露 Restful API 接口的服务端和负责将插件状态持久化存储的存储单元。智能化插件管理中心模块 IDE 子模块和 WsAgent 子模块关系如图 4.6 所示。

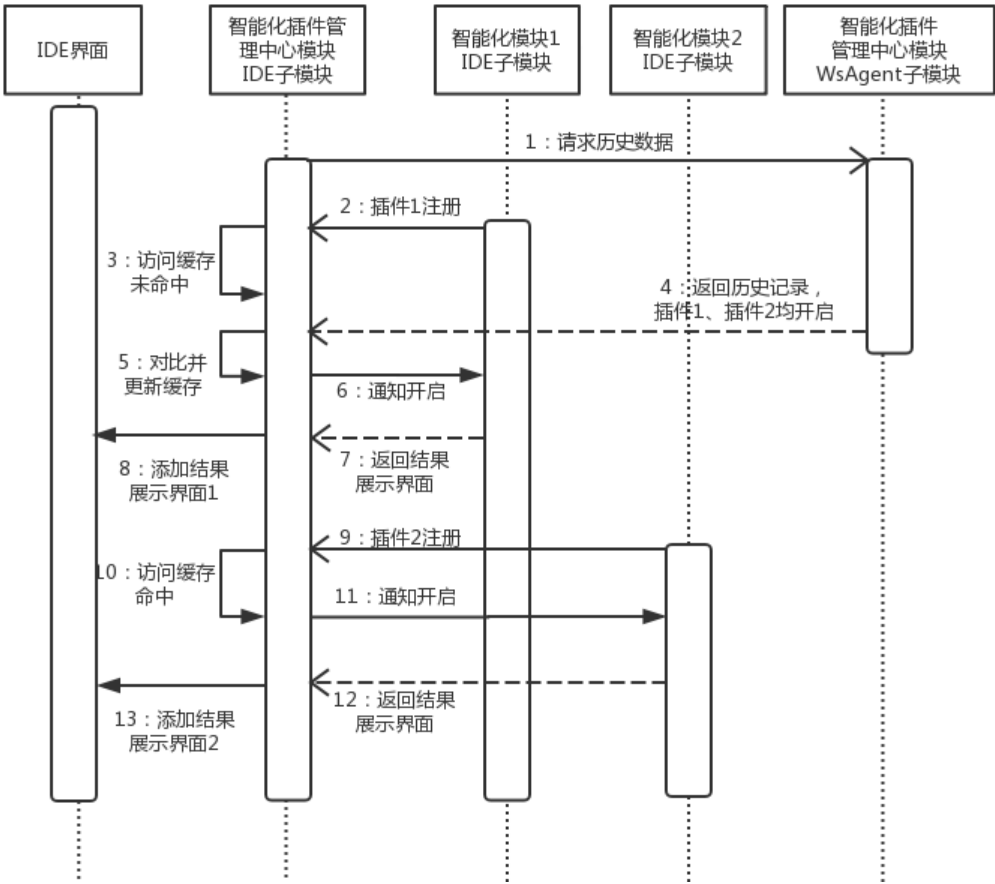


图 4.5 智能化插件管理中心缓存时序图



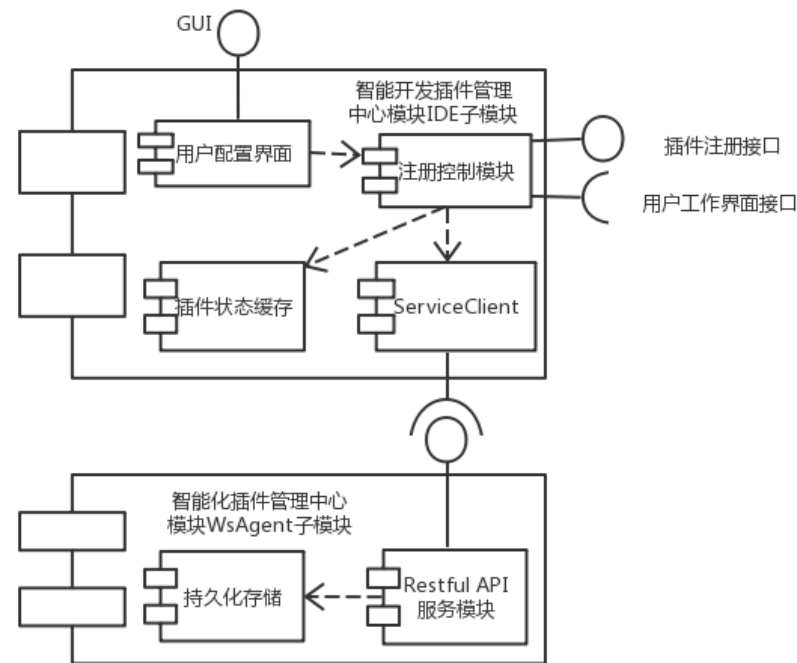


图 4.6 智能化插件管理中心组件图

### 4.5 本章小结

本章首先具体阐述了 Workspace Server, Workspace Agent 和 IDE 三层的联系与区别，并根据系统需求，对系统的总体架构进行了设计。之后详细设计了源代码实时分析插件，生产率统计插件，智能化插件管理中心三个功能模块各自在 IDE 层和 Workspace Agent 层的共计 6 个子模块。以 UML 的形式对部分较为复杂的功能进行了描述，对部分组件结构进行展示。

## 第 5 章 系统实现

### 5.1 开发环境配置

根据 Eclipse Che 的结构和本系统的需求与设计，本文的实现被分为不同的模块。在功能上分为代码分析，生产率统计，功能管理三大模块，而根据 Eclipse Che 架构分为两层，因此共产生了二乘三共计 6 个子模块。要让着 6 个子模块处于正确的位置，而发挥正确的功能，需要如下略微复杂的配置。

#### 5.1.1 相关软件安装

为了运行 Eclipse Che 实例，需要安装 Docker；为了编译打包项目，需要安装 Maven。Java 环境需要 Java 8 以上，因为项目中使用了 Java 8 新增的 Lamda 表达式的功能，Java 7 不支持该功能。

Docker 官网在明显的地方只提供了 Docker for Windows 和 Docker for Mac 下载，若要在 Linux 环境下安装 Docker，需要在 Docker Home Page->Resources->Docs->Get Docker->Docker CE->Linux 寻找对应版本的安装文档。值得一提的是，Docker 在 17.03 之后分化为企业版的 Docker EE 和社区版的 Docker CE，考虑到经济因素本文选择 Docker CE。在 Win 8.1 以及其他某些版本的 Windows，Docker 会要求在 BIOS 中开启 Hyper-V，但之后有仍有可能出现一些系统环境问题，因此强烈不建议在 Windows 上编译部署此项目。

Maven 的安装比较简单，在已经安装 Java 的情况下，下载安装包解压，配置 MAVEN\_HOME，并将%MAVEN\_HOME%/bin 添加到 Path 中即可。因为本文不涉及 Dashboard 部分的编译，因此可以跳过 Eclipse Che 指南中的 GO 语言安装。

#### 5.1.2 项目配置

Eclipse Che 有多种方法可以达到生成带插件的产品这一目的。其中包括 fork 完整源代码的项目，在运行的 Eclipse Che 中 fork 插件项目和在本地 fork 插件项目方法等。经过多次尝试，发现通过完整源代码部署存在配置复杂，部分依赖插件不易找到等问题；在运行的 Eclipse Che 中 fork 插件项目存在资源开销大，系统相应慢和版本不一致等问题（Che 环境会自动拉取最新版本的 che-dev 的 Docker 镜像，而非当前版本）。因此选用本地 fork 插件项目，再进行修改的开发方式。该方式具有资源开销小，项目结构简单，同时版本容易控制等优点。

根据设计需求分析和系统设计,本文需要设计三个插件,分别是源代码实时分析插件,生产率统计插件和智能化插件管理中心(其本身也是作为插件存在)。三个插件均需要在前端进行文件显示和在后端进行文件读写,因此选用 `ide-server-extension` 作为参考模板进行开发。Fork “`che-samples/che-ide-server-extension`”项目到个人仓库下并克隆到本地。克隆的模板项目有两个模块: `plugins` 和 `assembly`。`Plugins` 模块下包含本文开发的各个插件, `plugins` 模块下当前只有一个插件; `assembly` 模块中包含打包生成产品的控制代码,项目模板已经将对插件的添加到对应的包中,需要将 `plugin` 的文件目录名和相关类名改为自己的插件相关名称,这里我的文件改名为 `plugin-intelligent-management-center`,并对应修改其中子模块 `ide`, `server` 的文件名。之后再 `plugins` 模块下新建源代码实时分析插件模块,文件名为 `plugin-recommender-sample`,同样为其新建其 `ide`, `server` 子模块,添加 `pom.xml` 文件,在 `ide` 模块下添加 `gwt` 支持 `module.gwt.xml` 文件。重复这一步,建立 `plugin-productivity-statistician`。

接下来需要调整项目 POM 对象属性。首先需要确保每个文件夹已经添加到上层目录的 `pom.xml` 文件中的 `<modules>` 标签下。项目自带的 `pom.xml` 文件的 `groupId` 为 Eclipse Che 项目模板 `Id: org.eclipse.che.sample`。Maven 依靠 `groupId`, `artifactId` 和版本号来标识确定一个对象。在执行 Maven 命令 “`mvn clean install`” 时,可以将含有自己的扩展的对象发布到本地 Maven 仓库中,后续的打包过程会先从本地仓库中获取对应的扩展,可以获取到我开发的扩展对象并进行打包。但若代码编写出现问题部分编译失败, Maven 则会尝试从远程仓库获取扩展,并继续打包。这样在实例有可能仍然运行成功,但却不包含我的扩展。因此,我将 `groupId` 改为自己的标识,这样在扩展没有成功生成的情况下,后续打包会因找不到 Maven 对象而中断,方便错误的识别。本文共设计了四组 `groupId`,一组供项目根模块、`assembly` 模块和 `assembly` 的子模块下使用的,一组源代码实时分析插件是使用,一组供生产率统计插件使用,最后一组供产检管理模块使用

在此过程中,首先要将各个文件夹下的 `pom.xml` 文件的根节点下的 `<groupId>` 标签和 `<dependency>` 标签下的 `<groupId>` 标签中内容替换为自己的命名空间外,将每个插件的子模块添加到项目根目录的 `pom.xml` 文件的 `<dependencyManagement>` 标签下以及对应的 `assembly` 的子模块的 `pom.xml` 文件中。然后将各个插件的 `ide` 子模块的打包方式 (`<packaging>` 标签) 改为 `gwt-lib`, `server` 子模块的打包模式改为 `pom`。之后还需要将 `assembly-main` 下的 `assembly.xml` 中的 “`<dependencySets><dependencySet><includes><include>`” 标签下的 `assembly-ide-war` 和 `assembly-wsagent-server` 的 `groupId` 换成自己的命名空间。

同时对 assembly-wsagent-server 下的 assembly.xml 也做类似替换。图 5-1 中展示了使用本项目根目录命名空间进行配置的相关内容。

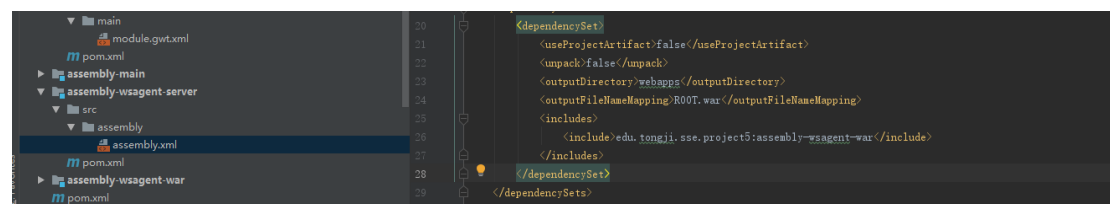


图 5.1(a) assembly-wsagent-server 中 assembly.xml 中需要修改的命名空间

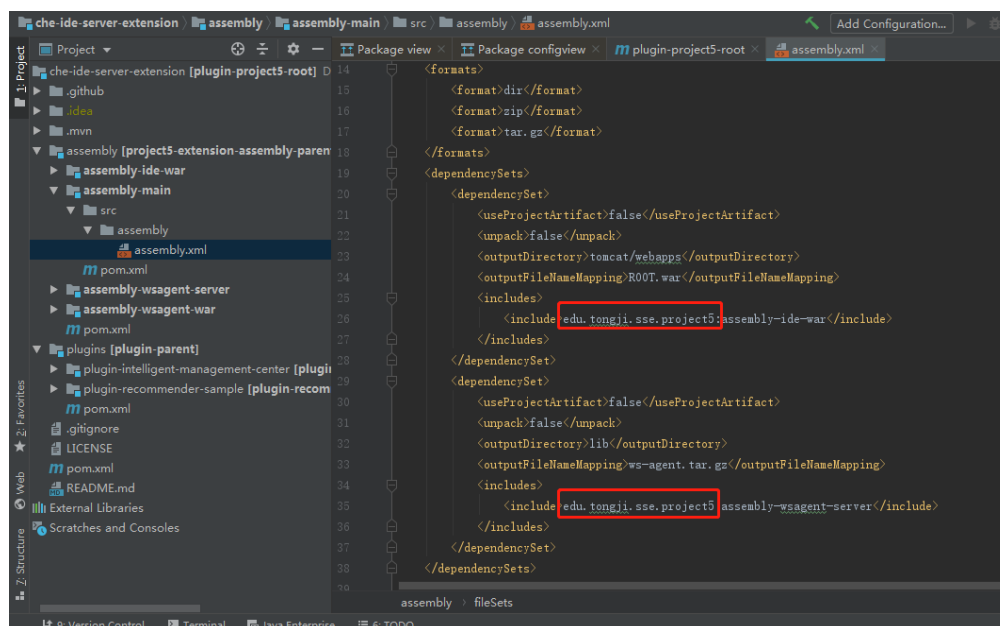


图 5.1(b) assembly-main 中 assembly.xml 中需要修改的命名空间

配置完成后，得到的目录结构如图 5.2，图中各子模块因为篇幅原因未展开，均含有源代码文件和 pom.xml 文件，插件的 ide 模块额外包含资源文件和 module.gwt.xml 文件。

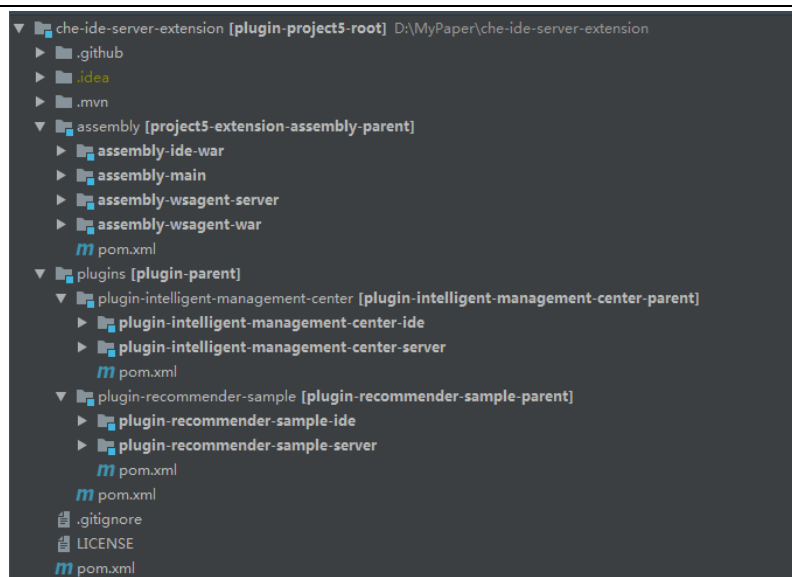


图 5.2 文件目录结构

之后可以在项目跟目录下执行“mvn clean install”命令，可能出现 license 不匹配，pom 文件没有排序，java 文件不符合规范等报错，原因是 Eclipse Che 上层 pom 配置了在 build 过程中相应检查。可以通过执行“mvn license:format”，“mvn sortpom:sort”以及“mvn fmt: format 命令”来格式化相关代码。除此之外，还可以直接在单个插件的根目录的 pom.xml 文件中的<builder>标签下添加上述插件的配置，具体配置如图 5.3 所示。

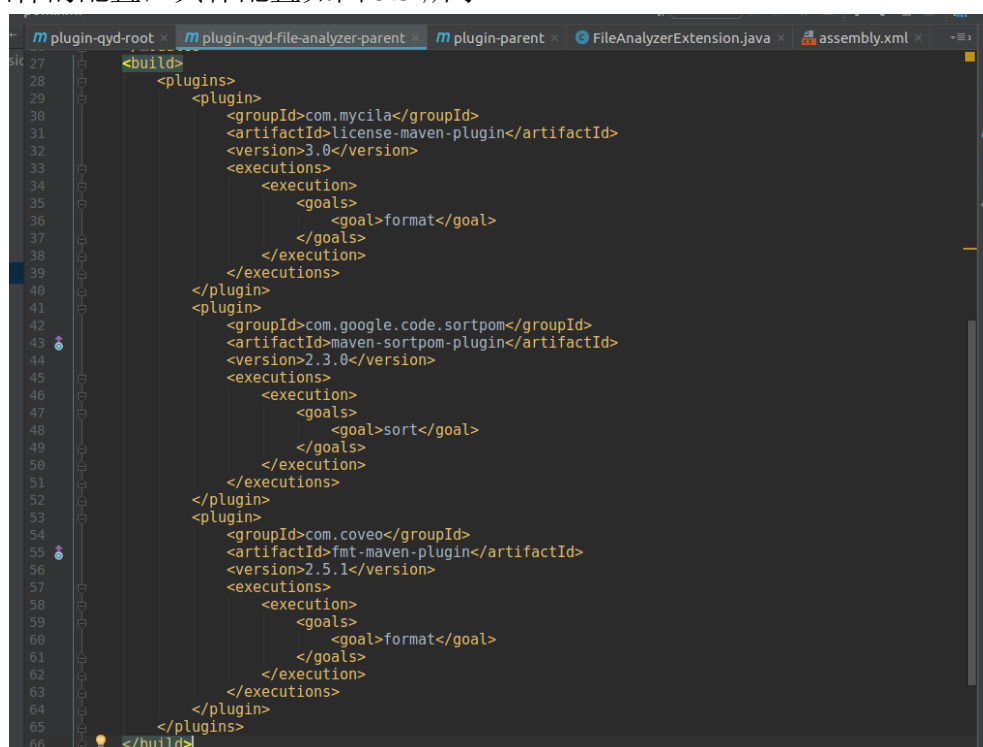


图 5.3 pom.xml 中&lt;build&gt;标签的详细配置

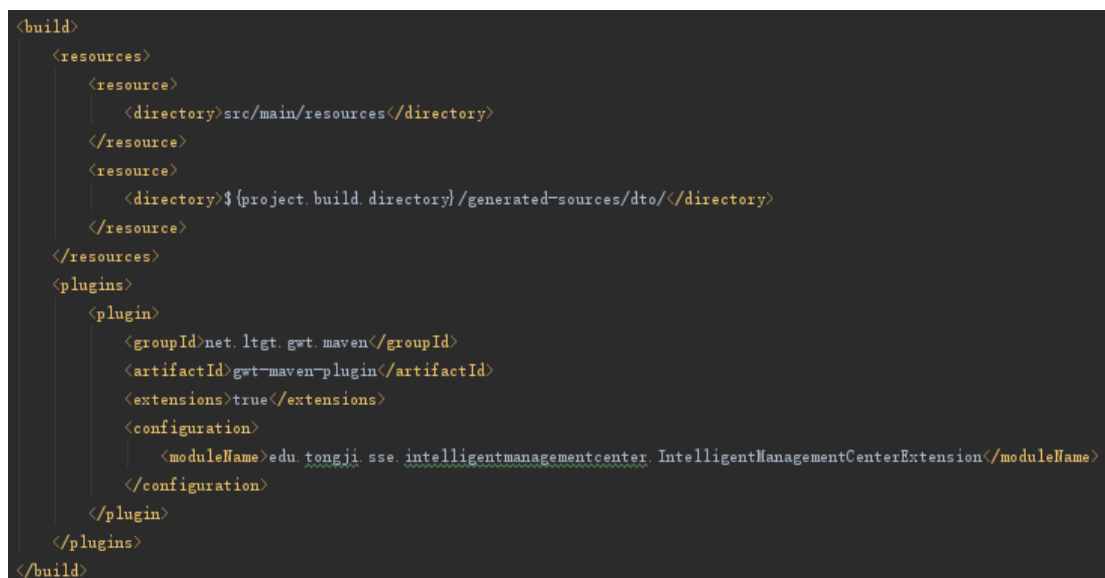
上述为支持项目文件结构的及基本配置，为了支持 GWT 配置，还需进行进一步配置。需要在每个插件的 ide 子模块的 pom.xml 编译过程中添加 GWT 相关过程。具体为添加 net.ltgt.gwt.maven 命名空间下的 gwt-maven-plugin 插件，并在配置中指定模块名称。模块名称的命名空间要与使用 Eclipse Che 的 “@Extension” 注解进行标记的 Java 类的命名空间前一部分保持一致。对于有资源文件需要一起编译的，在<build>标签下添加<resources>标签。

之后配置 src 文件夹的 module.gwt.xml 文件，只包含 ide 模块的 ide 扩展添加<source path=”ide”/>标签，对于要引用到的 share 模块的文件，需要添加<source path=”shared”/>， gwt 将根据此标签指定的路径在上文 pom 的命名空间下寻找扩展的源代码。同时在该文件中，还可以通过<inherits>标签来添加依赖，此处的依赖必须是被编译成 gwt-lib 的项目。这也导致了在前端能使用的项目依赖非常有限，对开发造成了一定困难。



```
<?DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.7.0//EN" "http://gwtproject.org/doctype/2.7.0/gwt-module.dtd">
<module>
  <inherits name="com.google.gwt.user.User"/>
  <inherits name="elemental.Elemental"/>
  <inherits name="org.eclipse.che.ide.Commons"/>
  <inherits name="org.eclipse.che.api.core.model.Model"/>
  <inherits name="com.google.gwt.json.JSON"/>
  <inherits name="com.google.gwt.inject.Inject"/>
  <inherits name="com.google.gwt.user.Debug"/>
  <source path="ide"/>
</module>
```

图 5.4(a) ide 模块下 module.gwt.xml 进行的配置



```
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
    </resource>
    <resource>
      <directory>${project.build.directory}/generated-sources/dto</directory>
    </resource>
  </resources>
  <plugins>
    <plugin>
      <groupId>net.ltgt.gwt.maven</groupId>
      <artifactId>gwt-maven-plugin</artifactId>
      <extensions>true</extensions>
      <configuration>
        <moduleName>edu.tongji.sse.intelligentmanagementcenter.IntelligentManagementCenterExtension</moduleName>
      </configuration>
    </plugin>
  </plugins>
</build>
```

图 5.4(b) ide 模块下 pom.xml 文件中为编译 GWT 进行的配置

在图 5.4 中，通过 pom.xml 文件中<moduleName>标签下的命名空间 edu.tongji.sse.intelligentmanagemnetcenter 加上 module.gwt.xml 文件中<source>标

签下的 `path` 属性组成新的命名空间 `edu.tongji.sse.intelligentmanagemnetcenter.ide`, GWT 在此命名空间下寻找前端扩展入口。同理, 为另两个插件做相同配置。

配置好之后, 在以后进行代码更改时, 直接运行 “maven clean install” 编译即可。在 `plugins` 目录下运行上述命令, 可编译所有新增的插件。在某个插件的目录下运行可编译单个插件。在 `assembly` 或 `assembly` 下运行可使用已经编译好的插件直接编译 `assembly` 组件。在项目根目录下运行可以连续编译插件和组件。插件或组件编译好之后均可在各自的 `target` 目录下找到已经封装好的包。同时程序包也会被添加到本地的 `maven` 仓库中。

### 5.2.3 运行参数配置

普通的 Eclipse Che 通过 Docker 启动时, 需要进行两个绑定, 一是需要将本地的 `data` 目录与 Eclipse Che 容器的 `/data` 目录绑定, 其中本地 `data` 目录可根据需要随机选择, Eclipse Che 会将一些配置文件与 `Workspace` 的数据存储到 `/data` 下。二是需要将本地套接字与 Eclipse Che 的套接字绑定。并且需要以交互模式运行容器并为容器重新分配一个伪输入终端 (参数: `-it`)。官方文档建议启动前清除已经停止的容器 (参数—`rm`)。启动前若通过本地 `assembly` 运行, 则还需要将本地的 `assembly` 文件夹与 Eclipse Che 的 `/assembly` 目录绑定。同时可以根据个人偏好, 将本地 `MAVEN_HOME` 与 Eclipse Che 容器进行绑定, 避免重复下载浪费时间。结合本地环境, Eclipse Che 启动命令为:

```
docker run -it -rm -v /var/run/docker.sock:/var/run/docker.sock -v
/root/che.data.16:/data -v /root/.m2:/home/user/.m2 -v
/root/qyd-ide-server-extension/assembly/assembly-main/target/eclipse-che-6.16.0:/assembly eclipse/che:6.16.0 start
```

## 5.2 IDE 层常用功能的实现

在系统实现过程中, 有些功能在过程上会被重复实现。例如三个 IDE 层模块均需要实现界面, 实现的过程大致一样。但是因为界面细节各不相同, 且界面的某些部分会被用 “@Singleton” 标注为单例, 因此在具体的模块中, 还是各模块分别实现。在本节, 先将这些常用的实现过程统一详细说明。之后为了叙述简洁, 在各个模块叙述的时会简要带过。

### 5.2.1 类的注册与绑定

一个插件，在 IDE 层和 Workspace Agent 层各自有一个 Extension。在 IDE 层，一般会有一个用 “@Extension” 标注的 Extension 类作为插件入口，负责注册或初始化对象和变量，这个类在上文配置过程中也提到过。一般一个 Extension 中会包含一个或多个 Action，负责实际功能。同时也会存在一个以 “@ExtensionGinModule” 标注的 GinModule 类来将接口和实现绑定，下文界面的 View 接口和 ViewImpl 将通过此类进行绑定。在 Workspace Agent 层，会有以 “@DynaModule” 来进行类的注册与绑定。在访问 Restful API 时 IDE 端可以通过 Eclipse Che 提供的 AsyncRequestFactory 工厂类提供的接口添加参数进行访问。

### 5.2.2 输入缓存实现

无论是生产率分析插件还是源代码实时分析插件，都需要输入缓存。Eclipse Che 通过 Google @Inject 注解来向插件中的类的构造方法中注入参数实例的。借助于此，在代码中很容易获得 EventBus 的实例，EventBus 中会传递各种扩展了 GWT 中抽象类 Event 的 Event 实例。开发者可以通过向其中添加事件监听类并实现其中的回调函数来处理各个类。

但是 EventBus 中并无法获得源代码文本变化的具体内容。在编辑器文本改变时只能获得对应编辑器是否存在脏数据的状态和当前光标的位置。要获得具体的输入字符串，需要获取编辑器本身的文本改变值，这样就需要获取 Editor 本身的 DocumentEventBus。获取 DocumentEventBus 需要 TextEditor 接口中的方法而 DocumentChangeEvent.getEditor 的返回类型 EditorPartPresenter 是 TextEditor 的父类，因此需要强制转换成 TextEditor 接口。转换之前建议判断编辑器是否实现了 TextEditor 接口，因为目前 Eclipse Che 只支持文本编辑器，但无法确定在将来其是否会支持其他编辑器。

最终具体实现为，在注入的 EventBus 中监听 EditorOpenedEvent 事件，在事件触发时获取打开的 EditorPartPresenter 转换为 TextEditor，从 TextEditor 中获取 DocumentEventBus 并监听其中的 DocumentChangeEvent 事件。当事件触发时，从 DocumentChangeEvent 事件实例中可以获取文本修改的相关性信息。建立一个 DocumentTextChangeSection 类，用以存储连续更新的代码片段。



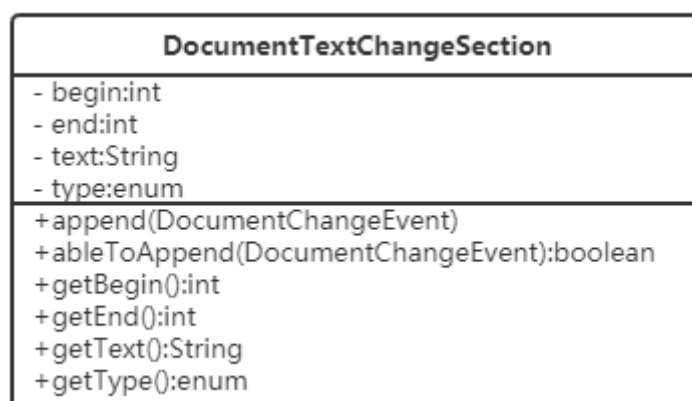


图 5.5 DocumentTextChangeSection 类图

其属性 `begin`, `end` 为当前修改的起点和重点, `text` 为文字内容, `type` 为 ADD 或 DELETE 的枚举类。从上文中的 `DocumentChangeEvent` 中, 可以获取到某次修改的起点, 终点和修改的类型与字符串。通过 `ableToAppend`, 判断出当前修改是否可以与之前的修改拼接, 若可以拼接, 通过 `append` 方法将本次修改的内容拼接上去, 否则新建 `DocumentTextChangeSection` 开始新的缓存片段。建立 `Map<String, Queue<DocumentTextChangeSection>>` 的实例来实现文件 URI 与其自身的输入缓存队列的映射。

### 5.2.3 界面实现

无论何种插件, 都需要一个界面来展示输出, Eclipse Che 使用 GWT 框架作为前端, 其框架已经固定, 在此我简要提出开发时的注意事项。

GWP 页面采用 MVP 模型, 分为 Model, View, Presenter。View 和 Presenter 互相交换数据, Presenter 和 Model 之间交换数据, Model 和 View 之间不直接交换数据。Eclipse Che 中 Model 的任务一般由访问 Workspace Agent Restful API 的 `ServiceClient` 承担, 一般并不和 View 放在一个包下。一般 Eclipse Che 中一个界面组件有 View 接口, `ViewImpl` 类, Presenter 类和 `ViewImpl.ui.xml` 页面布局文件组成。其具体关系如图 5.6。其中 `ViewImpl` 和 Presenter 的互相关连关系为, Presenter 在初始化时, 会通过注入, 获取接口 View 的实例, 接口 View 和 `ViewImpl` 类在 GWT 的 `GinModule` 中互相绑定, 因此会获得 `ViewImpl` 的实例, Presenter 通过调用 `ViewImpl` 的方法实现操控页面; Presenter 还会通过 `setDelegate` 方法将自身传入 `ViewImpl`, `ViewImpl` 通过调用自身成员 `delegate` 的方法实现将页面上的数据变化传给其他组件。

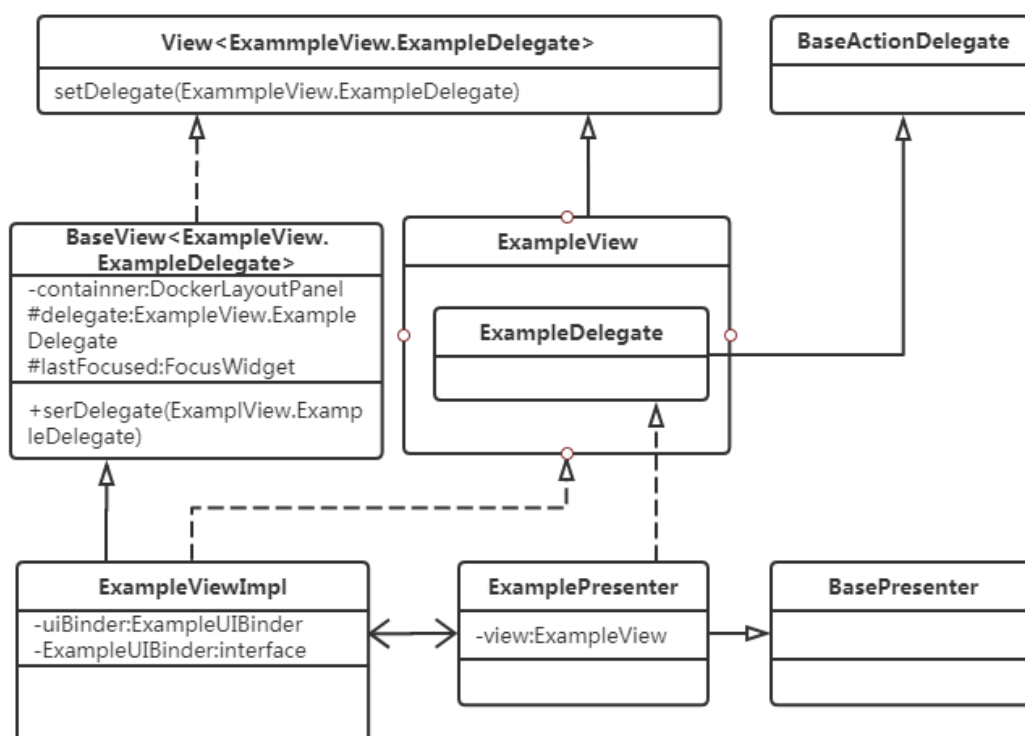


图 5.6 GWT 界面组件类图

**ViewImpl** 中包含一个内部接口，即图中的 **ExampleUI**，该接口扩展 **UiBinder<T1,T2>**，其中 **T1** 为 **ViewImpl.ui.xml** 中 **uibinder** 标签下根节点的类型，**T2** 为 **ViewImpl** 自身类型。**ViewImpl** 还在自身添加一个以 **ExampleUIBinder** 接口为数据类型的成员，这个接口可以通过 **GWT.create(ExampleUIBinder.class)** 来直接获取匿名实例，将 **ViewImpl** 与页面结构绑定并获取对应根节点。同时还可以在 **ViewImpl** 中使用 **@UIField** 标签加上和 **ViewImpl.ui.xml** 中标签类型一致，名称与标签 **ui:field** 一致的成员，直接访问 **ViewImpl.ui.xml** 中的元素。

在本文的三个插件中，均需要动态添加标签，因此需要使用 **panel** 作为容器。需要注意，部分 **panel** 扩展了 **simplePanel**，只能有一个子节点；其他部分 **panel** 扩展了 **complexPanel**，可以拥有多个子节点。在向某个节点下添加子节点时，尽量不要使用 **getElement().appendChildren(Element)** 方法，这种方法虽然可以正常展示元素内容，但监听函数不会被添加，因此强烈建议使用 **GWT** 框架下类自身的方法。

## 5.3 源代码实时分析模块实现

### 5.3.1 源代码实时分析 IDE 子模块实现

在 IDE 层，本插件创建了一个 `AbstractIntelligentPluginAction` 的子类 `AnalyzerAction`，按照下文 5.5 中的规范，注册到 `IntelligentPluginManager` 中。按照 5.2.2 中的方法监听活动的 Editor 中的输入事件。创建一个或多个内置的代码推荐结果解析服务的适配器，用于显示推荐结果。

当项目初始化时，若该 `Workspace` 的 `AutoSave` 功能没有开启，则通过 `Get` 方法从 `Workspace Agent` 中获取该 `Workspace` 的各个 `Project` 的上下文。当从 `EventBus` 监听到 Editor 启动的 `EditorOpenedEvent` 时，判断是否为 Java 的可执行源代码，若是，通过 `POST` 方法向 `Workspace Agent` 申请进行语法树分析。

`AnalyzerAction` 中添加枚举类型成员变量，来记录当前的状态，当监听到 `ActivePartChangeEvent` 时，判断新的 `Part` 是否为 `TextEditor`，若不是，保持原状态不变；若是则分析器所包含的 `URI`，判断其中内容是资源配置源代码还是可执行源代码。

当前代码若为资源配置源代码，在自动存储模式下，收到 `EditorDirtyStateChangedEvent`，Editor 没有脏数据且所有配置标签前后配对时，通过 `POST` 请求来让资源配置文件更新；在非自动存储模式下，通过正则匹配找到关注的标签，如 `<dependency>` 标签，`<build>` 标签等，通过从 Editor 获得到的输入缓存的位置判定其具体属于哪个功能块，在缓存内容具有完整标签结构之后，并分析出其具体内容，通过 `POST` 请求添加到 `Workspace Agent` 暂存的文件上下文信息中。

当前代码若为可执行源代码，在源代码加载时对源代码的逐行扫描，以及正则匹配拆分，对于变量名按照命名规则拆成单词，并进行词频统计，从而生成一个简单的 `API` 名称序列和代码中有意义的单词集。当开发者有输入未保存时，从开发者的输入缓存进行同样的操作，找到对应的有效的 `API` 调用，和有意义额单词集，在尝试获取推荐结果时，不拼接在一起，分开传给 `Workspace Agent`。

除此之外，依照 5.2.3 提供一个配置界面，供开发者选择内置的代码推荐服务器。

### 5.3.2 源代码实时分析 Workspace Agent 层实现

新建 `AnalyzerService` 类负责暴露 Restful `API` 接口。暴露“`/buildAST`”的 `Post` 接口，传入文件路径以及当前时间戳，进行语法树解析。因为 `Eclipse Che` 本身在编译时调用另外的 `Language Server`，`AST` 不暴露在外，因此本文引入项目“`INRIA/spoon`”<sup>[37]</sup>进行对语法树的解析，因为语法树解析需要一定能时间，因此创建线程进行分析，并将结果保存到 `Map<String, CtClass>` 中，`String` 为目录名，`CtClass` 为 `spoon` 解析出的语法树的根节点。同时用 `Map<String, Date>` 保存当前

语法树和时间戳的关系。

暴露 POST “/appendContextInfo” 和 “/refreshContextValue” 用于 IDE 前端在资源配置源代码分析模式下对项目上下文信息进行的临时添加，或请求 Workspace Agent 刷新项目上下文。提供文件的 URI 作为请求参数，使 Workspace Agent 可以更新部分文件对应的配置块。

暴露 POST “/selectRecommendServer” 接口，用于让开发者选择用于推荐的代码推荐服务器，并将此配置保存。

暴露 GET “/getRecommendation” API 来启动向远程代码推荐服务器的代码推荐请求。会收到 IDE 传来的源代码初始的 API 调用序列、有效词，用户最近输入的 API 调用序列、有效词集。检查当前文件的语法树是否成功生成，是否过期。若没有成功生成，则用源代码的初始的 API 调用序列代替，并添加用户最近输入的 API 调用序列和有效词；若有过期的语法树，则在使用语法树的同时添加用户最近输入的 API 序列和有效词；若语法树最新，则直接使用语法树。

Workspace Agent 建立与 IDE 对应的代码推荐服务适配器，将上述生成的内容转化成服务器可接受的数据格式。

### 5.3.3 推荐结果显示实现

在收到以后，IDE 层根据上文选择的内置的适配器分析返回值。对于其中的代码，新建 TextArea 对象，将内容放入，增加 “Accept” 按钮，绑定 onClick 事件将代码片段添加到当前光标位置。将每条返回结果放入 FlowPanel 中，作为一行返回结果，并放入由 ScrollPanel, VerticalPanel 封装好的结果显示区。

## 5.4 生产率统计模块实现

### 5.4.1 文件类型定义

根据之前的研究，本文把不同代码的工作量划归为不同的类型，分为如下类型：

5. 可执行代码，主要为在 src\main 下的各种语言的可执行代码文件，以后缀名作为识别标志，包括但不限于 .java, .js, .jsp, .html, .go, .py, .c, .cpp, .cs 等。
6. 自动化测试代码，主要包括 src\test 下的 .java 文件。
7. 依赖管理脚本，pom 的 <dependency>, <dependencyManagement> 标签等。
8. 编译部署管理脚本，比如 pom.xml 的 <build> 标签下的内容，makefile 文

件等。

目前，本插件支持以 Maven 管理的 Java 项目，产出量以产出代码行数来计算，消耗成本以“人·分”来计算，默认一个 web IDE 端只有一人进行工作。

#### 5.4.2 功能实现

在 Workspace Agent 层，本插件在 WorkspaceAgent 端通过 java.ws.rs 暴露一个“/getUniqueId”的 Restful API，每次通过 Get 请求获取一个唯一 Id。此举防止在单用户模式下无法分清是哪个用户在进行工作。在单用户模式下，IDE 端插件初始化时，尝试从 Workspace Agent 的“/getUniqueId”并临时保存，在 IDE 结束之前一直使用这个 Id 作为身份标识符与 Workspace 通讯。若为多用户模式，则使用 User Id 作为唯一 Id。另外，也将暴露一个 POST 接口，供前端写回统计数据用，该 POST 接口需要 userId（或上文的 UniqueId）、距上一次消息经过的时间、这段时间内的代码行数增量以及代码所属的具体类型。

在 IDE 层，本插件创建了一个 AbstractIntelligentPluginAction 的子类 ProductivityAction，按照下文 5.5 中的规范，注册到 IntelligentPluginManager 中。本插件设置了一个 ProductCostStatistician 类，其实例负责统计当前用户在某种文件上花费的时间与产出。其私有成员 productType 表明其属于上述哪种产出。在 ProductivityAction 初始化时监听 EventBus，与此同时初始化一系列的 ProductCostStatistician 的匿名子类的实例，每个实例用来统计一类文件的某种工作的产出的代码行数和所消耗的时间。其中 isFileInChargedByThisStatistician 用以根据文件路径名确定文件是否归属这个实例负责。在编辑器切换的时候，新的活跃编辑器逐个询问“统计员”是否负责统计新的活跃编辑器对应的文件，若是，则将自身通过 addEditor 方法传入“统计员”，“统计员”根据自身的设以及 5.2.2 中提到的方法监听 TextEditor。在开发者编辑过程中记录工作量。

ProductivityAction 也将监听 EventBus，在当前活跃的编辑器变更、文件保存或文件关闭时，从活跃的“统计员”处获取修改的代码行数作为工作输出，由自身记录从上次提交生产率信息的时刻到现在经过的时间，加上自己一直保存的 Id，以 POST 请求方式向 Workspace Agent 写入记录，在成功后清零统计员的记录，重新统计。若是文件自动保存则只清零记录，若是活跃编辑器变更，则再成功提交生产率消息之后重新选择“统计员”。

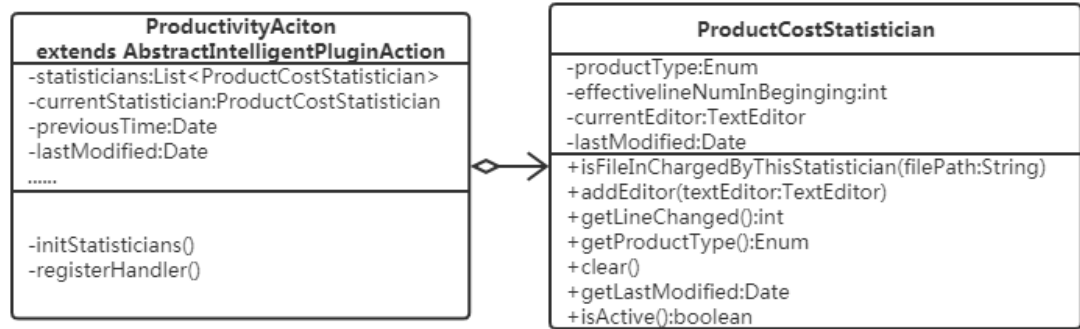


图 5. ? ProductCostStatistician 类图

### 5. 4. 3 心跳机制

为了防止因为包括网络在内的各种因素，使 IDE 端与 Workspace Agent 端的链接突然中断，导致大量缓存在 IDE 端的生产率数据丢失，并为了检测开发者本身的开发状态，IDE 端扩展将使用 GWT Timer scheduleRepeating 以 60 秒为周期定时触发上述的消息提交。提交的消息仍为正常的生产率信息。

对于心跳机制来说，有可能发生开发者下班或长时间离开之后长时间没有关闭浏览器的情况。这样会造成有很多空有时间消耗，但没有代码行数产出的消息，使生产率不正常地大幅下降。因此，在插件中加入 **ProductivityAction** 和 **ProductCostStatistician** 中加入 `lastModified` 成员变量，用于记录由开发者进行的最后一次操作的时间。若心跳消息发送时，距所有 `lastModified` 均已超过 10 分钟，记录这次心跳消息的时间。则记下当前时间，作为最后一次有效的心跳时刻，下次提交心跳消息时，不再记录消耗时间。之后若出现因开发者操作而出发的消息（如切换编辑器，输入文本并保存等）或心跳机制发现有编辑器重新处于活动中，若距最后一次有效时间未超过 1 小时，则补全这一段时间的记录，代码产出仍为 0 行，补充的时间记为距上次有效心跳时刻的时间。若超过 1 小时，则不记录这段时间，若有新增代码，以一个心跳周期作为这次消息的时间。

需要补充说明的是，触发 `lastModified` 更新的消息是编辑器中 **DocumentEventBus** 中的事件，可能是光标的移动，鼠标的点击等，并不一定会增加代码行数。

## 5. 5 智能化插件管理中心模块实现

### 5. 5. 1 智能化插件管理中心模块 IDE 子模块实现

本插件的核心内容是一个插件管理器。在 IDE 层，首先初始化一个插件管

理器的 Action, 从 Workspace Agent 端获取创建者的用户 Id 与当前的用户 Id 做对比, 若一致, 则将上一步创建的 Action 注册到 Workspace 菜单下, 其菜单项被点击时, 调用界面展示。插件定义了 IntelligentPluginManager 接口, 用 IntelligentPluginManagerImpl 进行了实现并使用 “@Singleton” 进行了注解, 在 GinModule 中将接口和实现记性绑定。其他类需要使用插件管理系统的时, 可以通过 @Inject 注解, 将 IntelligentPluginManager 作为构造函数参数传入, 则可获得唯一的 IntelligentPluginManagerImpl 实例。

插件中声明了抽象类 AbstractIntelligentPluginAction, 该抽象类扩展了 Eclipse Che 的 BaseAction 类, 作为各个插件前端的模板, 为各个中的 Action 提供设计规范。其中声明的主要方法有: getResultPresenter, 获取插件 Action 的结果界面; getResultPresenterPartStack, 获取用于展示结果界面的位置; getConfigWindow, 用于获取插件的配置界面。当有外部插件通过 IntelligentPluginManager 中的 registerPlugin(String, AbstractIntelligentPluginAction) 进行注册时, 插件的名称和其实例会被放入 Map<String, AbstractIntelligentPluginAction> 进行保存, 并且会在 Map<String, Boolean> 中查找其状态, 若其已经有被开启历史记录 (在之前 Workspace 启动时被开启), 则调用 setEnable 接口通知插件开启各种事件监听, 并将其界面添加到 Workspace 工作区中。

在 IntelligentPluginManager 初始化时, IntelligentPluginManager 会先尝试从 Workspace Agent 中获取上次的插件状态记录, 但是从 WorkSpace Agent 返回的数据可能会晚于 Intelligent 本身的注册, 因此状态记录返回时, 将检查当前的插件状态, 若当前插件状态和返回值不同, 将更新插件记录状态的同时更新插件的结果窗口的显示状态。这个功能被单独封装为一个方法, 也供插件配置页面结果保存时使用。

插件配置页面扩展的是 Eclipse Che 命名空间下的抽象类 Window 而非 BasePresenter, 但其余与 5.2.3 无太大差别。遍历 Map<String, Boolean>, 用 FlowPanel 作为一行, 每一行放入一个 Checkbox, 用 Map 中的 Boolean 初始化设置其 checked 的值, 来表示其是否插件的状态。之后放入以表示插件名字的 Label, 再从 Map<String, AbstractIntelligentPluginAction> 中查询对应的 Action 是否含有 ConfigWindow, 若有, 则生成一个可以弹出 ConfigWindow 的按钮。最后将所有行加入当前插件管理窗口。

### 5.5.2 WsAgent 层实现

Workspace Agent 暴露了 “/intelligentPluginAvailability” 的 GET 和 POST 接口, 用户获取和存储当前的配置。插件状态的记录文件被存在 Workspace 根目录

下的隐藏目录“.che”下。

## 5.6 本章小结

本章介绍了本系统的实现方案。首先介绍了搭建本系统环境的配置方法。然后介绍了三个插件共同所必须实现的基本功能，包括接口绑定与访问，用户输入缓存，界面显示。在上述功能基础上，进一步介绍了三个插件的各自功能的实现。

本章的实现可能有些繁琐，主要是因为目前关于该领域的研究和文档不多，除了官方文档基本没有文档参照；项目收到 Eclipse Che 自身架构和 GWT 的双重限制，需要在有限的框架内完成相应的功能。因此虽然功能虽然听上去简单，但有相当大的实现难度。



## 第 6 章 测试

### 6.1 源代码实时分析模块测试用例

Eclipse Che 并没有提供测试的相关文档，但是参照 Eclipse Che 源码下其他的插接，发现部分插件使用 Junit 和 Mockito 进行测试

### 6.2 生产率统计模块测试用例

首先进行

### 6.3 智能化插件管理中心

### 6.4 本章小结

## 第 7 章 结论与展望

### 7.1 结论

本文根据项目组对于项目展示的需求,结合我在研究生期间的工作,设计了三个基于云端集成开发环境 Eclipse Che 的插件,设计了在实时编写源代码的情况下简单解析 API 调用序列和源代码中关键字的解决方案和基于 Eclipse Che 来计算生产率的解决方案。三个插件均完成了最终设计的目的。实时文件解析在已经生成语法树和没有生成语法树的情况下都能完成文件的分析;生产率统计插件可以准确地、直观地实时搜集信息,计算并显示生产率;智能开发插件管理中心可以有效地整合各个智能化插件。本文提出的解决方案可被应用于实际,为后续开发提供有力的参考。

### 7.2 展望

目前 Eclipse Che 还有许多不成熟的地方,开发文档不够完全,有些功能不向下兼容。但是 Eclipse Che 的确在不断尝试,逐渐变好,在这三年的追踪过程中,也发现其功能越来越强大,淘汰了一些鸡肋的功能。希望随着其发展,与 API 的完善,本文的三个插件可以提供更多功能,项目组其他以本文插件为模板的新插件可以在功能上,界面美观性上有提升。

在本文之前,关于软件生产率的研究被提出了,但是不是很完善。通过本文这个插件,我们将难得地在生产环境中实时观测到生产率的变化,希望这些数据对以后进一步研究提供帮助。

## 致谢

历经

2019 年 5 月

## 参考文献

- [1] Raychev V, Vechev M T, Yahav E, et al. Code completion with statistical language models[J]. programming language design and implementation, 2014, 49(6): 419-428.
- [2] Ghafari, Mohammad & Moradi, Hamidreza. (2017). A Framework for Classifying and Comparing Source Code Recommendation Systems. . 10.1109/SANER.2017.7884674.
- [3] 聂黎明, 江贺, 高国军,等. 代码搜索与 API 推荐文献分析[J]. 计算机科学, 2017, 44(s1):475-482.
- [4] Zhong H, Xie T, Zhang L, et al. MAPO: Mining and Recommending API Usage Patterns[M]// ECOOP 2009 – Object-Oriented Programming. Springer Berlin Heidelberg, 2009:318-343.
- [5] Bajracharya S K, Ossher J, Lopes C V. Leveraging usage similarity for effective retrieval of examples in code repositories[J]. 2010.
- [6] Mcmillan C, Poshyvanyk D, Grechanik M, et al. Portfolio:Searching for relevant functions and their usages in millions of lines of code[J]. Acm Transactions on Software Engineering & Methodology, 2013, 22(4):1-30.
- [7] Keivanloo I, Rilling J, Zou Y. Spotting working code examples[C]// Proceedings of the 36th International Conference on Software Engineering. ACM, 2014:664-675.
- [8] Raghothaman M, Wei Y, Hamadi Y. SWIM: Synthesizing What I Mean[J]. Computer Science, 2015, 3(1).
- [9] Jiang H, Nie L, Sun Z, et al. ROSF: Leveraging Information Retrieval and Supervised Learning for Recommending Code Snippets[J]. IEEE Transactions on Services Computing, 2017, PP(99):1-1.
- [10] Li X, Wang Z, Wang Q, et al. Relationship-aware code search for JavaScript frameworks[C]// ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2016:690-701.
- [11] Wang S, Lo D, Jiang L. Active code search: incorporating user feedback to improve code search relevance[C]// ACM/IEEE International Conference on Automated Software Engineering. ACM, 2014:677-682.
- [12] Lu M, Sun X, Wang S, et al. Query expansion via WordNet for effective code search[C]// IEEE, International Conference on Software Analysis, Evolution and Reengineering. IEEE, 2015:545-549.
- [13] Lemos O A, De Paula A C, Zanichelli F C, et al. Thesaurus-based automatic query expansion for interface-driven code search[C]. mining software repositories, 2014: 212-221.
- [14] Nie L, Jiang H, Ren Z, et al. Query Expansion Based on Crowd Knowledge for Code Search[J]. IEEE Transactions on Services Computing, 2016, 9(5): 771-783.
- [15] Lv F, Zhang H, Lou J, et al. CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E)[J]. automated software engineering, 2015: 260-270.
- [16] Subramanian S, Inozemtseva L, Holmes R, et al. Live API documentation[C]. international conference on software engineering, 2014: 643-652.
- [17] Moreno L, Bavota G, Penta M D, et al. How can I use this method[C]. international

- conference on software engineering, 2015: 880-890.
- [18] Lemos O A, Bajracharya S K, Ossher J, et al. A test-driven approach to code search and its application to the reuse of auxiliary functionality[J]. Information & Software Technology, 2011, 53(4): 294-306.
- [19] Stolee K T, Elbaum S G, Dobos D, et al. Solving the Search for Source Code[J]. ACM Transactions on Software Engineering and Methodology, 2014, 23(3).
- [20] Inoue K, Sasaki Y, Xia P, et al. Where does this code come from and where does it go? - integrated code history tracker for open source systems -[C]. international conference on software engineering, 2012: 331-341.
- [21] Thung F, Wang S, Lo D, et al. Automatic recommendation of API methods from feature requests[J]. automated software engineering, 2013: 290-300.
- [22] Gu X, Zhang H, Zhang D, et al. Deep API learning[J]. foundations of software engineering, 2016: 631-642.
- [23] Niu H, Keivanloo I, Zou Y, et al. Learning to rank code examples for code search engines[J]. Empirical Software Engineering, 2017, 22(1): 259-291.
- [24] Rahman M M, Roy C K, Lo D. RACK: Automatic API Recommendation Using Crowdsourced Knowledge[C]// IEEE, International Conference on Software Analysis, Evolution, and Reengineering. IEEE, 2016:349-359.
- [25] Johannes Wettinger, Vasilios Andrikopoulos, Frank Leymann, Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Application. Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), 2015, pages 60-65, IEEE Computer Society.
- [26] J. Wettinger, U. Breitenbücher, and F. Leymann, "DevOpSlang - Bridging the Gap Between Development and Operations," in Proceedings of the 3rd European Conference on Service-Oriented and Cloud Computing (ESOCC), 2014.
- [27] M. Hußtermann, DevOps for Developers. Apress, 2012.
- [28] S. Nelson-Smith, Test-Driven Infrastructure with Chef. O'Reilly Media, Inc., 2013.
- [29] C. A. Cois, J. Yankel and A. Connell, "Modern DevOps: Optimizing software development through effective system interactions," Professional Communication Conference (IPCC), 2014 IEEE International, Pittsburgh, PA, 2014, pp. 1-7.
- [30] [https://en.wikipedia.org/wiki/Eclipse\\_Che](https://en.wikipedia.org/wiki/Eclipse_Che)
- [31] <https://github.com/eclipse/che-archetypes/tree/che6>
- [32] 刘国乐,余彦峰.浅析 Docker 容器技术[J].保密科学技术,2017(10):26-30.
- [33] 李娜.Docker 容器技术的发展及应用研究[J].数字技术与应用,2018,36(11):95-96.
- [34] 江日念, 林霞, 乔德新. Maven 在 Java 项目中的引入及应用[J]. 电脑知识与技术, 2013(21).
- [35] <http://maven.apache.org/plugins/index.html>
- [36] <http://www.gwtproject.org/overview.html>
- [37] Pawlak R, Monperrus M, Petitprez N, et al. SPOON: A library for implementing analyses and transformations of Java source code[J]. Software Practice & Experience, 2016, 46(9):1155-1179.

## 个人简历、在读期间发表的学术论文与研究成果

### 个人简历:

秦乙丹, 男, 1993 年 4 月生。

2016 年 7 月毕业于同济大学 软件工程专业 获学士学位。

2016 年 9 月入同济大学读硕士研究生

### 已发表论文:

[1] Qin Liu , Yidan Qin , Hongming Zhu , Hongfei Fan. Software Productivity in DevOps.  
Journal of Software, 2019, Vol.14(3): 129-137